

File Aggregation in Enstore Proposal

Alexander Moibenko
Alex Kulyavtsev
Vladimir Podstavkov

11/25/2009

Project Goals

- Writing Tape Marks at the end of the each file is expensive, it takes about 3 sec
 - At the rate 100 MB/sec it is the same as to write 300 MB of data
 - Files must be \gg 300 MB, few GB is good
- All files “become relatively small” as the tape capacity grows

Project Goals

- Experiments claim they do aggregate files, but “accidents” still happen per enstore monitoring
 - We can not depend on file aggregation by users
- Need to handle small files already on tapes during tape migration
- Automatically aggregate small files < 500 MB into larger “container” files with container size of few GB

What files to aggregate ?

- Typical write pattern : Raw **data** or results of data processing **are grouped in directory (sub)trees**
.../experiment/detector/Year/Month/Day/files
- Minos :
 - file is ~ one hour of raw data taking
 - “RUN” is one day (~24 files) is good size for aggregation
- Next generation of neutrino experiments will have similar data rates (driven by physics) and storage patterns
- Practice shows that experiments put the mixture of “large” and “small” files into the same directory

What files to aggregate ?

- Aggregation of files shall account for read access pattern. Only the experiment knows what read pattern will be.
- Stakeholders shall provide data access patterns.
- File aggregation policy must be flexible enough to adopt to different pattern without changing code.
- Per discussion with Minos data are stored and retrieved from tape as subdirectory tree.
- Aggregation of files by subdirectory tree is good to start with.

Pnfs Tags and File Containers

- The grouping of files for writing to tape is controlled by **file_family pnfs tag** .
- To control file aggregation the **new pnfs tag will be created** to mark top of the directory tree for aggregation. Files written in subtree underneath will be aggregated into container files before written to the tape.
- Initially **tar** is considered as a container but other choices are possible.

Required Features

- **Disk caching layer** in Enstore
 - temporary storage for incoming files, containers and staged files
 - full control over cache disk access to optimize
 - IO bandwidth to tape
 - concurrent Read and Write operations
 - **access from all nodes in cluster to any file in cache**
- Compatibility with existing pnfs namespace and enstore tools (encp)
- “Standard” clients / protocols to access system (e.g. Grid FTP)

Further Optimizations

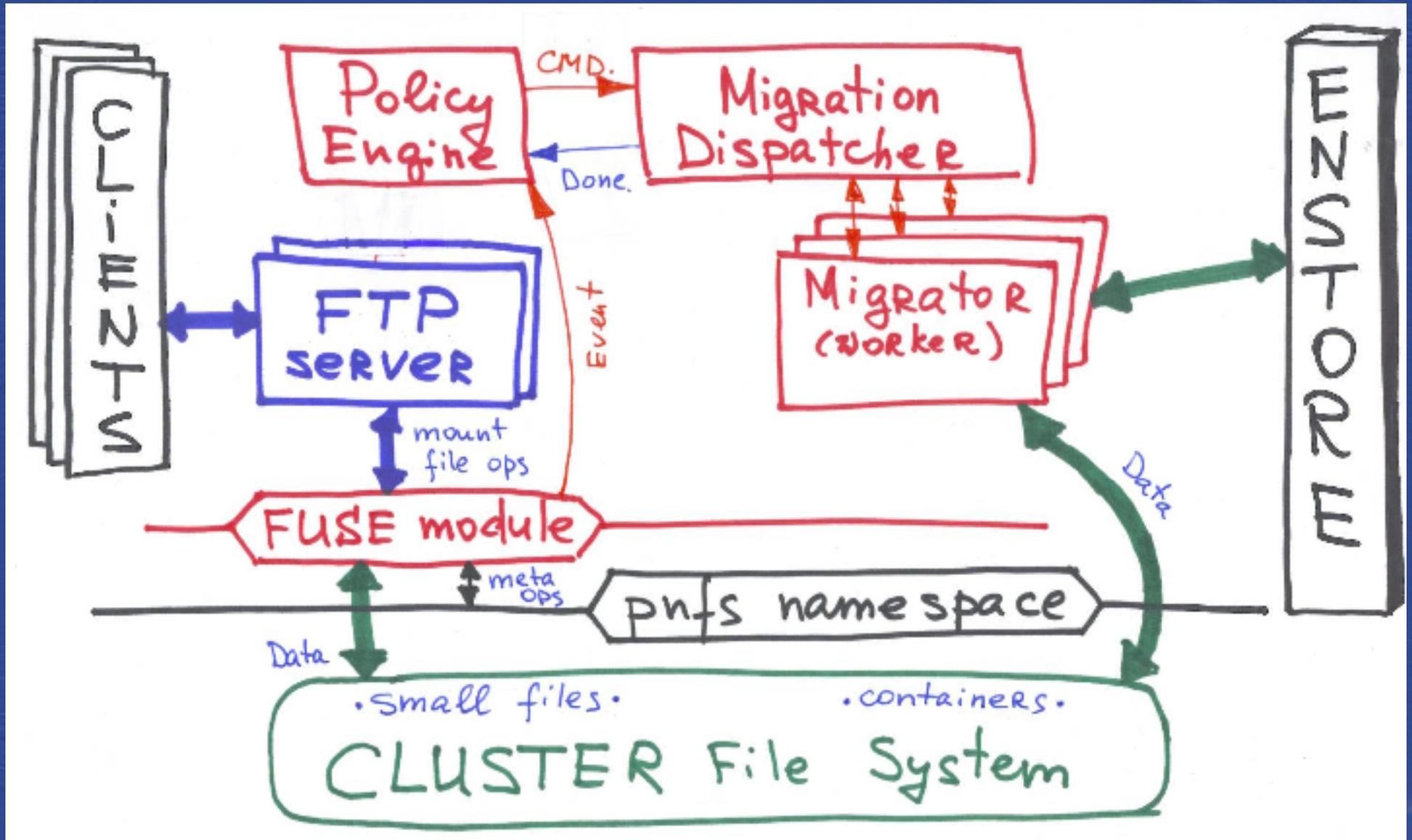
In addition to *File Aggregation into Containers* the system may provide *Write Requests Aggregation* in time.

- Enstore does all the work to optimize tape access (mount / seek / rewind)
- Data caching system shall accumulate requests and trigger tape writes by tunable parameters, for example
 - time window to minimize tape mounts and to maximize tape lifetime (mount in 12 hours, not 1 hr)
 - Number of files (N files > 50)
 - By total size of accumulated requests
- **Global optimization** for the whole system instead of each separate storage unit (like per disk pool).

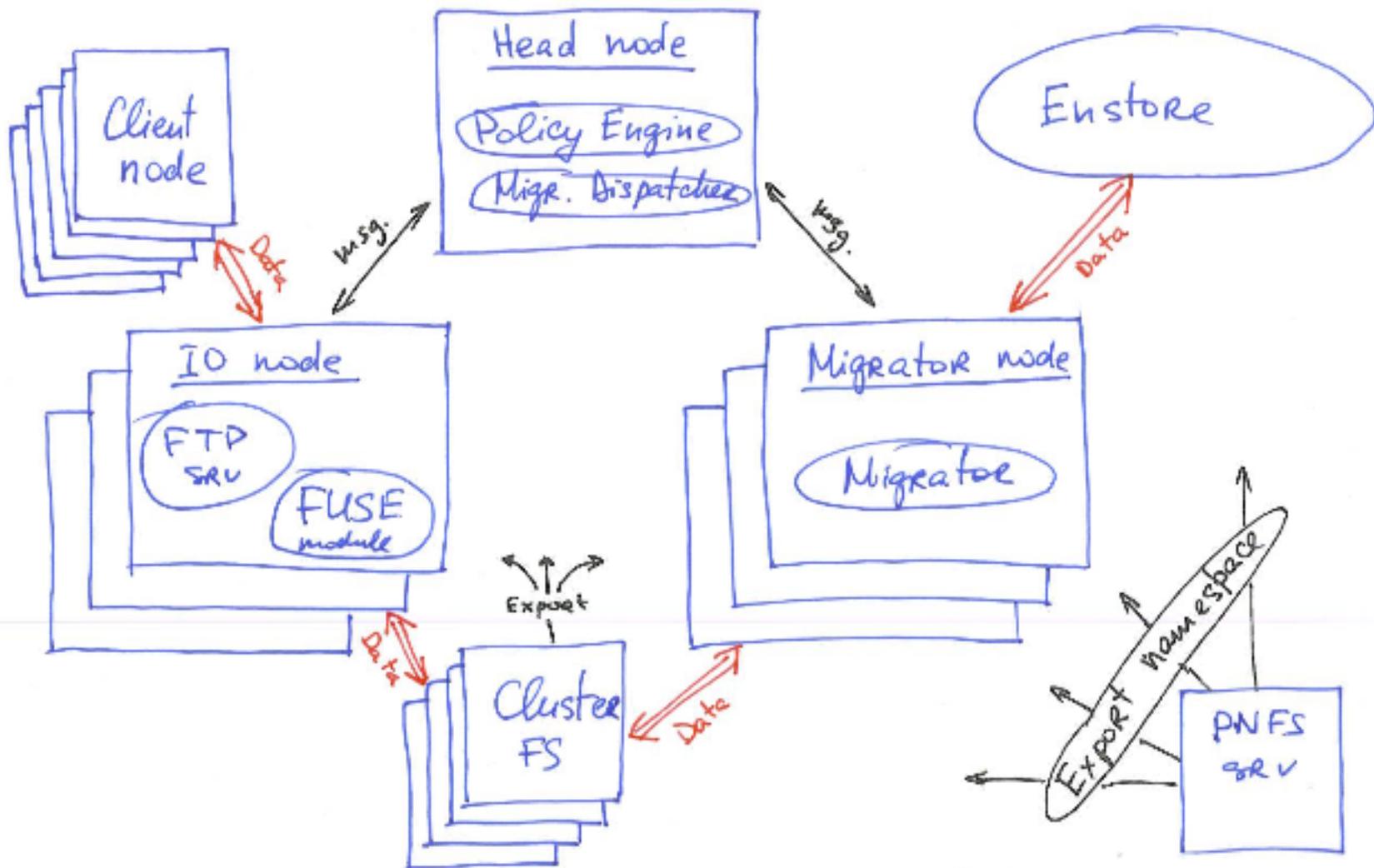
Enstore File Cache

- We introduce local Enstore File Cache
- User access : standard data transfer server
Grid FTP, ssh, etc.
- Files in pnfs namespace map to files in cache
- File System in User Space (FUSE) module provides
 - mapping of namespace operations to operations with pnfs, and
 - IO operations to operations with local file cache
 - Event feed to Policy Engine (open, close, ...)

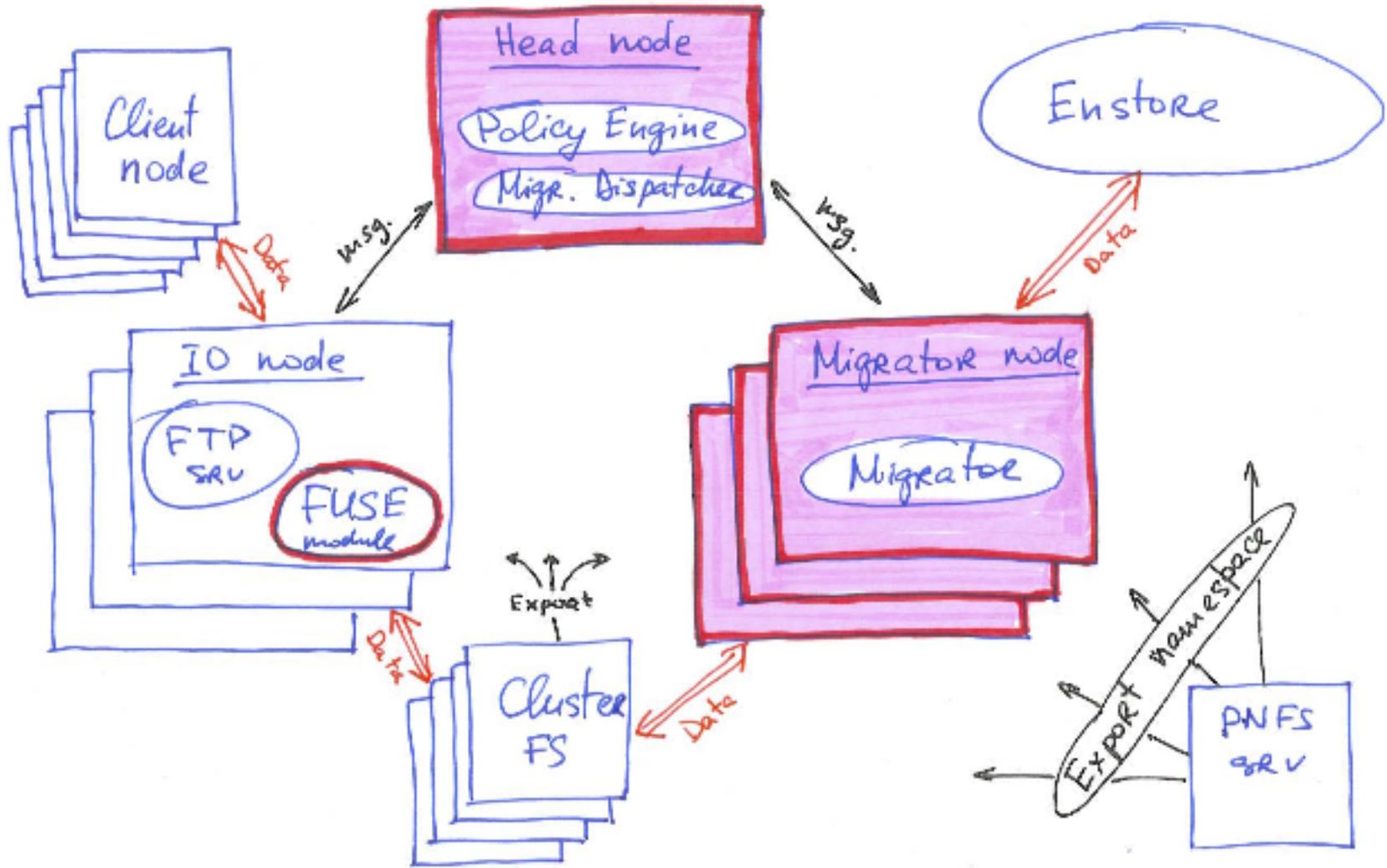
Software Architecture



Hardware Architecture



Hardware Architecture



Local Cache Organization

- Cached files are stored with name “pnfs ID”. Files are stored on cluster FS in a “well known” directory 'hash' subtree for even distribution
 - 012 / 345 / 678 / 9AB / 0123456789AB
- When user accesses the file in FS mounted over FUSE
 - FUSE module uses pnfs for metadata operations (find file path, check protections, etc.)
 - but for the data IO FUSE module opens file in cache residing on cluster storage

EVENTS

- Events are generated by Enstore Cache components and get served by Policy Engine
- Events are logged in persistent storage for recovery. Event consumers shall tolerate event replay in case of a failure.

Types of Events

- FUSE module generated events
 - `close()` on Write – FUSE module generates “event” when data are written in data layer and file is closed.
 - `open()` on Read if file is not present in file cache
 - User process opening file in FUSE fs is blocked on `open()` in FUSE module
- HSM events
 - Migrated
 - Purged
 - Staged

EVENTS

- FUSE Event is a tuple with
 - File ID (pnfs ID)
 - Operation (read, write, ...)
 - Time stamp
 - Pnfs directory tags (“File_family”, “Aggregate”)
- Format for other events is TBD
- We can retrieve more information about file when needed.

Policy Engine (PE)

- ***Policy Engine*** serves events received from different sources
- PE arranges files (file IDs) into groups with the same properties defined by *rules* (e.g. with the same *file_family* and *aggregation* tags, or file path pattern, etc) to be stored in the same container.
- PE checks if the container is “full” and it is time to switch to another container (by time window, number of files in container, total size of accumulated files per container). In such case PE communicates to Migration Dispatcher to start file aggregation and container transfer to the tape backend.

Candidates for PE implementation

- **Event Driven Application Servers (EDAS).**
 - Sun's **Intelligent Event Processing** engine (IEP), part of Glassfish Application Server. It has GUI to set and edit rules. Open Source.
 - Esper's Complex Event Processing engine. High performance Open Source.

Event Driven Application Servers

- Optimized to process and correlate streams of events (tuples) and can access historical data in DB. Used for network monitoring, security (intrusion detection), RFID tracking.
- Each event “record” is passed through set of rules (queries) instead of running query on full set of records in DB.
- Flexible. “Rules” are described in languages representing superset of SQL (CQL, EPL, ...). Thousands of rules can be chained or combined in network.

Event Driven Application Servers

- operate on events in time window
- process, split, combine event streams
- calculate aggregate properties of the (sub)stream, like number of events (files), total size
- store data in DB or call execution of the process at the end of processing chain
- API to provide input data feed and output sink
- Recovery mechanism can be provided outside of EDAS if not provided internally

More candidates for PE implementation

- **RobinHood**, policy engine for Lustre HSM project. Processes changelog events from MDT. Manages free space (based on space used and LRU).
- Lustre change logs are available in lustre v1.8 (current production)
- Written in “C”, may require source code modifications for integration

File Aggregation

- When PE has collected “enough” files it triggers aggregation of files into container.
- File name in container is “hash path / pnfs ID “.
- Ultimate disaster recovery - tape in hand but namespace is completely lost. Tool will be provided as part of each container to restore files with names as stored by user.

Writing Aggregated Files to Tape

- Container file gets copied to tape as a regular file using `encp`
- New record is created in `enstore DB file table` for each file in the container with fields:

`crc, volume, drive, location_cookie,`
`size, uid, gid, crc_seed`

having the same values as the container file

- `Enstore DB file table` is amended by new values
 - “`packaged`” = “`y`” or “`n`”

Writing Aggregated Files to Tape (cont.)

- *Enstore DB file_copies_map* table entries get created for each file

f1_bfid <-----> container bfid

f2_bfid <-----> container bfid

....

fN_bfid <-----> container bfid

- Container gets deleted immediately to save disk space in cache
- encp modifications may be needed

File Read

When file is not present in Enstore Cache :

- the process opening file in FUSE fs is blocked on open() in FUSE module
- FUSE module generates event to Policy Engine
- PE aggregates requests and eventually generates output “event” to retrieve File Container to be processed by Migration Dispatcher
- Migration Dispatcher controls container retrieval, unpacking, file purge operations

Reading Aggregated Files from Tape

- Container file is identified via *Enstore DB* table *file_copies_map* .
- Container is staged from tape.
- Container is unpacked in cluster FS in a separate subdirectory. Each unpacked file is renamed to the destination name “hash / pnfs ID ” so it can be accessed by FUSE.
- All files in the container are marked available for read and waiting processes are released.

Space Management

- Policy Engine is responsible to track file usage (LRU) and account for space usage on cluster FS and select files to be removed.
- LRU is to start with, high / low water marks for space.
- RobinHood Policy engine was developed to manage space on large FS. Implements high / low watermarks, LRU .

Scope of The Project

- Enstore Cache
 - integrate Cluster File System
 - develop FUSE module to merge pnfs with FS
- Policy Engine
 - Develop event feed, sink, rules
- Migration/Purge Dispatcher
- Select and integrate reliable messaging system (AMQP standard)
- Migration Modules integrated with Enstore