Lustre HSM HLD

J-Ch. Lafoucriere, A. Degrémont

22th January 2008

Contents

Architecture

This document follows the architecture documents available on Lustre Arch Wiki at the following URL:

HSM Migration *http://arch.lustre.org/index.php?title=HSM_Migration*

Space Manager *http://arch.lustre.org/index.php?title=Space_Manager*

Part I
Coordinator

1 Definitions

explicit administrative request Request send to the coordinator by a user or an administrator.

implicit/automatic request Request send to the coordinator by an automatic system, implicitly. This will be done mainly due to I/O accesses.

2 New entities

- External storage

  Each *external storage* Lustre will work with will be describe in Lustre in a specific entity. This entity will be composed of a specific unique identifier and a related copy tool and *external objects*.

- External object

  A external object describes data stored on an external storage and accessible with a specific identifier. Using this identifier Lustre will be able to read and remove this data when needed. All those objects are linked to a Lustre object.

3 Functional Specifications

The coordinator manages the migration requests and the external storages.

3.1 External copyout

This function requests that one specified Lustre object will be copied from Lustre to an external storage.

- Input elements:

  Lustre object reference Lustre unique reference to this object independently of its various versions.

  Lustre object version Version number of this object that will be copied. The tool can use this value to optimise the data storage if the external storage supports versioning.

  External storage reference A identifier to the external storage the caller wants the data be copied to.

  Mode Indicates whether this request is an explicit administrative request or not.

3.2 External copyin

This function requests that one specified external object should be copy from its external storage, into Lustre, to its associated Lustre object on a specific range.

- Input elements:

  Lustre object reference Lustre unique reference to this object independently of its various versions.

  Lustre object version Version number of this object that will be copied.

  External object A reference to the Lustre object copy store in an external storage.

  Range Contiguous range of data to be copied in.

  Mode Indicates whether this request is an explicit administrative request or not.

3.3 External remove

The specified external object will be deleted from its external storage.

- Input elements:

  Lustre object reference Lustre unique reference to this object independently of its various versions.

  Lustre object version Lustre object version number corresponding to the external object to be removed.

  External object A reference to the Lustre object copy to be removed.

  Mode Indicates whether this request is an explicit administrative request or not.

3.4 Internal duplicate

The function requests an internal Lustre source object to be copied to another internal Lustre object.

- Input elements:

  Source The source object that will be copied.

  Destination The destination object where the data will be put.

  Flags Generic value in order to alter the function behaviour.

  Mode Indicates whether this request is an explicit administrative request or not.

3.5 Cancel migration

Cancels all undergoing requests related to a Lustre object. The current migration will be aborted as soon as possible. As the data copy is not transactional, all the already done work will not be cancelled. The caller should deal with this. An explicit administrative request could only be cancelled by another administrative request.

- Input elements:

  Lustre object Lustre unique reference to this object independently of its various versions.

  Mode Indicates whether this request is an explicit administrative request or not.

3.6 Migration list

Returns a migration list starting from a specific rank and with a limited number or entries. Each entry will describe a migration with several information like the related Lustre object, the migration status and progress.

To get the full migration list, you need to call this function several times, updating the starting rank between each, using the last rank you have received the previous call. Iteration is finished when less migrations were returned than the required numbers.

- Input elements:

  Start rank The smaller migration ID which will be returned.

  Number of elements The max number of migration state to returned, starting from *start rank*.

- Output elements:

  Lustre object The Lustre object the migration is presently copying in or out.

  External storage A reference to the external storage used by this migration.

  Progress The copy progress.

  Status The last known state of the migration.

3.7 External storage administration

The *coordinator* will need to manage and store various external storage descriptions. For each of them, it will store their identifier, their copy tool and made this information available to every component which will need it, like the agents.

Thanks to a Lustre tool, the administrator will be able to:

3.7.1 Add a new external storage reference

This action defines a new external storage reference which could be later used by an agent.

- Input elements:

  Label User defined unique identifier for this *external storage*.

  Copy tool path Command line to a user-space tool which can interact with the *external storage*.

3.7.2 Destroy an external storage reference

This action will delete the declaration of an external storage identifying it with its defined label. The external objects referencing this external storage should be handled carefully.

- Input element:

  Label User defined unique identifying this *external storage*.

3.7.3 List the defined external storage references

This action will return a tuple for each defined external storage.

- Output elements:

  Label User defined unique identifier for an *external storage*.

  Copy tool path Command line to a user-space tool which can interact with the *external storage*.

3.8 Migration administration

New commands will be added to a user tool to manage migrations. These commands will provoke *explicit administrative requests* to the *coordinator*. Administrators and users could used those commands.

3.8.1 Manually copy-out a file

This action requires explicitly that a Lustre file should be copied to a defined external storage.

- Input elements:

  File path Local path to the file to copy out.

  External storage reference External storage label which will store the file data. This reference could be optional if the coordinator can guess it using policy or configuration.

3.8.2 Manually copy-in a file

This action requires explicitly that a specific version of a file, previously copied out should be bring back into the filesystem, filling a purge file or replacing it.

- Input elements:

  File path Local path to the file to copy in.

  File version Version available in an external storage. If this version is not precised, the last copied out version is used.

3.8.3 Manually purge file copies

This action requires that an identified file version copied out should be removed from the external storage.

- Input elements:

  File path Local path to the file to purge a copy.

  File version Version available in an external storage to purge. If this version is not precised, the last copied out version is removed.

3.8.4 List current migrations

List all the undergoing migrations and their details.

- Output elements:

  Lustre reference A path to the file currently under migration or an Lustre ID is no path is available.

  External source/destination A reference to an external storage or external object.

  Status The error status or the migration progress.

3.8.5 List current copies

This action displays all the available copied out versions of a specified Lustre file.

- Input elements:

  File path Local path to the file the user is interested in.

- Output elements:

  Version Version of the Lustre object.

  Date Date and time the object had when it was copied out.

  Size Size the object had when it was copied out.

  External object reference Reference describing the external object which owns the data.

4 Use Case Scenarios

4.1 Archiving a not-recently-accessed file

1. A Lustre component, like the Space Manager, requests the coordinator to copy out a specific Lustre file to a specific external storage.
2. The coordinator checks its internal records for an existing migration for this file and if needed starts a migration on an agent.
3. The request source can requires the migration current status to the coordinator if needed.
4. The coordinator stores the external object references to be able to restore this file later.
5. As soon as the migration is completed, the coordinator sends an acknowledgement to the request source.

4.2 Restoring a punch file

1. A Lustre component, like the an MDT or OST, sends a request to bring back a punch file to the coordinator.
2. The coordinator looks for the external copies of this Lustre file and starts the copy of the latest one on an agent.
3. When the copy is finished, the coordinator warns the request initiator of this completion.

4.3 Cleaning the external storage after a Lustre file has been deleted.

1. A user removes a Lustre file, all its references on MDTs and OSTs are purged. The reference to external objects are also removed. For each of those references, a request is send to the coordinator.
2. The coordinator receives a removal request for an external object and its corresponding specific Lustre object and version.
3. It starts a removal request on an agent for this external object. This removal is asynchronous.
4. When the removal is done, the request initiator is warned.
5. All the last references to those external objects could be cleaned.

4.4 A write I/O cancels an undergoing migration

1. The Space Manager initiates an *automatic copy-out request* for a Lustre file to the coordinator.
2. The coordinator starts the transfer on an agent.
3. The file content is read and the copy is undergoing, when a write request is received by an OST.
4. As this file is currently under migration, the OST requests to the coordinator to abort the migrations for this file.
5. As this migration was started by an *automatic request*, it could be cancelled by the OST trigger. The coordinator asks its agent to cancel the migration.
6. The migration is aborted, the coordinator acknowledge it to the OST.
7. The OST processes the I/O request.

4.5 An administrator wants details on current migrations

1. An administrator uses a user-space tool to list the undergoing migrations.
2. The tool asks the *coordinator* for a first list a migrations.
3. The *coordinator* returns a migration information list as required.
4. The tool asks the *coordinator* for a second list of migrations to complete the first one it has received.
5. The *coordinator* returns a second migration list. But this list is smaller than required because there is not so many migration presently.
6. The tool receives the second list, notices it has less results than required, and concludes they are the last current migrations.
7. The tool displays the information for all file migrations.

4.6 Declare a new external storage

1. An administrator uses the user-space tool to declare a new storage. It indicates the label it want for it and the path to the associated copy tool.
2. These informations are sent to the MGS which stores them in a specific log.
3. This log is sent to all its reader like the *coordinator*.
4. The *coordinator* updates its storage list.

5 Logic Specifications

5.1 External storage list

The external storage list will be stored in specific journal. This journal will be available on the MGS. Each external storage declaration and configuration will be stored in a journal record. This journal will be used by the *coordinator* to rebuild its internal list at start.

5.2 Agent list

Each *agent*, when started will announce itself to the *coordinator*. The *coordinator* will store in its internal log and into the MGS a record list indicating the *agent* list and their external storages. This log will be read by the *coordinator* to contact the *agent* and by the *agent* to know if they need to register themselves to the *coordinator*.

5.3 Coordinator implementation

The coordinator will be implemented as a classical Lustre RPC server implementing a new API set which will be used by the MDT, OST and the Space Manager. It will store the migration information in a local log saved in a dedicated storage device. This ables the coordinator to have failover nodes, like an OST.

6 State management

6.1 Scalability

The *coordinator* should scale up to 100,000 simultaneous migrations.

6.2 Recovery

The *coordinator* will use a dedicated storage device to save its internal record describing all current migrations in order to being able, when re-reading this journal to resume all migrations. Another node starting a *coordinator* should be able to read this device and replace a failing *coordinator*, with a failover mechanism.

6.3 Disk format changes

Lustre object copies Each Lustre object could have several copies of its data in different external storage. For each copy it will stored, on disk:

- The external storage reference.
- The external object reference on it.

- The Lustre object mtime when the object was copied out.
- The Lustre object size when the object was copied out.

Part II
Agent

Agents are Lustre components which reside on clients. They are responsible for spawning archiving tools when the *coordinator* requests it. Thanks to this tool, it moves data internally or externally of Lustre, manages several data movement in the same time, follows their progress and cancels them.

1 Functional Specifications

1.1 External copyout

A call which requests that one Lustre object specified will be copied from Lustre to an external storage.

- Input elements:

  Migration reference A migration identifier which should be used by the agent to identify this migration.

  Lustre object reference Lustre unique reference to this object independently of its various versions.

  Lustre object version Version number of this object that will be copied. The tool can use this value to optimise the data storage if the HSM supports versioning.

1.2 External copyin

A call which requests that one specified external object should be copied from its external storage into its associated Lustre object.

- Input elements:

  Migration reference A migration identifier which should be used by the agent to identify this migration.

  Lustre object reference Lustre unique reference to this object independently of its various versions.

  Lustre object version Version number of this object that will be copied.

  External object A reference to the Lustre object copy store in an external storage.

  Range Contiguous range of data to be copied in.

1.3 External remove

A call which requests that the specified external object will be deleted from its external storage.

- Input elements:

  Migration reference A migration identifier which should be used by the agent to identify this migration.

  Lustre object reference Lustre unique reference to this object independently of its various versions.

  Lustre object version Version number of this object that have been copied.

  External object A reference to the Lustre object copy to be removed.

1.4 Internal duplicate

A call which requests that the agent copy data from a internal Lustre source object to a internal other Lustre object.

- Input elements:

  Migration reference A migration identifier which should be used by the agent to identify this migration.

  Source The source object that will be copied.

  Destination The destination object where the data will be put.

  Flags Generic value in order to alter the function behaviour.

1.5 Data availability

A call which asks the agent to acknowledge that a specified data range have been totally copied for a particular migration. This call replies to the caller when the data is totally copied or if an error occurred.

- Input elements:

  Migration reference The migration identifier which was specified when the migration was requested.

  Range The contiguous range data the caller is interested in.

- Output element:

  Status Indicates if the data are copied or explains why this is not the case.

1.6 Migration cancel

This call requests to the agent that it cancels the specified migration, stopping all the corresponding processes.

- Input element:

  Migration reference The migration identifier which was specified when the migration was requested.

1.7 External storage association

When the *agent* is started by the administrator command, it specifies the *external storage* identifier this agent deals with. This information is sent to the MGS which will store it. The MGS will maintain in a journal the agent list and the external storage the agents manage.

2 Use Case Scenarios

2.1 An agent starts and registers

1. The administrator mounts a client specifying he wants that this client also starts an agent. It specifies which external storage this agent will communicate with.
2. The agent reads the agent list from the MGS and checks if it is declared in it. If not, it registers itself to the MGS, indicating the external storage label it will manage.
3. The agent also reads the external storage list from the MGS and get the copy tool command associated with its external storage.
4. The agent is ready to handle migrations.

2.2 An agent is requested to copy out a Lustre object

1. An *agent* received a migration request from the *coordinator* to copy out a Lustre object to the external storage this agent is connected to.
2. This agent spawns the copy tool defined for its external storage with the needed parameters.
3. Using the information this process is periodically sending, the agent updates its internal data for this processing.
4. The coordinator asks the agent for the availability of a specific data range.
5. The agent, using the information it knows on the requested migration, replies as soon as it knows the corresponding data range has been copied.

2.3 An agent is requested to copy in an external object in its Lustre object

1. An *agent* received a migration request from the *coordinator* to copy in a external object into a Lustre object. This external object is stored into the external storage this agent is connected to.
2. This agent spawns the copy tool defined for its external storage with the needed parameters.
3. Using the information this process is periodically sending, the agent updates its internal data for this processing.
4. The coordinator asks the agent for the availability of a specific data range.
5. The agent, using the information it has on the requested migration, replies as soon as it knows the corresponding data range has been copied.

2.4 An agent is requested to cancel its current work

1. An *agent* is processing a migration request.
2. It receives a cancellation request for this migration.
3. It requires to the *copy tool* process to stop itself.
4. When a timeout is reached, if the process has not stopped, the agent kills it.
5. It acknowledges the cancellation request to its initiator.

3 Logic Specifications

3.1 Migration cancel

The migration cancellation should stop the copy tool process which has been previously spawned. The agent should be able to stop this process by any means. To do so, it will do this in 3 steps at max:

- First, it signals the process to stop itself. The process can clean its connection and quickly quit.
- Then, the agent waits for the process end during a specific timeout.
- At end, if the timeout is reached, the agent kills the process.

4 State Machine Design

4.1 Scalability

An *agent* should scale up to 1,000 migrations in the same time.

4.2 Recovery

When an *agent* crashes down, all the migration it was dealing with are lost and it is the *coordinator* responsibility to initiates new migration requests to resume or restart the transfer on other agents. The agent do not deal with this itself, it has no recovery mechanism.

Part III
Archiving tool

The *archiving tool* is a userspace process associated with a HSM external storage which moves data between Lustre and its external storage. It is intended to be used by the *agent* to handle external copies.

1 Functional Specifications

1.1 Copyout

A list of tuples will be provided on input. They will be composed of a Lustre ID, a version number and a file path. For each tuple provided, the tool will manage a copy out and return another tuple composed by a reference to the corresponding input object, the external ID which was used, the status of the copy and its progress. Those informations will be returned periodically to the agent, updating the migration progress.

Optionally some hints could be provided to optimize the copy. Those informations are not mandatory and could be ignored by the tool.

- Input elements:

  Object ID Lustre unique ID to this object independently of its various versions.

  Object Version Version number of this object that will be copied. The tool can use this value to optimise the data storage if the HSM supports versioning.

  Special path Path to open the Lustre corresponding object by FID which will avoid triggering cache misses.

- Output elements:

  Lustre ID An identifier corresponding to one of the object provided on input, composed by the Object ID and Object Version.

  Status Returns the success or the error of the process.

  Progress Information on the data copy progress.

  External ID Identifier used by the HSM to identify this object copy.

- Hints:

  IO Size Preferred I/O size for file copy.

1.2 Copyin

A list of tuples will be provided on input. They will be composed of the Lustre object reference, an external ID, a file path and a range. For each tuple provided, the tool will manage a copy in and return another tuple composed by a reference to the corresponding input object, the status of the copy and its progress.

- Input elements:

  Lustre ID An identifier corresponding to one of the object provided on input, composed by the Object ID and Object Version

  Special path Path to open the Lustre corresponding object by FID which will avoid triggering cache misses.

  External ID HSM reference returned by the tool when the file was copied out.

  Range Range of data to be copied in. This could cover the whole file or only a part.

- Output elements:

  Lustre ID An identifier corresponding to one of the object provided on input, composed by the Object ID and Object Version

  Status Returns the success or the error of the process.

  Progress Information on the data copy progress.

1.3 Remove

A list of couples of Lustre object reference and an external ID will be provided on input. The tool will asked the external storage for the removal of the provided IDs. For each ID provided, the tool will returns a couple of ID and status.

- Input elements:

  Lustre ID An identifier corresponding to one of the object provided on input, composed by the Object ID and Object Version

  External ID HSM reference returned by the tool when the file was copied out.

- Output elements:

  Lustre ID An identifier corresponding to one of the object provided on input, composed by the Object ID and Object Version

  Status Returns the success or the error of the process.

1.4 Cancel

The tool should be able to cancel is undergoing work when it will receive some specific event like a signal. If this is not possible, it should not be problematic for Lustre or any external component if the tool process is killed during its work.

2 Use Case Scenarios

2.1 Archiving one Lustre file

One of the main use of this tool is to copy one specific Lustre object like a file to an external storage like a HSM.

1. A Lustre agent will spawn the archiving tool providing it the following informations :
   - An object id
   - An object version
   - A *open-by-fid* path
2. The tool will set up its connection with the external storage if needed.
3. The tool will open the file using the provided path. This action prevents the filesystem from triggering cache misses.
4. The tool will read the Lustre file and copy the data read to the external storage. At regular intervals it will returns informations to the Lustre component which has spawned it, indicating:
   - The Lustre object it is currently copying, indicating the object id and object version.
   - The copy state of this object, the possible errors.
   - The copy progression, like the data offset and data length or a percentage.
5. At the end, the tool will close the Lustre object, clean its connection and returns a last set of information like previously explained more the external storage ID used.

2.2 Restoring one Lustre file

The tool can be used to bring back a external entity, already known by Lustre like having being previously copied out from Lustre.

1. A Lustre component will spawn the archiving tool providing it the following informations:

- An object ID
- An object version
- A *open-by-fid* path
- An external ID

2. The tool will set up its connection with the external storage if needed.
3. The tool will open the file using the provided path. This action prevents the filesystem from triggering cache misses.
4. The tool will copy the data from the external storage and write them in the Lustre file. At regular intervals it will returns informations to the Lustre component which has spawned it, indicating:
   - The Lustre object it is currently copying, indicating the object id and object version.
   - The copy state of this object, the possible errors.
   - The copy progression, like the data offset and data length.
5. When the copy is finished, the tool will close the Lustre object, clean its connection and returns a last set of information indicating the success of this copy and a full progression.

2.3 Archiving several Lustre files

Lustre can ask the tool to migrate a list of files. It can use this information to optimize the migration. Possibly grouping all the provided files in one object in the HSM.

1. A Lustre component will spawn the archiving tool providing it *a list of tuples*. Each of this tuple will include the following informations :
   - An object ID
   - An object version
   - A *open-by-fid* path
2. The tool will set up its connection with the external storage if needed.
3. The tool will open the files using the provided paths. This action prevents the filesystem from triggering cache misses. The files will be opened one by one or all in the same time depending on the tool needs.
4. The tool will read the Lustre file and copy the data read to the external storage. At regular intervals it will returns informations to the Lustre component which has spawned it, indicating:

- The Lustre object it is currently copying, indicating the object id and object version.
- The copy state of this object, the possible errors.
- The copy progression, like the data offset and data length or a percentage.

Information on different files could be mixed. The tool can parallelize copies and send information for each them as soon as they are available. For the last display of information of each files, the external ID used will be returned too.

5. When the copy is finished, the tool will close the Lustre objects, clean its connection and returns a last set of information for each file indicating the success of the copy and a full progression.

2.4 Restoring several Lustre files

Lustre can ask the tool to bring back a list of files. Those files could be spread in various HSM entities or the same one if the tool has grouped them during copy out.

1. A Lustre component will spawn the archiving tool providing it *a list of tuples*. Each of this tuple will include the following informations :
   - An object ID
   - An object version
   - A *open-by-fid* path
   - An external ID
2. The tool will set up its connection with the external storage if needed.
3. The tool will open the files using the provided paths. This action prevents the filesystem from triggering cache misses. The files will be opened one by one or all in the same time depending on the tool needs.
4. The tool will copy the data from the external storage and write them in the Lustre files. At regular intervals it will returns informations to the Lustre component which has spawned it, indicating:
   - The Lustre object it is currently copying, indicating the object id and object version.
   - The copy state of this object, *whether some errors occurred.*
   - The copy progression, like the data offset and data length.

   Information on different files could be mixed. The tool can parallelize copies and send information for each them as soon as they are available.

When the copy is finished, the tool will close the Lustre objects, clean its connection and returns a last set of information for each file indicating the success of the copy and a full progression.

Part IV
Data purging

1 Definitions

punch Remove a data window from a file. The file becomes spare.

purge Removes data from a file without showing it to the user. Those data are available on another storage device. All access to the purge part will be trap and the data bring back before the access is processed.

purged window The range where the data have been removed and tagged as *purged*.

2 Functional Specifications

The MDT and OST will be improved with new functions to manipulate their object data.

A purge removes data from Lustre. They are considered being available in another storage. Even if the data are removed, the object previously owning them is still considered having the same size. From an external point of view, no data have been removed. If data are read in this purged window, zeroes will be returned or a cache-miss could be triggered to bring back the removed data.

Others purges could be done on an already-purged file, but the new range must be contiguous with the actual purged window. It should not have two distinct purged windows.

2.1 Data Purge

This function will punch data on a specific range for a Lustre object.

- Input elements:

  Lustre object Lustre unique reference to the object to be purged, independently of its various versions.

  Range The data window to purge. It could be an offset and a length by example.

  Object index Index identifying a file object to purge. If this index is not precised, the purge is apply to the whole file and will be split to match several file object.

2.2 Purge range information

Return the purged window of the specified object. A 0-length range means the object is not purged.

- Input element:

  Lustre object Lustre unique reference to the object independently of its various versions.

- Output element:

  Range The purged range if it exists.

3 Use Case Scenarios

3.1 Lustre detects access on a purge area

1. A client initiates a read request to an OST.
2. The OST verifies that this request does not overlap the purge area of this object, reading the object purged window.
3. If yes, the OST can trigger some events, like a cache-miss.
4. If no, the request is process normally.

3.2 Space Manager needs to make room

An event was raised indicating that space will be missing soon and the space manager decides to make room, purging candidates files.

1. The space manager requests to the MDT, which manage the candidate file, to purge a file on a defined range.
2. The MDT, reading the stripping information of this object, splits the purge request in several purge requests on various OSTs.
3. Each concerned OST receives a purge request for an object it managed. It will removes the corresponding data and note the range it has just punch has been in fact purged. This will update its purged window.
4. The OSTs acknowledge the purge to the MDT.
5. The MDT updates its local purged area and replies to the request initiator.

3.3 User wants to know whether his file data are available

1. Using a Lustre tool, a user requests informations on the file data concerning an optional migration.
2. The tool requests the data purge range to the MDT.
3. The tool checks whether this range is greater than 0.
4. If Yes, it displays to the user the file is partially or totally purge.
5. If No, the file is fully resident and could be accessed without trigerring cache misses.

4 Logic Specifications

4.1 Data purge

A data purge is composed of two phases:

- The first one consists in punching a data window from the object.
- The second one is to store, for each object, the data window that has been punched. This informs that this is not a simple sparse file but in fact a purged one.

It is important that those two modifications are done atomicly.

4.2 Purge area management

A purge could be apply to a file or a file object. All the purge requests are sent to the MDT, even if they concern only a file object. The MDT will propagate them to the right OSTs. This able the MDT to maintain its local purged window.

The *MDT purged window* will start at the smallest purge offset in the file objects and finish at the highest purge offset. This window could be describe as: all data outsite the window are garantied being resident in Lustre. All the data inside this window could possibly trigger a cache miss because the corresponding OST object has been purged.

As a consequence, the MDT will:

1. Receive all the external purge requests
2. Analyze which OST objects are concerned
3. Propagate the needed purge request. The MDT is the only component which send purge requests to OSTs
4. Update its local purged window as the OST acknowledges the requests the MDT has initiated.

5 State Management

5.1 Disk format changes

Purged data window The purge data window will be stored for each Lustre object, a OST object or a Lustre file, as an extended attribute which will be stored on-disk. It will be modified for each purge and when data are brought back.

Part V
Initiating

1 Functional Specifications

The OST data access will be modified to be able to work in two modes. The first one will handle classical I/O accesses and the second one will handle special accesses uses by user-space migration components.

1.1 Classical data access

In classical mode, for each I/O done on a file, the OST will verify that the accessed data are present, that is to say they have not been purged before. If data are present, the I/O is processed normally, if there not a migration request is sent to the *coordinator*.

1.2 Transparent data access

In transparent mode, the client opens the file specifying explicitly that this open is *special*. All I/O done on a file opened this way are processed directly without raising migration request.

2 Use Case Scenarios

2.1 A purge file is read and it triggers its staging

1. A client issues a read request to an OST on a purge part of a file.
2. The OST notices this request is a classical one, no special flag have been set. It checks the read request does not overlap the purged window.
3. If it overlaps, the OST triggers a cache-miss, requests a copy-in migration for this file to the coordinator, waits for its completion and then processes the request.
4. If not, the OST processes the request normally.

2.2 A copy tool uses transparent access to bring back data

1. The copy tool opened the file using special access, especially made for avoiding triggering cache-misses, to open the file.
2. It starts writing the data into the file.
3. The OST receives write request for this file.

4. It notices the file was opened in transparent access mode and so, does not check the purged window.

5. The write request is processed normally.