



SAGE

Percipient StorAGe for Exascale Data Centric Computing

FETHPC1 - 671500

WP3 Services and tools

D3.9 HSM for SAGE: Final Validation Report

SAGE_WP3_HSM_for_SAGE_Final_report_v1.1

Scheduled Delivery: 01.06.2018

Actual Delivery: dd.mm.yyyy

Version 1.1

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	



Responsible Partner: CEA

Revision history:

Date	Editor	Status	Version	Changes
04.04.2018	Thomas Leibovici	Draft	0	Document created
28.04.2018	Thomas Leibovici	Draft	0.1	Initial draft
09.05.2018	Thomas Leibovici	Draft	0.2	Ready for internal review
15.05.2018	Nicolas Vandenberg	Draft	0.3	Internal review
25.05.2018	Thomas Leibovici	Ready	1.0	Addressed review comments
28.05.2018	Thomas Leibovici	Final	1.1	Integrating additional comments

Authors

Thomas Leibovici (CEA), Philippe Deniel (CEA), J.-C. Lafoucrière (CEA)

Internal Reviewers

Nicolas Vandenberg (JUELICH)

Copyright

This report is © by CEA and other members of the SAGE Consortium 2015-2018. Its duplication is allowed only in the integral form for anyone's personal use and for the purposes of research or education.

Acknowledgements

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671500

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

Glossary of Acronyms

Acronym	Definition
AI	Artificial Intelligence
D	Deliverable
DRS	Document Review Sheet
DPA	Data Placement Advisor
EC	European Commission
FDMI	File Data Manipulation Interface (Mero component)
FOL	File Operation Log (Mero component)
HPC	High Performance Computing
HSM	Hierarchical Storage Management
I/O	Inputs and Outputs of computation (data)
IT	Information Technology
LSTM	Long Short-Term Memory
NN	Neural Network
NVRAM	Non-Volatile Random Access Memory
NVMe	Non-Volatile Memory express
RNN	Recursive Neural Network
SAGE	Percipient StorAGE for Exascale Data Centric Computing
SPCM	Semi-Persistent Cache Manager (SAGE WP4 runtime)
SSD	Solid State Disk
VMM	Virtual Memory Manager (SAGE WP4 runtime)
WP	Work Package

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

Table of Contents

1. Executive Summary.....	5
2. Introduction	6
3. Delivered interfaces and tools	7
4. Deployment and integration.....	27
5. Status, further work, and possible future directions	34
6. Conclusion	35

List of Figures

Figure 1: c0hsm_create prototype	8
Figure 2: demo of c0hsm_create	9
Figure 3: object layout after a write operation	9
Figure 4: c0hsm_copy function and its configurable behaviours	11
Figure 5: c0hsm_copy demo (MOVE behaviour).....	12
Figure 6: c0hsm_copy demo (MOVE+WRITE_TO_DEST behaviours).....	12
Figure 7: moving parts of objects.	13
Figure 8: c0hsm_copy demo (KEEP_PREV behaviour).	14
Figure 9: c0hsm_release function and its configurable behaviours.	15
Figure 10: c0hsm_release example.	16
Figure 11: c0hsm_release function (KEEP_LATEST behaviour).	16
Figure 12: c0hsm_set_write_tier prototype.	17
Figure 13: c0hsm_set_write_tier demo.	18
Figure 14: c0hsm_archive prototype.....	19
Figure 15: c0hsm_archive demo.....	20
Figure 16: c0hsm_stage prototype.....	21
Figure 17: c0hsm_stage demo.	21
Figure 18: c0hsm_multi_release prototype.	22
Figure 19: c0hsm_multi_release demo.	23
Figure 20: c0hsm command line syntax.....	24
Figure 21 - Definition of function "dpa_init"	25
Figure 22 - Definition of function "dpa_match".....	25
Figure 23 - Examples of DPA configuration.....	26
Figure 24 - Demonstration of tier selection using HSM DPA	26

List of Tables

Table 1 Partner Contributions for Deliverable	5
Table 2 - HSM storage tiers in the SAGE prototype	27

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

1. Executive Summary

Hierarchical Storage Management (HSM) makes it possible to build cost-effective storage systems by combining the speed of fast devices, e.g. SSDs and flash memory, and the large capacity that can be afforded by using cheaper storage, e.g. hard disk drives.

SAGE project task T3.1 implemented an HSM solution that can manage storage hierarchies of arbitrary depth in MERO, an object store designed for Exascale.

The prior deliverable D3.5 presented the detailed design of the HSM components to be implemented as part of the SAGE project and the progress of said components' development.

This final report describes the interfaces and tools to be provided to applications in detail, as well as their implementation on the SAGE prototype hosted at Jülich, including a demonstration of their operation in various use cases.

Partner	Contribution in Person Months
CEA	5

Table 1 Partner Contributions to the Deliverable

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

2. Introduction

The goal of Task T3.1 is to implement Hierarchical Storage Management for SAGE. This involves developing mechanisms for dynamic management of data location, and for movement of data between multiple storage tiers in a manner transparent to applications.

To achieve these goals, research and development in multiple directions has been undertaken as part of Task T3.1:

- Design and development of mechanisms and interfaces for management of data location across the storage tiers of a deep storage hierarchy made of two tiers or more.
- Design and development of mechanisms and interfaces for movement of data between tiers, which ought to be transparent to applications.
- Enabling smart data placement in order to take advantage of the deep storage hierarchies implemented by SAGE.
- Considering more scalable ways to manage data life cycle in large Exascale systems. This includes collecting and filtering information from the system using a Map-Reduce like mechanism. This also consists in allowing a policy engine to ingest and process this information efficiently so it can then make automated movement decisions.
- Research into the topics of AI and machine learning in order to create optimizing solutions for storage movement by predicting future data access patterns based on past application runs.

All these aspects have already been mentioned in the previous deliverable about HSM D3.5.

This follow-up report mainly focuses on the demonstration of HSM features: Providing interfaces to applications, demonstration of HSM commands in operation, implementation on the prototype, and tests of application access patterns derived from SAGE uses cases.

Section 3 of this document describes the implemented HSM API and tools that are made available to applications and users. Each of the implemented functions is illustrated by a short demonstration of its operation.

Section 4 focuses on the implementation of HSM on the SAGE prototype, as well as its integration to SAGE Runtime Environments developed in WP4.

Section 5 summarizes the current status of HSM implementation and the remaining work for the last months of the project. It also evokes possible future research directions for hierarchical storage management.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

3. Delivered interfaces and tools

Our HSM implementation provides the following interfaces and tools, allowing applications and users to manage their data location:

- A low level C API. This API allows for management of data location in a fine-grained fashion, i.e. moving specific pieces of data between specific storage tiers. It requires the caller to precisely keep track of data locations, since the caller has to be able to specify both source and target tier of data movements. In particular, this API can be called by a policy engine to trigger data movements between tiers.
- A high level C API. This interface is more designed for known use-cases. It is expected to be the preferred interface for most applications. The application merely specifies the storage tier on which it wishes the data to be located for access purposes; the HSM does what is needed to serve this request, independently of where the data is initially located.
- A command line tool, named *c0hsm*¹. This tool wraps around the calls of the above-mentioned two APIs. It allows users to query data location information for their objects and provides a command-line interface, making it possible to run a batch of HSM commands efficiently.
- A data placement advisor, named HSM DPA. It provides a light weight mechanism for determining the most appropriate tier for data given a set of application hints regarding data access patterns and I/O requirements.

This section describes these interfaces in details, and gives examples of their usage for various use-cases.

3.1. Low-level HSM API

The low-level HSM C API implements fine-grained data movements. Calls are kept generic with behaviour controlled by flags, allowing to use them in multiple use-cases.

In this section we describe the API calls and what they achieve. We demonstrate each call's effect by running its equivalent command in a *c0hsm* shell.

¹ The “c0” prefix is a common naming for tools based on Clovis API, e.g. *c0cp*, *c0cat*, ...

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

3.1.1. c0hsm_create

Description

The **c0hsm_create** function creates an object in a tier of the user's choice.

- Required input includes a 128-bits value (id) which will serve as the unique identifier for the created object. It is the same as its identifier in Mero (called *fid*).
- Furthermore, required input includes the tier where object's data is to be located initially. A tier is uniquely identified by an 8-bit value, called its index. Note that an object's tier information is not static; it can later be changed by other API calls like `c0hsm_set_write_tier()`.
- An `m0_clovis_obj` structure, which serves as a handle for the created object, is returned in the `obj` argument.
- Optionally, the created object can be left "open" so the application can immediately perform I/O operations on the object using standard Clovis calls avoiding unnecessary close/open.

```
/**
 * Create an object to be managed by HSM.
 * @param id Identifier of the object to be created.
 * @param obj Pointer to the created object
 *            (the structure is to be allocated by the caller).
 * @param tier_idx Index of the target tier (0 is the top tier).
 * @param keep_open Keep the object entity opened.
 * @return 0 on success, a negative error code on error.
 */
int c0hsm_create(struct m0_uint128 id, struct m0_clovis_obj *obj,
                uint8_t tier_idx, bool keep_open);
```

Figure 1: c0hsm_create prototype

Note: HSM reserves a range of object identifiers for internal use. Those identifiers cannot be used by users or applications to refer to objects via HSM. The reserved range is the set of 128-bit values with bit 95 set to 1.

Demo

In the example below, we create an object with id "0x1000000" in tier 2. Then we display the composite layout² structure for this object (note that for demo purposes, we are using `c0hsm` commands instead of their C API analogues).

² This structure was described in D2.1 and D3.5.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```
c0hsm> create 0x1000000 2
Composite object successfully created with id=0:0x1000000

c0hsm> show 0x1000000
- gen 0, tier 2, extents: (writable)
```

Figure 2: demo of c0hsm_create

From the output of the show command, we see that, initially, the current object generation (i.e. data version) is 0, and that it is located on tier 2. No data is present (i.e. no data has been written yet to the object). Furthermore, this only layer is writable, i.e. any write operation from applications will be directed to this layer.

To illustrate what occurs when we write to this object, we follow up by writing the contents of a file `ode_to_joy.txt` to this object, and see how the composite structure looks afterwards:

```
c0hsm> write_file 0x1000000 ./ode_to_joy.txt
36864 bytes successfully written at offset 0 (object id=0:0x1000000)

c0hsm> show 0x1000000
- gen 0, tier 2, extents: [0->0x8fff] (writable)
```

Figure 3: object layout after a write operation

The object layout still consists of a single layer in tier 2, but this layer now contains a range of data (called *extent*) from offset 0 to offset 0x8FFFF.

3.1.2. c0hsm_copy

Description

As its name suggests, the **c0hsm_copy** function is tasked with copying data between storage tiers. Due to its configurable behaviour (specified by flags), the function can be used for various operations and use-cases:

- It can archive data from a fast tier on a high-capacity tier, and additionally keep a copy of the data on the source tier, so the data can still be accessed quickly by the application. When performing such an archiving operation, the function can be configured to overwrite previous versions on the target tier or to keep them, thus implementing data versioning.
- It can move data from a fast tier to a high-capacity tier, removing this data from the source tier. In this case, the data is accessible only in the target tier after the operation, and disk space is released from the source. Again, when performing such a movement, previous versions of data can be preserved or overwritten (configurable behaviour).
- It can move or copy data the opposite way, from a high-capacity tier to a faster tier:

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

- Copying data keeps the data on the source (archive) tier, and stages a copy in a fast tier, which can be accessed fastly by the application. This use case could be described as “caching”. This is expected to be a good choice when many read-only I/O operations are to be on an object.
- Moving data drops the data from the source tier, which is a good choice when the object is likely to be widely modified or re-written and the user doesn't want to keep previous versions. In this case, the previous data doesn't need to be preserved and the HSM can immediately release the data from the archive tier as it is going to be made obsolete by the following modifications.

As input, `c0hsm_copy` takes the 128-bit identifier of the object to be managed (`obj_id`), the indices of the source and target tiers (`src_tier_idx` and `tgt_tier_idx`), and the range of data within the object to be moved (specified by `offset` and `length`). The last argument `flags` is a set of configurable behaviours (OR'ed values of `hsm_cp_flags`):

- `HSM_MOVE` indicates that the source data is to be dropped from the source tier after copying.
- `HSM_KEEP_OLD_VERS` preserves previous versions of data in the target tier, thus enabling the data versioning feature.
- `HSM_WRITE_TO_DEST` changes the write tier for the object. Setting it causes further write calls to be directed to the target tier of the copy (note that unsetting `HSM_WRITE_TO_DEST` does not set the write tier to the source tier, but leaves it as specified by previous API calls).

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```

/**
 * Copy options (configurable behaviours)
 */
enum hsm_cp_flags {
    HSM_MOVE = (1 << 0), /**< Remove source extent after copy
                             (default: leave it on the source
                             tier) */
    HSM_KEEP_OLD_VERS = (1 << 1), /**< Preserve previous versions of copied
                                      extent on the target tier
                                      (default: only keep the latest
                                      version). */
    HSM_WRITE_TO_DEST = (1 << 2), /**< Indicate the target tier becomes
                                      the preferred tier for next write
                                      operations.
                                      For example, when data needs fast
                                      write access after staging. Or to
                                      direct next writes to a slower tier
                                      after archiving.
                                      Default is to keep writing in the
                                      current tier. */
};

/**
 * Copy a region of an object from one tier to another.
 * This is transparent to applications if they use the I/O
 * calls defined above (c0hsm_create/pread/pwrite).
 * @param obj_id      Id of the object to be copied.
 * @param src_tier_idx Source tier index (0 is the top tier).
 * @param tgt_tier_idx Target tier index (0 is the top tier).
 * @param offset      Start offset of the region to be copied.
 * @param length      Size of the region to be copied.
 * @param flags        Set of OR'ed hsm_cp_flags.
 * @return 0 on success, a negative error code on error.
 */
int c0hsm_copy(struct m0_uint128 obj_id, uint8_t src_tier_idx,
               uint8_t tgt_tier_idx, off_t offset, size_t length,
               enum hsm_cp_flags flags);

```

Figure 4: `c0hsm_copy` function and the flags for configuring its behaviour.

Demo

As just explained, this API function offers multiple possible behaviours and addresses various use cases. We will demonstrate all flags, but not all combinations of them.

In the first example, we move all object data from tier 1 to tier 2. The `mv` argument to the `copy` command-line call is equivalent to the `HSM_MOVE` flag for `c0hsm_copy`.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```

c0hsm> copy 0x1000000 0x0 0xFFFFFFFF 1 2 mv
Archiving extent [0-0x8fff] (gen 0) from tier 1 to tier 2
36864 bytes successfully copied from subobj <0xffffffff01:0x1000000> to
<0xffffffff02:0x1000000> at offset 0
Extent [0-0x8fff] (gen 0) successfully released from tier 1

c0hsm> show 0x1000000
- gen 1, tier 1, extents: (writable)
- gen 0, tier 2, extents: [0->0x8fff]

```

Figure 5: c0hsm_copy demo (MOVE flag).

After the copy operation, the object consists of 2 layers:

- The bottom layer is located in the target tier, i.e. tier 2, and contains the generation 0 data.
- The top layer, located in the source tier, i.e. tier 1, currently contains no data. It is the target layer for follow-up write operations. Newly written data will be identified as generation 1.

Then, let's use the HSM_WRITE_TO_DEST flag. Its equivalent in the c0hsm shell is the w2dest option. In the example below we move the previous object from tier 2 to tier 3:

```

c0hsm> copy 0x1000000 0x0 0xFFFFFFFF 2 3 mv,w2dest
Archiving extent [0-0x8fff] (gen 0) from tier 2 to tier 3
36864 bytes successfully copied from subobj <0xfffffe02:0x1000000> to
<0xfffffe03:0x1000000> at offset 0
Extent [0-0x8fff] (gen 0) successfully released from tier 2

c0hsm> show 0x1000000
- gen 2, tier 3, extents: (writable)
- gen 0, tier 3, extents: [0->0x8fff]

```

Figure 6: c0hsm_copy demo (MOVE+WRITE_TO_DEST flags).

After this move, the writable layer is no longer located on tier 1, but on tier 3. This means that any further write operations will be directed to this tier. This is due to the "WRITE_TO_DEST" flag we set.

So far, examples demonstrated copies of whole objects. To complete this demo, we now demonstrate the ability of HSM to manage parts of objects: parts can be located on different tiers independently of each other. To show this, we move an extent of data in the middle of the object (specifically, bytes 0x2000 to 0x4FFF):

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```

c0hsm> copy 0x1000000 0x2000 0x3000 3 2 mv
Staging extent [0x2000-0x4fff] (gen 0) from tier 3 to tier 2
12288 bytes successfully copied from subobj <0xfffffe03:0x1000000> to
<0xfffffe02:0x1000000> at offset 0x2000
Extent [0x2000-0x4fff] (gen 0) successfully released from tier 3

c0hsm> show 0x1000000
- gen 2, tier 3, extents: (writable)
- gen 0, tier 2, extents: [0x2000->0x4fff]
- gen 0, tier 3, extents: [0->0x1fff] [0x5000->0x8fff]

```

Figure 7: moving parts of objects.

We see the data of generation 0 is now split onto 2 different tiers: extents [0 to 0x1fff] and [0x5000->0x8fff] are located on tier 3, and extent [0x2000->0x4fff] has been moved to tier 2.

For the rest of the demo, we work with a new object with a simpler configuration where we have a single copy of generation 0 located on tier 2, and a writable layer on tier 1 (see the first “show” command output in Fig.8).

Then we write new data to the object, and copy this new data to tier 2 using the HSM_KEEP_OLD_VERS flag to preserve previous versions (keep_prev option in c0hsm shell):

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```

# initial object state
c0hsm> show 0x1000000
- gen 1, tier 1, extents: (writable)
- gen 0, tier 2, extents: [0->0x8fff]

# writing a new version of data
c0hsm> write_file 0x1000000 ./ode_to_joy-v2.txt
36864 bytes successfully written at offset 0 (object id=0:0x1000000)

# object state after write
c0hsm> show 0x1000000
- gen 1, tier 1, extents: [0->0x8fff] (writable)
- gen 0, tier 2, extents: [0->0x8fff]

# copy this new data to tier 2, and keep previous versions
c0hsm> copy 0x1000000 0x0 0xFFFFFFFF 1 2 keep_prev
Archiving extent [0-0x8fff] (gen 1) from tier 1 to tier 2
36864 bytes successfully copied from subobj <0xfffffe01:0x1000000> to
<0xfffffe02:0x1000000> at offset 0

# final object state
c0hsm> show 0x1000000
- gen 2, tier 1, extents: (writable)
- gen 1, tier 1, extents: [0->0x8fff]
- gen 1, tier 2, extents: [0->0x8fff]
- gen 0, tier 2, extents: [0->0x8fff]

```

Figure 8: c0hsm_copy demo (KEEP_PREV flag).

After the object has been re-written, we see that a data extent [0-0x8fff] appears in the layer of tier 1 (generation 1). After the copy operation, we see 4 layers:

- The top layer is located in tier 1 and is empty. It is to receive follow-up write operations (data of generation 2).
- The second layer located in tier 1 contains data of generation 1, i.e. data of the file (ode_to_joy-v2.txt) we read from in this example. This data is left on tier 1 because we didn't set the *MOVE* flag for the copy operation.
- The third layer is the exact same version of the data (generation 1), but located in tier 2. This is the target of the performed copy operation.
- The last layer contains the data of the pre-existing version (generation 0) written prior to the events of Fig.8. This data has been preserved because we set the *HSM_KEEP_OLD_VERS* flag.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

3.1.3. c0hsm_release

Description

The **c0hsm_release** call releases space in a given tier by deleting a range of data (extent). To prevent data loss, it is not permitted to release data which has not been priorly copied to another tier.

As input, `c0hsm_release()` takes the 128-bit id of the object in question (`obj_id`), the index of the tier on which data is to be released (`tier_idx`) and the range of data to be released (`offset` and `length`).

By default, `c0hsm_release` tries to clean all generations of data in the specified tier. It can also be instructed to keep the last generation of data by setting the `HSM_KEEP_LATEST` flag.

```
/**
 * Release options (configurable behaviours)
 */
/** release options */
enum hsm_rls_flags {
    HSM_KEEP_LATEST = (1 << 0), /**< Release all data versions in the given
                                   tier except the freshest version.
                                   This makes it possible to free disk
                                   space from old data versions in
                                   the use-case of versioning (e.g.
                                   copy made with 'HSM_KEEP_OLD_VERS'
                                   flag). */
};

/**
 * Release a region of an object from the given tier.
 * @param obj_id      Id of the object to be released.
 * @param tier_idx      Tier to drop the data from.
 * @param offset       Start offset of the region to be released.
 * @param length       Size of the region to be released.
 * @param flags        Set of OR'ed hsm_rls_flags.
 * @return 0 on success, a negative error code on error.
 */
int c0hsm_release(struct m0_uint128 obj_id, uint8_t tier_idx,
                  off_t offset, size_t length, enum hsm_rls_flags flags);
```

Figure 9: `c0hsm_release` function and the flags for configuring its behaviour.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

Demo

We hereby illustrate the operation of `c0hsm_release()` under both possible behaviours: release all data versions from a tier, or keep the latest version.

The first example illustrates a caching use-case. A copy of object data was staged from tier 3 to a fast tier (tier 0). We release it from tier 0 as an application should do when it no longer needs the fast-tier copy.

```
# Initial objet state: a copy is cached in tier 0
c0hsm> show 0x1000001
- gen 1, tier 0, extents: (writable)
- gen 0, tier 0, extents: [0->0x8fff] <<<
- gen 0, tier 3, extents: [0->0x8fff]

# Now release the data from tier 0
c0hsm> release 0x1000001 0x0 0xFFFF 0
Extent [0-0x8fff] (gen 0) successfully released from tier 0

# final state
c0hsm> show 0x1000001
- gen 1, tier 0, extents: (writable)
- gen 0, tier 3, extents: [0->0x8fff]
```

Figure 10: `c0hsm_release` example.

In the second example, we start with multiple data generations. Then we release all generations of a given data range, except the latest.

```
# Initial objet state: multiple generations of data are located in tier 3
c0hsm> show 0x1000002
- gen 4, tier 0, extents: (writable)
- gen 3, tier 3, extents: [0x1000->0x2fff]
- gen 2, tier 3, extents: [0->0x8fff]
- gen 1, tier 3, extents: [0->0x8fff]
- gen 0, tier 3, extents: [0->0x8fff]

# Now release the data from tier 3
c0hsm> release 0x1000002 0x0 0xFFFF 3 keep_latest
Found no extent matching [0x1000-0x2fff] with generation >= 3: can't
release it from tier 3
Found no extent matching [0-0x8fff] with generation >= 2: can't release it
from tier 3
Extent [0-0x8fff] (gen 1) successfully released from tier 3
Extent [0-0x8fff] (gen 0) successfully released from tier 3

# Final state
c0hsm> show 0x1000002
- gen 4, tier 0, extents: (writable)
- gen 3, tier 3, extents: [0x1000->0x2fff]
- gen 2, tier 3, extents: [0->0x8fff]
```

Figure 11: `c0hsm_release` function (KEEP_LATEST flag).

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

As we see from the output of the “release” command, the following operations have been performed:

- Extents [0-0x8fff] of generation 0 and generation 1 have been deleted, because there is a more recent version of these extents: extent [0-0x8fff] of generation 2.
- Extent [0-0x8fff] of generation 2 has been kept because it was not fully covered by a more recent extent ([0x1000-0x2fff] doesn't fully cover it).
- Extent [0x1000-0x2fff] of generation 3 has been kept because it contains the most recent version for this range of data.

3.1.4. c0hsm_set_write_tier

Description

c0hsm_set_write_tier dynamically changes the tier where newly written data is directed to.

As input, it takes the 128-bit identifier of an object (**obj_id**) and a tier index (**tier_idx**).

```
/**
 * Selects the target tier for next write operations.
 * This does not move existing data.
 * @param id Identifier of the object.
 * @param tier_idx Index of the target tier.
 * @return 0 on success, a negative error code on error.
 */
int c0hsm_set_write_tier(struct m0_uint128 id, uint8_t tier_idx);
```

Figure 12: *c0hsm_set_write_tier* prototype.

Demo

In this example we demonstrate the changing of an object's write tier between 2 write operations:

- 1) The application writes an extent. It is written to the current writable tier.
- 2) Call `set_write_tier` to change the write tier.
- 3) The application writes another extent. This extent is stored in the newly designated writable tier.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```
# Initial object state: write tier is tier 0
c0hsm> show 0x1000005
- gen 0, tier 0, extents: (writable)

# Application writes data
c0hsm> write 0x1000005 0x0 0x1000 1
4096 bytes successfully written at offset 0 (object id=0:0x1000005)
c0hsm> show 0x1000005
- gen 0, tier 0, extents: [0->0xffff] (writable)

# change write tier
c0hsm> set_write_tier 0x1000005 1

# application writes new data
c0hsm> write 0x1000005 0x1000 0x1000 1
4096 bytes successfully written at offset 0x1000 (object id=0:0x1000005)
c0hsm> show 0x1000005
- gen 1, tier 1, extents: [0x1000->0x1fff] (writable)
- gen 0, tier 0, extents: [0->0xffff]
```

Figure 13: c0hsm_set_write_tier demo.

The final state shows the first extent has been written to tier 0, while the second extent is located on tier 1. This set of extents constitute the whole object: if an application reads data from offset 0x0 to 0x1FFFF, it will transparently get the union of the two extents even if they are actually located in different tiers.

3.2. Higher-level HSM API

The goal of this higher-level HSM API is to make HSM management simpler to applications, and to provide an API focused on known use-cases. These calls do not require applications to care about the current location of data; applications merely have to specify where they would like the data to be located after an API call. HSM does what is needed to move the data there, independently of where it is previously located.

In this section we describe the calls of this API and explain what they achieve. We demonstrate each call's effect by running its equivalent command in a c0hsm shell.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

3.2.1. c0hsm_archive

Description

c0hsm_archive is used to copy data from a fast tier to a slower one. Similarly to **c0hsm_copy**, the function's behaviour can be configured by flags to match the use-case; the available flags are the same as for *c0hsm_copy*. The main differences from **c0hsm_copy** are:

- The application doesn't need to specify the source tier. The HSM will copy all data which is currently located in tiers faster than the target one.
- This call only archives, i.e. copies data to a slower tier (tier with higher index).

```
/**
 * Request to archive data to a slower tier.
 * @param obj_id      Id of the object to be archived.
 * @param targer_tier  Target tier index (0 is the top tier).
 * @param offset       Start offset of the region to be staged.
 * @param length       Size of the region to be archived.
 * @param flags        Set of OR'ed hsm_cp_flags.
 * @return 0 on success, a negative error code on error.
 */
int c0hsm_archive(struct m0_uint128 obj_id, uint8_t target_tier,
                  off_t offset, size_t length, enum hsm_cp_flags flags);
```

Figure 14: *c0hsm_archive* prototype.

Demo

In this example, we use **c0hsm_archive** to archive multiple extents located in multiple tiers to an archive tier.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```
# Initial object state: 2 extents in tier 1, 1 extent in tier 0
c0hsm> show 0x1000005
- gen 3, tier 0, extents: [0x3000->0x3fff] (writable)
- gen 1, tier 1, extents: [0x1000->0x1fff]
- gen 0, tier 1, extents: [0->0xffff]

# Archive (move) all to tier 3
c0hsm> archive 0x1000005 0 0xFFFFF 3 mv
Archiving extent [0x3000-0x3fff] (gen 3) from tier 0 to tier 3
4096 bytes successfully copied from subobj <0xfffffc00:0x1000005> to
<0xfffffc03:0x1000005> at offset 0x3000
Extent [0x3000-0x3fff] (gen 3) successfully released from tier 0
Archiving extent [0x1000-0x1fff] (gen 1) from tier 1 to tier 3
4096 bytes successfully copied from subobj <0xfffffe01:0x1000005> to
<0xfffffe03:0x1000005> at offset 0x1000
Extent [0x1000-0x1fff] (gen 1) successfully released from tier 1
Archiving extent [0-0xffff] (gen 0) from tier 1 to tier 3
4096 bytes successfully copied from subobj <0xffffff01:0x1000005> to
<0xffffff03:0x1000005> at offset 0
Extent [0-0xffff] (gen 0) successfully released from tier 1

# Final state: all extents moved to tier 3
c0hsm> show 0x1000005
- gen 4, tier 0, extents: (writable)
- gen 3, tier 3, extents: [0x3000->0x3fff]
- gen 1, tier 3, extents: [0x1000->0x1fff]
- gen 0, tier 3, extents: [0->0xffff]
```

Figure 15: c0hsm_archive demo.

Initially, the object consists of 3 extents of different data generations, one of them located on tier 0, the other two located on tier 1.

c0hsm_archive() is then called to move all three extents to tier 3.

In the final state, one sees that all three extents are located on tier 3.

3.2.2. c0hsm_stage

Description

c0hsm_stage is used to copy data from a slow tier to a faster one. Unlike c0hsm_archive(), c0hsm_stage() only takes the WRITE_TO_DEST flag (see c0hsm_copy for explanation). The main differences from c0hsm_copy are:

- The application doesn't need to specify the source tier. The HSM will copy all data which is currently located in tiers slower than the target one.
- This call only stages, i.e. copies data to a faster tier (tier with lower index).

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```

/**
 * Request to stage data to a faster tier.
 * @param obj_id      Id of the object to be staged.
 * @param targer_tier  Target tier index (0 is the top tier).
 * @param offset       Start offset of the region to be staged.
 * @param length       Size of the region to be staged.
 * @param flags        HSM_WRITE_TO_DEST if next write operations should also
 *                    be directed to the same fast tier (intensive writes
 *                    expected).
 * @return 0 on success, a negative error code on error.
 */
int c0hsm_stage(struct m0_uint128 obj_id, uint8_t target_tier,
                off_t offset, size_t length, enum hsm_cp_flags flags);

```

Figure 16: *c0hsm_stage* prototype.

Demo

In this example, we use **c0hsm_stage** function to stage multiple data extents to a faster tier.

```

# initial state
c0hsm> show 0x1000000
- gen 2, tier 2, extents: (writable)
- gen 0, tier 2, extents: [0x2000->0x3fff]
- gen 0, tier 3, extents: [0->0x1fff] [0x4000->0x8fff]

# Staging to tier 1
c0hsm> stage 0x1000000 0x0 0xFFFFF 1
Staging extent [0x2000-0x3fff] (gen 0) from tier 2 to tier 1
8192 bytes successfully copied from subobj <0xffffffff02:0x1000000> to
<0xffffffff01:0x1000000> at offset 0x2000
Staging extent [0-0x1fff] (gen 0) from tier 3 to tier 1
8192 bytes successfully copied from subobj <0xffffffff03:0x1000000> to
<0xffffffff01:0x1000000> at offset 0
Staging extent [0x4000-0x8fff] (gen 0) from tier 3 to tier 1
20480 bytes successfully copied from subobj <0xffffffff03:0x1000000> to
<0xffffffff01:0x1000000> at offset 0x4000

# final state
c0hsm> show 0x1000000
- gen 2, tier 2, extents: (writable)
- gen 0, tier 1, extents: [0->0x8fff]
- gen 0, tier 2, extents: [0x2000->0x3fff]
- gen 0, tier 3, extents: [0->0x1fff] [0x4000->0x8fff]

```

Figure 17: *c0hsm_stage* demo.

Initially, the object consists of 3 extents located in tiers 2 and 3.

`c0hsm_stage()` is used to copy them all to tier 1, as the final state shows.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

3.2.3. c0hsm_multi_release

Description

c0hsm_multi_release's purpose is to release disk space from multiple levels of storage in a single operation. Just as `c0hsm_release()`, `c0hsm_multi_release()` can be configured to keep the latest data generation by setting the `HSM_KEEP_LATEST` flag.

Input to the `c0hsm_multi_release` function includes a maximal tier index, denoted `max_tier_idx` in the prototype. A release operation will be performed for all tier indices from 0 up to and including `max_tier_idx`.

```
/**
 * Release a region of an object from multiple tiers.
 * All data in tiers up to max_tier (including max_tier) are released.
 * @return 0 on success, a negative error code on error.
 * @param obj_id      Id of the object to be released.
 * @param targer_tier  Target tier index (0 is the top tier).
 * @param offset       Start offset of the region to be released.
 * @param length       Size of the region to be released.
 * @param flags        Set of OR'ed hsm_rls_flags.
 * @return 0 on success, a negative error code on error.
 */
int c0hsm_multi_release(struct m0_uint128 obj_id, uint8_t max_tier,
                        off_t offset, size_t length, enum hsm_rls_flags
                        flags);
```

Figure 18: `c0hsm_multi_release` prototype.

Demo

This example demonstrates the use of **c0hsm_multi_release** to release the disk space of multiple extents located in multiple tiers.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```
# initial state
c0hsm> show 0x1000000
- gen 2, tier 2, extents: (writable)
- gen 0, tier 0, extents: [0x1000->0x8fff]
- gen 0, tier 1, extents: [0->0xffff] [0x3000->0x8fff]
- gen 0, tier 2, extents: [0x2000->0x3fff]
- gen 0, tier 3, extents: [0->0x8fff]

# Release disk space in all top tiers (0, 1, 2)
c0hsm> multi_release 0x1000000 0x0 0xFFFFFFFF 2
Extent [0x1000-0x8fff] (gen 0) successfully released from tier 0
Extent [0-0xffff] (gen 0) successfully released from tier 1
Extent [0x3000-0x8fff] (gen 0) successfully released from tier 1
Extent [0x2000-0x3fff] (gen 0) successfully released from tier 2

# final state
c0hsm> show 0x1000000
- gen 2, tier 2, extents: (writable)
- gen 0, tier 3, extents: [0->0x8fff]
```

Figure 19: c0hsm_multi_release demo.

Initially, the example object is stored as a whole in tier 3, and various parts of the object are staged to tiers 0, 1 and 2.

Calling c0hsm_multi_release drops all extents up to tier 2, as long as a copy of the data remains available in tier 3.

The final state output indicates that the only remaining data extent is in tier 3.

3.3. c0hsm command-line interface

The **c0hsm** command-line interface is a wrapper around the HSM API calls detailed above. It can be used for triggering HSM actions manually. It also provides a “shell” mode that allows running batches of HSM operations without initialising/finalising Clovis connections for each command. It can display the location of objects managed by HSM; and can also be useful for HSM administration.

The possible actions are listed below:

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

```

Usage: c0hsm <action> <fid> [...]
  actions:
    create <fid> <tier>
    show <fid>
    dump <fid>
    write <fid> <offset> <len> <seed>
    write_file <fid> <path>
    read <fid> <offset> <len>
    copy <fid> <offset> <len> <src_tier> <tgt_tier>
        [options: mv,keep_prev,w2dest]
    move <fid> <offset> <len> <src_tier> <tgt_tier>
        [options: keep_prev,w2dest]
    stage <fid> <offset> <len> <tgt_tier> [options: w2dest]
    archive <fid> <offset> <len> <tgt_tier>
        [options: mv,keep_prev,w2dest]
    release <fid> <offset> <len> <tier> [options: keep_latest]
    multi_release <fid> <offset> <len> <max_tier>
        [options: keep_latest]
    set_write_tier <fid> <tier>
    shell

```

Figure 20: c0hsm command line syntax

The following table details the correspondence between c0hsm commands and HSM API calls.

c0hsm command	Equivalent HSM API call
create	c0hsm_create
copy	c0hsm_copy
move	c0hsm_copy with HSM_MOVE flag
stage	c0hsm_stage
archive	c0hsm_archive
release	c0hsm_release
multi_release	c0hsm_multi_release
set_write_tier	c0hsm_set_write_tier

The following commands provide additional services that are not provided by the HSM APIs:

- **show**: displays a summary of data extents over all tiers for a given object.
- **dump**: it is a more detailed version of “show”. It displays more details about data extents present in all tiers, like their subobject ids. Indeed, each layer of a composite object can be accessed using a specific identifier. This can be useful for reading previous versions of objects.
- **read**: dumps the contents of an object.
- **write**: (for testing) writes dummy data to a given extent of an object.
- **write_file**: write a file’s contents to a given object.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

- **shell**: start a shell to execute a set of HSM commands interactively

3.4. HSM Data Placement Advisor

Principle

This library implements the “storage pool selection” mechanism presented in deliverable D3.5. Its purpose is to help applications with selection of the most appropriate storage tier for a given I/O pattern.

We have seen in the previous sections that HSM expects a tier index as input of most of its API calls. A simple approach be to have applications loading data into the fastest tier when they need to access it, and archiving data to a higher-capacity tier when they no longer use it. However, we also want to take advantage of the deep storage hierarchy implemented by SAGE, with three tiers or more, and it seems smarter to reserve the top tier for fast random access while intermediate tiers are used for purposes like e.g. sequential access of large blocks.

Interface

The library implements and provides two main calls. Library calls are prefixed by “dpa”, which stands for “data placement advisor”:

- **dpa_init**: Initializes the library by reading a configuration file. This configuration file contains the tiers’ performance characteristics.

```
/**
 * Load tier characteristics from a configuration file.
 * @param[in]  cfg    Path to the configuration file.
 * @param[out] hdl    Handle to the library resources.
 * @return 0 on success, -errno on error.
 */
int dpa_init(void **hdl, const char *cfg);
```

Figure 21 - Definition of function “dpa_init”

- **dpa_match**: analyse access pattern hints and returns the index and name of the tier that matches these hints best.

```
/**
 * Match a coma-separated list of criteria to the closest tier.
 * @param[in] hdl    Handle to the library resources given by dpa_init.
 * @param[in] str    Input string is of the form "x=1, y=2, z=24".
 * @param[out] tier_index Index of the suggested tier.
 * @return The name of the suggested tier.
 */
const char *dpa_match(void *hdl, const char *str, uint8_t *tier_index);
```

Figure 22 - Definition of function “dpa_match”

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

Configuration file syntax

An HSM DPA configuration file consists of a list of tiers and their characteristics. The following example defines 2 tiers: one on SSDs, one on HDD.

```
[tier0_ssd]
io_rate = 10k
sync = 1
random_io = yes
r_speed = 2GB/s
w_speed = 1GB/s
rewrite = 1
volume = 1-100GB

[tier1_disk]
io_rate = -1k
io_size = +512
r_speed = 100MB/s
w_speed = 80MB/s
rewrite = 1
volume = -4T
```

Figure 23 - Examples of DPA configuration

Demo

In the example below, we use the test program “hsm_dpa” as a wrapper around the DPA library. We pass it a configuration file that contains tier specifications for a three-tier setup (hsm.cfg), and a string that represents some application hints. The program then displays the tier that matches the input hints best. In the output below, we also included some debug traces to showcase how the pool selection decision was made:

```
> hsm_dpa hsm.cfg random_io=1,io_rate=1000,r_speed=10M

REGION SCORES:
    tier0_flash: 1.90
    tier1_disk: 0.90
    tier2_archive: -0.30
best match: tier0_flash

> hsm_dpa hsm.cfg random_io=0,io_size=10M,volume=1T

REGION SCORES:
    tier0_flash: -0.20
    tier1_disk: 2.00
    tier2_archive: 3.00
best match: tier2_archive
```

Figure 24 - Demonstration of tier selection using HSM DPA

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

4. Deployment and integration

This section describes our implementation of HSM on the SAGE prototype. It describes our integration of HSM with SAGE use-cases, and evaluates it by taking performance measurements for data workflows for these use-cases.




4.1. Deployment on SAGE prototype

On the SAGE prototype cluster, HSM is deployed to manage 4 storage tiers illustrated in Table 2:

- The fastest tier, with index 0, uses NVRAM.
- The second tier, with index 1, uses SSDs.
- The third tier, with index 2, uses fast SAS disks.
- The slowest tier, with index 3, uses archive disks, optimized for high capacity and low cost.

As described in Section 3, HSM API calls and c0hsm commands take a tier index as an argument for data placement.

Table 2 - HSM storage tiers in the SAGE prototype

Tier index	Storage Technologies	Storage Enclosure
0	NVRAM (NVDIMM / new tech)	
1	NVRAM – FLASH 2.5" SSD	
2	Fast 3.5" SAS disk	 Fast disk

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

3	Cost optimised 3.5" SATA disk	 <p>Archive</p>
---	-------------------------------	---------------------------------------------------------------------------------------------------

Example 1:

This command creates an object located on the SSD storage tier (tier 1) of the SAGE prototype:

```
# Create an object located on SSDs (tier 1)
# 0x1000000 is the object identifier
c0hsm create 0x1000000 1
```

Example 2:

This command moves an object located on the NVRAM tier (tier 0) to the archive tier (tier 3):

```
# Move an object from NVRAM (tier 0) to archive (tier 3)
# 0x1000000 is the object identifier
# 0x0 0xFFFFFFFF represent the data range to be moved
c0hsm move 0x1000000 0x0 0xFFFFFFFF 0 3
```

4.2. Integration with SAGE Runtime Environment

4.2.1. Virtual Memory Manager (VMM)

Description and principle

The Virtual Memory Manager is developed in WP4 to manage the data workflow for interactive visualization of large three-dimensional data arrays.

The use-case includes 4 steps:

1. A pre-processing step generates data that is ingested to the SAGE object store in an archive tier.
2. Data is loaded to a fast cache tier using HSM primitives.
3. The visualization application intensively reads data from the cache tier.
4. When the application terminates, all data is removed from the fast cache tier using HSM calls.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

HSM integration

The implementation of this use-case relies on the following calls:

Task	Used API calls
Create an object in archive	<code>c0hsm_create(id, obj, archive, 0)</code>
Read data object	Standard clovis read: <code>m0_clovis_obj_op(obj, M0_CLOVIS_OC_READ, ...)</code>
Load data object into cache	<code>c0hsm_stage(obj, cache, 0, ext, 0)</code>
Evict data object from cache	<code>c0hsm_multi_release(obj, cache, 0, ext)</code>

Test

The purpose of this test is to evaluate the HSM's capability to implement this use-case. It reproduces the steps of a VMM workflow-based visualisation use case:

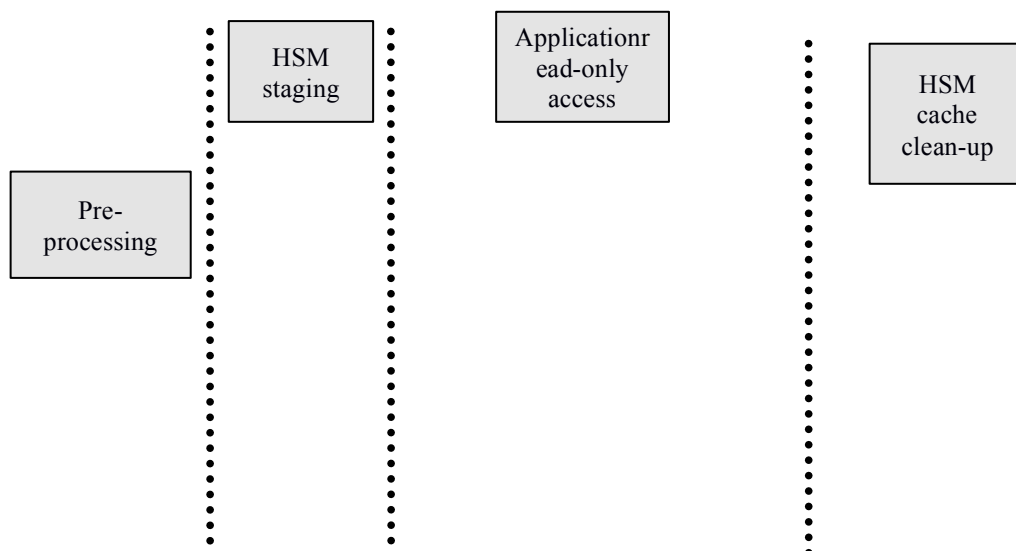
- 1) Populating a SAGE object store with a set of data, written to an archive tier (tier 3).
- 2) Loading (copying) this set of data into tier 0.
- 3) Perform read-only access of the data set.
- 4) Release data from tier 0.

During this test, we measure the volume of data located in each tier.

To highlight the different data management phases, we arbitrarily reduced the time spent in the application access phase (3rd phase). This phase could actually be much longer in a real use-case, which would make the benefits of pre-loading data to a fast storage tier even more important.

The result of this test is presented below:

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	



- In the first phase, the chart shows that the amount of data in the “archive” tier grows. This growth represents the data which is written during the pre-processing step.
- Then, in the second phase, the chart indicates the previously written data being copied to the fast storage tier, thus doubling the total amount of data in the system (1 copy in archive tier, 1 copy in fast disk cache).
- In the 3rd phase, the application performs read-only operations (in the fast storage level), so the data volume doesn't evolve.
- The final step consists in freeing used resources in the fast storage level.

4.2.2. Semi-Persistent Cache Manager (SPCM)

Description and principle

The SPCM is developed in WP4. Its current main use-case is the JURASSIC code. Unlike the previous use-case, the SPCM is designed to manage object modification. The SPCM system loads data to a fast tier when application needs it. The application can then modify the object and benefits of the fast device speed. Once the application no longer needs to access the object, the SPCM archives it back to a slower tier.

HSM integration

The implementation of this use-case uses the following HSM API calls:

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

Task	Used API calls
Read data object	Standard clovis read: m0_clovis_obj_op(obj, M0_CLOVIS_OC_READ, ...)
Stage data object by move	c0hsm_stage(obj, cache, 0, ext, HSM_MOVE)
Stage data object by copy	c0hsm_stage(obj, cache, 0, ext, 0)
Migrate unaltered data object	c0hsm_multi_release(obj, cache, 0, ext)
Migrate altered data object by copy	c0hsm_archive(obj, arch, 0, ext, HSM_MOVE)
Migrate altered data object with versioning	c0hsm_archive(obj, arch, 0, ext, HSM_MOVE HSM_KEEP_PREV_VERS)

The following HSM calls are used for the final cleanup phase:

Task	Used API calls
Evict unaltered objects from cache	c0hsm_multi_release(obj, cache, 0, ext)
Evict altered objects from cache	c0hsm_archive(obj, arch, 0, ext, HSM_MOVE)

Test

This test reproduces the data workflow of a JURASSIC use-case which works with the SPCM.

The test consists of the following steps:

0) Initially, data is located in an archival tier (tier 3)

1.1) Objects that are not expected to be modified are copied to the fast storage tier (tier 0), with their original copy remaining in the archive.

1.2) Objects that are expected to be modified are moved to the fast storage tier. The copy is cleared from the archived tier, as it will be outdated after modification.

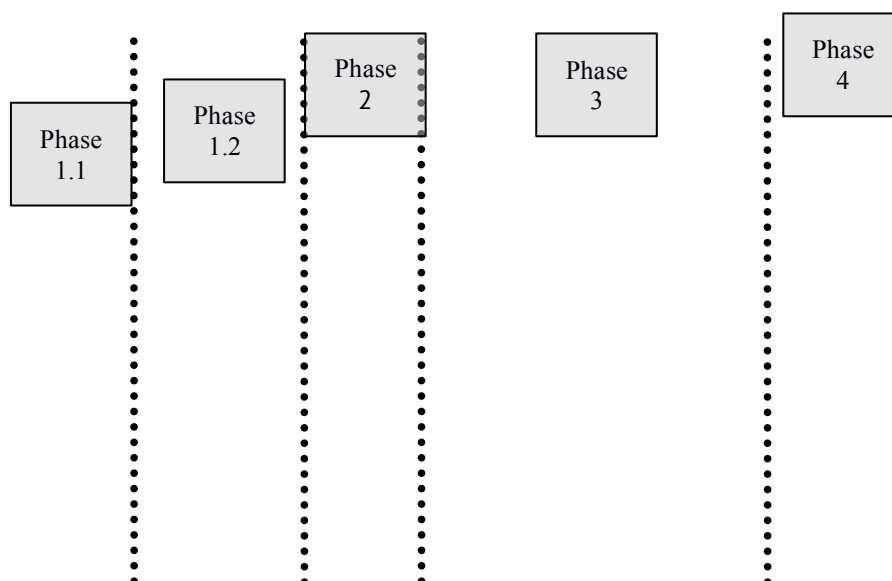
2) Once all needed data are staged to the fast tier, the application runs. It can read data from the fast tier, and write (append) new data to the objects.

3) Modified objects are fully copied back to the archive tier.

4) Non-modified objects are cleared from the fast storage tier.

Data location during this test run is represented in the chart below:

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGE for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	



- During phase 1.1, the volume of data in tier 0 increases as objects are copied from tier 3 to tier 0. Volume in tier 3 remains constant as original copies are left of in tier 3.
- In phase 1.2, objects are moved from tier 3 to tier 0. Thus, the volume in tier 3 decreases and the volume in tier 0 is increased by an equal amount. The total (cumulated) volume of data remains constant.
- The application runs in phase 2. It accesses data available in the fast tier and appends new data to objects. The chart thus indicates that the amount of data in tier 0 increases.
- During phase 3, the volume in the archive tier increases as objects are moved from tier 0 to tier 3. Volume in tier 0 is reduced by the same amount, thus preserving the total cumulated volume in the system.
- Finally, in phase 4, non-modified objects are released from the fast tier, so the volume in tier 0 decreases as the volume in the archive is unchanged.

Possible optimization for append case

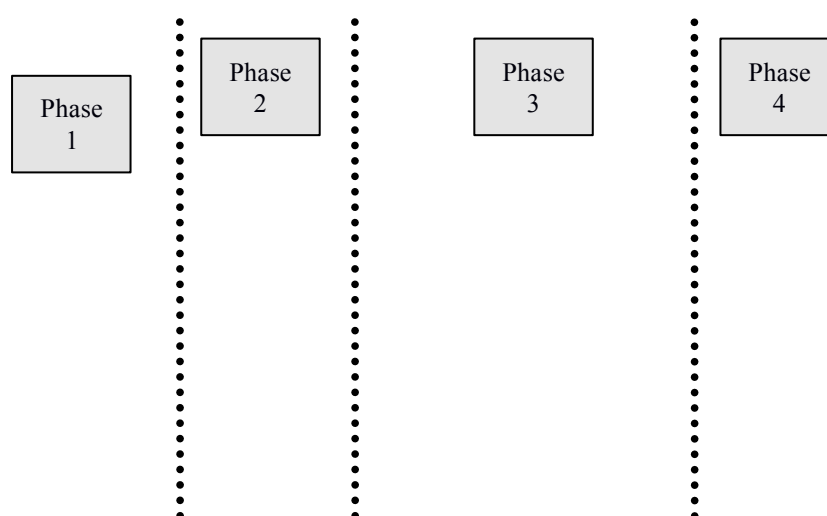
In the case objects are appended, the use-case can be optimized by taking advantage of some HSM features developed in WP3.1. As this HSM implementation can handle parts of objects, it is not needed to move whole objects to the fast disk cache if they are likely to be modified. Indeed, as the modification consists of appending objects, the initial version of data can remain in the archive (in phase 1.2), then HSM will only have to archive the new data range to the archive in phase 3, thus reducing the total amount of data to be moved at this step.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

Test (optimised version for append case)

To implement this optimization, we merge the 2 steps 1.1 and 1.2 in a single step 1. This step simply consists in copying (staging) data from the archive to the fast tier, without taking care if the object will be modified or not by the application.

The result of this test is represented below:



- In phase 1, volume in tier 0 increases as objects are copied to it. Volume in tier 3 is unchanged as original object copies remain in the archive.
- Phase 2 is similar to the previous one: volume in tier 0 increases as application writes new data. The volume in tier 3 is constant.
- In phase3, only the newly written data is to be moved to tier 3. Non-modified parts of the cached objects are already present in tier 3 so it can be dropped from the fast level without being copied again. As fewer data is copied to the archive compared to the amount of data that is released from tier 0, the total amount of data in the system decreases at this step (unlike the previous test).
- Phase 4 is similar to the previous test run.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

5. Status, further work, and possible future directions

As stated in the introduction, the design steps of this project task T3.1 lead us to consider various innovative approaches for Hierarchical Storage Management, and data life-cycle management in general.

The main work of implementing HSM mechanisms on top of the Mero object store is done. The API and tools presented in section 3 are available on the SAGE prototype hosted at Jülich.

In the last remaining months of the project, the following points will have to be finalised:

- The “multi-pool support” feature recently implemented as part of WP2 is to be integrated to finalize the HSM implementation and properly support all storage tiers on the prototype hardware.
- To perfectly achieve a transparent behaviour with regard to applications, it should be made possible to rely on future Clovis capability of managing concurrency internally without relying on application-level locking.

At this point, it also seems interesting to bind the HSM implementation to the pNFS interface developed in WP3.2. This way, the pNFS gateway could provide a high-performance POSIX access to objects stored in MERO and managed by HSM. Additionally, HSM-specific information such as data location across storage tiers can be exposed to pNFS clients through standard extended attributes. Last but not least, writing specific values to particular extended attributes could trigger HSM movements from a standard pNFS client.

Some other aspects have been considered with a more prospective approach. It is expected that they cannot be addressed during the project as they would require more resources, but they seem interesting avenues of development in the future:

- Considering more scalable ways to manage data life cycle in large Exascale systems. This can be done by using a map-reduce-style mechanism like FDMI. This feature was used in WP2 to keep track of associations between objects and containers. FDMI would offer further possibilities for HSM, e.g. tracking the usage of data ranges in each tier, thus providing useful information to make policy decisions. The duration of the project did not allow for going deeper into that direction.
- Using machine learning and AI to guide data movement decisions. The idea is to optimize data movement by predicting future application access patterns based on past runs. As described in D3.5, we explored some algorithms to detect recurrent access patterns, more or less successfully. It would certainly require a lot more research to test and compare various predictions methods, tune them, and implement them for HSM use cases. This could not be deeply investigated in the time of the SAGE project, but the idea is still interesting to consider in the future.

SAGE	SAGE_WP3_HSM_for_SAGE_Final_report_v1.1
Percipient StorAGe for Exascale Data Centric Computing	Created on 04/04/2018
D3.9 HSM for SAGE: Final Validation Report	

6. Conclusion

The SAGE project made it possible to overtake the state of the art in Hierarchical Storage Management:

- The implementation of HSM made in T3.1 can handle an arbitrary number of storage tiers, while most of today's systems only manage 2 levels (one active and one archive backend). Managing such deep storage hierarchies makes it possible to take advantage of the wide variety of storage devices available today: Non-Volatile Memory, Solid State Disks, Hard Drive Disks, archive disks...
- SAGE HSM is capable of managing data at various granularities: It can handle full objects, parts of objects, but also sets of objects by using the container feature developed in WP2. This provides a high flexibility in data management, not previously available in existing systems.
- The provided HSM primitives offer configurable behaviour, which enables other features than HSM. In particular, the implemented HSM primitives could also be used for other use cases like backup, versioning, etc.

Besides T3.1, this successful implementation is also the result of an intense, exciting, and efficient collaboration between the different SAGE Work Packages.

Thanks to all participants of the project, and to everyone who made this project possible.