Tape migration and recall

castor development team

October 8, 2007

Reference Guide

Issue :	1
Revision :	0
Reference :	http://www.cern.ch/castor
Created :	March 02, 2005
Last modified :	October 8, 2007

Contents

1 Who should use this guide **3**

2 How this guide is organized **4**

3 Terminology **5**

4 Overview of tape migration and recall in CASTOR **6**

4.1 Components 6

4.2 Handling of multiple requests for same volume 6

4.3 Interaction with other CASTOR services 7

5 Detailed description **9**

5.1 rtcpclientd 9

5.1.1 Work flow 9

5.1.2 Log messages 11

5.2 migrator 13

5.2.1 Signature of a migration candidate 14

5.2.2 Work flow 15

5.2.3 Dual copies 17

5.2.4 Log messages 17

5.3 recaller 19

5.3.1 Origin and signature of a recall candidate 19

5.3.2 Work flow 21

5.3.3 Segment checksum verification 23

5.3.4 Log messages 23

5.3.5 System messages and warnings 23

5.4 MigHunter: the default migration candidate hunter daemon 24

5.4.1 Shortcomings/simplifications in current implementation and options for the future 25

5.4.2 Work flow 26

5.4.3 Log messages 27

5.5 TapeErrorHandler: the default tape error handler daemon 28

5.5.1 Work flow . . . 29
5.5.2 Policy parameters and return value . . . 29
5.5.3 Log messages . . . 31

6 Operating the CASTOR tape migration and recall 32
6.1 Installation and configuration of tape migration/recall services . . . 32
6.1.1 Installation with RPM . . . 32
6.1.2 Installation from source tar-file . . . 33
6.1.3 CASTOR name server (Cns) privileges . . . 34
6.1.4 Configuration . . . 34
6.2 Importing FileClasses from Cns . . . 37
6.3 Enabling tape migration for a service class (SvcClass) . . . 37
6.3.1 The steps to take . . . 37
6.3.2 Creating a service class . . . 38
6.3.3 Add/remove tape pools . . . 39
6.3.4 Starting the MigHunter . . . 40
6.3.5 Dual tape copy migrations . . . 42
6.3.6 Tailoring selective migration streams using policies . . . 42
6.4 Tape error handling and retries . . . 45
6.4.1 Migration retry policy example . . . 46
6.5 SvcClass and FileClass concepts . . . 46

7 Trouble shooting 48
7.1 Killing stuck tape requests . . . 48
7.2 Stopping the service . . . 48
7.3 Stopping the MigHunter . . . 48
7.4 Cleanup after a server or node crash . . . 48
7.4.1 Full reset of the migration streams . . . 49
7.4.2 Resetting status of individual migration stream in the catalogue DB . . . 49
7.4.3 Resetting TapeCopy status in the catalogue DB . . . 50
7.4.4 Cleanup orphaned migration candidates . . . 51
7.4.5 Resetting Tape status for recall candidates . . . 52
7.5 Unbalanced Streams . . . 52

A Log messages 54

`($Id: castorMigrationRecall.tex,v 1.4 2005/05/03 16:33:00 obarring Exp $)`

Chapter 1

Who should use this guide

($Id: mrGuideReaders.tex,v 1.3 2005/05/12 09:22:01 obarring Exp $)

This manual provides an overview for anyone who is interested in the functioning and administration of the tape migration and recall in CASTOR. Chapter 4, “Overview of tape migration and recall in CASTOR”, is for all users. Chapter 5, “Detailed description” contains information of interest mainly to programmers (CASTOR developers), system- and group administrators. Chapter 6 “Operating the CASTOR tape migration” and chapter 7, “Trouble shooting”, contain information of interest mainly to system administrators.

Chapter 2

How this guide is organized

($Id: mrGuideOrganization.tex,v 1.2 2005/05/10 13:55:40 obarring Exp $)

This manual is organized in the following sections:

- **Overview of tape migration and recall in CASTOR:** gives an overview of the services involved with tape migration and recall
- **Detailed description:** detailed work-flow and catalog database interactions of the components involved with tape migration and recall
- **Operating the CASTOR tape migration and recall:** explains how to setup and operate the tape migration and recall. Various examples show how the tape migration can be configured to meet operational and user requirements. This chapter also points out the differences to the old stager and the role of the CASTOR name server *FileClass* definition.
- **Trouble shooting:** various tips that can help solving some known problematic situations

Chapter 3

Terminology

($Id: mrTerminology.tex,v 1.5 2005/05/12 09:22:01 obarring Exp $)

- **Migration Candidate** A disk resident file that is waiting to be migrated to tape. In the stager catalog, a migration candidate is always associated with a **TapeCopy** (see below) with status TAPECOPY_CREATED (0), TAPECOPY_TOBEMIGRATED (1), TAPECOPY_WAITINSTREAMS (2).
- **FileClass** Stager catalog class that defines the fileresidence on tape. The CASTOR file - file class name association is defined in the CASTOR name server. The only really relevant attribute defined in the **FileClass** is the number of copies on tape. The file residence on disk and the resource specific attributes for how the file is copied from disk to tape (e.g. which tape pools to use, number of drives) are defined in the **SvcClass** table.
- **Stream** A Stream is a virtual container of migration candidates. In the catalog database schema it provides the link between **TapeCopy** and **TapePool**
- **SvcClass** Defines the all resource attributes for a service: the disk pools and tape pools to be used; the number of drives used for migration; policies for tape migration, recall, garbage collection and disk-to-disk file replication.
- **TapeCopy** For tape recalls, the **TapeCopy** link the **CastorFile** to the tape segment(s) attributes in **Segment** table. For tape migration, the **TapeCopy** links the **CastorFile** to the migration **Stream**s. For a given **CastorFile** the number of **TapeCopy** rows is given by the nbCopies attribute in the **FileClass**.
- **TapePool** The **TapePool** table contains the names of VMGR tape pools to be used for the tape migration. A given **SvcClass** can be linked to zero or several **TapePool**s from which the tapes should be selected when the migration starts.

Chapter 4

Overview of tape migration and recall in CASTOR

($Id: mrOverview.tex,v 1.6 2005/05/12 09:22:01 obarring Exp $)

4.1 Components

The servers and programs involved with tape migration and recall are shown in Figure 4.1. The interactions with other CASTOR services are also shown in the figure.

The migration and recall of files to and from tape are controlled by the *rtcpclientd* daemon. The daemon periodically queries the catalog database for migration and recall tasks and submits the corresponding tape requests to VDQM. When a tape mover starts up with a new tape request it connects to the *rtcpclientd* daemon that immediately spawns the *recaller* or *migrator* program for handling the request. Given that there is a one-to-one mapping between tape movers and *recaller*/*migrator* processes they could eventually become part of the mover itself. This would, however, require the Oracle client to be installed and configured on all tape servers.

While the tape requests are queued in VDQM there is no instantiated process associated with the request, which means that there at most be as many *recaller*/*migrator* processes as there are tape drives.

In addition to the core framework (*rtcpclientd*, *migrator*, *recall*) directly dealing with the copying of files between tape and disk, there are two policy enforcement components: *MigHunter* and *TapeErrorHandler*. The *MigHunter* creates the migration streams and attach the migration candidates allowed by the migration policy defined for the service. The *TapeErrorHandler* calls the global retry policy deciding whether a retry is allowed or not after a failing tape recall or migration.

The *MigHunter* can either run as a daemon or directly from the command line. The *TapeErrorHandler* is normally launched by the *rtcpclientd* when a *recaller* or *migrator* exits with non-zero exit status. It can also be run from the command line.

4.2 Handling of multiple requests for same volume

The *recaller*/*migrator* processes query the catalog database for the files to be copied. The RTCOPY mover protocol supports adding of new files to running requests and this feature is systematically used by the *recaller*/*migrator* processes: the catalog is queried whenever the mover is ready to receive more work. Thus, for instance, many concurrent recall requests for files on the same volume are automatically grouped together in the same tape request.

If many files remain to be copied, they are not all sent to the mover at the same time. Instead, the file to be

migrated or the file-system to use for a recall is decided in the moment the tape mover is ready for more work. This allows to always pick the best migration candidate or file-system (recall) given the instant load or status.

4.3 Interaction with other CASTOR services

Figure 4.1 shows the different components involved with tape migration/recall and their links to other CASTOR services.

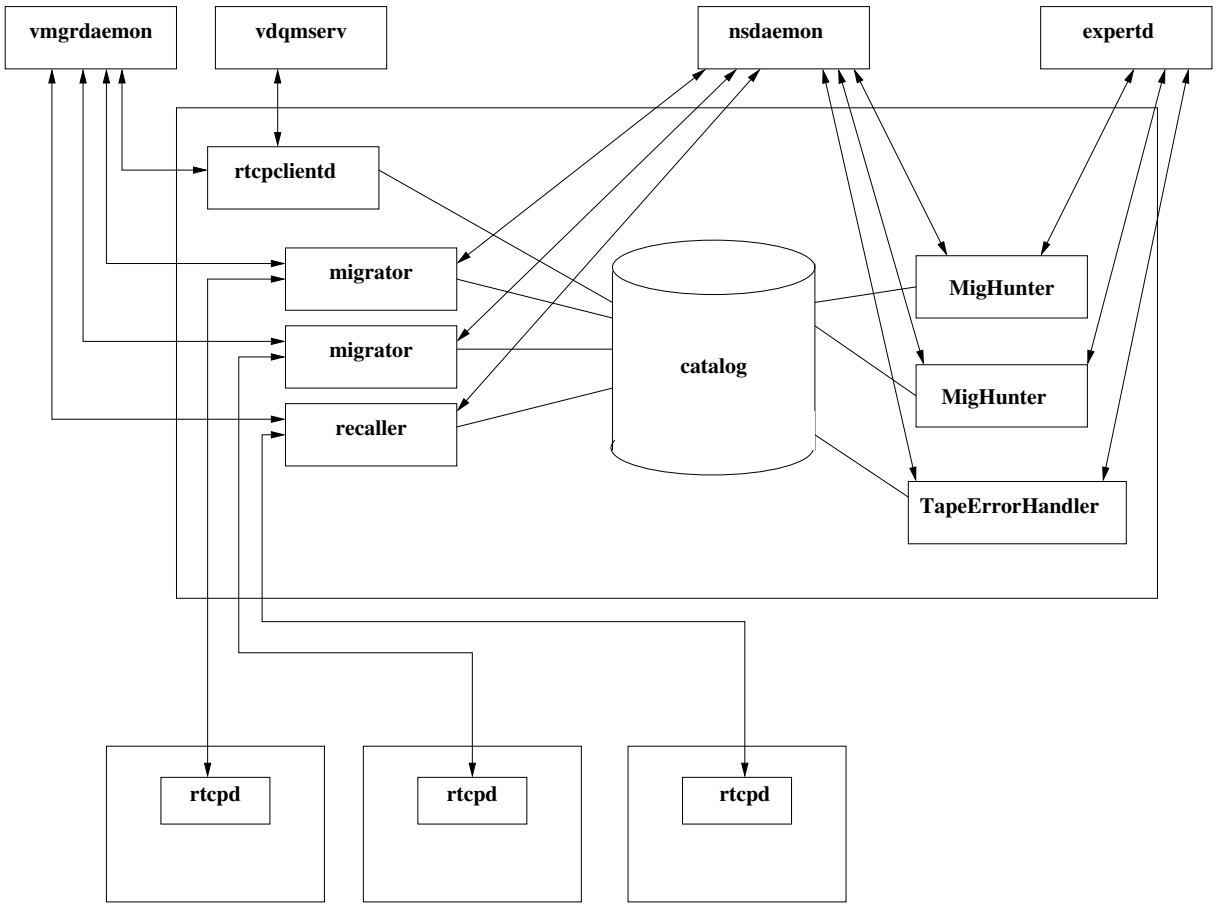

Figure 4.1: The tape migration/recall components and their interaction with other CASTOR services

The *rtcpclientd* daemon can run on anywhere. There are no clients to the *rtcpclientd* daemon. Its coupling to the stager is entirely through the catalog database and there is no messages going directly between the two (or any other service) apart from an optional UDP notification message.

Because of its dependence on VDQM, the *rtcpclientd* is not entirely stateless. It may therefore be necessary to perform a cleanup of the status of some rows in the **Stream** and **Tape** tables after a restart. This cleanup is explained in the "trouble shooting" chapter (see 7.4.2 and 7.4.5).

When the *rtcpclientd* daemon starts a *migrator* it first calls VMGR to get the tape to be used for the migration. It is also the *rtcpclientd* daemon that resets the tape **BUSY** status in VMGR when the *migrator* has exit or if the *migrator* terminates abnormally (killed or crash). This means that there is a small timewindow during which the tape can be **BUSY** in VMGR but no longer visible in VDQM (in the *showqueues* output).

The *rtcpclientd* daemon submits the tape requests to VDQM and receives the connection from the *rtcpd* mover process when it starts processing a new tape request. This means that all VDQM requests submitted by *rtcpclientd* will have the same client address (host:port). Once the *recaller*/*migrator* starts up, a new

temporary port will be opened for all its further communication with *rtcpd*. The temporary port is not known to VDQM, which means that the *killjid* command will not work on requests started by the *rtcpclientd* daemon. Instead, the *kill* command should be used if the request is running. The best is to kill the *recaller*/*migrator* client rather than the mover (*rtcpd*). If the request has not started yet, it may be removed from the VDQM queue using the *vdqm_admin* command but this is not recommended unless the migration or recall candidates for that tape have been removed from the catalog database.

The *migrator* process updates both VMGR and Cns for every file copied. The tape (and tape-pool) space counters are updated in VMGR and the CASTOR file segment attributes (VID, tape file sequence, checksum, etc.) are updated in Cns.

The *recaller* process calls Cns to retrieve the segment attributes required for the checksum verification.

Chapter 5

Detailed description

($Id: mrDetailedDescription.tex,v 1.2 2005/04/14 08:51:56 obarring Exp $)

5.1 rtcpclientd

($Id: rtcpclientd.tex,v 1.5 2005/05/12 09:22:01 obarring Exp $)

The *rtcpclientd* [2] is a single-threaded daemon. It is a client to both the catalog database,DLF (for the logging), VDQM and VMGR (for migration requests) but no other CASTOR service.

5.1.1 Work flow

The Figure 5.1 shows the basic processing flow in the *rtcpclientd* daemon. The daemon first checks the catalogue database for pending tape recall and migration requests:

- Tape recall requests are identified by a row in the **Tape** table where the `status` column is equal to `TAPE_PENDING` (1) and the `tpmode` attribute is equal to `0`. The corresponding database query is: `SELECT id FROM Tape WHERE status = 1`. The `status` is atomically updated to `TAPE_WAITDRIVE` (2) to reflect that the tape request is being queued in VDQM. The update of the `status` also prevents that the same tape is selected again.
- Tape migration requests are identified by a row in the **Stream** table where the `status` column is equal to `STREAM_PENDING` (0). The corresponding database query is: `SELECT id FROM Stream WHERE status = 0`. The `status` is atomically updated to `STREAM_WAITDRIVE` (1) in order to avoid that the same stream is selected twice. For each **Stream** the *rtcpclientd* calls VMGR (*vmgr_gettape()*) and creates (or re-uses) the corresponding row in the **Tape** table. The **Tape** `status` attribute is set to `TAPE_WAITDRIVE` (2) to reflect that the tape request is being queued in VDQM. The **Tape** `tpmode` attribute is equal to `1`.

Note that the flow for migration requests is identical to the recalls, once the tape has been returned by VMGR.

Since the access mode (`tpmode`) is an attribute of the **Tape**, there may be two rows for the same volume (one for read and one for write).

Once all new tape requests have been submitted to VDQM, the *rtcpclientd* waits for one of the following three events:

- A tape mover connection when one of the queued tape requests is being started by VDQM. The *rtcpclientd* forks the corresponding *migrator* or *recaller* process that will handle the process. This

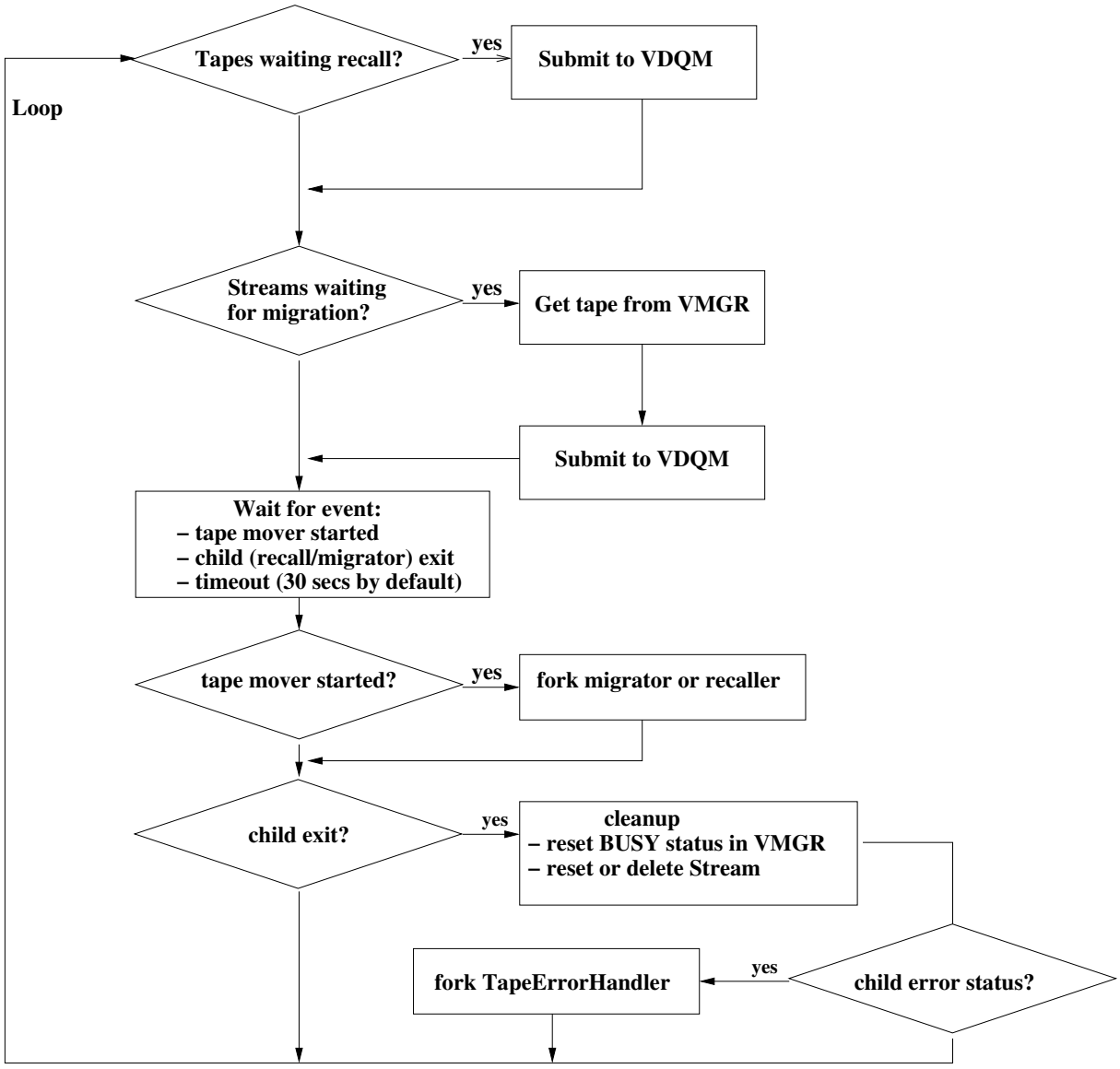

Figure 5.1: The *rtcpclientd* work flow

means that *migrator* and *recaller* processes are only instantiated for running requests and hence the number of such processes cannot exceed the number of tape drives.

- A child *migrator* or *recaller* process exits. If it was a *migrator* process, the *rtcpclientd* daemon is responsible for the necessary cleanup:
  - Reset the corresponding **BUSY** tape status in VMGR
  - Reset or delete the corresponding **Stream** row in the catalog. If the *migrator* exited before all TapeCopies had been processed (e.g. the tape was full), the **Stream - Tape** association is removed and the **Stream** `status` is reset to `STREAM_PENDING` so that it can be immediately restarted in the next *rtcpclientd* iteration. If there are no more TapeCopies, the corresponding **Stream** row is deleted from the catalog.
- A (configurable) timeout after which the *rtcpclientd* checks the catalog for new requests. Default is 30 seconds.

5.1.2 Log messages

This section lists and explains the most common DLF messages logged by the *rtcpclientd* daemon.

5.1.2.1 System messages and warnings

Table 5.1 lists the most common system and warning messages logged by *rtcpclientd* (see appendix A for a list of all messages).

Message nb	Message text	Comments
0	rtcpcld server started	First message logged when server starts
5	starting worker request for VDQM VolReqID	Logged by *rtcpclientd* before forking a worker (*migrator*/*recaller*)
17	Request successfully submitted to VDQM	
20	recaller ended	
21	migrator ended	
25	Execute command	Logged by the forked worker before calling execv() with the *migrator* or *recaller* command
26	New connection	A tape mover has started for one of the queued tape requests
31	Got tape from VMGR	Successful call to *vmgr_gettape()*
34	Updated tape info in VMGR	The tape status is being update in VMGR
52	Reset Stream	A **Stream** is reset after a *migrator* exit. The **Stream** will normally continue immendiately with a new tape
61	Lost VDQM queue position. Resubmitting request	*rtcpclientd* detects that a submitted request has disappeared (e.g. VDQM crashed)
64	Re-enable tape+segments for selection	A *recaller* exits with a failed **Segment**. The **Tape** and all non-failing **Segments** are reset to allow for a new recall.

Table 5.1: Most common system messages and warnings

The excerpt below is an example showing the startup of a migration stream:

```
DATE=20050223121653.972530 HOST=lxs5008.cern.ch LVL=System PRGN=0 PROG=rtcpcld P
ID=31681 TID=-1 MSGN=31 MSG="Got tape from VMGR" RQID=c9641c4200000010b8d49a3ca1
836893 CNS=N/A FID=0000000000000000 DLF.TPVID1=P13471 DGN="994BR7" SIDE=0 STARTF
SEQ=53

DATE=20050223121654.143159 HOST=lxs5008.cern.ch LVL=System PRGN=0 PROG=rtcpcld P
ID=31681 TID=-1 MSGN=17 MSG="Request successfully submitted to VDQM" RQID=c9641c
4200000010b8d49a3ca1836893 CNS=N/A FID=0000000000000000 VID="P13471" MODE=1 VDQM
RQID=479912

DATE=20050223121654.163671 HOST=lxs5008.cern.ch LVL=System PRGN=0 PROG=rtcpcld P
ID=31681 TID=-1 MSGN=26 MSG="New connection" RQID=c9641c4200000010b8d49a3ca18368
93 CNS=N/A FID=0000000000000000 REMHOST="tpsrv032.cern.ch"

DATE=20050223121654.211818 HOST=lxs5008.cern.ch LVL=System PRGN=0 PROG=rtcpcld P
ID=31996 TID=-1 MSGN=25 MSG="Execute command" RQID=c9641c4200000010b8d49a3ca1836
```

```
893 CNS=N/A  FID=0000000000000000 COMMAND="/usr/bin/migrator -V P13471 -s 0 -U c
9641c42-00000010-b8d49a3c-a1836893 -k 1885887 -g 994BR7 -f 53 -d 200GC -i 479912
-l aul -u 994B53A8 -T 1109157414 "

DATE=20050223121654.212394 HOST=lxs5008.cern.ch LVL=System PRGN=0 PROG=rtcpcld P
ID=31681 TID=-1 MSGN=5 MSG="starting worker request for VDQM VolReqID" RQID=c964
1c4200000010b8d49a3ca1836893 CNS=N/A FID=0000000000000000 DLF.TPVID1=P13471 MODE
=1 VDQMID=479912 WORKPID=31996
```

As can be seen, each message is logged with a number of relevant parameters. The request identifier (RQID) is always the same during the full lifecycle of the *rtcpclientd* daemon.

5.1.2.2 Error messages

The most common *rtcpclientd* error messages are listed in Table 5.2.

Message nb	Message text	Comments
3	failed system call	A system or CASTOR call failed (see below)
7	rtcopy client daemon internal error	This error may happen when *rtcpclientd* detects an internal inconsistency. Typical example is when the *rtcpclientd* was restarted while it had queued requests in VDQM. When VDQM starts one of the requests, the *rtcpclientd* does not know about the request anymore.
10	catalogue lookup error	The check for new **Segments** or **Streams** to do failed in this iteration. This error is usually preceeded by the "Database service error" message (see below).
16	External logger error message	Error logged by a log-function in the RTCOPY API. These messages normally follows (or preceeds) a "normal" error message and are always logged when an error is propagated between the mover and its client.
22	Database service error	

Table 5.2: Most common error messages logged by *rtcpclientd*

Most error messages are either due to a failing system call or catalog interface call. The following DLF parameter convention is used for the logging of system and catalog interface errors:

- **failed system call**: five parameters
  - parameter name **SYSCALL**: names the system call that failed
  - parameter name **ERROR_STR**: the error string corresponding to the (thread) global error variable (**errno** or **serrno**)
  - parameter names **File** and **Line**: the source file and line where the error happened
  - parameter names **errno** and **serrno**: the values of the corresponding global error variables. **serrno** is set for CASTOR library calls while **errno** is set for all normal system calls
- **Database service error**: six parameters
  - parameter name **DBCALL**: names the catalog interface call that failed
  - parameter name **ERROR_STR**: the error string corresponding to the (thread) global error variable (**errno** or **serrno**)
  - parameter **DB_ERROR**: the detailed database error (Oracle code)

- parameter names **File** and **Line**: the source file and line where the error happened
- parameter names **errno** and **serrno**: the values of the corresponding global error variables. **serrno** is set for CASTOR library calls while **errno** is set for all normal system calls

Examples of the two error messages types are:

```
DATE=20050227073611.389554 HOST=lxs5008.cern.ch LVL=Error PRGN=0 PROG=rtcpcld PI
D=8929 TID=-1 MSGN=3 MSG="failed system call" RQID=84c51c4200000010951a92b0eff26
893 CNS=N/A FID=0000000000000000 SYSCALL="rtcp_RecvReq()" ERROR_STR="Connection
closed by remote end" File="rtcpclientd.c" Line=168 errno=0 serrno=1016

DATE=20050301102843.244995 HOST=lxs5008.cern.ch LVL=Error PRGN=0 PROG=rtcpcld PI
D=8929 TID=-1 MSGN=22 MSG="Database service error" RQID=0000000000000000000000000
00000000 CNS=N/A FID=0000000000000000 DBCALL="Cstager_IStagerSvc_streamsToDo()"
ERROR_STRING="Invalid argument" DB_ERROR="Error in update request : ORA-00060: d
eadlock detected while waiting for resource  Statement was : UPDATE Stream SET i
nitialSizeToTransfer = :1, status = :2 WHERE id = :3 and id was 420573 " File="r
tcpcldCatalogueInterface.c" Line=643 errno=0 serrno=22
```

5.1.2.3 Alert messages

The alert messages usually requires manual intervention. Table 5.12 lists the possible alert messages logged by the *rtcpclientd* daemon.

Message nb	Message text	Comments
1	rtcpcld_InitNW() failed to initialise network ports	*rtcpclientd* failed to initialise its network port. Probably there is already a *rtcpclientd* running on the system
50	VMGR error requiring admin intervention	Usually due to an empty tape pool but there can be other reasons (e.g. initialSizeToTransfer column in the **Stream** table is zero)
71	Service shutdown	The *rtcpclientd* server shutdown

Table 5.3: Alert messages logged by *rtcpclientd*

Example:

```
DATE=20050218075210.538427 HOST=lxs5008.cern.ch LVL=Alert PRGN=0 PROG=rtcpcld PI
D=24156 TID=-1 MSGN=50 MSG="VMGR error requiring admin intervention" RQID=26e514
4200000010b370cdbac98868a3 CNS=N/A FID=0000000000000000 POOLNAME="alicemdc6_613"
File="rtcpcldVmgrInterface.c" Line=303 errno=0 serrno=28
```

5.2 migrator

```
($Id: migrator.tex,v 1.13 2005/05/13 09:16:20 obarring Exp $)
```

The *migrator* is a multi-threaded program that is usually started by the *rtcpclientd* daemon. It is a client to both the catalog database, DLF (for the logging), tape mover (*rtcpd*), VMGR and Cns. For debugging purposes, the *migrator* process can also run stand-alone, in which case it is also a client to the VDQM service. When running it stand-alone, the `-i <VDQM-reqid>` switch should be omitted.

The number of threads should usually match the "disk I/O" threadpool size of the tape mover. Compiled default is 3. There is usually one open database session per thread, which means that the *migrator* can have up to 3 database sessions. If the files are larger than the *rtcpd* memory buffers, there will only be two database sessions.

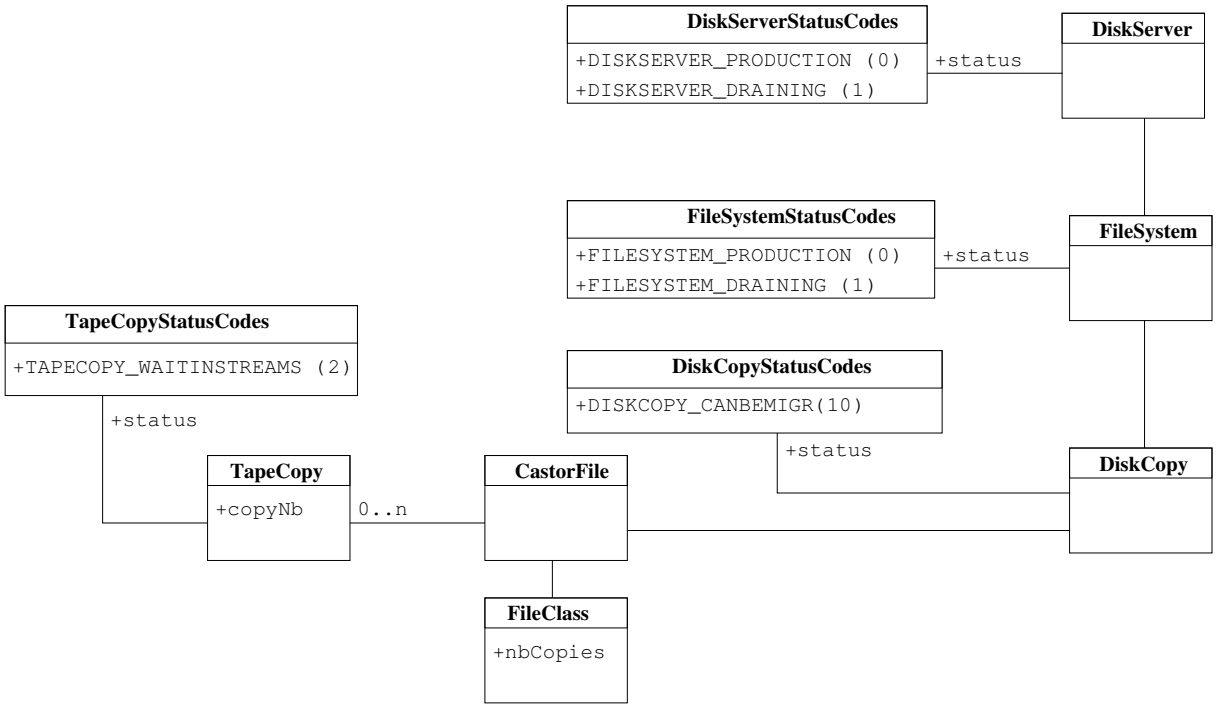

Figure 5.2: The signature of migration candidates

5.2.1 Signature of a migration candidate

The catalog database signature of a file ready for migration is shown in Figure 5.2. The candidate is identified by the following series of `status` codes for the various tables involved:

- The disk server (**DiskServer**) on which the file reside must be available for migration. A draining disk server is considered as available for migration (but not for recall).
- The file system (**FileSystem**) on which the file reside must be available for migration. A draining file system is considered as available for migration (but not for recall).
- The disk file (**DiskCopy**) must be in status `DISKCOPY_CANBEMIGR`, which is normally set by the "put" job at the end when the disk mover has closed the file.
- There must be a **TapeCopy** linked to the corresponding **CastorFile**. The **TapeCopy** is created by the "put" job at the same time as the **DiskCopy** is flagged `DISKCOPY_CANBEMIGR` [1]. The **TapeCopy** status should be set to `TAPECOPY_WAITINSTREAMS`, which is normally done in the stream creation (or update) phase of the *MigHunter* program.

The corresponding catalog database query to find all files waiting to be picked up by a migration stream looks like:

```
SELECT TapeCopy.id
FROM TapeCopy,CastorFile,DiskCopy,FileSystem,DiskServer
   WHERE TapeCopy.status = 2 -- TAPECOPY_WAITINSTREAMS
   AND TapeCopy.castorFile = CastorFile.id
   AND DiskCopy.castorFile = CastorFile.id
   AND DiskCopy.status = 10 -- DISKCOPY_CANBEMIG
   AND DiskCopy.fileSystem = FileSystem.id
   AND FileSystem.status IN (0,1) -- FILESYSTEM_PRODUCTION, FILESYSTEM_DRAINING
   AND FileSystem.diskServer = DiskServer.id
   AND DiskServer.status IN (0,1) -- DISKSERVER_PRODUCTION, DISKSERVER_DRAINING;
```

[1] Several **TapeCopy** rows may be created if it is a multi-copy fileclass

5.2.2 Work flow

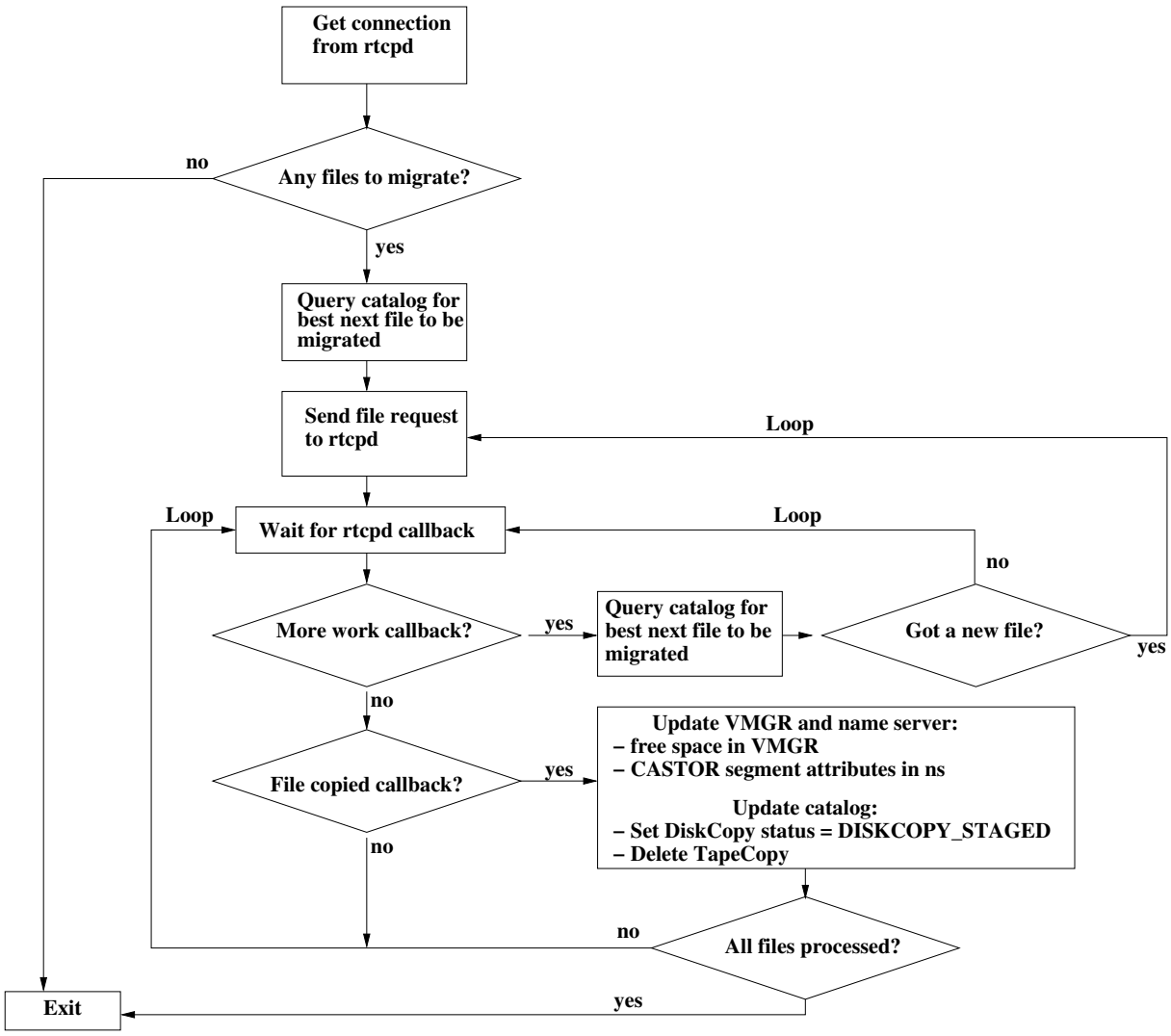

Figure 5.3: The *migrator* work flow

The Figure 5.3 shows the basic processing flow in the *migrator* program. The *rtcpclientd* transfers the *rtcpd* connection to the *migrator* via the `-s <socket-number>` option on the *migrator* command-line. When the *migrator* starts up it first checks if there still are files to be copied since some other parallel migration **Stream** may have already copied all files. If it finds no more files to process, it simply drops the *rtcpd* connection in order to force the *rtcpd* to exit without mounting the tape. **Warning**, although this is not strictly an error, the dropped connection is logged as such by the tape mover (*rtcpd*), e.g.

```
May  3 04:05:11 rtcopyd[18244]: rtcp_Transfer() netread(0,HDR): connection dropped
May  3 04:05:11 rtcopyd[18244]: rtcpd_GetRequestList() rtcp_RecvReq(): Connection
 closed by remote end
May  3 04:05:11 rtcopyd[18244]: rtcpd_MainCntl() request loop finished with error
May  3 04:05:11 rtcopyd[18244]: rtcpd_Deassign() ASSIGN volreq ID 279813 -> jobID 18244
May  3 04:05:11 rtcopyd[18244]: rtcpd_Deassign() RELEASE job ID 18244
May  3 04:05:11 rtcopyd[18244]: request failed
```

However, because the error appears at the startup of the tape mover (*rtcpd*), it is not added to the RTCOPY accounting records and hence will not appear in the *rtstat* statistics.

The corresponding (ORACLE) query to the catalog database is[2]:

```
SELECT COUNT(*)
FROM DiskServer, FileSystem, DiskCopy, CastorFile, TapeCopy, Stream2TapeCopy
    WHERE DiskServer.id = FileSystem.diskserver
    AND DiskServer.status IN (0, 1) -- DISKSERVER_PRODUCTION, DISKSERVER_DRAINING
    AND FileSystem.id = DiskCopy.filesystem
    AND FileSystem.status IN (0, 1) -- FILESYSTEM_PRODUCTION, FILESYSTEM_DRAINING
    AND DiskCopy.castorfile = CastorFile.id
    AND TapeCopy.castorfile = Castorfile.id
    AND Stream2TapeCopy.child = TapeCopy.id
    AND Stream2TapeCopy.parent = streamId
    AND TapeCopy.status = 2; -- TAPECOPY_WAITINSTREAMS
```

(`streamId` is the **Stream** being started). In case the query returns true (more than zero candidates), the `status` of the corresponding **Stream** is atomically updated to `STREAM_WAITMOUNT`.

The *rtcpd* tape mover usually has a large memory buffer that allows it to start copying the first disk-file while the tape is being mounted. This is the reason why the *migrator* immediately sends the first file to be processed.

After having sent the tape mount and the first file request to *rtcpd*, the *migrator* enters its main loop to process *rtcpd* callbacks. There are three types of callbacks:

- Tape file positioned callback when the tape is ready to receive (or send, in case of recall) the next file. The tape file position callback is not used by the *migrator*.
- Tape file copied callback when the file has been completely copied to tape (or disk, in case of recall). The tape file copied callback is used by the *migrator* to update VMGR, Cns and the catalog database as follows:
  - If the copy was successful the VMGR free-space counter is decremented with the file-size and the number-of-files counter is incremented
  - If the copy was successful the Cns (CASTOR name server) is updated with the segment attributes: tape VID, tape file sequence number, tape position block identifier (if supported), segment checksum.
  - In the catalog database:
    1. Single-tapecopy CASTOR files: the corresponding **TapeCopy** row is deleted and the **DiskCopy** `status` is set to `DISKCOPY_STAGED` (0)
    2. Multi-tapecopy CASTOR files: update `status` of **TapeCopy** to `TAPECOPY_STAGED` (5). If all **TapeCopy** rows associated with the **CastorFile** are in `status` `TAPECOPY_STAGED` (5), they are all deleted and the **DiskCopy** `status` is set to `DISKCOPY_STAGED` (0)
    3. In case the file copy failed, the **TapeCopy** row is left in `TAPECOPY_SELECTED` (3) status and associated with a new row in the **Segment** table containing the error information. The **Segment** `status` is set to `SEGMENT_FAILED`. This is the only case when the **Segment** table is used for file migration (usually it is only used for tape recall). The error information stored in the **Segment** row is used by the *TapeErrorHandler* program to determine if a retry is appropriate (see Section 5.5). Finally, the failed **TapeCopy** is detached from the **Stream**, allowing the **Stream** to be deleted when it runs out of eligible migration candidates.
- "More work" callback when the *rtcpd* has free resources (memory buffers and threads) to process more files. When it receives a "More work" callback the *migrator* calls a catalog database query to find the next **TapeCopy** to process. This query is explained in more details below.

[2] This may not be the final implementation because it has been found that the execution plan for this query can become sub-optimal

The catalog database query to find the next **TapeCopy** to process is using the `weight`, `deltaWeight` and `fsDevaiation` columns of the **FileSystem** table to determine which is the best filesystem to select a **TapeCopy** from. The `weight`, `deltaWeight` and `fsDeviation` columns are periodically updated by the CASTOR resource monitoring. Between two updates, the `deltaWeight` and `fsDeviation` are used to correct the `weight` with the anticipated load from every new migration stream added to a filesystem. This is to avoid that the same filesystem is always selected between two monitoring updates. The exact query for selecting the best filesystem is a rather complex join of four tables and will not be shown here (the implementation can be found in the `bestTapeCopyForStream` Oracle procedure in *castor/db/oracle.sql* in the CASTOR CVS. Once the best filesystem containing eligible migration candidates has been found, one of the candidates is selected (in database order) and returned to the *migrator*. The `status` column of the corresponding **TapeCopy** is atomically updated to `TAPECOPY_SELECTED` (3).

5.2.3 Dual copies

The number of tape copies for a particular CASTOR file is defined by the `nbCopies` attribute in the **FileClass** class, see Figure 5.2. When the **CastorFile** is prepared for migration at the end of a "put" job (see Section 5.2.1), a `nbCopies` number of rows will be inserted in the **TapeCopy** table and associated to the **CastorFile** row. Each **TapeCopy** row is independent and they are equally treated by the *MigHunter*. However, the **TapeCopy**s for a given **CastorFile** will be assigned an unique `copyNb` starting from 1, which can be used for applying a selective migration policy. An exampleof this is given in Section 6.3.6.2. If no selective migration policy is applied, there is a protection in the *migrator* program preventing the two copies to go onto a same tape. When the *migrator* detects this condition it will log a warning, "Dual copy on same tape. Detached from stream" (see Section 5.2.4.1) and detach the **TapeCopy** from the **Stream**. If there are multiple **Stream**s, the **TapeCopy** will be automatically picked up by another **Stream** otherwise the *MigHunter* will put it back on the **Stream** in its next iteration (in this latter case the **TapeCopy** may be bounced between the *migrator* and the *MigHunter* for quite a few iterations until a new tape is selected for the **Stream**).

5.2.4 Log messages

This section lists and explains the most common DLF messages logged by the *migrator* program.

5.2.4.1 System messages and warnings

Table 5.4 lists the most common DLF messages logged by *migrator* (see appendix A for a list of all messages).

The log excerpt below is an example showing the position and successful migration of a file:

```
DATE=20050310163702.783540 HOST=lxs5008.cern.ch LVL=System PRGN=1 PROG=migrator P
ID=12557 TID=0 MSGN=8 MSG="rtcopy client daemon callback: file position" RQID=0d6
9304200000010a78acd9dee2c04a5 CNS=lxs5009 FID=00000000030eb8da DLF.TPVID1=P03222
TPSERV="tpsrv110" TPDRIVE="994B42A2" DLF.SRQID1=0c69304200000010aff6d471d6570000
FSEQ=3 BLKID="0000aa18" PATH="lxfsrk5702:/shift/lxfsrk5702/data02//74/51296474@lx
s5009" STATUS=2

DATE=20050310163742.692614 HOST=lxs5008.cern.ch LVL=System PRGN=1 PROG=migrator P
ID=12557 TID=0 MSGN=34 MSG="Updated tape info in VMGR" RQID=0d69304200000010a78ac
d9dee2c04a5 CNS=lxs5009 FID=00000000030eb8da DLF.TPVID1=P03222 BYTESWRT=108114471
6 COMPRFAC=99 NBFILES=1 STATUS="BUSY"

DATE=20050310163742.808631 HOST=lxs5008.cern.ch LVL=System PRGN=1 PROG=migrator P
ID=12557 TID=0 MSGN=55 MSG="File staged" RQID=0d69304200000010a78acd9dee2c04a5 CN
S=lxs5009 FID=00000000030eb8da DLF.TPVID1=P03222 TPSERV="tpsrv110" TPDRIVE="994B4
```

Message nb	Message text	Comments
8	File position callback	Mover callback when tape file is positioned
14	migrator started	The *migrator* program startup message
18	Nothing left to do for this VID	No migration candidates found. This message is logged at startup when the *migrator* checks if it is worth mounting the tape or not.
19	Get more work callback: add file	Mover callback when it is ready to receive more files. The message contains the disk path and CASTOR `fileid` of the selected candidate.
21	migrator ended	
28	No more segments to process	There are no more eligible migration candidates left for this stream.
31	Got tape from VMGR	Successful call to *vmgr_gettape()*
34	Updated tape info in VMGR	The tape status is being update in VMGR
51	Re-enable TapeCopy for selection	The *migrator* has hit an error and before exit it resets the `status` column of the **TapeCopy** table for all outstanding selected migration candidates.
53	Dual copy on same tape. Detached from stream.	The **CastorFile** is associated with a dual tape copy fileclass (`nbCopies>1` in the **FileClass** table). The *migrator* has detected that the current tape already holds one copy of the file and will hence refrain from migrating the second copy to the same tape. See Section 5.2.3 for more details
55	File staged	Mover callback when the file has been copied to tape

Table 5.4: Most common *migrator* system messages and warnings

```
2A2" DLF.SRQID1=0c69304200000010aff6d471d6570000 FSEQ=3 DISKPATH=lxfsrk5702:/shif
t/lxfsrk5702/data02//74/51296474@lxs5009 FILESIZE=1081144716 TOTFILES=3 TOTBYTES=
2507008792
```

Each message is logged with a number of DLF parameters providing more detailed information. The request identifier (RQID) is unique to a migrator, which means that one can easily follow its lifecycle. Other important parameters are the the CASTOR bitfile id (FID), which helps the tracing of a particular CASTOR file through out the whole stager system.

5.2.4.2 Error messages

Table 5.5 lists the most common DLF messages logged by the *migrator* program (see appendix A for a list of all messages).

An example of a failed copy message:

```
DATE=20050304220319.940373 HOST=lxs5008.cern.ch LVL=Error PRGN=1 PROG=migrator P
ID=16797 TID=0 MSGN=16 MSG="External logger error message" RQID=b6af284200000010
a5d6ecdcd6c884a8 CNS=N/A FID=0000000000000000 RTCP_LOG=" CPDSKTP ! I/O ERROR WRI
TING ON TAPE: Medium error ASC=C ASCQ=0 (BLOCK # 2220)  CPDSKTP ! TAPE IS NOW IN
CORRECTLY TERMINATED  "
```

5.2.4.3 Alert messages

The alert messages usually requires manual intervention. Table 5.9 lists the possible alert messages logged by *migrator* program.

Message nb	Message text	Comments
3	failed system call	A system or CASTOR call failed (see Section 5.1.2.2 for further explanation)
16	External logger error message	Error logged by a log-function in the RTCOPY API. These messages normally follows (or preceeds) a "normal" error message and are always logged when an error is propagated between the mover and its client.
22	Database service error	A system or catalog database interface call failed (see Section 5.1.2.2 for further explanation)
33	Tape unavailable	The has been DISABLED, EXPORTED or ARCHIVED in VMGR while the *migrator* was processing it.
44	Failed to update segment in name server	Usually preceeded by a failed system call (see above) for a *Cns_setsegattr()* call (or some other Cns API call).
45	Failed to update VMGR	Similar to previous item but for a VMGR API call (usually *vmgr_updatetape()*).
47	Segment check failed	Internal error when checking some segment attributes returned by the mover. Most likely due to a mismatch between the number of bytes copied and the filesize assigned to the **CastorFile** row in the catalog.
58	Copy failed	The mover got an error while copying the file. The corresponding error code, message text and severity are logged.

Table 5.5: Most common *migrator* error messages

5.3 recaller

`($Id: recaller.tex,v 1.14 2005/07/06 11:08:55 obarring Exp $)`

Exactly like the *migrator* (see Section 5.2) the *recaller* is a multi-threaded program that is usually started by the *rtcpclientd* daemon. It is a client to both the catalog database, DLF (for the logging), tape mover (*rtcpd*) and Cns. For debugging purposes, the *recaller* process can also run stand-alone, in which case it is also a client to the VDQM service. When running it stand-alone, the `-i <VDQM-reqid>` switch should be omitted.

The number of threads should usually match the "disk I/O" threadpool size of the tape mover. Compiled default is 3. There is usually one open database session per thread, which means that the *recaller* can have up to 3 database sessions. If the files are larger than the *rtcpd* memory buffers, there will only be two database sessions.

5.3.1 Origin and signature of a recall candidate

Recall candidates are usually created on demand, meaning that there is always a **SubRequest** associated with the file to be recalled. The exception is when a `recallPolicy` has been defined for the **SvcClass**, in which case the policy could be to pre-stage some files for which there are not necessarily any associated active request. In the normal case, however, the user request to access a CASTOR file is scheduled to one of the disk servers where the job immediately exits. This seemingly useless scheduling of a job that anyway will exit is motivated by the fact that also tape recalls must be subjected to fair-share and other access policies enforced by the scheduler. Before returning, the job will update the catalog as follows:

- set the **DiskCopy** `status` to `DISKCOPY_WAITTAPERECALL`

Message nb	Message text	Comments
23	Retrieved inconsistent tape request	The **Tape** row database key passed with the `-k <dbKey>` command option to the *migrator* program does not correspond to the specified tape VID. Unless the catalog database is corrupted (this has never happened up to now) this error can only happen if the *recaller* program has been run directly from the command line (e.g. for debugging purposes).

Table 5.6: Most common *migrator* alert messages

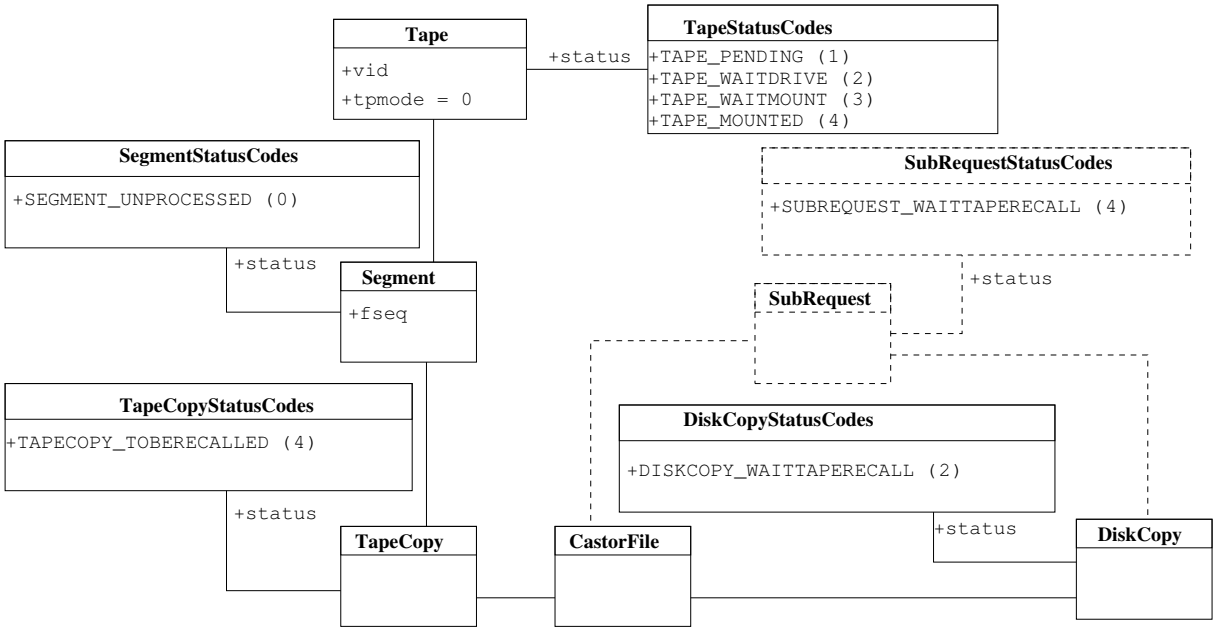

Figure 5.4: The signature of recall candidates

- set the **SubRequest** `status` to `SUBREQUEST_WAITTAPERECALL`
- create a **TapeCopy** row with `status` = `TAPECOPY_TOBERECALLED` and link it with the **CastorFile**
- create one or more **Segment** and **Tape** (if necessary) rows for each of the segments to be recalled and link each **Segment** to the **TapeCopy**
- set the **Segment** `status` to `SEGMENT_UNPROCESSED`
- for each **Tape** row, check the `status` and update from `TAPE_UNUSED`, `TAPE_FINISHED` or `TAPE_FAILED` to `TAPE_PENDING`. Note that for running requests the **Tape** `status` is one of `TAPE_PENDING`, `TAPE_WAITDRIVE`, `TAPE_WAITMOUNT` or `TAPE_MOUNTED` and the newly attached **Segment** will automatically be added to the running request.

The catalog database signature of a file ready for recall is shown in Figure 5.4.

It should be noted that the **DiskCopy** is *not* linked to a **FileSystem**. This is because the selection of the target file system is done when the tape is mounted and positioned, which allows to always select the "best" (in terms of load and free space) just in time before the file is transferred.

The corresponding catalog database query to find all segments waiting to be picked up by a recaller looks like:

```
SELECT Tape.vid,Segment.fseq
FROM Tape, Segment
   WHERE Tape.status IN (1,2,3,4)           -- TAPE_PENDING, TAPE_WAITDRIVE,
                                            -- TAPE_WAITMOUNT, TAPE_MOUNTED
   AND Segment.status = 0                   -- SEGMENT_UNPROCESSED
   AND Segment.tape = Tape.id;
```

5.3.2 Work flow

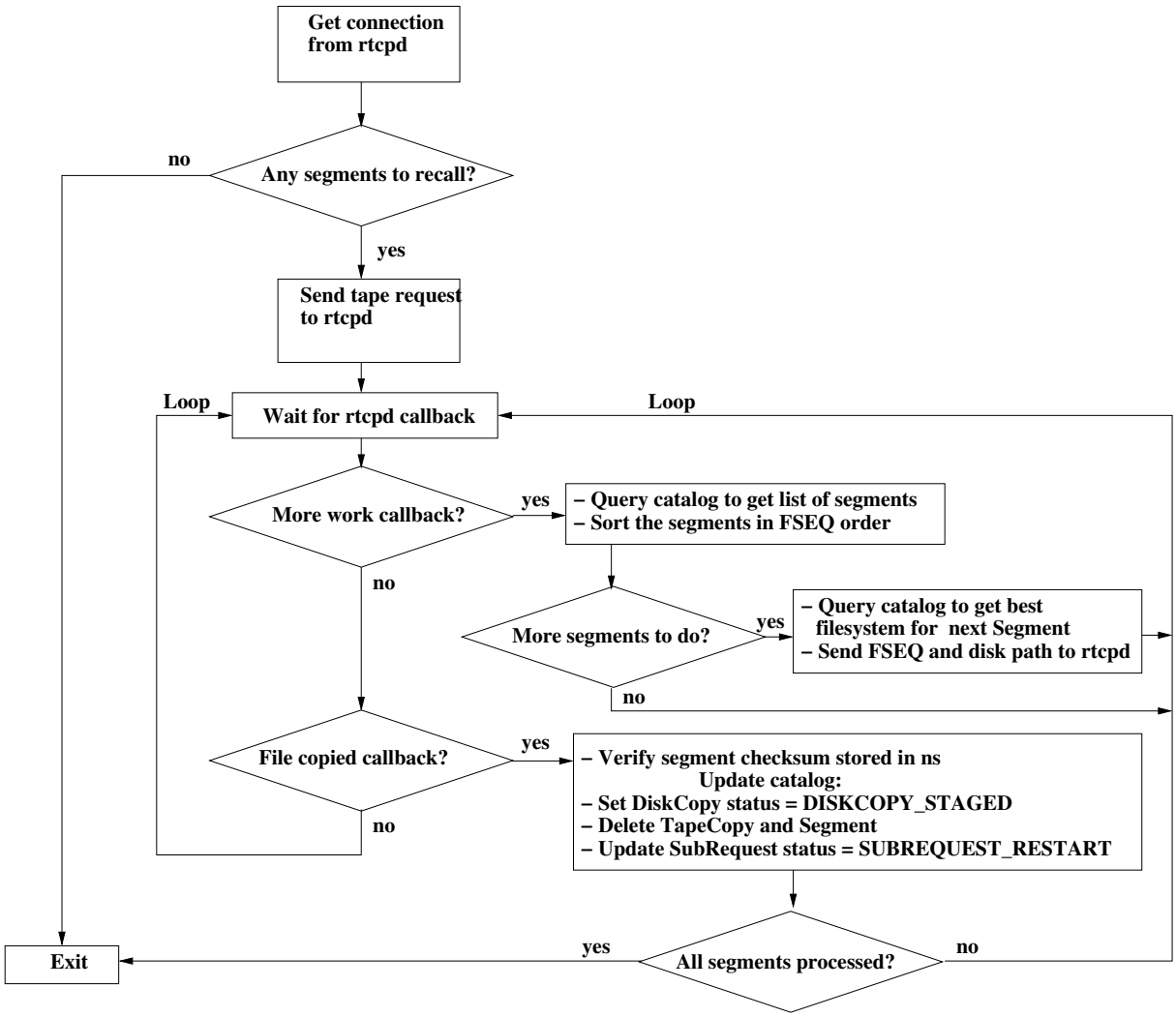

Figure 5.5: The *recaller* work flow

The Figure 5.5 shows the basic processing flow in the *recaller* program. The *rtcpclientd* transfers the socket number of the *rtcpd* connection to the *recaller* via the `-s <socket-number>` option on the *recaller* command-line. When the recaller starts up it first checks if there still are files to be copied since some clients may have canceled their recalls in the meanwhile [3]. If it finds no more files to process, it simply drops the *rtcpd* connection in order to force the *rtcpd* to exit without mounting the tape. **Warning**, although this is not strictly an error, the dropped connection is logged as such by the tape mover (*rtcpd*), e.g.

```
May  3 04:05:11 rtcopyd[18244]: rtcp_Transfer() netread(0,HDR): connection dropped
May  3 04:05:11 rtcopyd[18244]: rtcpd_GetRequestList() rtcp_RecvReq(): Connection
 closed by remote end
```

[3] By default, a recall is on demand, which means that it is always associated with an active **SubRequest**

```
May  3 04:05:11 rtcopyd[18244]: rtcpd_MainCntl() request loop finished with error
May  3 04:05:11 rtcopyd[18244]: rtcpd_Deassign() ASSIGN volreq ID 279813 -> jobID 18244
May  3 04:05:11 rtcopyd[18244]: rtcpd_Deassign() RELEASE job ID 18244
May  3 04:05:11 rtcopyd[18244]: request failed
```

However, because the error appears at the startup of the tape mover (*rtcpd*), it is not added to the RTCOPY accounting records and hence will not appear in the *rtstat* statistics.

The corresponding (ORACLE) query to the catalog database is:

```
SELECT count(*) FROM Segment
    WHERE Segment.tape = tapeId
    AND Segment.status = 0;            -- SEGMENT_UNPROCESSED
```

where `tapeId` is the catalog database identifier (64 bit number) of the **Tape** being started. In case the query returns true, the **Tape** `status` is atomically updated to `TAPE_WAITMOUNT`.

Once the tape mount request has been sent the *recaller* enters its main loop for processing the *rtcpd* callbacks. There are three types of callbacks:

- Tape file positioned callback when the tape is ready to send (or receive, in case of migration) the next file. The tape file position callback is not used by the *recaller*
- Tape file copied callback when the file has been completely copied to disk (or tape, in case of migration). When it receives a tape file copied callback the *recaller* verifies the segment checksum calculated by *rtcpd* with its reference value in the CASTOR name server, and if OK, it updates the catalog database as follows:
  1. Single-segment CASTOR files: the corresponding **Segment** and **TapeCopy** rows are deleted and the **DiskCopy** `status` is set to `DISKCOPY_STAGED` (0). The **SubRequest** `status` is updated to `SUBREQUEST_RESTART` (1) in order to release (and schedule) the request waiting for the recall of this file.
  2. Multi-segment CASTOR files: update the **Segment** `status` to `SEGMENT_FILECOPIED` (5). If all **Segment** rows associated with the **TapeCopy** are in `status` `SEGMENT_FILECOPIED` (5), they are all deleted and the **DiskCopy** and **SubRequest** are updated like above (single-segment CASTOR files)
  3. In case the file copy (or position) failed, the **Segment** `status` is set to `SEGMENT_FAILED` (6) and the *rtcpd* error information is copied to the `errMsgTxt`, `errorCode` and `severity` columns. The error information stored in the **Segment** row should be used by the *TapeErrorHandler* process to determine if a retry is appropriate (see Section 5.5).
- "More work" callback when the *rtcpd* has free resources (memory buffers and threads) to process more files. When it receives a "More work" callback the *recaller* first calls the catalog to get list of new recall candidates (if any) that has been inserted to the **Segment** table since the last lookup. The `status` column for each selected **Segment** is atomically updated from `SEGMENT_UNPROCESS` to `SEGMENT_SELECTED` (to avoid that the segment is selected twice). The list of new candidates is sorted in tape file sequence order and appended to the list of candidates that are being processed. Thereafter the *recaller* picks the first candidate in the sorted list and queries the catalog for the best target filesystem to which the file can be copied. The complete file request (tape FSEQ and disk path) is finally returned to *rtcpd*.

The catalog database query to find the best target **FileSystem** to recieve the current **Segment** is using the `free`, `deltaFree`, `reservedSpace`, `weight`, `deltaWeight` and `fsDeviation` columns of the **FileSystem** table. Those columns are periodically updated by the CASTOR resource monitoring. Between two updates, the `deltaFree`, `reservedSpace`, `deltaWeight` and `fsDeviation` are used to correct the `free` and `weight` with the anticipated (decreased) free space and load from every new recall stream added to a filesystem. This is to avoid that the same filesystem is always selected between two monitoring updates. The exact query for selecting the best filesystem is a rather complex join of four tables and will not be shown here (the implementation can be found in the `bestFileSystemForSegment` Oracle procedure in *castor/db/oracle.sql* in the CASTOR CVS.

5.3.3 Segment checksum verification

By default the *recaller* verifies the segment checksum calculated by the tape mover (*rtcpd*). If the CASTOR name server segment attribute for the file does not yet have a checksum (i.e. the file has not been touched since the segment checksum verification was deployed), the *recaller* updates the name server with the checksum algorithm and value returned by the mover. If the CASTOR name server segment attribute already have a checksum but the algorithm name differs from the name returned by the tape mover (*rtcpd*), the name server segment attribute is updated if:

- `rtcpcld CHANGE_CHECKSUM_NAME YES` is defined in */etc/castor/castor.conf*, or
- the environment variable `RTCPCLD_CHANGE_CHECKSUM_NAME=YES`

The checksum verification can be switched off completely through:

- `rtcpcld USE_CHECKSUM NO` is defined in */etc/castor/castor.conf*, or
- the environment variable `RTCPCLD_USE_CHECKSUM=NO`

5.3.4 Log messages

This section lists and explains the most common DLF messages logged by the *recaller* program.

5.3.5 System messages and warnings

Table 5.7 lists the most common DLF messages logged by *recaller* (see appendix A for a list of all messages).

The log excerpt below is an example showing the position and successful recall of a file:

```
DATE=20050405171625.916486 HOST=tbed0083.cern.ch LVL=System PRGN=2 PROG=recaller
 PID=28030 TID=0 MSGN=8 MSG="rtcopy client daemon callback: file position" RQID=a
aab5242000000109a269f9ba2adf4a7 CNS=tbed0082 FID=000000000078e1b5 DLF.TPVID1=R043
79 TPSERV="tpsrv010" TPDRIVE="984051A4" DLF.SRQID1=aaab524200000010be39fd9ccb5800
00 FSEQ=142 BLKID="00031010" DISKPATH="lxfsrk4303:/shift/lxfsrk4303/data02//01/79
22101@tbed0082.309109" STATUS=2

DATE=20050405171626.729816 HOST=tbed0083.cern.ch LVL=System PRGN=2 PROG=recaller
PID=28030 TID=1 MSGN=39 MSG="Checksum is OK" RQID=aaab5242000000109a269f9ba2adf4a
7 CNS=tbed0082 FID=000000000078e1b5 DLF.TPVID1=R04379 SIDE=0 MODE=0 FSEQ=142 BLOC
KID="00031010" SEGCKSUM=-1652288696

DATE=20050405171626.949275 HOST=tbed0083.cern.ch LVL=System PRGN=2 PROG=recaller
PID=28030 TID=1 MSGN=55 MSG="File staged" RQID=aaab5242000000109a269f9ba2adf4a7 C
NS=tbed0082 FID=000000000078e1b5 DLF.TPVID1=R04379 TPSERV="tpsrv010" TPDRIVE="984
051A4" DLF.SRQID1=aaab524200000010be39fd9ccb580000 FSEQ=142 DISKPATH="lxfsrk4303:
/shift/lxfsrk4303/data02//01/7922101@tbed0082.309109" OFFSET=0 FILESIZE=9675420 T
OTFILES=1 TOTBYTES=9675420
```

Each message is logged with a number of DLF parameters providing more detailed information. The request identifier (RQID) is unique to a recaller, which means that one can easily follow its lifecycle. Other important parameters are the the CASTOR bitfile id (FID), which helps the tracing of a particular CASTOR file through out the whole stager system.

Message nb	Message text	Comments
8	File position callback	Mover callback when tape file is positioned
13	recaller started	The *recaller* program startup message
18	Nothing left to do for this VID	No recall candidates found. This message is logged at startup when the *recaller* checks if it is worth mounting the tape or not.
19	Get more work callback: add file	Mover callback when it is ready to receive more files. The message contains the disk path and CASTOR `fileid` of the selected candidate.
20	recaller ended	
24	New file request	A new **Segment** (tape file) added to list of segments to be process
28	No more segments to process	There are no more eligible recall candidates left for this tape.
39	Checksum is OK	The tape segment checksum verification was successful
41	Update checksum in Cns segattrs	The checksum is not set for this segment, or it is set but the algorithm has changed since (provided the `rtcpcld CHANGE_CHECKSUM_NAME` configuration is set in */etc/castor/castor.conf*
54	Not all segments staged for file	The file is segmented over several tapes and not all segments have not yet been staged. When this message is logged, it means that the **DiskCopy** `status` will remain in `DISKCOPY_WAITTAPERECALL`
55	File staged	Mover callback when the file has been copied from tape
59	Unusual position method	A warning notifying that *blockid* position is not used

Table 5.7: Most common *recaller* system messages and warnings

5.3.5.1 Error messages

Table 5.8 lists the most common DLF messages logged by the *recaller* program (see appendix A for a list of all messages).

An example of a failed copy message:

```
DATE=20050414053331.272678 HOST=tbed0083.cern.ch LVL=Error PRGN=2 PROG=recaller P
ID=7491 TID=0 MSGN=58 MSG="Copy failed" RQID=dfe35d4200000001096d3fe6d436944ba CNS
=tbed0082 FID=000000000167f944 DLF.TPVID1=P01331 TPSERV="tpsrv031.cern.ch" TPDRIV
E="994B53A3" DLF.SRQID1=00000000000000000000000000000000 FSEQ=237 BLKID="000de1e8
" DISKPATH="lxfsrk4303:/shift/lxfsrk4303/data01//36/23591236@tbed0082.120716" STA
TUS=1 CPRC=-1 ERROR_CODE=5 ERROR_STR="locate: TP042 - /dev/sga : scsi error : Med
ium error ASC=15 ASCQ=0  " RTCP_SEVERITY=68
```

5.3.5.2 Alert messages

The alert messages usually requires manual intervention. Table **??** lists the possible alert messages logged by *recaller* program.

5.4 MigHunter: the default migration candidate hunter daemon

```
($Id: MigHunter.tex,v 1.7 2005/08/03 13:12:54 obarring Exp $)
```

Message nb	Message text	Comments
3	failed system call	A system or CASTOR call failed (see Section 5.1.2.2 for further explanation)
16	External logger error message	Error logged by a log-function in the RTCOPY API. These messages normally follows (or preceeds) a "normal" error message and are always logged when an error is propagated between the mover and its client.
22	Database service error	A system or catalog database interface call failed (see Section 5.1.2.2 for further explanation)
58	Copy failed	The mover got an error while copying the file. The corresponding error code, message text and severity are logged.

Table 5.8: Most common *recaller* error messages

Message nb	Message text	Comments
23	Retrieved inconsistent tape request	The **Tape** row database key passed with the `-k <dbKey>` command option to the *recaller* program does not correspond to the specified tape VID. Unless the catalog database is corrupted (this has never happened up to now) this error can only happen if the *recaller* program has been run directly from the command line (e.g. for debugging purposes).
60	Checksum algorithm name too long	The name of the checksum algorithm returned by the *rtcpd* tape mover is too long (as defined by `CA_MAXCKSUMNAMELEN` in *h/Castor_limits.h*).
76	Recalled disk file has wrong size	The recaller is configured to check the size of the disk files (see Section 6.1.4.4) and the size does not match the size in the stager catalog database.

Table 5.9: Most common *recaller* alert messages

The *MigHunter* is a single-threaded that can be started as a daemon or run as a normal command [3]. It is a client to both the catalog database, the expert system (calling the migrator policy), DLF (for the logging) and Cns.

5.4.1 Shortcomings/simplifications in current implementation and options for the future

In its current implementation the *MigHunter* works at a **SvcClass** (service class) level. This allows for setting different stream creation conditions (migration trigger level) between service classes. The disadvantage with the current implementation is that while a **TapePool** can be shared among several **SvcClass**es, a given **Stream** always belong to one **SvcClass**.

Another simplification in the current implementation is the selection of migration candidates, which blindly selects all **TapeCopy**s with status `TAPECOPY_CREATED` and `TAPECOPY_TOBEMIGRATED`. To allow for more control of what should be migrated and when, the implementation can trivially be changed to only select candidates with status `TAPECOPY_TOBEMIGRATED` and delegate the decision to update from `TAPECOPY_CREATED` to `TAPECOPY_TOBEMIGRATED` to an external policy program/script.

For operational backward compatibility, a *legacy* mode is supported where all the migration attributes (tape pools, nb drives) are taken from the file class definition in the CASTOR name server. This mode is not recommended since it will create tape pools on the fly and attempt to distributes the streams over the tape pools while at the same time respecting the number of streams constrain (`nbDrives`) defined in the

SvcClass.

5.4.2 Work flow

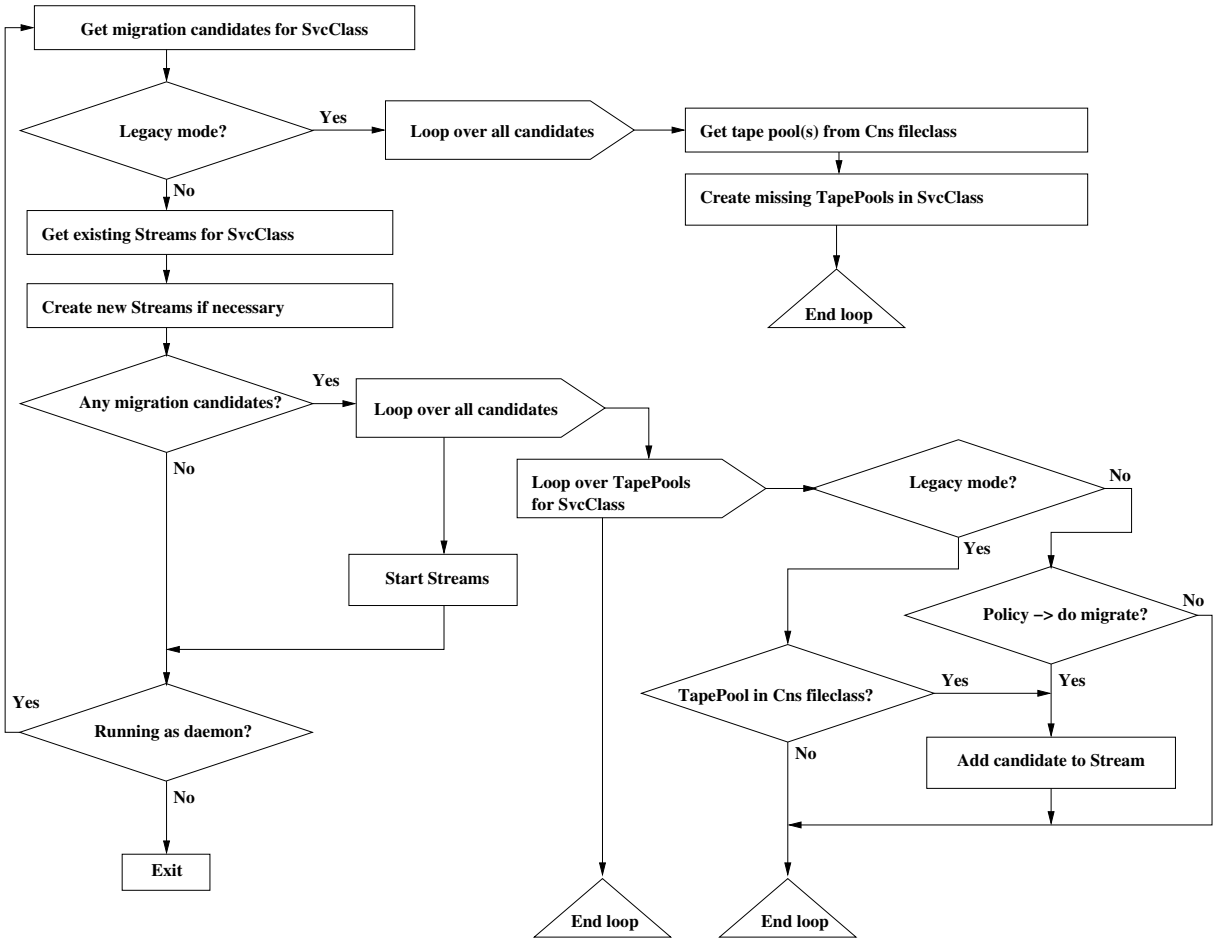

Figure 5.6: The *MigHunter* work flow

The Figure 5.6 shows the basic processing flow in the *MigHunter* program. The processing is composed of three parts:

- creation of new streams (if necessary), given the `nbDrives` attribute in the **SvcClass** and the associated **TapePool**s
- for each migration candidate, call the policy defined by the `migratorPolicy` attribute of the **SvcClass** and attach the candidate to the **Stream**s allowed by the policy.
- start the streams if not already running

The first step is implemented as a simple query to get all **TapePool**s linked to the **SvcClass**. The *MigHunter* assumes that all **TapePool**s are of equal importance and attempts to loadbalance the **Stream**s over them given the constraint imposed by the `nbDrives` attribute in the **SvcClass**. For instance, assume a **SvcClass** 'userSvc', which is linked to the two **TapePool**s 'smallFiles' and 'bigFiles'. If the `nbDrives` attributes of the 'userSvc' **SvcClass** is equal to '1', only one **Stream** will be created and associated with the first **TapePool** returned (the order depends on database implementation). If the `nbDrives` attribute is equal to '2', two **Streams** will be created distributed equally over the **TapePool**s (one **Stream** each). If the `nbDrives` attribute is equal to '3', there will be two **Stream**s linked to the first **TapePool** (database order)

and one **Stream** linked to the second **TapePool**. Clearly the first case (`nbDrives=1`) it does not make sense since the second **TapePool** will never be used. A better way to address this type of configurations is described in the Section 6.3.

The selection of candidates for migration is implemented with the following query:

```
SELECT TapeCopy.id
FROM TapeCopy, CastorFile, SvcClass
    WHERE TapeCopy.castorFile = CastorFile.id
    AND CastorFile.svcClass = SvcClass.id
    AND SvcClass.name = 'userSvc'
    AND TapeCopy.status IN (0, 1); -- TAPECOPY_CREATED, TAPECOPY_TOBEMIGRATED
```

To avoid that two *MigHunter* processes concurrently select the same set of migration candidates, the `status` attribute is atomically updated to `TAPECOPY_WAITINSTREAMS` for all **TapeCopy** rows returned by the above query.

For each combination of migration candidate and tape pools the migration policy given by the `migratorPolicy` in the **SvcClass** is called with the following arguments:

- tape pool name (17 chars), e.g. '*test_LTO2*'
- castor file name, e.g. '*/castor/cern.ch/user/m/me/myfile*' (1024 chars)
- copy number (dual tapecopy files) (11 chars)
- fileid, 64bit castor bit-file id (22 chars)
- filesize (64bit) (22 chars)
- filemode, file permission mode, e.g. 0644 (11 chars)
- uid, owner uid (11 chars)
- gid, owner gid
- atime, last access to file (11 chars)
- mtime, last file modification (11 chars)
- ctime, last file metadata modification (11 chars)
- fileclass, castor file class identifier (11 chars)

Example:

```
test_LTO2 /castor/cern.ch/alice/mdc6/tapes/tbed0043_00018688_2_1_20050429_111352.root
1 55101111 400 33206 30132 1395 1114766059 1114766101 1114766101 126
```

The policy returns '0' = do not migrate, '1' = do migrate. More examples for how to write migration policies are given in Section 6.3.

After all migration candidates have been inspected and attached to the **Stream**s, all **Stream**s with `status=STREAM_CREATED` are updated to `STREAM_PENDING` to allow for them to be picked up by the *rtcpclientd* (see Section 5.1).

Note that when attaching the migration candidates to the streams, the *MigHunter* do not make any distinction between newly created or running streams. This means that new candidates can immediately be picked up by a running *migrator*.

5.4.3 Log messages

This section lists and explains the most common DLF messages logged by the *MigHunter* program.

5.4.3.1 System messages and warnings

Table 5.10 lists the most common DLF messages logged by *migrator* (see appendix A for a list of all messages).

Message nb	Message text	Comments
72	Checked for new migration candidates	Parameters give the **SvcClass** name and the number of candidates returned by query.
73	Number of streams for SvcClass	
74	SvcClass has too many streams	This is a warning message logged when the *MigHunter* finds more streams than what is allowed by the `nbDrives` attribute in the **SvcClass**. Since removing a stream can potentially be a heavy operation, the *MigHunter* will not do it.
75	Migration volume below threshold	The amount of data to be migrated is below the limit specified with the `-v` option.

Table 5.10: Most common *MigHunter* system messages and warnings

Example showing the successful creation of 8 new migration streams:

```
DATE=20050429124702.818795 HOST=lxs5008.cern.ch LVL=System PRGN=6 PROG=MigHunter
 PID=14291 TID=-1 MSGN=73 MSG="Number of streams for SvcClass" RQID=a41072420000
0010b5e5f2ba0f950033 CNS=N/A FID=0000000000000000 SVCCLASS="default" NBOLDSTR=0
NBNEWSTR=8
```

5.4.3.2 Error messages

Table 5.11 lists the most common DLF messages logged by the *MigHunter* program (see appendix A for a list of all messages).

Message nb	Message text	Comments
3	failed system call	A system or CASTOR call failed (see Section 5.1.2.2 for further explanation)
22	Database service error	A system or catalog database interface call failed (see Section 5.1.2.2 for further explanation)
77	No tape pool for service class with migr candidates	The service class has no associated tape pool but there are waiting migration candidates.

Table 5.11: Most common *migrator* error messages

5.4.3.3 Alert messages

The alert messages usually requires manual intervention. Table 5.12 lists the possible alert messages logged by the *MigHunter* daemon.

5.5 TapeErrorHandler: the default tape error handler daemon

`($Id: TapeErrorHandler.tex,v 1.7 2005/05/13 15:05:02 obarring Exp $)`

The *TapeErrorHandler* is a single-threaded program that is usually started by the *rtcpclientd* daemon whenever a *recaller* or *migrator* child process exits with error status. The program can also be started from the

Message nb	Message text	Comments
68	Migration failed for file	(PUT_FAILED) This message, means disk file has not been migrated to tape. The reason is usually given in a preceeding error message. The most probable reason is that the Cns_statx() system call failed.
71	Service shutdown	The *rtcpclientd* server shutdown

Table 5.12: Alert messages logged by *rtcpclientd*

command line and it does not take any arguments. The *TapeErrorHandler* queries the catalog for all rows in the **Segment** table where the status is SEGMENT_FAILED (6). Thereafter it calls a retry policy via the expert system for each segment to find out whether the error is retry-able or not according to the current policy. The *TapeErrorHandler* is a client to both the catalog database, DLF (for the logging), VMGR and Cns.

5.5.1 Work flow

The Figure 5.7 shows the basic processing flow in the *TapeErrorHandler* program. When the program starts up it queries the catalog database for all **Segment** rows with status set to SEGMENT_FAILED. The corresponding query is:

```
SELECT id FROM Segment WHERE status = 6; -- SEGMENT_FAILED
```

The *TapeErrorHandler* loops over all failed segments (if any) and calls the expert system to apply the current retry policy. Different retry policies apply for migration and recall.

Depending on the policy output and whether the failed segment comes from a migration or recall, the catalog is updated as shown in Table 5.13.

type	policy	Catalog updates and actions
migration	no retry	**TapeCopy**.status = TAPECOPY_FAILED. An alert message is logged in DLF
migration	do retry	**Segment**.status = SEGMENT_RETRIED, **TapeCopy**.status = TAPECOPY_CREATED
recall	no retry	**DiskCopy**.status = DISKCOPY_FAILED, **SubRequest**.status = SUBREQUEST_FAILED for all waiting subrequests. An error is propagated back to all waiting clients.
recall	do retry	**Segment**.status = SEGMENT_RETRIED, clone the failed segment and set **Segment**.status = SEGMENT_UNPROCESSED for the new segment. The corresponding **Tape** status is checked and if the tape is not running (not mounted or waiting in VDQM), set **Tape**.status = TAPE_PENDING

Table 5.13: Catalog updates

5.5.2 Policy parameters and return value

In the current implementation, the input to the retry policy is an array with the following information for the failed segment and all previous retries:

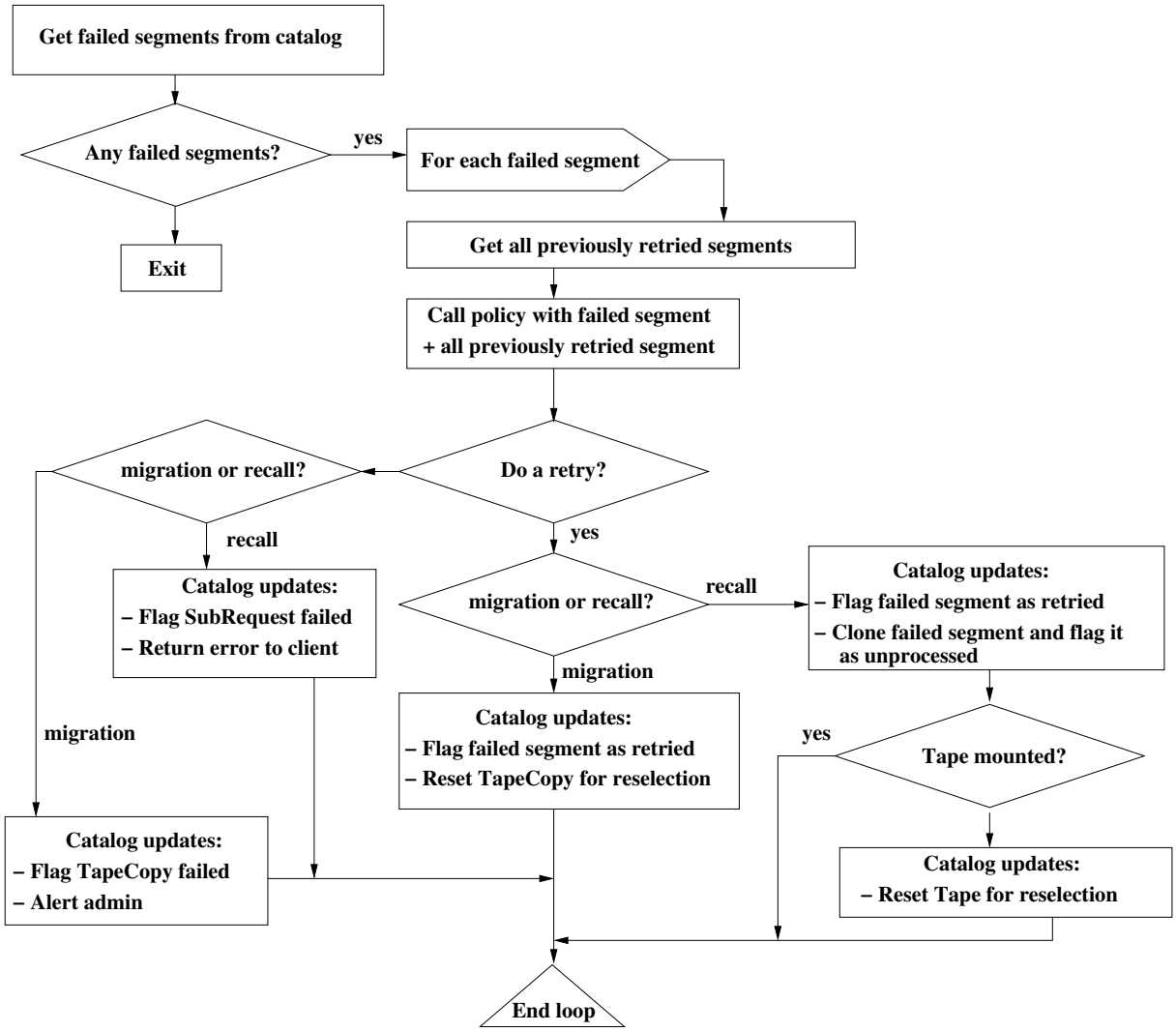

Figure 5.7: The *TapeErrorHandler* work flow

- **errorCode**: is usually the `errno`, `serrno` or `rfio_errno` set by the failing subcomponent (tape drive, disk server, network, ...). An example is errorCode=1917 (`ETPARIT`: tape media error)

- **severity**: is the severity flagged by the tape mover *rtcpd*. It is bitwise 'or' of failure conditions constants defined in *h/rtcp_constants.h* in the CASTOR CVS. An example is 12 = `RTCP_FAILED`(8) + `RTCP_RESELECT_SERV`(4).

- **errorMsgTxt**: message text giving more details about the error. An example is "CPTPDSK ! locate: TP042 - /dev/sga : scsi error : Medium error ASC=15 ASCQ=0"

The retry policy module is called as follow:

```
migratorRetryPolicy.pl error1 severity1 "msg1" error2 severity2 "msg2" ...
```

The policy returns a number, 0 == *no retry*, 1 == *do retry*.

5.5.3 Log messages

5.5.3.1 System messages and warnings

Table 5.14 lists the most common DLF messages logged by *TapeErrorHandler* (see appendix A for a list of all messages).

Message nb	Message text	Comments
67	Tape Error retry postponed	There is no currently available disk file (file system or disk server are out of service). This message is only logged for failed migrations.
69	Recall retry rejected by policy	

Table 5.14: Most common *TapeErrorHandler* system messages and warnings

The log excerpt below is an example of a rejected tape-recall retry:

```
DATE=20050401181311.136223 HOST=tbed0083.cern.ch LVL=Warning PRGN=7 PROG=TapeErro
rHandler PID=524 TID=-1 MSGN=69 MSG="Recall retry rejected by policy" RQID=000000
00000000000000000000000 CNS=tbed0082 FID=00000000016b158d DBKEY=184202 File="T
apeErrorHandler.c" Line=719 errno=0 serrno=0
```

Each message is logged with a number of DLF parameters providing more detailed information, for instance (in the above example) the CASTOR bitfile id (FID), which helps the tracing of a particular CASTOR file through out the whole stager system.

5.5.3.2 Error messages

Table 5.15 lists the most common DLF messages logged by the *TapeErrorHandler* program (see appendix A for a list of all messages).

Message nb	Message text	Comments
3	failed system call	A system or CASTOR call failed (see Section 5.1.2.2 for further explanation)
22	Database service error	A system or catalog database interface call failed (see Section 5.1.2.2 for further explanation)
70	Name server segment no longer valid	The corresponding CASTOR name server segment is invalid: the file has been deleted or the tape is in DISABLED, EXPORTED or ARCHIVED status in VMGR.

Table 5.15: Most common *TapeErrorHandler* error messages

5.5.3.3 Alert messages

The alert messages usually requires manual intervention. Table 5.16 lists the possible alert messages logged by *TapeErrorHandler* program.

Message nb	Message text	Comments
68	Migration failed for file	(`PUT_FAILED`) This message means that a disk file has not been migrated to tape. An administrator intervention is required to fix the problem.

Table 5.16: Most common *TapeErrorHandler* alert messages

Chapter 6

Operating the CASTOR tape migration and recall

```
($Id: mrOperation.tex,v 1.17 2007/10/02 12:00:46 obarring Exp $)
```

In this chapter it is explained how to setup and operate the CASTOR tape migration and recall. The first section explains how to install and configure the *rtcpclientd* service for proper functioning. Thereafter it is explained how to enable migration for a *SvcClass*. Various examples show how the different attributes can be used to Tailor the migration to meet the needs for a particular service class. Advanced features such as selective migration given important file attributes are also discussed.

6.1 Installation and configuration of tape migration/recall services

6.1.1 Installation with RPM

The *rtcpclientd*, *migrator*, *recaller*, *TapeErrorHandler* and *MigHunter* programs are packaged in the *castor-rtcopy-clientserver* RPM. The RPM depends on *castor-lib*, *castor-lib-oracle* and castor-dbtools RPMs. Once installed the service is managed under the name *rtcpclientd*, e.g.

```
[root@castor2srv02 root]# chkconfig --list rtcpclientd
rtcpclientd     0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

The service can be stopped or (re-)started manually using the service script, e.g.

```
service rtcpclientd stop
service rtcpclientd start
```

Using the service scripts to stop the service is recommended. If service is stopped using other means or if the node crashes a manual cleanup of **BUSY** tapes in VMGR may be required.

For tape recalls it is enough with the *rtcpclientd* service running while tape migrations will not start unless the *MigHunter* process is also running. The *MigHunter* can run as a manged service. However, since the *MigHunter* is specific to each *SvcClass* there should usually be one *MigHunter* process per *SvcClass*. The service script lists uses the *listSvcClass* command to list all defined service classes and starts one *MigHunter* for each. The file */etc/sysconfig/MigHunter* defines the parameters (e.g. migration triggering interval) used when starting the *MigHunter* processes. If each service class is required to have its own tailored migration triggers, it is better to disable the *MigHunter* service and start the processes manually.

Each program write log-messages in its own DLF (Distributed Logging Facility) facility, which therefore must be created before the service is started. The corresponding DLF facility names are:

- **rtcpcld** - the *rtcpclientd* daemon
- **migrator** - the *migrator* program forked by *rtcpclientd*
- **recaller** - the *recaller* program forked by *rtcpclientd*
- **MigHunter** - the *MigHunter* daemon
- **TapeErrorHandler** - the *TapeErrorHandler* program forked by *rtcpclientd*

See the *dlfenterfacility* man-page [4] for how to create new DLF facilities. Example:

```
dlfenterfacility -F rtcpcld -n 10
dlfenterfacility -F migrator -n 11
dlfenterfacility -F recaller -n 12
dlfenterfacility -F MigHunter -n 13
dlfenterfacility -F TapeErrorHandler -n 14
```

(requires the DLF client RPM, castor-dlf-client, to be installed and configured).

6.1.2 Installation from source tar-file

When installing from a source tar-file, the build macros in *config/site.def* should be set as follows:

```
#define BuildRtcopyClient       YES
#define BuildRtcopyServer       NO
#define BuildRtcopyLibrary      YES
...
#define BuildOraCpp             YES
```

The software can be then built and installed with the commands:

```
make depend
make install
```

Note that the Oracle (or, later, MySQL) environment must be set, e.g.

```
export ORACLE_CERN=/afs/cern.ch/project/oracle
export ORACLE_HOME=/usr/lib/oracle/10.1.0.2/client
export TNS_ADMIN=${ORACLE_CERN}/admin
export PATH=${PATH}:${ORACLE_CERN}/script:${ORACLE_HOME}/bin
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${ORACLE_HOME}/lib
```

Once the binaries and libraries have been installed, the service can be started from the command line:

```
/usr/bin/rtcpclientd
```

In order to allow for the necessary cleanup of **BUSY** tapes in VMGR, it is recommended to stop the service by killing the *rtcpclientd* daemon with one the signals SIGINT (2) or SIGTERM (15). Example:

```
[root@tbed0083 root]# ps -ef |grep rtcpclientd
stage     1999     1  0 May30 ?        00:00:04 /usr/bin/rtcpclientd
root     29003 28935  0 09:38 pts/1    00:00:00 grep rtcpclientd
[root@tbed0083 root]# kill -INT 1999
```

6.1.3 CASTOR name server (Cns) privileges

All tape migration and recall take place on behalf of the client user under a generic account, usually login name 'stage' and group 'st'. This generic account must therefore be privileged for reading and writing any file in the castor name server, which is granted by giving it **ADMIN** privilege in the CASTOR User Privilege Verification (Cupv) service, e.g.

```
Cupvadd --src castor2srv02 --tgt castorsrv1 --user stage --group st --priv ADMIN
Cupvadd --src castor2srv02 --tgt castorsrv2 --user stage --group st --priv ADMIN
Cupvadd --src castor2srv02 --tgt castorsrv3 --user stage --group st --priv ADMIN
```

where `castor2srv02` is the hostname of the node where the *rtcpclientd* daemon runs and `castorsrv1,2,3` are the hosts where the CASTOR name server runs. Also the *MigHunter* process needs **ADMIN** privileges and if it runs under a different account (e.g. 'root') it must be added using the *Cupvadd* command like above.

6.1.4 Configuration

The tape migration/recall is configured in */etc/castor/castor.conf*.

6.1.4.1 Catalog configuration

The following configuration defines the Oracle conversion services to be used. It is mandatory where Oracle is used (similar configuration configuration parameters will be used for MySQL):

```
OraCnvSvc       user    [dbUser]
OraCnvSvc       passwd  [dbPassword]
OraCnvSvc       dbName  [dbInstanceName]
OraStagerSvc    user    [dbUser]
OraStagerSvc    passwd  [dbPassword]
OraStagerSvc    dbName  [dbInstanceName]
```

where the "[dbUser]", "[dbPassword]", etc. should be replaced with the catalog database username, password and instance name to be used.

6.1.4.2 Log-streams configuration

The DLF logging destinations should be defined for each facility. The *rtcpclientd*, *migrator*, *recaller* and *MigHunter* programs use the following DLF loglevels:

- **LOGDEBUG** - exhaustive debug logging aimed for debugging
- **LOGUSAGE** - trace (log entry and exit) for some blocking calls. Can be used to check timing of catalog data-base calls.
- **LOGSYSTEM** - all normal operation information messages
- **LOGWARNING** - non-fatal exceptions
- **LOGERROR** - system and catalog data-base interface errors
- **LOGALERT** - operational alerts usually requiring an intervention
- **LOGSECURITY** - failed authentication

Here follows an example */etc/castor/castor.conf* excerpts for how the logging could be configured:

```
rtcpcld LOGSYSTEM     file:///var/spool/rtcpclientd/rtcpclientd.log x-dlf://lxs5008.cern.ch/
rtcpcld LOGWARNING    file:///var/spool/rtcpclientd/rtcpclientd.log x-dlf://lxs5008.cern.ch/
rtcpcld LOGERROR      file:///var/spool/rtcpclientd/rtcpclientd.log x-dlf://lxs5008.cern.ch/
rtcpcld LOGALERT      file:///var/spool/rtcpclientd/rtcpclientd.log x-dlf://lxs5008.cern.ch/
rtcpcld LOGSECURITY   file:///var/spool/rtcpclientd/rtcpclientd.log x-dlf://lxs5008.cern.ch/
migrator LOGUSAGE     file:///var/spool/rtcpclientd/migrator.log
migrator LOGSYSTEM    file:///var/spool/rtcpclientd/migrator.log x-dlf://lxs5008.cern.ch/
migrator LOGWARNING   file:///var/spool/rtcpclientd/migrator.log x-dlf://lxs5008.cern.ch/
migrator LOGERROR     file:///var/spool/rtcpclientd/migrator.log x-dlf://lxs5008.cern.ch/
migrator LOGALERT     file:///var/spool/rtcpclientd/migrator.log x-dlf://lxs5008.cern.ch/
migrator LOGSECURITY  file:///var/spool/rtcpclientd/migrator.log x-dlf://lxs5008.cern.ch/
recaller LOGSYSTEM    file:///var/spool/rtcpclientd/recaller.log x-dlf://lxs5008.cern.ch/
recaller LOGWARNING   file:///var/spool/rtcpclientd/recaller.log x-dlf://lxs5008.cern.ch/
recaller LOGERROR     file:///var/spool/rtcpclientd/recaller.log x-dlf://lxs5008.cern.ch/
recaller LOGALERT     file:///var/spool/rtcpclientd/recaller.log x-dlf://lxs5008.cern.ch/
recaller LOGSECURITY  file:///var/spool/rtcpclientd/recaller.log x-dlf://lxs5008.cern.ch/
MigHunter LOGSYSTEM   file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
MigHunter LOGWARNING  file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
MigHunter LOGERROR    file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
MigHunter LOGALERT    file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
MigHunter LOGSECURITY file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
TapeErrorHandler LOGSYSTEM   file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
TapeErrorHandler LOGWARNING  file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
TapeErrorHandler LOGERROR    file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
TapeErrorHandler LOGALERT    file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
TapeErrorHandler LOGSECURITY file:///var/spool/rtcpclientd/MigHunter.log x-dlf://lxs5008.cern.ch/
```

With the above configuration, all facilities log the levels **LOGSYSTEM**, **LOGWARNING**, **LOGERROR**, **LOGALERT** and **LOGSECURITY** both to local log-file and to a DLF server instance running on *lxs5008.cern.ch*. In addition, the *migrator* also writes **LOGUSAGE** (trace information) messages to its local logfile.

6.1.4.3 Expert system policy configurations for migration/recall

Both tape migration, recall and automatic retries can optionally be controlled with policies. The policy input data is formatted for policies written in some script language (e.g. *perl*). Since each **SvcClass** may have its own migrator and recaller policies, the convention for dynamically finding the corresponding policy scripts is required. The convention used for migrator and recaller policies is that the first argument of the policy input data is always the policy name itself.

All migration and recall policies are called via a global policy script configured in the */etc/castor/EXPCONFIG* on the host running the *expertd* daemon. An example for the global policy scripts configuration:

```
[root@lxs5009 root]# cat /etc/castor/EXPCONFIG
EXP_RQ_FILESYSTEM       /etc/expert/    /usr/bin/clips -f bestfs.clp
EXP_RQ_REPLICATION      /etc/expert/    /usr/bin/perl replica.pl
EXP_RQ_MIGRATOR /etc/castor/expert/     /usr/bin/perl migrator.pl
EXP_RQ_RECALLER /etc/castor/expert/     /usr/bin/perl recaller.pl
[root@lxs5009 root]#
```

An example of the policy scripts (*migrator.pl* and *recaller.pl*) implementations:

```
#!/usr/bin/perl -w

use strict;
use diagnostics;
use POSIX;
```

```
my @input = split(" ",<>);

#
# First argument == name of policy to execute
#
my $policyName = shift @input;

my $rc = system("/usr/bin/perl -w $policyName @input");

exit($rc);
```

The host name where the *expertd* daemon runs must be configured in */etc/castor/castor.conf* hosts running the *MigHunter*. Example:

```
EXPERT  HOST    lxs5009
```

6.1.4.4 Miscellaneous migration/recall configuration

The tape migration/recall service checks the following configuration in */etc/castor/castor.conf*:

migrator HANDLE_TPERROR YES/NO (default is **YES**): enable/disable migrator for automated action upon tape errors. If enabled and one of the following errors is encountered, the tape status will be set to `RDONLY` in VMGR and the corresponding *TapeCopy* will be re-enabled for migration (to another tape):

- ETPARIT: parity error (media error)
- ETUNREC: unrecoverable media error
- ETLBL: bad label structure

Note that this feature only exists for the *migrator*. The *recaller* will never decide to retry or change tape status in VMGR by itself.

migrator CHECKFILE YES/NO The migrator will check the disk files (successful *rfio_stat64()* before sending them to the mover. This is only useful if the catalog contains references to disk files that have been physically deleted, which should normally never be the case. Default is **NO**.

recaller CHECKFILE YES/NO The recaller will check the disk files (successful *rfio_stat64()* after they have been staged by the mover. An alert message is logged (see Section 5.3.5.2 if the disk file size does not match the size in the CASTOR name server. This extra check is not needed unless one suspects a problem with the recall of multi-segment files (single-segment files must match the segment checksum). Default is **NO**.

migrator CHECKFSEQ YES/NO If set to **YES** the *migrator* will check the start filesequence number (FSEQ) of the tape given by VMGR with the last FSEQ known by the CASTOR name server. This is an extra check to assure that the VMGR and CASTOR name server are consistent.

migrator THREAD_POOL thread-poolsize (*migrator* only). Size of the migrator thread pool (default = 3)

recaller THREAD_POOL thread-poolsize (*recaller* only). Size of the recaller thread pool (default = 3)

rtcpcld NOTIFY_HOST hostname Hostname where the *rtcpclientd* daemon is running. This is used for clients, such as the stager, that wish to notify the *rtcpclientd* about new requests in the catalog.

rtcpcld NOTIFY_PORT portnumber UDP port on which the *rtcpclientd* daemon listens for notification messages . This is used for clients, such as the stager, that wish to notify the *rtcpclientd* about new requests in the catalog.

rtcpcld USE_CHECKSUM YES/NO Switch on/off segment checksum support (default is **YES**)

rtcpcld CHANGE_CHECKSUM_NAME YES/NO overwrite existing checksum name in CASTOR name server in case the tape mover uses a different algorithm (default is **NO**)

TAPEERRORHANDLER MIGRATOR_RETRY_POLICY retryPolicyName Name of the migrator retry policy to be used by the *TapeErrorHandler*. A policy implementation (script) with the same name should exist in the */etc/castor/expert* directory on the host running the *expertd* daemon.

TAPEERRORHANDLER RECALLER_RETRY_POLICY retryPolicyName Name of the recaller retry policy to be used by the *TapeErrorHandler* (see previous point).

None of those configurations are required for normal functioning of the system. They may, however, be useful under certain circumstances. Examples are:

- A global change of the checksum algorithm (**rtcpcld CHANGE_CHECKSUM_NAME YES**)
- A broken file system has been put back in production but the repair unfortunately scratched all files (**migrator CHECKFILE YES** instructs the *migrator* to check and skip lost disk files)

6.2 Importing FileClasses from Cns

Before tape migration/recall, and indeed the stager as a whole, can function correctly, the Cns fileclass names must be imported (cached) in the **FileClass** table. The stager will refuse to process file requests associated with a Cns fileclass that is not defined in the **FileClass** table. The *enterFileClass* [5] and *modifyFileClass* [6] commands both support the option `--GetFromCns`. The commands will create/modify the **FileClass** rows in the catalog database. The only relevant attribute is the `nbCopies`, which defines the number of tape-copies to be created for a given **CastorFile**. See Section 6.5 for an explanation of the role of the **FileClass**, and Sections 5.2.3 and 6.3.5 for how the dual copy migration works.

6.3 Enabling tape migration for a service class (SvcClass)

This section describes the different steps necessary for tailoring the tape migration for a service class. Examples are given to illustrate the use of the different tools and also how to write migration policies for different purposes.

6.3.1 The steps to take

The following steps are required for enabling a **SvcClass** for migration:

Create the SvcClass Before anything else, the service class itself has to be created in the catalog database. This is described in Section 6.3.2.

Add/remove tape pools Add/remove VMGR tape pools to the **SvcClass** and set the number of tape drives to be used for the migration, see Sections 6.3.2.2 and 6.3.3.

Start the MigHunter The *MigHunter* process is responsible for creating **Stream**s and connect migration candidates to those. The *MigHunter* triggers migration on either time or data volume. The details on how to start the *MigHunter* and how to set the migration triggers are given in Section 6.3.4.

Tailored migration streams Use of migration policies is described in Section 6.3.6.

Imortant: all of the steps can be performed prior to having started the *rtcpclientd* daemon. It should be noted, however, that no data will be moved to/from tape until the *rtcpclientd* daemon has been started. See Section 6.1.1 or 6.1.2 for how to start and stop the service.

6.3.2 Creating a service class

The *enterSvcClass* command [8] can be used to create a row in the **SvcClass** table. If not all **SvcClass** attributes have been decided when the service class is created, they can later be updated using the *modifySvcClass* command [9]. The `printSvcClass` command can be used to list the attributes a **SvcClass** and its associated tape and disk pools.

6.3.2.1 enterSvcClass example: SvcClass with default settings

Create a **SvcClass** called `test` with default settings. A warning is printed in case the **SvcClass** already exists.

```
[root@lxs5008 root]# enterSvcClass --Name test
Adding SvcClass: test
[# SvcClass #]
nbDrives : 0
name : test
defaultFileSize : 0
maxReplicaNb : 1
replicationPolicy :
gcPolicy :
migratorPolicy :
recallerPolicy :
id : 0
TapePools :
DiskPools :
[root@lxs5008 root]#
```

The attributes of the created **SvcClass** can be checked at any time with the *printSvcClass* command:

```
[root@lxs5008 root]# printSvcClass test
[# SvcClass #]
nbDrives : 0
name : test
defaultFileSize : 0
maxReplicaNb : 1
replicationPolicy :
gcPolicy :
migratorPolicy :
recallerPolicy :
id : 59556
TapePools :
DiskPools :
   ............... SvcClass has no tape pools ........
   ............... SvcClass has no disk pools ........
```

6.3.2.2 enterSvcClass example: SvcClass with tape pools

Create a **SvcClass** called `user` with two tape pools `smallFiles`, `largeFiles` and one disk pool `userDisks`. The **SvcClass** is configured to use at most two tape drives for migration.

```
[root@lxs5008 root]# enterSvcClass --Name user --NbDrives 2 --TapePools smallFiles:largeFiles \
--DiskPools userDisks
Adding SvcClass: user
[# SvcClass #]
nbDrives : 2
name : user
defaultFileSize : 0
```

```
maxReplicaNb : 1
replicationPolicy :
gcPolicy :
migratorPolicy :
recallerPolicy :
id : 0
TapePools :
DiskPools :
Add tape pool smallFiles to SvcClass user
Add tape pool largeFiles to SvcClass user
Add disk pool userDisks to SvcClass user
[root@lxs5008 root]#
```

6.3.3 Add/remove tape pools

Adding and removing of tape pools (and disk pools) can be performed using the *modifySvcClass* command, see [9].

6.3.3.1 modifySvcClass example: adding tape pools

Adding two tape pools `test1` and `test2` to an existing **SvcClass** `test`:

```
[root@lxs5008 hsmtools]# modifySvcClass --Name test --AddTapePools test1:test2
Add tape pool test1
Add tape pool test2
[root@lxs5008 hsmtools]#
```

Print the **SvcClass** attributes and all its tape pools and disk pools using the *printSvcClass* command:

```
[root@lxs5008 hsmtools]# printSvcClass test
[# SvcClass #]
nbDrives : 0
name : test
defaultFileSize : 0
maxReplicaNb : 1
replicationPolicy :
gcPolicy :
migratorPolicy :
recallerPolicy :
id : 59564
TapePools :
  0 :
    [# TapePool #]
    name : test1
    id : 59565
    SvcClasses :
      0 :
        [# SvcClass #]
        Back pointer, see above
    Streams :
  1 :
    [# TapePool #]
    name : test2
    id : 59566
    SvcClasses :
      0 :
        [# SvcClass #]
        Back pointer, see above
```

```
    Streams :
DiskPools :
[root@lxs5008 hsmtools]#
```

The **SvcClass** now has 2 tape pools (`test1`, `test2`). However, it is not yet ready for tape migration since the `nbDrives` attribute in the **SvcClass** is equal to zero. With two tape pools, the `nbDrives` should be greater or equal to two otherwise one of the pools will never be used (see Section 5.4.2). The *modifySvcClass* program can be used to set the number of drives:

```
[root@lxs5008 root]# modifySvcClass --Name test --NbDrives 2
[root@lxs5008 root]# printSvcClass test
[# SvcClass #]
nbDrives : 2
name : test
defaultFileSize : 0
maxReplicaNb : 1
replicationPolicy :
gcPolicy :
migratorPolicy :
recallerPolicy :
id : 59564
TapePools :
  0 :
    [# TapePool #]
    name : test1
    id : 59565
    SvcClasses :
      0 :
        [# SvcClass #]
        Back pointer, see above
    Streams :
  1 :
    [# TapePool #]
    name : test2
    id : 59566
    SvcClasses :
      0 :
        [# SvcClass #]
        Back pointer, see above
    Streams :
DiskPools :
[root@lxs5008 root]#
```

Removal of tape pools from a **SvcClass** can be performed using the `--removeTapePools` option to the `modifySvcClass` command. Note that the **TapePool** is not removed from the catalog (since other **SvcClass**es may be using it) - it is only the association with the **SvcClass** that is removed. Example:

```
[root@lxs5008 root]# modifySvcClass --Name test --RemoveTapePools test1
Remove tape pool test1
```

6.3.4 Starting the MigHunter

Once a **SvcClass** has configured with some tape pools and the `nbDrives` attribute is greater than zero, it is ready for tape migration. The final step is to start the *MigHunter* program for the service class. In each iteration the *MigHunter* program first checks the existing **Stream**s (if any) for the service class and creates new ones if ncessary. Thereafter it loops through the migration candidates for the service class and attach them one-by-one to all the **Stream**s. The detailed workflow for the *MigHunter* program is described in Section 5.4.

6.3.4.1 Run MigHunter in foreground

The first time a *MigHunter* is started for a new **SvcClass** it may be useful to run it in foreground. In foreground mode, the *MigHunter* goes through one iteration and exits.

```
[root@lxs5008 root]# /usr/bin/MigHunter test
Get service class test
Get migration candidates for test
Found 0 candidates for test
Create new tape pools for test if necessary
Initial size to transfer: 0
Get existing streams for test
Trapped signal 0: Restore 0 migration candidates
[root@lxs5008 root]#
```

In this case, the **SvcClass** did not contain any files to migrate and no streams were created. An example with an active **SvcClass**:

```
[root@lxs5008 root]# /usr/bin/MigHunter default
Get service class default
Get migration candidates for default
Found 1287 candidates for default
Create new tape pools for default if necessary
Initial size to transfer: 953200330222
Get existing streams for default
Found 0 streams for default. Create new streams if necessary
SvcClass allows for 8 streams, found 0
Create 8 new streams
8 new streams created for default
Add migration candidates to streams for default
Start the streams for default
Start 0 stream for tape pool test_LTO2
Start 1 stream for tape pool test_LTO2
Start 2 stream for tape pool test_LTO2
Start 3 stream for tape pool test_LTO2
Start 4 stream for tape pool test_LTO2
Start 5 stream for tape pool test_LTO2
Start 6 stream for tape pool test_LTO2
Start 7 stream for tape pool test_LTO2
Trapped signal 0: Restore 1287 migration candidates
Restoring unattached migration candidates. Check 1287 candidates
Restore procedure finished
[root@lxs5008 root]#
```

6.3.4.2 Run MigHunter as a daemon with time or volume triggered migration

The *-d* switch will start the *MigHunter* program in background daemon mode. The *-t time (secs)* and *-v dataVolume* options allows for setting time and data volume triggered migrations. If the *-t time* option is not given, the *MigHunter* will check for new candidates every 10 minutes.

```
[root@lxs5008 root]# /usr/bin/MigHunter -t 300 -d default
[root@lxs5008 root]# ps -ef |grep MigHunter
root     29468     1  1 12:06 ?        00:00:00 /usr/bin/MigHunter -t 300 -d default
root     29470 29327  0 12:06 pts/1    00:00:00 grep MigHunter
[root@lxs5008 root]#
```

In the example above, the *MigHunter* will check for new migration candidates every 5 minutes (300 seconds). Below is an example of a data volume triggered migration

```
[root@lxs5008 root]# /usr/bin/MigHunter -v 100G -d test
[root@lxs5008 root]# ps -ef |grep MigHunter
root     29468     1  0 12:06 ?        00:00:00 /usr/bin/MigHunter -t 300 -d default
root     29480     1  1 12:08 ?        00:00:00 /usr/bin/MigHunter -v 100G -d test
root     29483 29327  0 12:08 pts/1    00:00:00 grep MigHunter
[root@lxs5008 root]#
```

The *MigHunter* for the `test` service class is not start any new migrations unless there are at least 100GB of data to migrate. Note that the data volume trigger is only respected when starting new streams. If the service class has already running streams, new candidates will automatically be added to those.

6.3.5 Dual tape copy migrations

The number of copies to be migrated to tape for a given **CastorFile** is decided by the `nbCopies` attribute in the **FileClass** table. The `nbCopies` can be modified with *enterFileClass* [5] or *modifyFileClass* [6] commands. Example:

```
[root@lxs5008 root]# enterFileClass --Name dualCopy --NbCopies 2
Adding FileClass: dualCopy
[# FileClass #]
name : dualCopy
minFileSize : 0
maxFileSize : 0
nbCopies : 2
id : 0
[root@lxs5008 root]#
```

See Sections 5.2.3 and 6.2 for more details about the **FileClass** class.

6.3.6 Tailoring selective migration streams using policies

The *MigHunter* program checks if the **SvcClass** has an non-empty `migratorPolicy` attribute. The policy name can be set/reset using the *modifySvcClass* command:

```
[root@lxs5008 rtcopy]# modifySvcClass --Name default --MigratorPolicy testMigPolicy.pl
```

sets the policy and

```
[root@lxs5008 rtcopy]# modifySvcClass --Name default --MigratorPolicy ""
```

resets the policy (== no policy).

The corresponding script must exist in the */etc/castor/expert* policy repository on the host running the *expertd* daemon (see Section 6.1.4.3).

The migrator policy script is called for *each* migration candidate *and* tape pool. The input data is:

tape pool name Name of the **TapePool** associated with the migration stream

castor file name e.g. `/castor/cern.ch/user/m/me/myfile`

copy number sequence number of the tape copy (dual copy files)

fileid 64bit castor bit-file id

filesize (64bit)

filemode file permission mode

uid file owner uid

gid file owner gid

atime last access to file

mtime last file modification

ctime last file metadata modification

fileclass id castor file class identifier

For instance, a **SvcClass** is configured with two tape pools: `test_LTO2` and `alicemdc6_3592`:

```
...
test_LTO2 /castor/cern.ch/alice/mdc6/tapes/tbed0058_00018875_1_2_20050503_123511.root
1 55328119 12582912 33206 30132 1395 1115116511 1115116543 1115116543 126
alicemdc6_3592 /castor/cern.ch/alice/mdc6/tapes/tbed0058_00018875_1_2_20050503_123511.root
1 55328119 12582912 33206 30132 1395 1115116511 1115116543 1115116543 126
...
```

The following subsections give examples for different migration policy implementations.

6.3.6.1 Use-case 1: selective migration based on filesize

A **SvcClass** `userSvc` is configured with two tape pools: `fastAccess` and `largeCapacity`. The former contains with relatively low capacity but with fast mount/load/position (e.g. STK 9840 media). The `largeCapacity` tape pool contains high capacity media with relatively slow mount/load/positioning time (e.g. STK 9940 media). The use-case to be addressed is the following: it is desirable to be able to selectively construct the migration streams so that small files, say `file-size < 100MBytes`, should be migrated to the `fastAccess` pool while large files should go to the `largeCapacity` pool. To achieve this the **SvcClass** is configured with two streams (`nbDrives = 2`) and a `migratorPolicy=userMigrPolicy.pl`, like follows:

```
[root@lxs5008 root]# printSvcClass userSvc
[# SvcClass #]
nbDrives : 2
name : userSvc
defaultFileSize : 0
maxReplicaNb : 1
replicationPolicy :
gcPolicy :
migratorPolicy : userMigrPolicy.pl
recallerPolicy :
id : 109577
TapePools :
  0 :
    [# TapePool #]
    name : fastAcces
    id : 109578
    SvcClasses :
      0 :
        [# SvcClass #]
        Back pointer, see above
    Streams :
  1 :
    [# TapePool #]
    name : largeCapacity
    id : 109579
    SvcClasses :
```

```
      0 :
        [# SvcClass #]
        Back pointer, see above
    Streams :
DiskPools :
  0 :
    [# DiskPool #]
    name : default
    id : 2
    FileSystems :
    SvcClasses :
      0 :
        [# SvcClass #]
        Back pointer, see above
[root@lxs5008 root]#
```

The implementation of the `userMigrPolicy.pl` would look something like:

```
#!/usr/bin/perl -w

use strict;
use diagnostics;
use POSIX;
my $doMigrate = 0;

END {print "$doMigrate\n";}

my ($tapePool,$castorFile,$fileid,$copynb,$fileSize,$mode,$uid,$gid,$atime,$mtim
e,$ctime,$fileClassId) = @ARGV;

if ( (($tapePool =~ "largeCapacity") && ($fileSize >= 100*1024*1024)) ||
     (($tapePool =~ "fastAccess") && ($fileSize < 100*1024*1024)) ) {
        $doMigrate = 1;
}

exit($doMigrate);
```

With the expert system correctly configured (see 6.1.4.3) it is enough to add the above perl script in */etc/castor/expert* policy repository for the **SvcClass** to be meet the use-case requirement.

6.3.6.2 Use-case 2: migration of dual copies to different pools

A **SvcClass** `rawDataSvc` is configured with two tape pools: `copy1` and `copy2`. The files written to `rawDataSvc` all belong to the Cns file-class `dualCopy`, which is defined with `nbCopies = 2`. The default behaviour of the *MigHunter* program is to assign both copies to all available **Stream**s. The desired behaviour is to selectively write files with `copyNb = 1` (in **TapeCopy** table) to tape-pool `copy1` and the files with `copyNb = 2` to `copy2`. To achieve this the **SvcClass** is configured with two streams (`nbDrives = 2`) and a `migratorPolicy=dualCopyMigrPolicy.pl`, like follows:

```
[root@lxs5008 root]# printSvcClass rawDataSvc
[# SvcClass #]
nbDrives : 2
name : rawDataSvc
defaultFileSize : 0
maxReplicaNb : 1
replicationPolicy :
gcPolicy :
migratorPolicy : dualCopyMigrPolicy.pl
recallerPolicy :
```

```
id : 1043201
TapePools :
  0 :
    [# TapePool #]
    name : copy1
    id : 1043202
    SvcClasses :
      0 :
        [# SvcClass #]
        Back pointer, see above
    Streams :
  1 :
    [# TapePool #]
    name : copy2
    id : 1043203
    SvcClasses :
      0 :
        [# SvcClass #]
        Back pointer, see above
    Streams :
DiskPools :
[root@lxs5008 root]#
```

The implementation of the `dualCopyMigrPolicy.pl` would look something like:

```
#!/usr/bin/perl -w

use strict;
use diagnostics;
use POSIX;
my $doMigrate = 0;

END {print "$doMigrate\n";}

my ($tapePool,$castorFile,$fileid,$copynb,$fileSize,$mode,$uid,$gid,$atime,$mtim
e,$ctime,$fileClassId) = @ARGV;

if ( (($tapePool =~ "copy1") && ($copynb == 1) ||
     (($tapePool =~ "copy2") && ($copynb == 2)) ) {
        $doMigrate = 1;
}

exit($doMigrate);
```

With the expert system correctly configured (see 6.1.4.3) it is enough to add the above perl script in */etc/castor/expert* policy repository for the **SvcClass** to be meet the use-case requirement.

6.4 Tape error handling and retries

Whenever a tape migration or recall encounters an error, the corresponding tape mover (*rtcpd*) returns the error to the client and exits. The *migrator* and *recaller* programs update the corresponding **Segment** row in the stager catalog with the error information. A special component, the *TapeErrorHandler* (see Section 5.5), is provided for enforcing the global retry policies ('global' here means that the retry policy is common to all **SvcClass**es). The are two retry policies: one for recall and one for migration. The *TapeErrorHandler* is normally launched by the *rtcpclientd* daemon when a *migrator* or *recaller* exits with error status. It can also be launched manually from the commandline if necessary (e.g. to try out a new retry policies before starting the *rtcpclientd*).

The names of the recall and migration retry policies to be used are configured in the */etc/castor/castor.conf* file on the host where the *TapeErrorHandler* runs, see Section 6.1.4.4. Example:

```
TAPEERRORHANDLER MIGRATOR_RETRY_POLICY retryForEverPolicy.pl
TAPEERRORHANDLER RECALLER_RETRY_POLICY noRetriesPolicy.pl
```

The default names used if the above configuration is not provided are: *migratorRetryPolicy.pl* and *recallerRetryPolicy.pl*. The same dynamic policy lookup that is used for the migration and recall policies, see Section 6.1.4.3, is used for the global retry policies.

6.4.1 Migration retry policy example

There is no reason why a tape migration retry should fail unless there are problems with the disk server or filesystem holding the file. If this is really the case, the disk server or filesystem should eventually be put in DISABLED status, either by an external (to CASTOR) fault-tolerance actuator detecting the hardware failure or by an administrator.

Tape media or drive error should normally always be retried no matter what the error was. Note that under some circumstances, the *migrator* may decide to put a tape in `RDONLY` status in VMGR, see Section 6.1.4.4 (under the **migrator TAPE_TPERROR** configuration description). In particular this guarantee that another tape will be used in a retry after a media error.

Thus, a retry-for-ever policy may make sense for tape migration. The policy implementation is trivial:

```
#!/usr/bin/perl -w

use strict;
use diagnostics;
use POSIX;
my $doRetry = 1;

END {print "$doRetry\n";}

exit(EXIT_SUCCESS);
```

6.5 SvcClass and FileClass concepts

With the previous CASTOR stager, all important migration attributes were taken from the Cns file-class definitions. For each migration candidate, the stager looked up the file-class attributes to find out how the file should be migrated. Since nothing prevented files from different fileclasses to be mixed within a given migration, the creation and composition of the migration streams could be problematic. The problem was that volatile storage resource attributes like how many drives and which tape pools should be used, were mixed with permanent store file attributes such as the number of copies on tape. Another problem was that while the assocation CASTOR file - fileclass belongs to the namespace, the fileclass attributes (maxdrives, tapepools, nbcopies, ...) themselves have nothing to do with the name-space. The CASTOR namespace maps logical filenames to their residence on permanenet store (tape) while it is the stager catalog that provides the volatile disk-residence mapping.

To cope with the problems mentioned above, the old stager had to maintain and regularly update a cache (for efficiency) of the Cns fileclass attributes in its own catalog. In the design phase of the new stager it was decided to correct the design flaw once and forever, and hence move the definition of the Cns fileclass attributes into the stager catalog.

The new stager introduces the **SvcClass** class in the catalog database schema. The **SvcClass** is the place where resources and their usage are tighed together to define a complete service. The **SvcClass** is independent of the CASTOR file and its associated fileclass. Unless the *MigHunter* is run in legacy mode (see

Section 5.4.1), the migration attributes and tapepool association defined in the Cns fileclass are ignored. In the longer term those attributes should be removed from the Cns fileclass definition.

As was stated above the only attribute remaining in the fileclass definition that really belongs to the permanent store, is the number of copies on tape. This information still needs to be cached in the stager catalog (the **FileClass** class). A set of administrative commands (*enterFileClass* [5], *modifyFileClass* [6], *deleteFileClass* [7] and *printFileClass*) are provided for this purpose. Whether the `nbCopies` attribute belongs to the name space (and hence the Cns fileclass definition) or should also be moved to the stager catalog can be debated. In principle, following the argumentation above, it should probably continue to belong to the Cns fileclass definition. The *enterFileClass* [5] and *modifyFileClass* [6] commands therefore supports an option `--GetFromCns`. If the `nbCopies` definition is moved to the **FileClass** in the stager catalog, it means that the Cns fileclass is empty and thus only the file - fileclass name (or id) is maintained in the CASTOR namespace. In any case, with the reduced number of attributes there is no need for more than two or three fileclasses in total once the new stager has been deployed (e.g. “singleCopy”, “dualCopy”).

Chapter 7

Trouble shooting

($Id: mrTroubleShooting.tex,v 1.9 2007/10/08 10:09:57 obarring Exp $)

7.1 Killing stuck tape requests

Killing stuck tape requests (e.g. bad drive or stuck RFIO connections) is best done by using the *kill* command on the *rtcpd* process on the tape server or the corresponding *recaller/migrator* process.

Use of the *killjid* command is not recommended. The command has no effect on the request itself because the client network port number known to VDQM has been re-negociated between *rtcpd* mover and the *recaller/migrator* client. Instead the *killjid* abort message will be received, logged and ignored by the *rtcpclientd* daemon.

7.2 Stopping the service

The stopping of *rtcpclientd* and its *migrator* and *recaller* child processes should normally be performed using the *castor-rtcopy-clientserver* service script. The script kills the *rtcpclientd* daemon using the SIGINT signal (`kill -2`, which allows the daemon to properly stop its *migrator* and *recaller* child processes, reset the streams and cleanup **BUSY** tapes in VMGR.

7.3 Stopping the MigHunter

If the *MigHunter* is killed while it is attaching migration candidates to the **Stream**s, the unprocessed **TapeCopy** candidates will remain in status `TAPECOPY_WAITINSTREAMS` (2) but without links to any **Stream**. Unless this inconsistent state is corrected, the unprocessed **TapeCopy** candidates will never be migrated. The *MigHunter* will do this cleanup automatically if it was killed with one of the signal `SIGINT`, `SIGTERM` or `SIGABRT`. If killed with some other signal, the orphaned migration candidates must be cleaned up manually (see Section 7.4.4).

7.4 Cleanup after a server or node crash

Under normal circumstances the appropriate cleanup after a failed migration or recall is performed by the corresponding *recaller/migrator* process and/or the *rtcpclientd*. If the machine running the *rtcpclientd* crashes or a process terminates abnormally it may be necessary with manual cleanup. The stager catalogue cleanup, for instance resetting the *Stream* and *TapeCopy* status for incompletely processed requests are

automatically run when the *rtcpclientd* process restarts. It is, however, still necessary to cleanup falsely **BUSY** tapes in VMGR.

7.4.1 Full reset of the migration streams

If one suspect problems with the tape migrations but it is unclear if it is related to a particular stream or due to some internal inconsistency (or bug), it may be useful to do a complete and clean reset of the tape migrations:

1. Stop all *MigHunter* processes
2. Delete the **TapeCopy** - **Stream** links in the catalogue
3. Wait for the **stream**s to be deleted by the *rtcpclientd*
4. Once all **Stream**s have disappeared, update the status of all **TapeCopies** in status `TAPECOPY_WAITINSTREAMS` (2) or `TAPECOPY_SELECTED` (3) to `TAPECOPY_TOBEMIGRATED` (1) (or `TAPECOPY_CREATED` (0)).
5. Restart the **MigHunter** processes

The deletion of the **TapeCopy** - **Stream** links can be done:

```
SQL> DELETE FROM Stream2TapeCopy;
SQL> COMMIT;
```

Once all **Stream**s have disappeared the **TapeCopies** are updated:

```
SQL> UPDATE TapeCopy SET Status=1 WHERE Status in (2,3);
SQL> COMMIT;
```

The deletion of the **Stream**s may take some time since it may require tape requests queued in VDQM to be started. Also the running **Stream**s should be allowed to finish with the **TapeCopies** that has been selected for migration. If some **Stream**s do not disappear, it may be necessary to reset the status to `STREAM_CREATED` in order for the *rtcpclientd* to pick up the **Stream**.

7.4.2 Resetting status of individual migration stream in the catalogue DB

To see the migrations streams and their status, do in *sqlplus*:

```
SQL> SELECT * FROM STREAM;
```

Example:

```
SQL> SELECT ID,TAPE,TAPEPOOL,STATUS FROM STREAM;
        ID       TAPE   TAPEPOOL     STATUS
---------- ---------- ---------- ----------
    293814     243443       1026          3
    293815       1053       1027          3
    293816     184299       1026          3
    293817     244234       1027          1
    293818      60452       1026          3
    293819      12637       1027          3
    293820     277656       1026          3
    293821     242627       1027          3
    293822      45654       1026          3
```

```
    293823     294532       1027          1
    293824      36473       1026          3

        ID       TAPE   TAPEPOOL     STATUS
---------- ---------- ---------- ----------
    293825     266437       1027          3

12 rows selected.
```

The Stream status codes can be found in the file `castor/stager/StreamStatusCodes.hpp` in the CASTOR CVS. The current definitions are:

```
    enum StreamStatusCodes {
      STREAM_PENDING = 0,
      STREAM_WAITDRIVE = 1,
      STREAM_WAITMOUNT = 2,
      STREAM_RUNNING = 3,
      STREAM_WAITSPACE = 4,
      STREAM_CREATED = 5
    }; // end of enum StreamStatusCodes
```

For all streams with `STREAM_RUNNING` (3) status, there should be a corresponding migrator process running. The migrator can be matched to the stream through the value of the `-k <dbKey>` switch, which should correspond to the number found in the TAPE column of the **Stream** table, e.g.

```
/usr/bin/migrator -V P13541 -s 0 -U 46850342-00000010-b87facaf-946d6873
 -k 184299 -g 994BR7 -f 69 -d 200GC -i 115815 -l aul -u 994B5492 -T 1107528216
```

corresponds to the STREAM row with ID=293816 since its TAPE value is 184299.

The two streams ID=293817 and ID=293823 both have STATUS=1, which corresponds to `STREAM_WAITDRIVE`. This should usually correspond to a tape request queued in VDQM, which can be checked by looking up the tape VID in the TAPE table, e.g.

```
SQL> SELECT VID,STATUS FROM TAPE WHERE STREAM=293817;
```

If no corresponding queued VDQM request is found, the stream is most probably a zombie that will never start unless a manual action is taken. This can happen if a previous *rtcpclientd* was stopped abruptly. Another cause can be that the *rtcpclientd* failed to receive the startup message from the *rtcpd* mover. The startup message contains the tape VID, without which the *rtcpclientd* cannot determine the associated stream (and hence cannot execute a cleanup). To clear the situation it is sufficient to reset the stream STATUS to 0 and the stream will automatically be picked up at the next iteration of *rtcpclientd* (usually within 30 seconds). For instance (in *sqlplus*)

```
SQL> UPDATE STREAM
       SET STATUS=0,                -- STREAM_PENDING
           TAPE=0                   -- remove tape link if any
       WHERE ID=293817;
SQL> UPDATE TAPE
       SET STREAM=0                 -- remove stream link if any
     WHERE STREAM=293817;
SQL> COMMIT;
```

WARNING: *while the above operation is safe, it is in general dangerous to manually update/insert/delete rows in the stager catalog database*

7.4.3 Resetting TapeCopy status in the catalogue DB

When a migrator fails, e.g. due to a tape mover error, it usually tries to reset the status of all TapeCopies it has selected for processing. For the TapeCopies that were not affected by the error, the status is reset from

TAPECOPY_SELECTED (3) to TAPECOPY_TOBEMIGRATED (1), which allows for them to be picked up and attached to the **Stream**s in the next MigHunter iteration. The **TapeCopy** for which the tape request failed, is left in *TAPECOPY_SELECTED* status in the catalogue database and the links to the **Stream**s are cut. In addition the failed **TapeCopy** also has an associated **Segment** in status SEGMENT_FAILED (6) and the relevant error information stored in errMsgTxt, errorCode (== errno, serrno or rfio_errno depending on the error type) and severity columns. Failed tape copies are later processed by the *TapeErrorHandler*, see Section 5.5, and should normally not require any manual intervention.

The TapeCopy cleanup must take place directly in the migrator because it is the only process knowing about which TapeCopies it has selected. Thus, if the migrator stops abruptly (killed or crashed), the selected TapeCopies must be reset manually. To find the tape copies, it is necessary to look at the migrator log for all entries for the migrator RQID and check find all records with MSG="rtcopy client daemon callback: get more work" without a corresponding MSG="File staged" for the same disk path. To find the corresponding TapeCopies it is necessary find the CastorFile corresponding to the found disk paths using the castor fileid, e.g.

```
DATE=20071002115719.306522 HOST=c2alicesrv02.cern.ch LVL=System FACILITY=migrator
PID=20625 TID=1 MSG="Get more work callback: add file" REQID=4701ac5f-0000-1000-8
47e-dedbebea0000 NSHOSTNAME=castorns NSFILEID=154943160 TPVID=I06985 TPSERV="tpsr
v154" TPDRIVE="35922018" SUBREQID=4701ac5f-0000-1000-b796-869ce553376e FSEQ=16043
 BLKID="00000000" PATH="lxfsrc4801.cern.ch:/srv/castor/03//60/154943160@castorns.
539488660" STATUS=1
```

where the CastorFile can be found via the fileid (NSFILEID=154943160),

```
SQL> SELECT id FROM CASTORFILE WHERE FILEID=154943160;

        ID
----------
     11381

SQL> UPDATE TAPECOPY
       SET STATUS=1                  -- TAPECOPY_WAITINSTREAMS
       WHERE STATUS=3 AND CASTORFILE=11381;
SQL> COMMIT;
```

WARNING: *while the above operation is safe, it is in general dangerous to manually update/insert/delete rows in the stager catalog database*

Note that the status should be reset to TAPECOPY_TOBEMIGRATED (or TAPECOPY_CREATED) since, the **TapeCopy** - **Stream** link is cut when the **TapeCopy** was selected for migration.

7.4.4 Cleanup orphaned migration candidates

If *MigHunter* was killed abruptly (see Section 7.3) for instance after a node crash, a manual cleanup of orphaned migration candidates is needed. An orphaned migration candidate is a **TapeCopy** in TAPECOPY_WAITINSTREAMS (2) status but without links to any **Stream**.

The query to find orphaned migration candidates is:

```
SQL> SELECT * FROM TapeCopy,Stream2TapeCopy
       WHERE TapeCopy.status = 2              -- TAPECOPY_WAITINSTREAMS
       AND Stream2TapeCopy.child (+)= TapeCopy.id
       AND Stream2TapeCopy.child is null;
```

WARNING: the subquery can be heavy and should be avoided if there are many rows in the **Stream2TapeCopy** table. Check with a SELECT count(*) FROM Stream2TapeCopy first and wait with the cleanup if there are thousands of rows. If possible, the easiest is to let the migration of all non-orphaned candidates finish before doing the cleanup.

The cleanup consists of resetting the status of the **TapeCopy** to TAPECOPY_TOBEMIGRATED (1):

```
SQL> UPDATE TapeCopy SET status=1  -- TAPECOPY_TOBEMIGRATED
     WHERE id IN (SELECT TapeCopy.id FROM TapeCopy,Stream2TapeCopy
                    WHERE TapeCopy.status = 2         -- TAPECOPY_WAITINSTREAMS
                    AND Stream2TapeCopy.child (+)= TapeCopy.id
                    AND Stream2TapeCopy.child is null);
SQL> COMMIT;
```

WARNING: *while the above operation is safe, it is in general dangerous to manually update/insert/delete rows in the stager catalog database*

7.4.5 Resetting Tape status for recall candidates

If the *rtcpclientd* daemon is restarted while it had outstanding requests in the VDQM queue, those requests are lost. The corresponding rows in the **Tape** table will then remain in status TAPE_WAITDRIVE (2) forever. If the daemon was killed without allowing cleanup (e.g. kill -9) there may also be tapes in status TAPE_WAITMOUNT, TAPE_MOUNTED) or TAPE_FAILED. In future versions there will be an automatic cleanup in this case but until then it is necessary to reset the status to TAPE_PENDING preferrably *before* restarting the *rtcpclientd* daemon. The cleanup can be performed as follows:

```
SQL> UPDATE TAPE
       SET STATUS=1                 -- TAPE_PENDING
       WHERE STATUS (2,3,4,6)       -- TAPE_WAITDRIVE,WAITMOUNT,MOUNTED,FAILED
       AND TPMODE=0                 -- ONLY READ REQUESTS
       AND ID IN (SELECT TAPE FROM SEGMENT WHERE STATUS IN (0,7));
SQL> COMMIT;
```

WARNING: *while the above operation is safe, it is in general dangerous to manually update/insert/delete rows in the stager catalog database*

7.5 Unbalanced Streams

When adding a new stream to an already running migration, the new stream will by default only get the new migration candidates that appear after the creation of the stream. This means that the migration candidates that existed before the new stream was created can only be picked up by the old streams. This imbalance can become problematic when the flow of input files ends because the newer streams will run out of migration candidates and end long before the older streams.

To cope with this problem there is an option -C to the *MigHunter* program that will cause it to ''clone'' the list of migration candidates from an existing stream whenever it creates a new one. The reason why this behaviour is not set by default is because the cloning can become heavy in terms of load on the database if there are many migration candidates to clone.

Bibliography

[1] CASTOR man pages, http://cern.ch/castor/DOCUMENTATION/MAN/

[2] *rtcpclientd* man pages, http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/rtcopy/rtcpclien

[3] *MigHunter* man pages, http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/rtcopy/MigHunter

[4] *dlfenterfacility* man-page, http://cern.ch/castor/DOCUMENTATION/MAN/CASTOR/dlf/dlfenterfac

[5] *enterFileClass* man-page, http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/hsmtools/enterF

[6] *modifyFileClass* man-page http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/hsmtools/modify

[7] *deleteFileClass* man-page http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/hsmtools/delete

[8] *enterSvcClass* man-page http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/hsmtools/enterSv

[9] *modifySvcClass* man-page http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/hsmtools/modify

[10] *deleteSvcClass* man-page, http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/hsmtools/delete

[11] *expertd* man-page, http://cern.ch/castor/DOCUMENTATION/MAN/PROTO2/expert/expertd.man.h

Appendix A

Log messages

($Id: mrAppendixLogMsgs.tex,v 1.6 2005/08/03 13:12:54 obarring Exp $)

The list of DLF messages used by *rtcpclientd*, *recaller*, *migrator* *TapeErrorHunter* and *MigHunter* programs is defined in the *h/rtcpcld_messages.h* file in the CASTOR CVS. An up-to-date list of messages can always be obtained using the *dlflisttext* command for the *rtcpcld* facility:

```
dlflisttext -F rtcpcld
```

The following messages are defined so far:

Message number	Message text	Severity level
0	rtcpcld server started	System
1	rtcpcld_InitNW() failed to initialise network ports	Alert
2	rtcp_Listen() failed	Error
3	failed system call	Error
4	rtcpd did not return VDQM VolReqID	Error
5	starting worker request for VDQM VolReqID	System
6	rtcopy client daemon UID/GID configuration error	Error
7	Internal error	Error
8	File position callback	System
9	File copy callback	Debug
10	catalogue lookup error	Error
11	VDQM call failed	Error
12	Failed to start worker	Error
13	recaller started	System
14	migrator started	System
15	External logger info/debug message	Debug
16	External logger error message	Error
17	Request successfully submitted to VDQM	System
18	Nothing left to do for this VID	System
19	Get more work callback: add file	System
20	recaller ended	System
21	migrator ended	System
22	Database service error	Error
23	Retrieved inconsistent tape request	Alert

Message number	Message text	Severity level
24	New file request	System
25	Execute command	System
26	New connection	System
27	Unknown option	Error
28	No more segments to process	System
29	Segment counts	Debug
30	FSEQ exceeds maximum allowed for label type	Error
31	Got tape from VMGR	System
32	Retry limit exceeded	Error
33	Tape unavailable	Error
34	Updated tape info in VMGR	System
35	No tape associated with segment	Error
36	Unexpected tape access mode detected	Error
37	Adding segment to castor file	Debug
38	Wrong checksum	Error
39	Checksum is OK	System
40	Name server segment not found	Error
41	Update checksum in Cns segattrs	System
42	Wrong checksum algorithm detected but not changed	System
43	No tape pool found for stream	Error
44	Failed to update segment in name server	Error
45	Failed to update VMGR	Error
46	There are remaining tape copies to do	System
47	Segment check failed	Error
48	Bad path component	Error
49	Security layer error	Security
50	VMGR error requiring admin intervention	Alert
51	Re-enable TapeCopy for selection	System
52	Reset Stream	System
53	Dual copy on same tape. Detached from stream.	Warning
54	Not all segments staged for file	System
55	File staged	System
56	File removed during migration. Error ignored	Warning
57	Zero size disk file rejected	System
58	Copy failed	Error
59	Unusual position method	Warning
60	Checksum algorithm name too long	Alert
61	Lost VDQM queue position. Resubmitting request	Warning
62	Ping VDQM server	Debug
63	Get more work callback: add placeholder	Debug
64	Re-enable tape+segments for selection	System
65	Tape request failed without failed segments	Alert
66	Tracing statement	Usage
67	Tape Error retry postponed	Warning
68	Migration failed for file	Alert
69	Recall retry rejected by policy	Warning
70	Name server segment no longer valid	Error
71	Service shutdown	Alert
72	Checked for new migration candidates	System
73	Number of streams for SvcClass	System
74	SvcClass has too many streams	System
75	Migration volume below threshold	System
76	Recalled disk file has wrong size	Alert
77	No tape pool for service class with migr candidates	Error

Table A.1: DLF log messages defined for the tape recall and migration components