Centre of Excellence in Simulation of Weather and Climate in Europe
Phase 2

Report of Appliances Available for Testing

Deliverable D4.2

The project Centre of Excellence in Simulation of Weather and Climate in Europe Phase 2 (ESiWACE2) has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 823988

About this document

Work package in charge: WP4 Data Systems for Scale

Actual delivery date for this deliverable: September 30th 2022
Dissemination level: PU (for public use)

Lead authors:
Seagate Systems UK (Seagate): Ganesan Umanesan, Sai Narasimhamurthy
Data Direct Networks (DDN): Jean-Thomas Acquaviva, Konstantinos Chasapis

Other contributing authors:
Fondazione Centro Euro-Mediterraneo sui Cambiamenti Climatici (CMCC): Donatello Elia

Table of Contents

1. Abstract / publishable summary

This deliverable reports on the two storage technologies from two vendors, DDN and Seagate, which were designed to work with the Earth System Data Middleware (ESDM) - details of which were reported in Deliverable D4.1 “Advanced software stack for Earth system data”[ESiWACE2 D4.1]. The expectation was that various storage appliances would have support for ESDM and could be deployed with ESDM at the weather and climate sites as ESDM would become available.

Seagate demonstrated the CORTX Motr (formerly Mero) software installation being capable of working with commodity hardware, with the ability to be installed at the German Climate Computing Center (DKRZ) and other centres in the future as per their requirements. This is a change in strategy from Seagate for CORTX Motr. At the beginning of the project, the plan was to have an appliance based on Mero – as Mero was closed source software. However, during the time frame of ESiWACE2, Mero was open sourced as CORTX Motr – making it feasible to be deployed very widely on commodity hardware. Showcasing CORTX Motr’s capability to work on commodity hardware working with ESDM, we believe is a powerful message for the community rather than providing just a “closed” appliance. This deliverable hence takes that approach.

DDN demonstrated the relevance and the efficiency of the Earth Systems Data Middleware (ESDM) developed during the ESiWACE project on state-of-the-art industrial storage technologies. DDN implemented a sophisticated integration of ESDM with the flash native, burst buffer solution product of DDN’s portfolio, Infinite Memory Engine (IME). As part of the integration ESDM now supports the IME native interface that bypasses the FUSE, allowing to obtain the maximum performance of IME. The code of the integration is available and accessible to everyone through GitHub. Using IME, DDN was able to produce performance results significantly beyond the state of the art in term of hardware / software cooperation. Furthermore, ESDM provides performance in the range of the low-level API while still offering a higher level of semantic to the end-user. The results were obtained within one of DDN’s labs using 5 IME appliances and 10 clients. With an aggregated peak I/O bandwidth of 100 GB/sec, this setting is representative of the core of the HPC / technical computing market. In addition to the lab set-up and the integration effort, DDN has led more theoretical and research-oriented work. In respect to the Active Storage task, DDN provided a complete design architecture and implemented a proof of concept (PoC). The prototype is expected to be ready within the Q1 of 2023. The PoC code of the active storage is available on GitHub.

2. Conclusion & Results

The following were the main conclusions and outcomes from the work:

1. ESDM works with CORTX Motr from Seagate through the CORTX Motr API backend. The adaptation for ESDM on Mero (through the “Clovis backend”) was implemented early on in the project and reported in D4.1. The Clovis backend was renamed to CORTX Motr API after it was open sourced. ESDM was hence made to work with that backend and proven in this deliverable.
2. The CORTX Motr software with ESDM was successfully deployed on a virtual machine emulating commodity hardware and requirements at DKRZ - suitable for adoption by other weather and climate centres.
3. DDN has ported ESDM on its IME flash native appliance. Obtained results on ESDM are extremely good: ESDM provides a semantic consistent with the expectations of scientists while achieving performance comparable to the one obtained with low level proprietary API from DDN.
4. DDN designed the architecture of the active storage/computational storage feature. An initial prototype is already available and the solution is planned to be ready within the Q1 of 2023.

3. Project objectives

This deliverable contributes directly and indirectly to the achievement of all the macro-objectives and specific goals indicated in section 1.1 of the Description of the Action:

Macro-objectives	Contribution of this deliverable?
(1) Enable leading European weather and climate models to leverage the available performance of pre-exascale systems with regard to both compute and data capacity in 2021.	Yes
(2) Prepare the weather and climate community to be able to make use of exascale systems when they become available.	Yes

Specific goals in the workplan	Contribution of this deliverable?
Boost European climate and weather models to operate in world-leading quality on existing supercomputing and future pre-exascale platforms	
Establish new technologies for weather and climate modelling	Yes
Enhance HPC capacity of the weather and climate community	Yes
Improve the toolchain to manage data from climate and weather simulations at scale	Yes
Strengthen the interaction with the European HPC ecosystem	Yes
Foster co-design between model developers, HPC manufacturers and HPC centres	Yes

4. Detailed report on the deliverable

We will detail the approaches taken by Seagate and DDN on the appliances for weather and climate centres.

4.1 CORTX Motr (Seagate)

This section will discuss the implementation and deployment of CORTX Motr [Motr] on a virtual machine with ESDM. The original plan was to deploy CORTX Motr on the DKRZ working environment. However, after discussion with the DKRZ system administrators, it was decided to do this in a phased approach where:

(a) We demonstrate CORTX Motr in User Space – as it minimises risks from a system admin perspective, rather than deploying experimental software in kernel space
(b) Show that CORTX Motr in User Space with ESDM passes all the tests and satisfies the requirements for installation at DKRZ. This also gives tremendous flexibility for deploying CORTX Motr on any commodity hardware, if it is a User Space program
(c) Start the work on moving this to an actual deployment

We show (a) and (b) in this deliverable. (c) will be work that will continue in earnest in the future even after the end of the project – where Seagate will also continue to support DKRZ through the CORTX community.

We next discuss the successful deployment of CORTX Motr in user space.

4.1.1 Introduction and Background

CORTX Motr is an object storage system developed by Seagate and now fully open sourced. CORTX works on adapting any commodity storage hardware into a storage server suitable for HPC and cloud applications. In the following, the overall architecture of CORTX Motr server (also called just as "Motr") is described and presented in Figure 1.

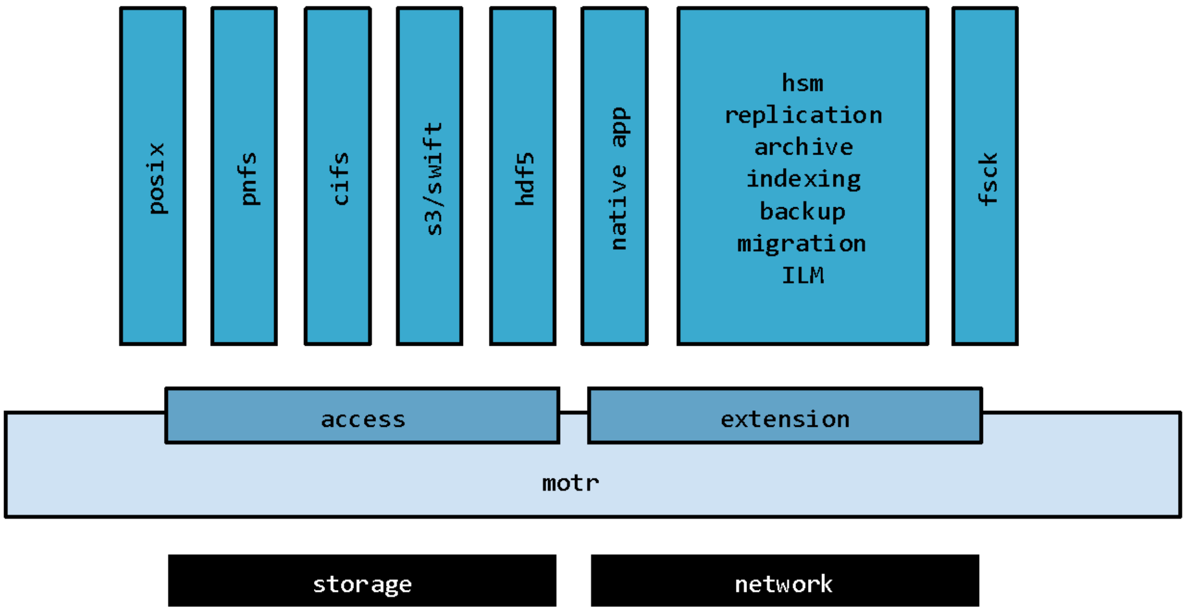


Figure 1: CORTX Motr Architecture.

Motr exposes an object I/O and Key Value (KV) interface through an access API and an extension API. The Key Values can be used to describe the Objects and overlay any higher level semantics on top of the object store such as POSIX, pNFS, S3, etc as indicated above. So the applications can access CORTX Motr natively or through such "gateways" through their respective clients. The extension interface is used to build various third party applications on top of the object store such as those needed for Hierarchical Storage Management (HSM), Replication, Indexing, File System Consistency Checking (FSCK), etc. Motr works on top of any commodity hardware server with HDDs, SDDs or NVRAM. The individual nodes of Motr are connected through a network that uses protocols such as "libfabric" [libfabric].

Many HPC applications can use the CORTX Motr native access interface. In ESiWACE2 the native access interface is accessed through the "ESDM" gateway.

A user space version of the CORTX Motr object store suitable for ESDM, and, weather and climate applications (and its usage and setup in a virtual environment) and its demonstration is described next.

4.1.2 CORTX Motr User Space Demonstration Steps

The original CORTX Motr is developed as a kernel module and it was dependent on various kernel mode modules such as "lnet" [lnet] which runs in kernel space – required for running the networking services. Most of the motr services used to run with root user privileges and hence in root user session. The result was that one machine (hence all its storage pools) could be part of only one motr cluster. Also running a motr cluster required to have "Sudo" or equivalent privileges which posed security concerns if the cluster is to be controlled by a normal user and not admin/sudo-user. Further bugs in the kernel module could result in system crashes and making the system unusable.

In order to eliminate the above-mentioned limitations and security concerns, motr code is further enhanced to work as a user-mode driver and use libfabric instead of lnet. Libfabric is a user-mode library providing similar network functionality as lnet as required by motr. Also, all the sudo/root access requirements have been removed and now it can be run by a normal user without any special sudo privilege requirement. Each user session is independent and is expected to be capable of running multiple motr instances on a single machine and users can control the cluster independently. All user specific code/data resides in respective user home directory to avoid any interference.

These outcomes were as a result of continued discussions with DKRZ, including the system administrators and the ESiWACE2 project stakeholders from WP4.

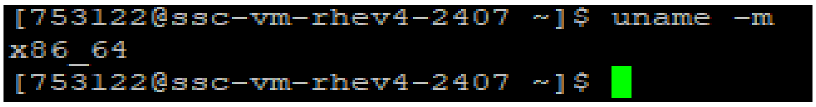
We next demonstrate CORTX Motr working in user space and the steps that need to be undertaken to achieve it – as presented to someone that wants to perform it. A package usermode-motr.tar.gz[1] was supplied by Seagate to run a single node CORTX Motr cluster through the following steps.

With this package, the following steps demonstrated CORTX Motr in user space (suitable for deployment on commodity hardware at DKRZ) along with ESDM tests on top of it. Please note that we summarise the steps – but then also provide some commands and screen shots inline, used to reproduce these. We will also make the below instructions openly available.

Step 1 - Checking Operating System (OS) and machine requirements

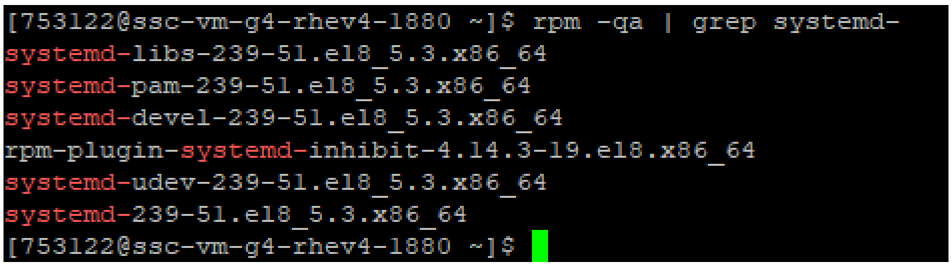
The following are the hardware and Operating System (typical of commodity hardware) that was successfully tested for the user space version of CORTX Motr (in a virtual machine). A similar environment needs to be used for CORTX Motr deployment.

(a) Hardware tested: x86_64 machine was tested as below



```
[753122@ssc-vm-rhev4-2407 ~]$ uname -m
x86_64
[753122@ssc-vm-rhev4-2407 ~]$
```

(b) OS tested: Centos version 8 was tested (with systemd>=239) as below



```
[753122@ssc-vm-g4-rhev4-1880 ~]$ rpm -qa | grep systemd-
systemd-libs-239-51.el8_5.3.x86_64
systemd-pam-239-51.el8_5.3.x86_64
systemd-devel-239-51.el8_5.3.x86_64
rpm-plugin-systemd-inhibit-4.14.3-19.el8.x86_64
systemd-udev-239-51.el8_5.3.x86_64
systemd-239-51.el8_5.3.x86_64
[753122@ssc-vm-g4-rhev4-1880 ~]$
```

Step 2 - Checking Red Hat Package Managers (RPMS) requirements

The usermode-motr.tar.gz package that is supplied has the following RPMs that are available. These include core CORTX Motr core functionality, CORTX Utilities and development tools, the configuration and high availability component called CORTX Hare, Intel's intelligent storage acceleration library that contains erasure coding API functions and some python utility programs provided by Seagate.

1) isa-l-2.30.0-1.el7.x86_64.rpm (Intelligent storage acceleration library)

[1] https://seagatetechnology.sharepoint.com/:u:/r/sites/gteamdrv1/tdrive1224/Shared Documents/Components/EsiWACE Motr Cluster/Vimlesh/USERMODE/usermode-motr.tar.gz?csf=1&web=1&e=DCZRQe Available upon request

2) cortx-py-utils-2.0.0-2_72f9d68.noarch.rpm (Utilities)
3) cortx-motr-2.0.0-1_git450998d6.el8.x86_64.rpm (Core Functionality)
4) cortx-motr-devel-2.0.0-1_git450998d6.el8.x86_64.rpm (Development Tools)
5) cortx-hare-2.0.0-1_git032746b.el8.x86_64.rpm (Configuration and High Availability)

Step 3 - Building the RPMS

The following RPMs are built as per the below instructions:

motr rpms:

```
$ git clone https://github.com/Seagate/cortx-motr.git
$ cd ~/cortx-motr
$ git checkout -b esiwace remotes/origin/esiwace2-dkrz
$ ./autogen.sh
$ ./configure  --with-user-mode-only
$ make clean; make rpms
```

cortx-utils rpms:

```
$ cd ~/cortx-utils
$ ./jenkins/build.sh -v 2.0.0 -b 2
```

cortx-hare rpms:

Please install cortx-motr rpms before building hare rpms and erase/uninstall after cortx-hare rpm is built.

```
$ cd ~
$ git clone https://github.com/Seagate/cortx-hare.git
$ cd cortx-hare
$ git checkout -b esiwace remotes/origin/esiwace2-dkrz
$ make rpm
```

isa-l rpms:

```
$ git clone https://github.com/daos-stack/isa-l
$ cd isa-l
$ git checkout -b isl-2.30 845f488c54ef27dfbee66b498099acda65344cf1
$ make
```

Step 4 - Setting up the Config requirements

The below scripts available in usermode-motr.tar.gz are then run.

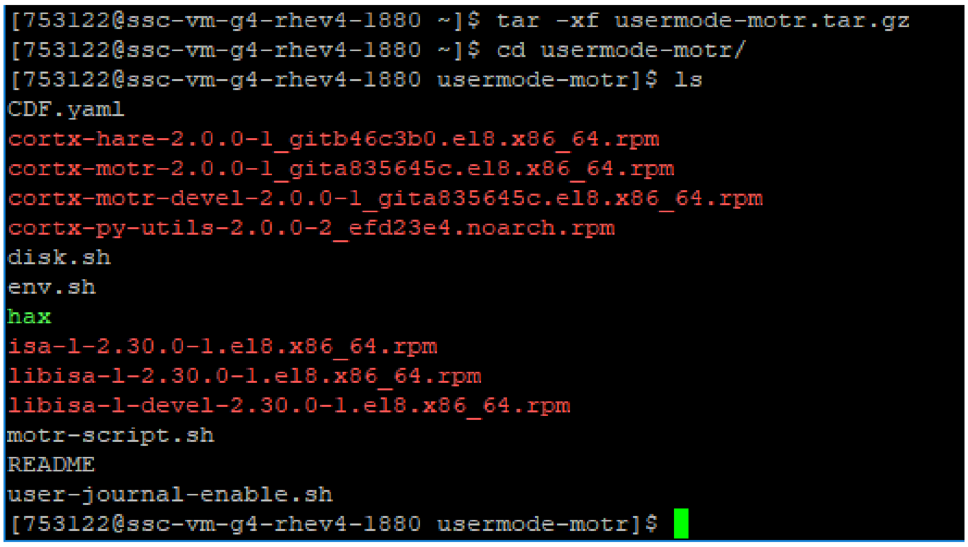
- user-journal-enable.sh
- motr-script.sh

“user-journal-enable.sh” is an optional script and is required to provide user mode journalctl access to non-root user. Journalctl is a utility for querying and displaying logs from journald, systemd's logging service. “motr-script.sh” script is required to perform initial user-mode environment setup from available rpms and other sources inside the usermode-motr directory obtained after running untar of usermode-motr.tar.gz

Step 5 - Setting up motr cluster environment

The following are the steps needed to set up the Motr cluster in user space.
Untar usermode-motr.tar.gz:

$ tar –xf usermode-motr.tar.gz
$ cd usermode-motr



```
[753122@ssc-vm-g4-rhev4-1880 ~]$ tar -xf usermode-motr.tar.gz
[753122@ssc-vm-g4-rhev4-1880 ~]$ cd usermode-motr/
[753122@ssc-vm-g4-rhev4-1880 usermode-motr]$ ls
CDF.yaml
cortx-hare-2.0.0-1_gitb46c3b0.el8.x86_64.rpm
cortx-motr-2.0.0-1_gita835645c.el8.x86_64.rpm
cortx-motr-devel-2.0.0-1_gita835645c.el8.x86_64.rpm
cortx-py-utils-2.0.0-2_efd23e4.noarch.rpm
disk.sh
env.sh
hax
isa-l-2.30.0-1.el8.x86_64.rpm
libisa-l-2.30.0-1.el8.x86_64.rpm
libisa-l-devel-2.30.0-1.el8.x86_64.rpm
motr-script.sh
README
user-journal-enable.sh
[753122@ssc-vm-g4-rhev4-1880 usermode-motr]$
```

Configure user-mode access to journalctl:
The script user-journal-enable.sh is run to enable/configure user-mode access to journalctl logs by non-root user so that users can view logs generated by user-unit services run by them.

$ sudo sh user-journal-enable.sh

Configure /etc/libfab.conf:
The user needs to configure libfabric by providing network protocol (like tcp/ib) and interface (like eth0/enp0s3, as on your system) to be used.
Example:

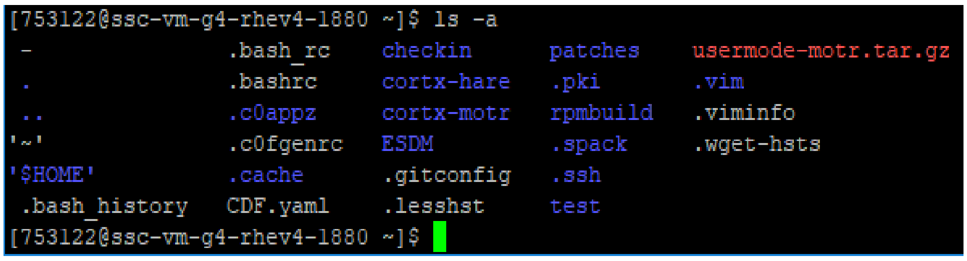
$ sudo sh -c 'echo "networks=tcp(enp0s3)" > /etc/libfab.conf'
$ cat /etc/libfab.conf networks=tcp(enp0s3)

Step 6 - Operating the CORTX Motr user mode cluster

The below steps demonstrate the basic operation of the CORTX cluster in user mode. We demonstrate successful starting, shutting down and querying the status of the CORTX Motr cluster (single node).
"hctl" or "Hare control" commands control the operation of the CORTX Motr cluster through the Hare module described earlier.
All the operations/services are executed on behalf of user executing hctlcluster commands. So, each user session is independent. All directories generated during cluster operations reside within the user home directory inside *$HOME/seagate/*.
Examples:

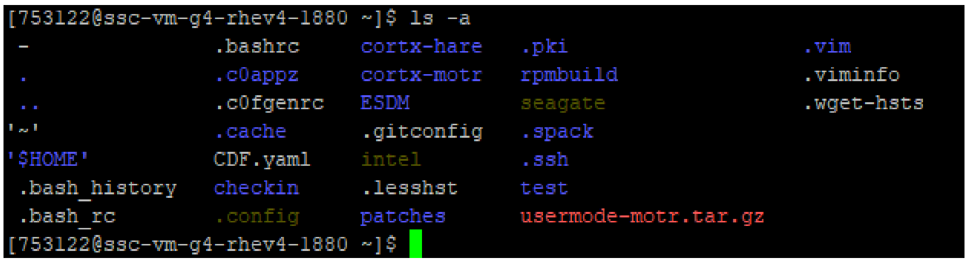
$HOME/seagate/var/lib/hare,
$HOME/seagate/var/motr,
$HOME/seagate/var/log/motr,
$HOME/seagate/var/log/seagate/,
$HOME/seagate/etc/motr

Before setting up cluster



```
[753122@ssc-vm-g4-rhev4-1880 ~]$ ls -a
 -                .bash_rc     checkin       patches     usermode-motr.tar.gz
 .                .bashrc      cortx-hare    .pki        .vim
 ..               .c0appz      cortx-motr    rpmbuild    .viminfo
'~'               .c0fgenrc    ESDM          .spack      .wget-hsts
'$HOME'           .cache       .gitconfig    .ssh
 .bash_history    CDF.yaml     .lesshst      test
[753122@ssc-vm-g4-rhev4-1880 ~]$
```

After setting up cluster (/bootstrapping]), we get new directories created as *intel/* for (isa-l), *.config/* and Seagate under *$HOME/.*



```
[753122@ssc-vm-g4-rhev4-1880 ~]$ ls -a
 -                .bashrc      cortx-hare   .pki                        .vim
 .                .c0appz      cortx-motr   rpmbuild                    .viminfo
 ..               .c0fgenrc    ESDM         seagate                     .wget-hsts
'~'               .cache       .gitconfig   .spack
'$HOME'           CDF.yaml     intel        .ssh
 .bash_history    checkin      .lesshst     test
 .bash_rc         .config      patches      usermode-motr.tar.gz
[753122@ssc-vm-g4-rhev4-1880 ~]$
```

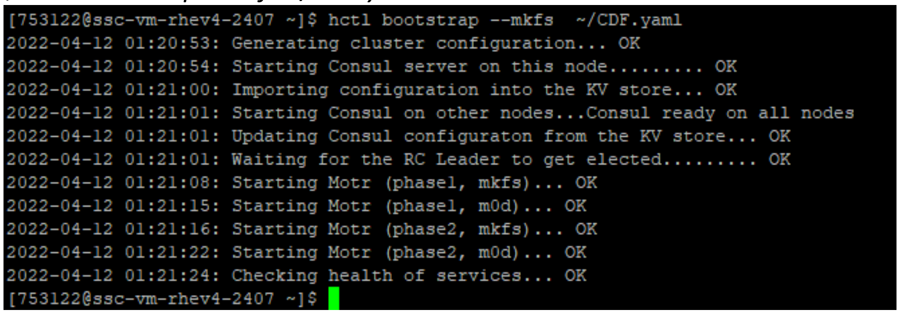
Setting up (per-user) environment:

The following script is run to setup motr preboot user mode environment. Cortx-motr/utils/hare/isa-l rpms are by-default taken from usermode-motr directory, the rpms names/location in the script are modified if required.

$ sh motr-script.sh
$ source env.sh

Bootstrapping the cluster:

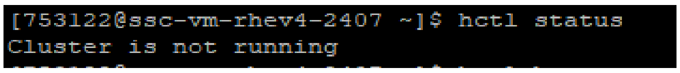
$ htcl bootstrap --mkfs ~/CDF.yaml



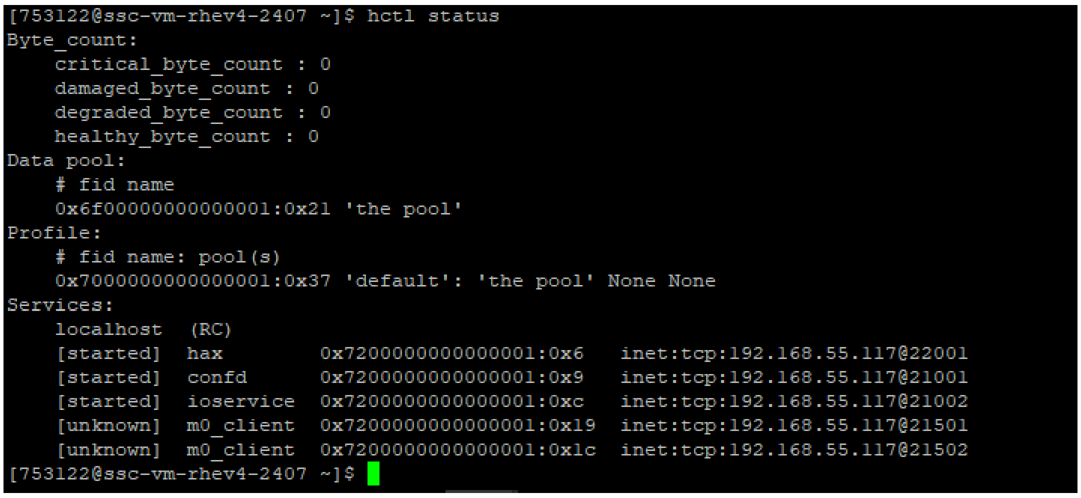
```
[753122@ssc-vm-rhev4-2407 ~]$ hctl bootstrap --mkfs  ~/CDF.yaml
2022-04-12 01:20:53: Generating cluster configuration... OK
2022-04-12 01:20:54: Starting Consul server on this node......... OK
2022-04-12 01:21:00: Importing configuration into the KV store... OK
2022-04-12 01:21:01: Starting Consul on other nodes...Consul ready on all nodes
2022-04-12 01:21:01: Updating Consul configuraton from the KV store... OK
2022-04-12 01:21:01: Waiting for the RC Leader to get elected......... OK
2022-04-12 01:21:08: Starting Motr (phase1, mkfs)... OK
2022-04-12 01:21:15: Starting Motr (phase1, m0d)... OK
2022-04-12 01:21:16: Starting Motr (phase2, mkfs)... OK
2022-04-12 01:21:22: Starting Motr (phase2, m0d)... OK
2022-04-12 01:21:24: Checking health of services... OK
[753122@ssc-vm-rhev4-2407 ~]$
```

Status of the cluster:

$ hctl status



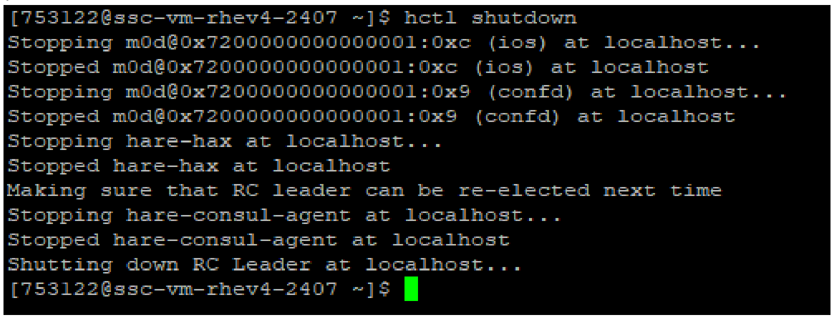
```
[753122@ssc-vm-rhev4-2407 ~]$ hctl status
Cluster is not running
```



```
[753122@ssc-vm-rhev4-2407 ~]$ hctl status
Byte_count:
    critical_byte_count : 0
    damaged_byte_count : 0
    degraded_byte_count : 0
    healthy_byte_count : 0
Data pool:
    # fid name
    0x6f00000000000001:0x21 'the pool'
Profile:
    # fid name: pool(s)
    0x7000000000000001:0x37 'default': 'the pool' None None
Services:
    localhost  (RC)
    [started]  hax        0x7200000000000001:0x6   inet:tcp:192.168.55.117@22001
    [started]  confd      0x7200000000000001:0x9   inet:tcp:192.168.55.117@21001
    [started]  ioservice  0x7200000000000001:0xc   inet:tcp:192.168.55.117@21002
    [unknown]  m0_client  0x7200000000000001:0x19  inet:tcp:192.168.55.117@21501
    [unknown]  m0_client  0x7200000000000001:0x1c  inet:tcp:192.168.55.117@21502
[753122@ssc-vm-rhev4-2407 ~]$
```

Shutting down the cluster:

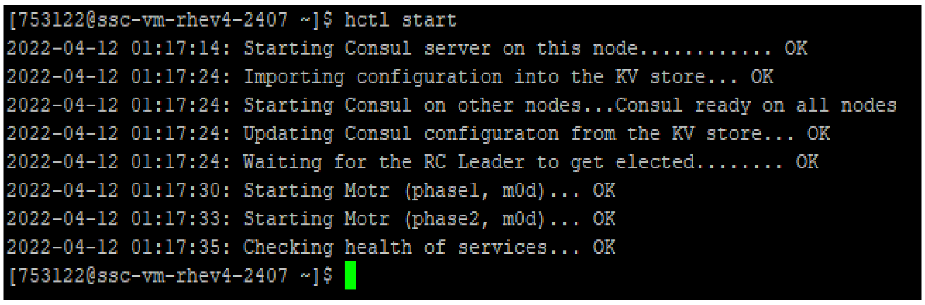
$ htcl shutdown



```
[753122@ssc-vm-rhev4-2407 ~]$ hctl shutdown
Stopping m0d@0x7200000000000001:0xc (ios) at localhost...
Stopped m0d@0x7200000000000001:0xc (ios) at localhost
Stopping m0d@0x7200000000000001:0x9 (confd) at localhost...
Stopped m0d@0x7200000000000001:0x9 (confd) at localhost
Stopping hare-hax at localhost...
Stopped hare-hax at localhost
Making sure that RC leader can be re-elected next time
Stopping hare-consul-agent at localhost...
Stopped hare-consul-agent at localhost
Shutting down RC Leader at localhost...
[753122@ssc-vm-rhev4-2407 ~]$
```

Starting the cluster:

$ htcl start



```
[753122@ssc-vm-rhev4-2407 ~]$ hctl start
2022-04-12 01:17:14: Starting Consul server on this node........... OK
2022-04-12 01:17:24: Importing configuration into the KV store... OK
2022-04-12 01:17:24: Starting Consul on other nodes...Consul ready on all nodes
2022-04-12 01:17:24: Updating Consul configuraton from the KV store... OK
2022-04-12 01:17:24: Waiting for the RC Leader to get elected........ OK
2022-04-12 01:17:30: Starting Motr (phase1, m0d)... OK
2022-04-12 01:17:33: Starting Motr (phase2, m0d)... OK
2022-04-12 01:17:35: Checking health of services... OK
[753122@ssc-vm-rhev4-2407 ~]$
```

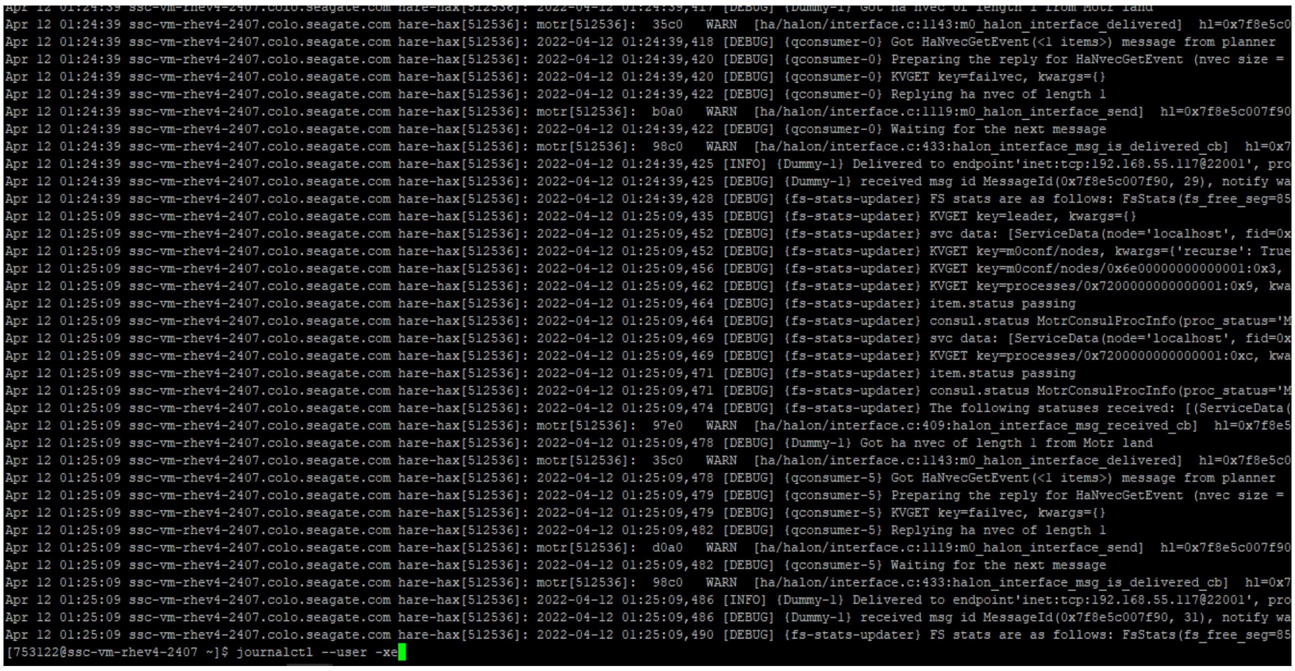
Step 7 - Checking CORTX Motr logs in user-mode

Use journalctl facility to view logs generated by motr. Journalctl provides user-specific log messages and the user running journalctl is able to see only logs which belong to them and not from any other users or system logs.

Example commands:

$ journalctl –-user –xe

$ journalctl --since "2 min ago"



```
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,417 [DEBUG] {Dummy-1} Got ha nvec of length 1 from Motr land
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: motr[512536]:  35c0   WARN  [ha/halon/interface.c:1143:m0_halon_interface_delivered]  hl=0x7f8e5c0
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,418 [DEBUG] {qconsumer-0} Got HaNvecGetEvent(<1 items>) message from planner
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,420 [DEBUG] {qconsumer-0} Preparing the reply for HaNvecGetEvent (nvec size =
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,420 [DEBUG] {qconsumer-0} KVGET key=failvec, kwargs={}
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,422 [DEBUG] {qconsumer-0} Replying ha nvec of length 1
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: motr[512536]:  b0a0   WARN  [ha/halon/interface.c:1119:m0_halon_interface_send]  hl=0x7f8e5c007f90
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,422 [DEBUG] {qconsumer-0} Waiting for the next message
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: motr[512536]:  98c0   WARN  [ha/halon/interface.c:433:halon_interface_msg_is_delivered_cb]  hl=0x7
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,425 [INFO] {Dummy-1} Delivered to endpoint'inet:tcp:192.168.55.117@22001', pro
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,425 [DEBUG] {Dummy-1} received msg id MessageId(0x7f8e5c007f90, 29), notify wa
Apr 12 01:24:39 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:24:39,428 [DEBUG] {fs-stats-updater} FS stats are as follows: FsStats(fs_free_seg=85
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,435 [DEBUG] {fs-stats-updater} KVGET key=leader, kwargs={}
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,452 [DEBUG] {fs-stats-updater} svc data: [ServiceData(node='localhost', fid=0x
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,452 [DEBUG] {fs-stats-updater} KVGET key=m0conf/nodes, kwargs={'recurse': True
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,456 [DEBUG] {fs-stats-updater} KVGET key=m0conf/nodes/0x6e00000000000001:0x3,
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,462 [DEBUG] {fs-stats-updater} KVGET key=processes/0x7200000000000001:0x9, kwa
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,464 [DEBUG] {fs-stats-updater} item.status passing
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,464 [DEBUG] {fs-stats-updater} consul.status MotrConsulProcInfo(proc_status='M
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,469 [DEBUG] {fs-stats-updater} svc data: [ServiceData(node='localhost', fid=0x
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,469 [DEBUG] {fs-stats-updater} KVGET key=processes/0x7200000000000001:0xc, kwa
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,471 [DEBUG] {fs-stats-updater} item.status passing
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,471 [DEBUG] {fs-stats-updater} consul.status MotrConsulProcInfo(proc_status='M
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,474 [DEBUG] {fs-stats-updater} The following statuses received: [(ServiceData(
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: motr[512536]:  97e0   WARN  [ha/halon/interface.c:409:halon_interface_msg_received_cb]  hl=0x7f8e5
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,478 [DEBUG] {Dummy-1} Got ha nvec of length 1 from Motr land
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: motr[512536]:  35c0   WARN  [ha/halon/interface.c:1143:m0_halon_interface_delivered]  hl=0x7f8e5c0
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,478 [DEBUG] {qconsumer-5} Got HaNvecGetEvent(<1 items>) message from planner
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,479 [DEBUG] {qconsumer-5} Preparing the reply for HaNvecGetEvent (nvec size =
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,479 [DEBUG] {qconsumer-5} KVGET key=failvec, kwargs={}
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,482 [DEBUG] {qconsumer-5} Replying ha nvec of length 1
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: motr[512536]:  d0a0   WARN  [ha/halon/interface.c:1119:m0_halon_interface_send]  hl=0x7f8e5c007f90
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,482 [DEBUG] {qconsumer-5} Waiting for the next message
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: motr[512536]:  98c0   WARN  [ha/halon/interface.c:433:halon_interface_msg_is_delivered_cb]  hl=0x7
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,486 [INFO] {Dummy-1} Delivered to endpoint'inet:tcp:192.168.55.117@22001', pro
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,486 [DEBUG] {Dummy-1} received msg id MessageId(0x7f8e5c007f90, 31), notify wa
Apr 12 01:25:09 ssc-vm-rhev4-2407.colo.seagate.com hare-hax[512536]: 2022-04-12 01:25:09,490 [DEBUG] {fs-stats-updater} FS stats are as follows: FsStats(fs_free_seg=85
[753122@ssc-vm-rhev4-2407 ~]$ journalctl --user -xe
```

4.1.3 ESDM on user space CORTX Motr - Steps

The following steps demonstrate ESDM on the user space instantiation of CORTX Motr.

Step 1 – Building CORTX Motr for ESDM integration

Cortx-motr should be built with the following configuration for ESDM integration.

$ git clone https://github.com/Seagate/cortx-motr.git
$ cd cortx-motr
$ ./autogen.sh
$ ./configure --enable-finject
$ make rpms
$ yum localinstall cortx-motr-2.0. 0-1_git.el8.x86_64.rpm*

Step 2 – Building & Installing ESDM with NetCDF

Before embarking for building ESDM, the following dependencies need to be installed.

Step 2-1 – Installing Spack tool

"Spack" [Spack] is a multiplatform package manager that builds and installs multiple versions and configurations of software. It is installed as follows.

Clone the git repository

$ git clone --depth=2 https://github.com/spack/spack.git spack
$. spack/share/spack/setup-env.sh

Set a gcc version to be used

$ export gccver=8.5.0

Install dependencies

$ spack install gcc@$gccver +binutils
$ spack compiler find

```
$ spack install autoconf%gcc@$gccver
$ spack install openmpi%gcc@$gccver
$ spack install gettext%gcc@$gccver
$ spack install jansson%gcc@$gccver
$ spack install glib%gcc@$gccver
$ spack install cmake%gcc@$gccver
```

Load dependencies

```
$ spack load gcc@$gccver
$ spack load autoconf%gcc@$gccver
$ spack load libtool%gcc@$gccver
$ spack load openmpi%gcc@$gccver
$ spack load jansson%gcc@$gccver
$ spack load cmake%gcc@$gccver
$ spack load glib%gcc@$gccver
```

These dependencies need to be loaded every time, enabling a user login into new session.

Step 2-2 – Installing ESDM with NetCDF enablement

ESDM from the GitHub repository is cloned and configured as below.

```
$ git clone https://github.com/ESiWACE/es
```

Configure the repository as follows:

```
$ cd esdm
$ pushd deps
$ ./prepare.sh
$ popd
$ ./configure \$    --prefix=${PREFIX} \$    --enable-netcdf4
$ cd build
$ make
$ make install
```

Step 2-3 – Installing netCDF with ESDM

After the installation of ESDM, the NetCDF module is installed as below. The steps below are followed for the cloning, configuration and installation.

```
$ git clone https://github.com/ESiWACE/esdm-netcdf-c.
$ cd esdm-netcdf-c
$ ./bootstrap
$ export INSTAPATH=/path-to-ESDM-installation/
$ ./configure \
    --prefix=$prefix \
    --with-esdm=$INSTPATH \
    LDFLAGS="-L$INSTPATH/lib" \
    CFLAGS="-I$INSTPATH/include" \
    CC=mpicc \
    --disable-dap
$ make -j
$ make install
```

Step 3 – Testing ESDM with Motr Cluster

Step 3-1 – Generation of the ESDM.conf file

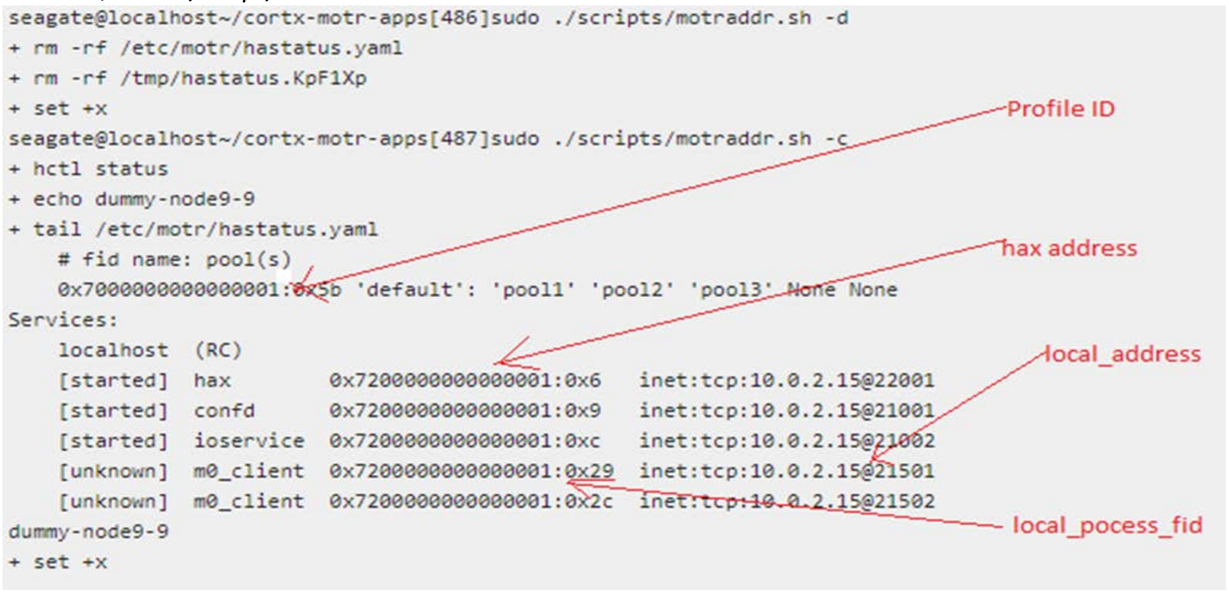
For testing with the motr cluster, ESDM should be provided with the Motr connection parameters in the esdm.conf file located in the search path. esdm.conf will be loaded with the following contents for the same purpose.

```
{
    "esdm": {
        "backends": [
            {
                "type": "MOTR",
                "id": "c1",
                "target": "inet: tcp:192.168.50.39@21501
inet: tcp:192.168.50.39@22001 0x7000000000000001:0x4f 0x7200000000000001:0x29",
"performance-model”: {"latency”: 0.0137, "throughput”: 3218.0},
                "max-threads-per-node”: 8,
                "max-fragment-size”: 268435456,
                "max-global-threads”: 64,
                "accessibility”: "global"
            }
        ],
        "metadata": {
            "type": "metadummy",
            "id": "md",
            "target": “. /metadummy"
        }
    }
```

The contents in bold type in the above snippet should be replaced with the corresponding parameters specific to the cluster (DKRZ cluster).
The parameters “target” are interpreted as : *<**local address**>, <**hax_address**>, <**Profile ID**>, <**local_process_fid**>.*

The same could be obtained, if the cluster had been started already, and then using the following command located in the clovis (cortx-motr-apps) application, if installed in the system.

$ cd cortx-motr-apps
$ sudo ./script/motraddr.sh -d



```
seagate@localhost~/cortx-motr-apps[486]sudo ./scripts/motraddr.sh -d
+ rm -rf /etc/motr/hastatus.yaml
+ rm -rf /tmp/hastatus.KpF1Xp
+ set +x
seagate@localhost~/cortx-motr-apps[487]sudo ./scripts/motraddr.sh -c
+ hctl status
+ echo dummy-node9-9
+ tail /etc/motr/hastatus.yaml
    # fid name: pool(s)
    0x7000000000000001:0x5b 'default': 'pool1' 'pool2' 'pool3' None None
Services:
    localhost  (RC)
    [started]  hax        0x7200000000000001:0x6   inet:tcp:10.0.2.15@22001
    [started]  confd      0x7200000000000001:0x9   inet:tcp:10.0.2.15@21001
    [started]  ioservice  0x7200000000000001:0xc   inet:tcp:10.0.2.15@21002
    [unknown]  m0_client  0x7200000000000001:0x29  inet:tcp:10.0.2.15@21501
    [unknown]  m0_client  0x7200000000000001:0x2c  inet:tcp:10.0.2.15@21502
dummy-node9-9
+ set +x
```

The cortx-motr-app client application can be installed using the following GitHub link: https://github.com/Seagate/cortx-motr-apps.git [cortx apps]. These apps perform four basic operation such as 1.) creating an object, 2.) writing to an object, 3.) reading from an object, and 4.) deleting an object.

Step 3-2 – Running test cases designed for ESDM testing

For running the esdm test cases located in the esdm/src/test directory, the below steps should be followed. The cortx-motr user mode tar file (usermode-motr.tar.gz) is downloaded into the user's local home directory.

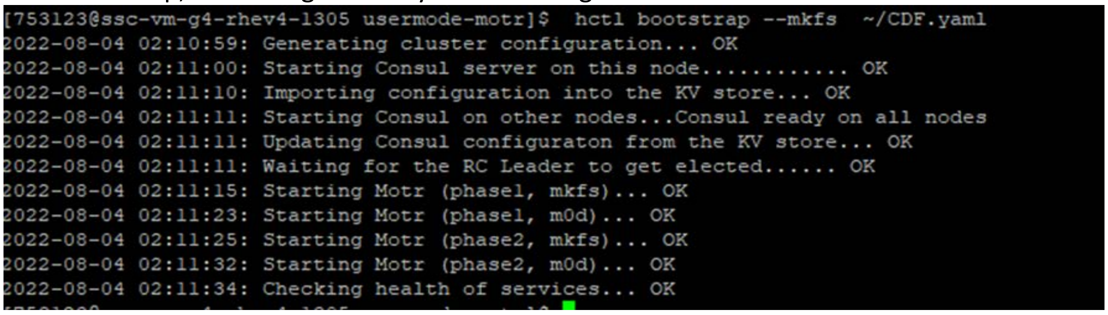
The user mode cortx motr code base is untarred in the home directory using the following commands and a few bash scripts are executed as shown to setup the user mode cortx.

$ tar -xf usermode-motr.tar.gz
$ cd usermode-motr
$ source ./motr-script.sh
$ source ./env.sh

The cluster is then bootstrapped using the following commands

$ hctl bootstrap --mkfs ~/CDF.yaml

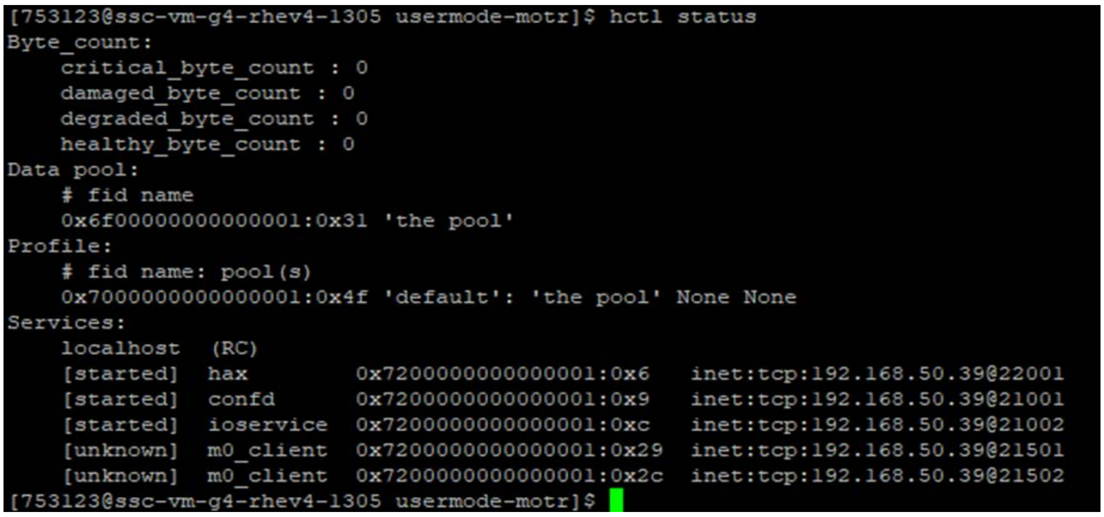
If the cluster comes up, it will be greeted by the following screen.



```
[753123@ssc-vm-g4-rhev4-1305 usermode-motr]$  hctl bootstrap --mkfs  ~/CDF.yaml
2022-08-04 02:10:59: Generating cluster configuration... OK
2022-08-04 02:11:00: Starting Consul server on this node............ OK
2022-08-04 02:11:10: Importing configuration into the KV store... OK
2022-08-04 02:11:11: Starting Consul on other nodes...Consul ready on all nodes
2022-08-04 02:11:11: Updating Consul configuraton from the KV store... OK
2022-08-04 02:11:11: Waiting for the RC Leader to get elected...... OK
2022-08-04 02:11:15: Starting Motr (phase1, mkfs)... OK
2022-08-04 02:11:23: Starting Motr (phase1, m0d)... OK
2022-08-04 02:11:25: Starting Motr (phase2, mkfs)... OK
2022-08-04 02:11:32: Starting Motr (phase2, m0d)... OK
2022-08-04 02:11:34: Checking health of services... OK
```

If required, the following command can be used to assess the state of the cluster.

$ hctl status



```
[753123@ssc-vm-g4-rhev4-1305 usermode-motr]$ hctl status
Byte_count:
    critical_byte_count : 0
    damaged_byte_count : 0
    degraded_byte_count : 0
    healthy_byte_count : 0
Data pool:
    # fid name
    0x6f00000000000001:0x31 'the pool'
Profile:
    # fid name: pool(s)
    0x7000000000000001:0x4f 'default': 'the pool' None None
Services:
    localhost  (RC)
    [started]  hax        0x7200000000000001:0x6    inet:tcp:192.168.50.39@22001
    [started]  confd      0x7200000000000001:0x9    inet:tcp:192.168.50.39@21001
    [started]  ioservice  0x7200000000000001:0xc    inet:tcp:192.168.50.39@21002
    [unknown]  m0_client  0x7200000000000001:0x29   inet:tcp:192.168.50.39@21501
    [unknown]  m0_client  0x7200000000000001:0x2c   inet:tcp:192.168.50.39@21502
[753123@ssc-vm-g4-rhev4-1305 usermode-motr]$
```

After going through the above steps successfully, we can run the esdm test cases on the CORTX Motr user mode cluster.

$ cd ~/esdm/build
$ make test

Step 3-3 – Verifying results

The current motr cluster has been tested with ESDM using:

> *$ cd ~/esdm/build*
> *$ make test*

```
[753123@ssc-vm-g4-rhev4-1305 build] # make test
Running tests...
Test project /root/pkg/esdm/build
      Start  1: basic-types-conversion-tst
 1/38 Test  #1: basic-types-conversion-tst        .......  Passed    0.00 sec
      Start  2: basic types
 2/38 Test  #2: basic types        ......................  Passed    0.00 sec
      Start  3: derived-attr-serde
 3/38 Test  #3: derived-attr-serde        ...............  Passed    0.00 sec
      Start  4: derived-types
 4/38 Test  #4: derived-types        ....................  Passed    0.00 sec
      Start  5: ser-dtype
 5/38 Test  #5: ser-dtype        ........................  Passed    0.00 sec
      Start  6: serde
 6/38 Test  #6: serde        ............................  Passed    0.00 sec
      Start  7: string-stream-benchmark
 7/38 Test  #7: string-stream-benchmark        ..........  Passed    2.88 sec
      Start  8: tst-conversion-all
 8/38 Test  #8: tst-conversion-all        ...............  Passed    0.00 sec
      Start  9: tst-conversion-lu
 9/38 Test  #9: tst-conversion-lu        ................  Passed    0.00 sec
      Start 10: tst-conversion
10/38 Test #10: tst-conversion.        ..................  Passed    0.00 sec
      Start 11: tst-precision
11/38 Test #11: tst-precision        ....................  Passed    0.00 sec
      Start 12: metatest
12/38 Test #12: metatest        .........................  Passed    0.03 sec
      Start 13: a-many-dims-stress-test
13/38 Test #13: a-many-dims-stress-test        ..........  Passed   22.21 sec
      Start 14: a-many-fragments-stress-test
14/38 Test #14: a-many-fragments-stress-test        .... ***Failed    0.02 sec
      Start 15: a-writeback-benchmark
15/38 Test #15: a-writeback-benchmark        ........... ***Failed    0.04 sec
      Start 16: attributes
16/38 Test #16: attributes        ......................   Passed    0.02 sec
      Start 17: data-conversion
17/38 Test #17: data-conversion        ..................  Passed    0.02 sec
      Start 18: data-copy-benchmark
18/38 Test #18: data-copy-benchmark        ..............  Passed    1.80 sec
      Start 19: datatype-serializer
19/38 Test #19: datatype-serializer        ..............  Passed    0.03 sec
      Start 20: fill-value
20/38 Test #20: fill-value        .......................  Passed    0.58 sec
      Start 21: fragment-selection-benchmark
21/38 Test #21: fragment-selection-benchmark        ....   ***Failed 1.03 sec
      Start 22: fragmentation-method
22/38 Test #22: fragmentation-method        .............  ***Failed 0.03 sec
      Start 23: grid
23/38 Test #23: grid        .............................  Passed    0.67 sec
      Start 24: hypercube
24/38 Test #24: hypercube        ........................  Passed    0.07 sec
      Start 25: incomplete-copy
25/38 Test #25: incomplete-copy        ..................  Passed    0.02 sec
      Start 26: init
26/38 Test #26: init        .............................  Passed    1.42 sec
      Start 27: metadata_nc
27/38 Test #27: metadata_nc        ......................  Passed    0.94 sec
      Start 28: readwrite-benchmark
28/38 Test #28: readwrite-benchmark        ..............  Passed    7.24 sec
      Start 29: readwrite-conversion
29/38 Test #29: readwrite-conversion        .............  Passed    0.65 sec
      Start 30: readwrite
30/38 Test #30: readwrite        ........................  Passed    1.51 sec
      Start 31: scil
31/38 Test #31: scil        .............................  Passed    0.63 sec
      Start 32: unlim-dim
32/38 Test #32: unlim-dim        ........................  Passed    1.79 sec
```

```
      Start 33: write-simple
33/38 Test #33: write-simple        .....................   Passed    0.57 sec
      Start 34: write-stream
34/38 Test #34: write-stream .....  Subprocess aborted***Exception:   0.37 sec
      Start 35: write
35/38 Test #35: write       ............................   Passed    2.74 sec
      Start 36: zopen-dset-2x
36/38 Test #36: zopen-dset-2x        ....................   Passed    0.53 sec
      Start 37: zread-stream
37/38 Test #37: zread-stream        .....................   Passed    0.52 sec
      Start 38: zread
38/38 Test #38: zread       ............................   Passed    0.68 sec

87% of the tests passed, and 5 tests out of 38 failed.

Total Test time (real) = 49.07 sec

The following tests FAILED:
         14 - a-many-fragments-stress-test (Failed)
         15 - a-writeback-benchmark (Failed)
         21 - fragment-selection-benchmark (Failed)
         22 - fragmentation-method (Failed)
         34 - write-stream (Subprocess aborted)

Errors while running CTest
Output from these tests can be found in: /root/pkg/esdm/build/Testing/Temporary/LastTest.log
Use "--rerun-failed --output-on-failure" to re-run the failed cases verbosely.
make: *** [Makefile:71: test] Error 8
```

33 out of 38 tests have passed.
Test case #34 failed due to missing implementation of the one ESDM motr API

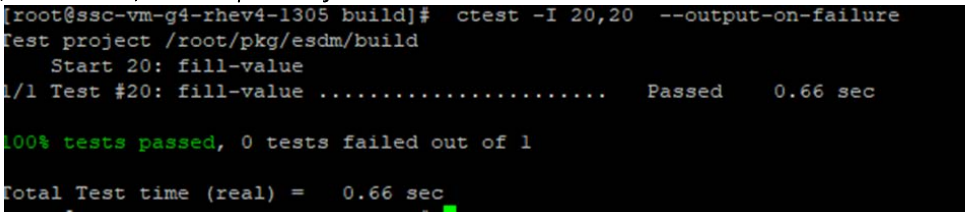
*(static int esdm_backend_t_motr_fragment_write_stream_blocksize (esdm_backend_t * b, estream_write_t * state, void * c_buf, size_t c_off, uint64_t c_size).*

And 4 tests failed due to their Posix setup requirement.
Please note that the failed tests, however, have been confirmed as not critical from the perspective of a demonstration of ESDM on CORTX Motr.

To run one testcase at a time, the following commands should be followed.

For example, if it is required to run testcase #20 only, the command is composed using the following template: *ctest -I <**start test case no**> <**end test case no**> --output-on-failure*

$ cd ~/esdm/build
$ ctest -I 20,20 --output-on-failure



```
[root@ssc-vm-g4-rhev4-1305 build]#  ctest -I 20,20  --output-on-failure
Test project /root/pkg/esdm/build
    Start 20: fill-value
1/1 Test #20: fill-value ......................   Passed    0.66 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.66 sec
```

To view the test log, use the following command:

$ cat ~/ esdm/build/Testing/Temporary/LastTest.log

4.1.4 Issues and Challenges

We next discuss some of the implementation issues and challenges that we faced in the Cortx Motr user mode deployment with ESDM. More general challenges are discussed later.

<u>**Challenges**</u>

- **Understanding the CMake build system**
  ESDM follows the CMake type build system, but due to its versatileness, and not having proper understanding initially, it took a little arduous effort to assimilate and to fix the motr connection issue with ESDM. But however, after having put in a fair share of effort, it was fixed successfully.

<u>**Issues**</u>

- **Motr was not connecting with ESDM**
  Solution: In the current ESDM build, there happens to be a lack of a few motr-specific changes in order to connect to cortx motr. This build only serves the purpose of **Posix**-based unit testing. So, the required CMake files were modified to make cortx motr connection in order.

- **A few ESDM test cases were failing due to block aligned issue with Cortx-motr**
  Solution: Since cortx motr has the constraint of only accepting block aligned amounts of data either for reading or writing, a few ESDM test cases failed. The required code changes were made in the ESDM code base to take care of the same. Most of the failed test cases ran successfully after that.

- **Test #34: write-stream failed even after the block aligned issue removal**
  ```
  34/38 Test #34: write-stream .....................Subprocess aborted***Exception:   1.11 sec
  ```
  The above-mentioned test case was failing, even after the block aligned issue had been fixed. This is due to the lack of implementation of motr specific API. This API is currently under development.

- **User mode cortx-motr cluster setup should be made generic and run by any user non-intrusively**
  Motr-script.sh was modified appropriately with generic variables, so that any user can set up the cluster without worrying about any user-specific changes in their configuration files.

4.1.5 Next steps (Seagate)

The user mode version of CORTX Motr + ESDM, as demonstrated, was agreed to be sufficient to move forward with the actual installation at DKRZ. To start with, the installation will be done on a test cluster. This is work in progress. Seagate will continue to support the installation even after the end of the project (through the CORTX Community). The user mode VM will be made available to other weather and climate data centres – which we hope will be a precursor for actual deployment. The performance will depend on the hardware on which CORTX Motr is installed since its open source software. Performance tests will be planned once we have the hardware instantiation at DKRZ.

4.2 ESDM with IME integration (DDN)

4.2.1 Introduction and background

The goal pursued by this work is two-fold, first to help application developers to keep the focus on their problem without requesting in-depth analysis of the storage system, second to deliver the highest level of I/O performance. ESDM is targeting to improve I/O performance for earth system simulation as used in climate and weather applications from the perspective of an I/O library. It exploits structural information exposed by workflows, applications as well as data description formats such as HDF5 and NetCDF to organise metadata and data more efficiently across a variety of storage backends. ESDM provides as well a

higher level of abstraction, thus allowing scientists to keep the focus on the expression of their problems, while shielding them from the underlying storage complexity.
On the other hand, storage is still undergoing a deep revolution with the arrival of Flash devices, storage class memory and cloud. Within the scope of the project, DDN worked on integrating Flash storage devices in ESDM, to provide the best performance without impacting the end users' codes. The flash integration within ESDM is implemented with the Infinite Memory Engine (IME) [IME] product of DDN. IME is a Scale-Out Flash Cache Layer using NVMe SSDs inserted between the compute cluster and the back-end parallel file system. It leverages the intrinsic flash properties, such as low latency, high bandwidth and the capability to be byteaddressable at minimal performance cost.

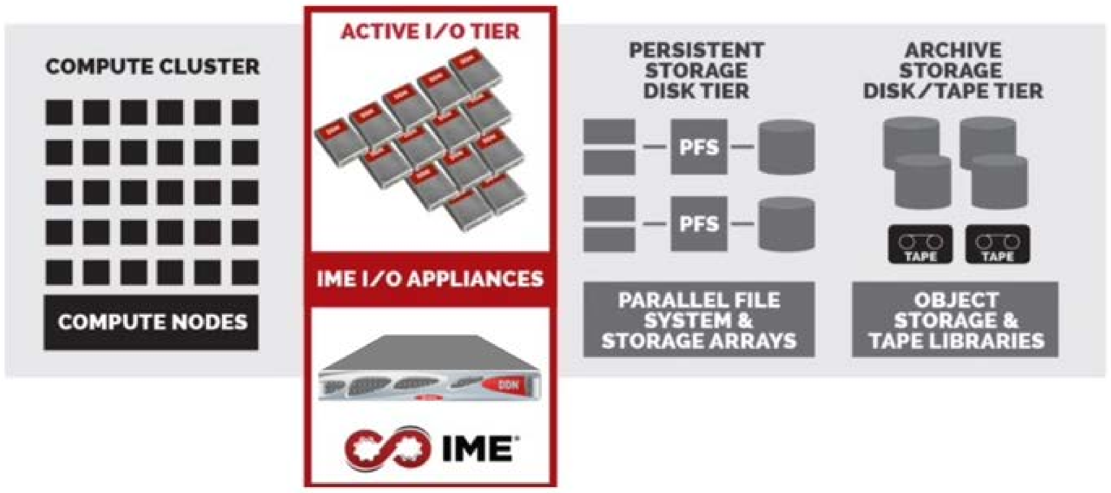

COMPUTE CLUSTER
COMPUTE NODES
ACTIVE I/O TIER
IME I/O APPLIANCES
IME
PERSISTENT STORAGE DISK TIER
PFS
PFS
PARALLEL FILE SYSTEM & STORAGE ARRAYS
ARCHIVE STORAGE DISK/TAPE TIER
TAPE
TAPE
OBJECT STORAGE & TAPE LIBRARIES

Figure 2: IME as part of the I/O path from the compute node to the parallel file system.

Figure 2 illustrates the I/O path from the compute nodes to the parallel file system with the IME inserted between them. IME is configured as a cluster with multiple NVMe servers. IME handles the I/O data traffic and forwards the metadata requests to the parallel file system. The purpose of IME is to accelerate difficult I/O patterns such as small or random I/O, and targeting shared files with high concurrency by taking advantage of Flash device characteristics. IME is implemented as thin software I/O management and shifts the burden of complex metadata operations to the underlying file system layer. IME is not a full-fledged file system and thus relies on the underlying file system to serve metadata calls. Furthermore, IME storage capacity is usually more limited than the file system's one, as IME is fully built with expensive NVMe devices. A spill-over mechanism is implemented in IME to move data to the file system when IME reaches capacity saturation.
From an end user standpoint, there are two major methods that ESDM can utilise to interact with IME. Without any modification to the ESDM, it can benefit from IME using the IME FUSE client utilising the POSIX backend. However, the IME FUSE client adds overhead, which includes a system call and the use of the fuse module with an additional copy from user to kernel space. These software overheads are known issues, but were considered as negligible in respect of the high latency of rotating devices. The disruptive apparition of Flash devices, with order of magnitude lower latency, suddenly shifts the bottleneck from the hardware to the software layers. Figure 3 shows a few of the layers within the kernel space that must be crossed to serve an I/O system call. Apart from the cost of the hardware operation itself, the software overhead is also significant. Fuse adds a copy of the data from user space to kernel space using a 128 KB buffer (The 128 KB limit has been extended to 1 MB for kernel starting 4.2). The size of this buffer prevents large I/O request to mitigate the latency cost for the call chain. For I/O of large size, the call stack will have to be traversed multiple times.

A second induced bottleneck is that fuse requires a lock between processes writing from the same node to a shared file. This lock can have a dramatic performance overhead for I/O highly concurrent accesses. It should be noted that this lock is not needed in the case of Flash systems which provide byte addressable accesses, thus avoiding the false sharing issue. In addition to the system call overhead, any fuse file system implementation must pay an extra penalty for crossing twice from user to kernel space and vice versa. As illustrated in Figure 3, any fuse call from the user application must cross from the user space to kernel, is forwarded to the fuse kernel module, propagated to the fuse user space library, served from the user level implementation of the file system, and finally returned through all the layers. IME tackles this overhead by offering applications a custom I/O interface called IME native. The IME native interface is publicly available at [IME_NATIVE]. This interface remains similar to POSIX but requires explicit calls of the API. IME provides a client library that can be exposed to the application directly with the IME client eliminating additional layers and minimising overhead as shown in Figure 5.

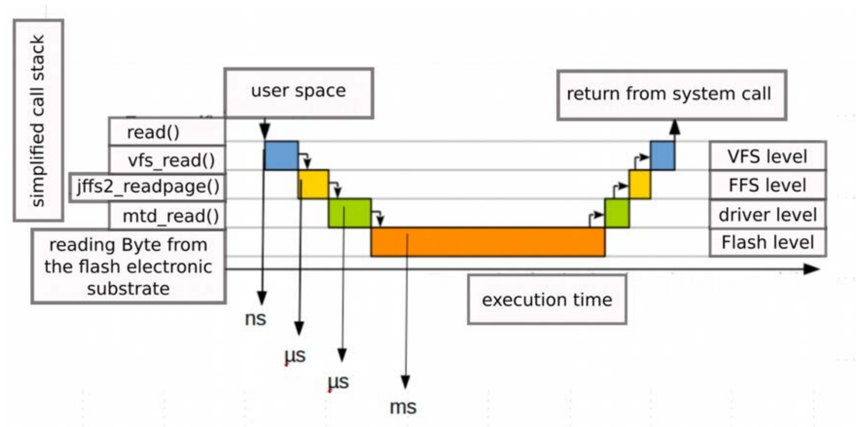


Figure 3: Summary of function calls and layers to serve an I/O call from user space up to storage device in respect to the execution time.

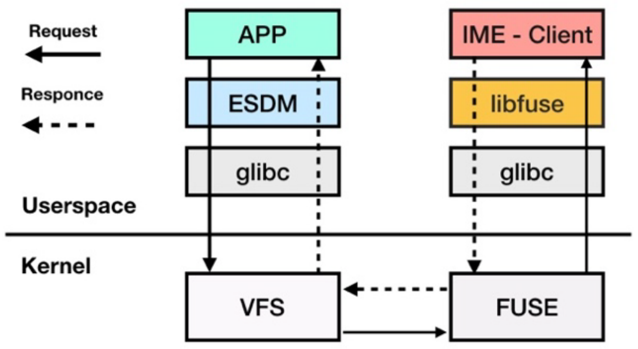


Figure 5: I/O path with ESDM and IME fuse client

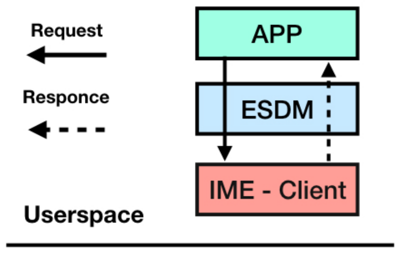


Figure 4: I/O path with ESDM and IME native integration.

As illustrated by the previous figures, Fuse is a critical component for the performance of the data flows. Out of this work on Flash, DDN acknowledges the importance of Fuse, both the kernel module and the user-space library. While not being directly within the scope of the ESiWACE project, following IME development, DDN is dedicating engineering resources to deliver improvements (software patches) to the fuse open-source code. All the produced code is pushed upstream to the fuse and kernel community.

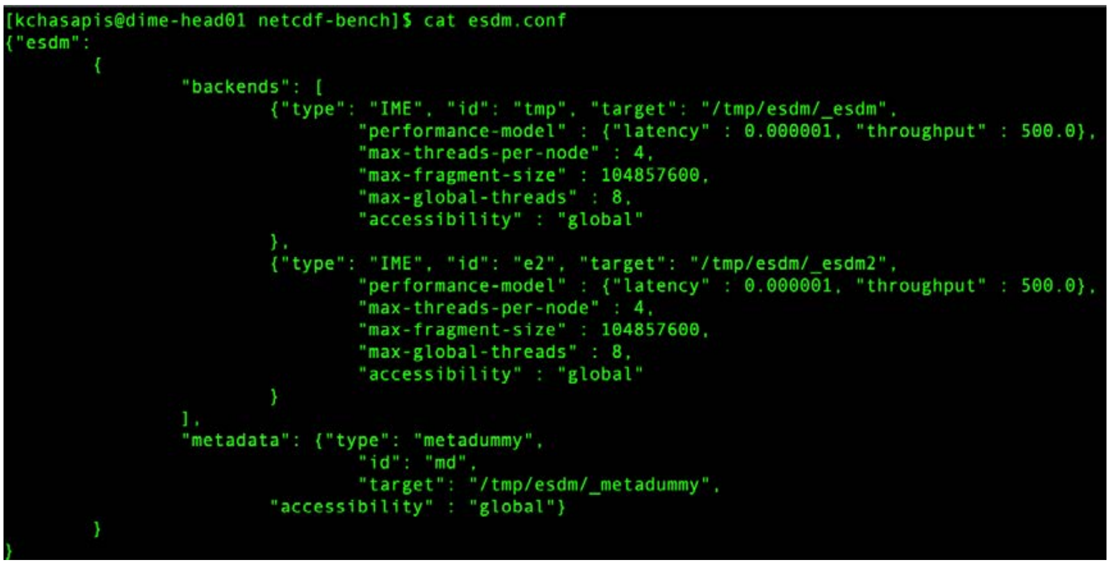
4.2.2 ESDM with IME native interface integration

To integrate the ESDM with the IME native interface we implemented a new data backend for ESDM. The I/O path from the application to ESDM and to the IME client is illustrated in Figure 4 for the case of the IME Fuse and in Figure 5 for the case of IME native. Using Fuse offers genericity at the cost of a more complex data flow, form a portability perspective it is possible to run directly ESDM software on IME Fuse interface. However, at the expense of performance degradation. The IME backend support also the Native interface, where application directly call the IME client API, bypassing all the need of buffering attached to POSIX calls managed by Fuse. The goal of our work was to interface the ESDM software with the IME native client API. Any applications using ESDM will then transparently get the high performance provided by IME native API.
To summarize, IME native interfaces the ESDM links directly to the IME client, in contrast to IME fuse, which requires crossing the user to kernel space barrier two times. Furthermore, IME native is an application-level call, thus saving the overhead of a system call. Therefore, the IME Native API is the most performant and shortest call path from an application to the IME client.

The new IME native ESDM data backend is implemented at: *src/backemds-data/ime*[2]. The implementation is based on the POSIX backend and has been modified accordingly to support the IME native interface. The following commands demonstrate how one can utilise the IME backend for ESDM. The initial step is to clone the GitHub repository of ESDM, the integration of IME native has been already merged in the master brunch of ESDM. The second step is to configure ESDM with IME support. In order to properly configure ESDM, a configuration parameter is needed to specify the paths to IME libraries and header files. The rest of the compilation and installation steps are the same and are described in section 4.1.2 Step 8.

```
$: git clone https://github.com/ESiWACE/esdm.git
$: cd esmd
$: ./configure --with-ime-include=/opt/ddn/ime/include/ --with-ime-lib=/opt/ddn/ime/lib
--enable-netcdf4
```

To utilise the IME native interface, one must specify IME as the backend type and provide the path to the IME mount point in the target field in the ESDM configure file. Figure 6 illustrates an example configuration file of where IME is mounted in the */tmp/esdm* directory. Last, we have verified that the integration of the IME native interface did not affect the results of the internal ESDM test results.



```
[kchasapis@dime-head01 netcdf-bench]$ cat esdm.conf
{"esdm":
        {
                "backends": [
                        {"type": "IME", "id": "tmp", "target": "/tmp/esdm/_esdm",
                                "performance-model" : {"latency" : 0.000001, "throughput" : 500.0},
                                "max-threads-per-node" : 4,
                                "max-fragment-size" : 104857600,
                                "max-global-threads" : 8,
                                "accessibility" : "global"
                        },
                        {"type": "IME", "id": "e2", "target": "/tmp/esdm/_esdm2",
                                "performance-model" : {"latency" : 0.000001, "throughput" : 500.0},
                                "max-threads-per-node" : 4,
                                "max-fragment-size" : 104857600,
                                "max-global-threads" : 8,
                                "accessibility" : "global"
                        }
                ],
                "metadata": {"type": "metadummy",
                                "id": "md",
                                "target": "/tmp/esdm/_metadummy",
                        "accessibility" : "global"}
        }
}
```

Figure 6: ESDM configuration with IME native data backend.

[2] https://github.com/kchasapis/esdm/tree/master/src/backends-data/ime

4.2.3 ESDM with IME data backend benchmarking

Evaluation methodology

To measure the performance improvement of the IME native integration with ESDM we leverage the “netcdf-bench” [Netcdf-Bench] and run tests in an internal DDN cluster. The netcdf-bench is a tool dedicated to measure the I/O performance of scientific workloads on large scale systems. One of the motivations of the development of netcdf-bench is to fill the gap between pure I/O benchmarks, which tend to be focused on the lower level of the software stack, and application coarse grain measurements, which tend to overlook I/O. This tool mimics the typical I/O behaviour of scientific climate applications and in combination with the support of the ESDM backend makes it relevant for performance testing. To leverage the ESDM backend with the netcdf-bench, one must link ESDM with the custom ESDM-NetCDF. For our experiments we perform all the steps needed to compile and link ESDM with ESDM-NetCDF as described in detail in the ESDM GitHub [ESDM].

The experimental parameter space of our testing is structured by 4 parameters of *netcdf-bench*:

- Type of I/O to be performed: Independent / Collective I/O,
- Chunk geometry,
- Block geometry,
- Interface to be selected for the low level I/O calls: IME Fuse (POSIX) or IME native.

By experimenting with different chunk/block geometries and the type of I/O, we can quantify the sensitivity of performance in relation to the low-level interface used. We focus our evaluation on the write path, since this is the most demanding scenario from a scientific applications perspective, specifically for those applications using check-pointing.

Although in our experiments we do not compare the performance of IME against traditional file systems, this comparison has been already published at [IME_PERF]. This publication utilised the IO500 [IO500] benchmark and clearly illustrates the benefits of IME in contrast to a traditional file system.

Experimental Results

Our experimental testbed consists of 10 client/compute nodes connected to 5 IME servers which themselves are connected to the backing parallel file system (BFS) as illustrated in Figure 7.

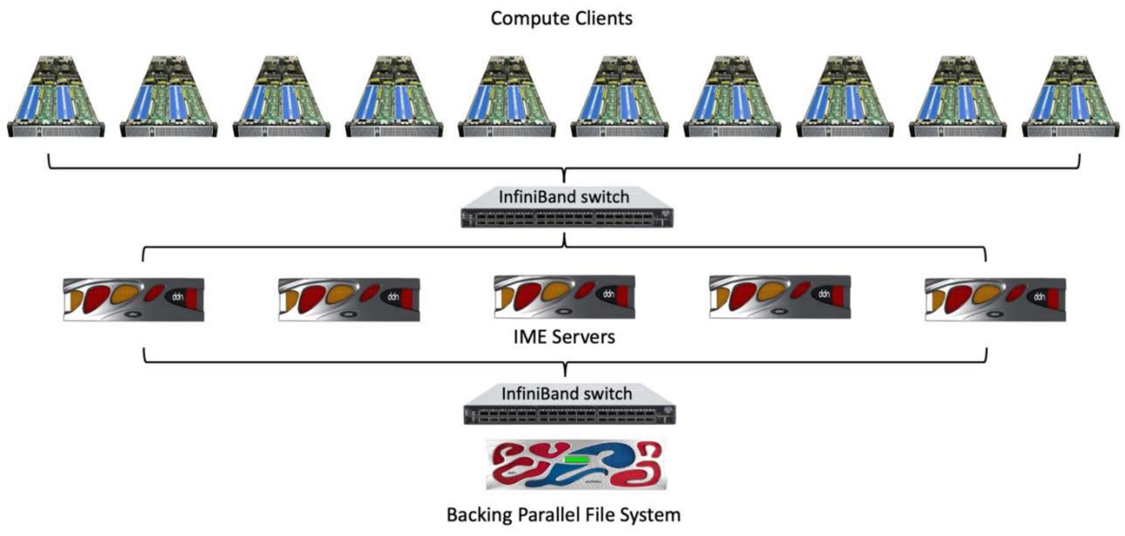


Figure 7: Benchmarking Testbed setup.

Each client node is equipped with 2 X Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz, 128 GB DDR4 RAM and a Mellanox Technologies MT27500 Family [ConnectX-3] InfiniBand card. As for the IME servers, we use the IME 240 machines, which have: 2 X Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, 128 GB RAM, 2x InfiniBand cards Mellanox Technologies MT27700 Family [ConnectX-4] and 21 Intel Corporation DC P3520 SSD. Each IME server has a data capacity of 25 TB. The 4 servers' configuration can host up to 100 TB of data on its flash devices. However, since the number of compute clients is a bit limited in respect of real-world systems, we have configured the IME cluster to utilise only one InfiniBand card on each server. This allows compute nodes to put more pressure on the server and measure the impact of saturation.

As for the backing parallel file system we use a single server with Lustre that is equipped with Mellanox Technologies MT27700 Family [ConnectX-4], 2 X Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz and 256 GB DDR4 RAM. For all the experiments the BFS is used only to serve the metadata requests and the IME servers' data inject performance is not impacted by the fact that a single node with limited bandwidth is used as a BFS.

For all the experiments we write a total of 327 GB per node, which is sufficient to fill caches and provides stable results across multiple runs and makes sure that each test result is reproduceable dropping caches between each run. All the performance experiments have been conducted in exclusive mode, i.e., to limit interference, no other users are allowed on the testbed, neither on IME servers nor on compute nodes.

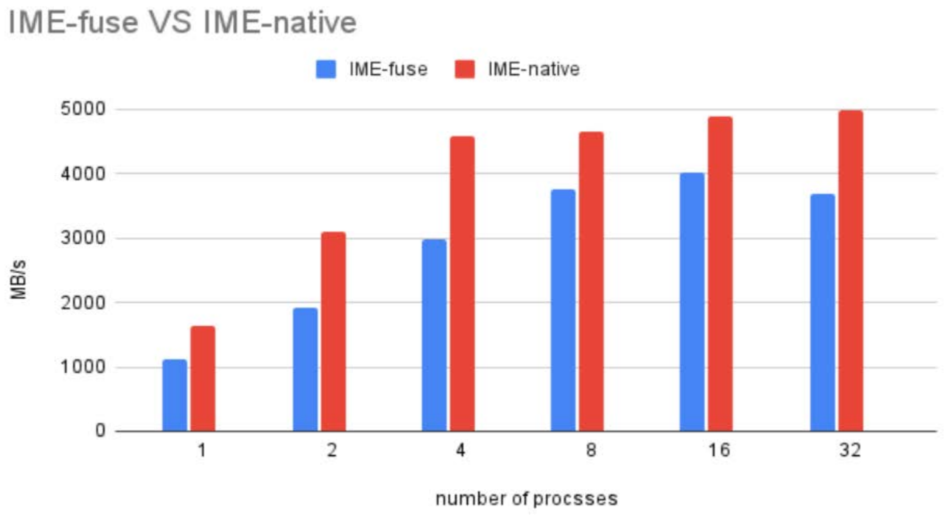


Figure 8: netcdf-bench reported write throughput varying the number of processes in a single client.

Figure 8 presents the aggregated write throughput reported by the netcdf-bench varying the number of processes (strong scaling) from 1 to 32 in a single client. The main takeaway from this figure is the performance gap between ime-fuse and ime-native. As we can see, IME native outperforms IME-fuse and this is aligned with the expectations, since fuse is introducing an additional lock, thus serialising accesses. In terms of actual performance gain, depending on the number of processes, improvement ranges from 1.2x to 1.6x. Furthermore, IME-native can achieve close to max performance with only 4 processes. We can also notice a slight performance reduction of IME-fuse when using more than 16 processes, which reflects the overhead of parallelism (lock overhead). Regarding IME-Native, performance still takes advantage of a higher number of threads, but gains remain marginal.

Figure 9 illustrates the aggregated write throughput reported by the netcdf-bench varying the number of client nodes (weak scaling). For this set up we use 16 processes per node. Like for the case above, IME-

native outperforms IME-fuse even in the case that we utilise more client nodes. The performance benefit of IME-native over IME-fuse is 1.2x to 1.3x.
Another set of tests that we did was with the type of I/O that the netcdf-bench will perform. The benchmark supports three different types of I/O: i) independent, ii) collective and iii) file-per-process. In a typical file system and without the use of an I/O library, the different I/O types would have resulted in different I/O performance outcomes. However, with the case of IME and ESDM the performance remainsidentical and independent on the type of I/O selected.

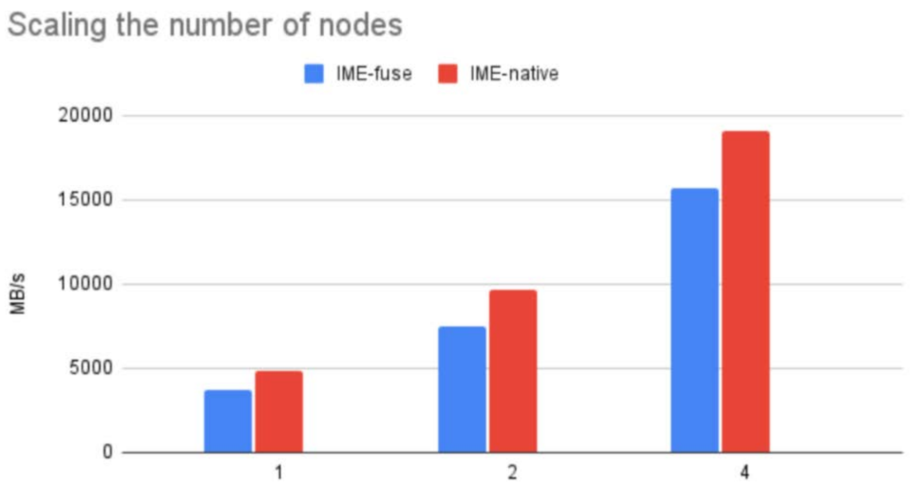


Figure 9: netcdf-bench reported aggregated write throughput varying the number of nodes with 16 processes per node.

Conclusions
From our evaluation we demonstrate the performance improvement of the IME-native interface over IME-fuse. Depending on the case we can reach a performance improvement from 1.2x up to 1.6x. Similar results have also been obtained for the integration of IME with a different scientific I/O library named SION I/O. The results of the integration have been published at [IME_SION].

4.2.4 Active Storage Introduction and background

Active Storage (AS), sometimes referred to as Computational Storage (CS), represents architectures in which the storage devices/platforms not only host data but can also perform computation on these hosted data. AS architectures can dramatically reduce data movements between storage plane and the compute plane of an IT system. The benefits of such data reduction are several including: i) reduce network utilisation ii) faster time to solution iii) cost efficiency especially in edge and cloud environments and iv) energy savings.

Within the scope of the project, we worked on an AS design for the use case of a supercomputer that utilises a parallel file system. In the following section we provide a detailed overview of the architecture that we propose.

4.2.5 Active Storage Architecture Overview

The emergence of Flash storage devices has dramatically changed the I/O path in supercomputers. The availability of a new technology with superior performance but higher cost has led to an increased complexity of storage architectures with multiple tiers. The goal of these new architectures is to extract the best performance from Flash while mitigating the cost. Nowadays, the storage system of a supercomputer usually includes several tiers. Tiers can be implemented in several ways, as a discrete resource hosted in

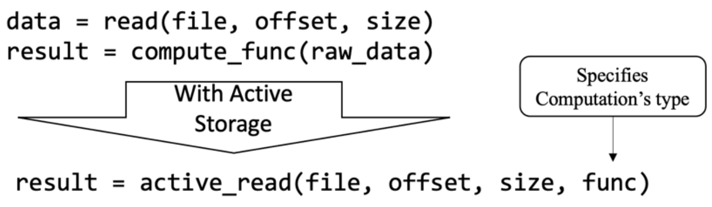
the compute nodes themselves (local storage) or as a disaggregated resource, i.e. a shared pool of resources, typically instantiated as a large burst buffer tier in between the compute nodes and the archive tier. Notice that both architectures can be combined with local storage and a disaggregated pool as well.
Designing an AS solution in such a complex storage platform is not a trivial task since several layers are involved in the I/O path. For our solution we view a storage server, a machine that holds an array of storage devices, as a storage unit that provides read/write requests and we aim to add an extra request type to perform a rather simple computation on the data requested and return the result of the computation rather than the actual data. After a thorough examination we decided to apply the AS at the file system (FS) level.

There are two main advantages of an AS implementation at the FS layer. Firstly, at the file system layer we have the semantic information in respect of data distribution across storage servers. Therefore, we can point to the exact storage servers that hold the data we are interested in. Secondly, I/O libraries that are implemented on top of the file system can integrate AS without any additional effort. We expect that the AS will be transparent to the end user and that the I/O libraries will handle the additional steps internally. Such integration is crucial for the success of the AS. Otherwise, each application would have to manually integrate AS hampering its acceptance by the scientific community. ESDM is an ideal candidate for such an I/O library.

To implement AS at the FS layer, we utilise the input/output control (IOCTL) [IOCTL] mechanism. The IOCTL is a system call that allows device specific I/O operations that are not able to be expressed by the regular system calls. As a parameter it takes the file descriptor (the file must be already opened previously by a regular open system call), the command code to execute and then a variable number of arguments depending on the needs of each command. For our use case, the IOCTL that are implemented in addition to the command code and a pointer to a buffer where the result will be stored get the same arguments as a read system call.

The type of computations that would be possible to execute in an AS should be lightweight (with a low arithmetic intensity) to prevent saturation of the storage servers CPU, which could harm the performance of the file system service. Targeting to support a subset of computations offered with MPI reduce [MPI_REDUCE] are examples of such operations. The calculation of the min/max/sum over some values is another example of light weight operations. Such calculations can be heavily utilised by the libraries/tools that operate on high level data formats that contain semantic information and a description of the actual data, such as ESDM.

Figure 10 demonstrates an example of a pseudo-code with and without AS that would appear in an I/O library. As we can see, without the use of the AS one must first issue the read request to fetch the data from the storage servers to the client and then issue the compute function call. With the AS we merge these two requests into one that is polymorphic and has as a parameter the type of computation to be executed.



```
data = read(file, offset, size)
result = compute_func(raw_data)
```

```
result = active_read(file, offset, size, func)
```

Figure 10: I/O calls pseudo-code with and without Active Storage

The complete architecture of the I/O path from our AS solution is presented in Figure 11. As can be seen, there are two major components for the AS implementation: i) the client side and ii) the storage server side.

On the client side we have the application connecting with the I/O library that will have the AS integrated. The I/O library then will issue the AS call to the file system client (in the example illustrated, it involves a fuse implementation of a client-side file system that is the more complicated case in terms of the I/O path). The file system client has the following tasks to serve and AS request: i) receive the AS request from the I/O library directly or from the fuse client, ii) identify the storage servers that hold the pieces of data that the request is for, iii) propagates the call to the storage servers, iv) retrieve the partial result from each storage server, v) aggregate the partial result from each server and vi) return the final result to the I/O library.

On the storage server side, the following tasks are executed to serve an AS request. Firstly, the server receives the AS request from the FS Client. Then it gathers file data belonging to the requested region. In case that the actual data are not present in the high-performance storage it will send a read request to the high-capacity storage servers to fetch all data. Once all data have been collected it executes the function indicated by the AC request on the data. As a final step it submits the generated result back to the FS Client.

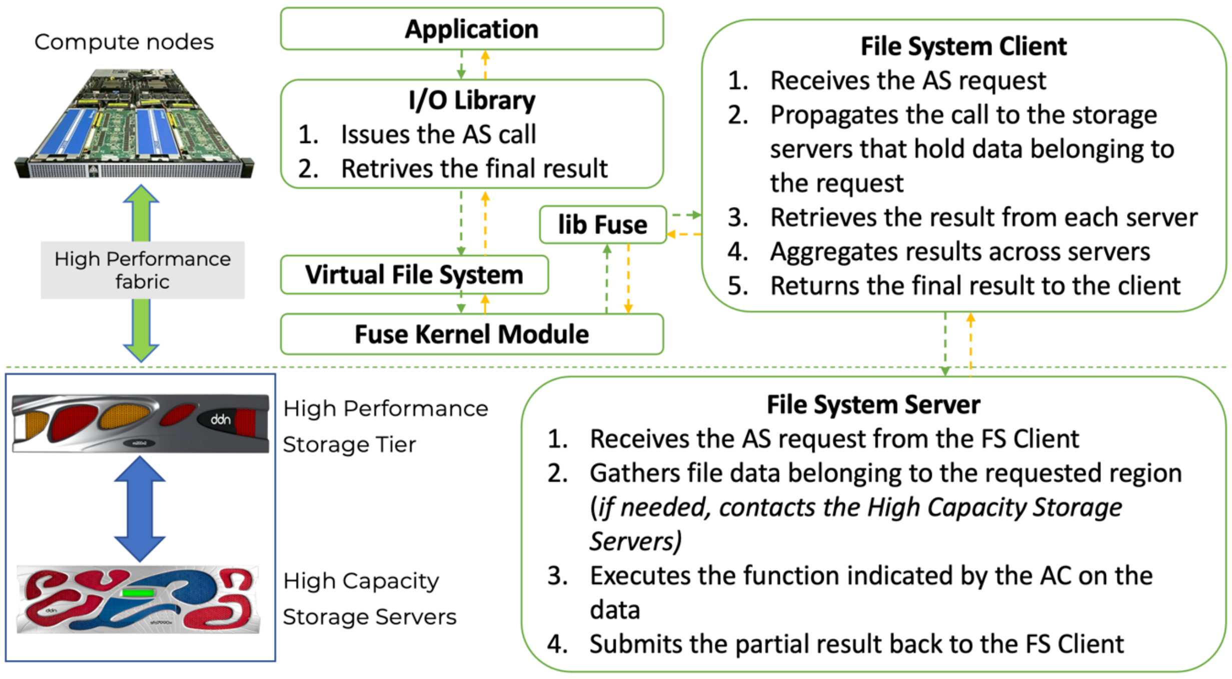

Compute nodes
High Performance fabric
Application
I/O Library
1. Issues the AS call
2. Retrives the final result
lib Fuse
Virtual File System
Fuse Kernel Module
File System Client
1. Receives the AS request
2. Propagates the call to the storage servers that hold data belonging to the request
3. Retrieves the result from each server
4. Aggregates results across servers
5. Returns the final result to the client
High Performance Storage Tier
High Capacity Storage Servers
File System Server
1. Receives the AS request from the FS Client
2. Gathers file data belonging to the requested region (if needed, contacts the High Capacity Storage Servers)
3. Executes the function indicated by the AC on the data
4. Submits the partial result back to the FS Client

Figure 11: I/O PATH from the Application to the Storage Servers with Active Storage.

5. References

[ESiWACE2 D4.1] ESiWACE2 D4.1. (2021). *Advanced softwarestack of Earth system data - Deliverable D4.1.* On *Zenodo* https://zenodo.org/record/4651493.

[cortx apps] https://github.com/Seagate/cortx-motr-apps.git, Accessed 30/08/2022

[libfabric] https://ofiwg.github.io/libfabric/, Accessed 30/08/2022

[lnet] https://wiki.lustre.org/Lustre_Networking_(LNET)_Overview, Accessed 30/08/2022

[Motr] https://github.com/Seagate/cortx-motr, Accessed 30/08/2022

[Spack] https://github.com/spack/spack, Accessed 30/08/2022

[IME] https://www.ddn.com/products/ime-flash-native-data-cache/, Accessed 30/08/2022

[IME_NATIVE] https://github.com/DDNStorage/ime_native, Accessed 30/08/2022

[ESDM] https://github.com/kchasapis/esdm, Accessed 30/08/2022

[IO500] https://io500.org, Accessed 30/08/2022

[IOCTL] https://en.wikipedia.org/wiki/Ioctl, Accessed 30/08/2022

[Netcdf-Bench] https://github.com/joobog/netcdf-bench, Accessed 30/08/2022

[IME_PERF] Konstantinos Chasapis, Jean-Yves Vet, and Jean-Thomas Acquaviva (2019). *Benchmarking Parallel File System Sensitiveness to I/O patterns*. In IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, 427–428.

[IME_SION] Konstantinos Chasapis, Jean-Thomas Acquaviva, and Sebastian Lührs (2021). *Integration of Parallel I/O Library and Flash Native Ac- celerators: an Evaluation of SIONlib with IME*. In Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS ’21), April 26, 2021, Online, United Kingdom. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3439839.3458736

[MPI_REDUCE] https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/, Accessed 30/08/2022

6. Changes made and/or difficulties encountered, if any

The main change from Seagate’s side was to show the deployment of CORTX Motr on commodity hardware at a Weather and Climate centre (DKRZ) since Mero was fully open sourced as CORTX Motr. The original plan had been to provide an appliance based on Mero – considering that Mero was a closed source product. This plan was changed after CORTX Motr had been made open source as discussed earlier. It was decided to implement a user mode version of CORTX Motr in a VM, which can be easily installed on commodity hardware.

The original cortx-motr and its support packages such as cortx-hare/cortx-utils etc are written to be installed and operate system wide. All the required directories/binaries/logs etc used to be installed/created in /root space. There used to be single motr session on a system to be operated by an authorised user. Systemctl commands used to create motr sessions are not available to a normal user (before Centos 8) since there used to be single system wide instance of Motr on sudo behalf and most of the directories/binaries/logs location were hardcoded in Motr/Hare code.

So the major challenges encountered were:

1) Modification of the code to install all the required rpms in the user’s home directory instead of /root space. This has been resolved by extracting all the required rpms in the user’s home directory, and by performing the required environment setup to make it runnable/accessible by the respective user. Mainly cortx-motr, cortx-hare, cortx-utils and isa-l rpms are used for this.

2) Modification of the code to make it runnable per user instead of only for a single root user, so that individual users can run a motr session from their home directory location. It required larger scale code changes to be made in cortx-motr as well as its support packages like cortx-hare to make it runnable from the user's home directory independent of each other (cortx-motr and cortx-hare)

3) Find a way to run systemctl/motr-services on non-sudo user behalf so that different users can have their own independent motr sessions. It required code changes to be made to run systemctl services in user mode and use Centos 8 with updated systemd support, which allows running systemctl services on non-sudo user behalf as well.

4) Once RPMS have been installed in a user’s home directory, make the required code changes so that all the required binaries, as well as the required libraries, are searched and found at runtime, as by default in the search is performed in system standard locations. Libmotr and libisal are major libraries that required code changes in cortx-hare to pick it from the respective home directory at runtime.

Updated version of motr obtained after overcoming these challenges does not require any sudo privilege for any motr cluster operations. On a single system, different users are expected to run and control their respective motr cluster session independently without interfering with each other. All the required code is installed/extracted in the seagate subfolder of the respective user's home directory. Also, any logs or other runtime-generated data are kept in user-specific directories to avoid any interference.

On the side of DDN the main challenge was related to the integration of IME within a new I/O library. While this had been done previously on another scientific I/O library (SionLib), entering into the code of another library is always technically committing. The validation of the code changes both from a functional and performance standpoint is also an important element that requires a considerable engineering effort.The most challenging part has been to set-up dedicated hardware resources in one of DDN's lab, furthermore as we are targeting large systems the testbed had to be of meaningful size. This means not only aggregating the needed computing and storage resources but also the set-up and maintenance of the testbed (including a back-end file system with a Lustre deployment).This effort was required because IME servers are instantiated as appliances and cannot be easily shipped and installed on a partner cluster. Inserting storage servers in an existing environment would have required root access and non-negligible security and administrative hurdles. A lightweight validation on virtual machines or ad-hoc servers (white boxes) would have been possible but at the cost of limiting the validation to functional validation and not performance validation. Thus, we believe that with the help of netcdf-bench and the dedicated testbed in the DDN lab we have offered the best validation process, even if this has required a large devops effort.We do believe that future actions or projects should take into account these aspects and integrate a dedicated budget line for in-house hosting of appliances and the corresponding process for remote root access at a scientific partner site.

7. How this deliverable contributes to the European strategies for HPC

This deliverable is very much aligned to the EuroHPC Joint Undertaking's goals and missions as adopted by the European Council in July 2021[3] to develop world class HPC capabilities in Europe. We provide below the stated goals and show the alignment.

a) further develop, deploy, extend and maintain in the EU a world-class supercomputing and data infrastructure, driven by key scientific, industrial and social applications;
b) develop and deploy a quantum computing and quantum simulation infrastructure
c) reach the next frontier of high-performance computing by acquiring the first exascale supercomputers. These supercomputers are capable of more than a billion billion operations per second (when compared to ten billion operations per second of an ordinary laptop device);
d) federate European supercomputing and quantum computing resources and make them accessible to a wide range of public and private users everywhere in Europe, including for the European public data spaces, as presented in the 2020 European Data Strategy;
e) provide secure cloud-based supercomputing services for a wide range of public and private users everywhere in Europe;
f) support the development of innovative supercomputing technologies and applications to underpin a world-class European HPC ecosystem;

[3] https://digital-strategy.ec.europa.eu/en/policies/high-performance-computing-joint-undertaking#:~:text=EuroHPC%20JU%20mission,top%205%20in%20the%20world.

g) develop greener computing, and exploit the synergies of HPC with artificial intelligence, big data and cloud technologies;
h) extend and widen the use of supercomputing to a wide range of scientific and industrial users, for instance by helping SMEs develop innovative business cases using supercomputers and providing them with training opportunities and the critical HPC skills they need via National HPC Competence Centres;
i) deploy Centres of Excellence in HPC applications and the industrialisation of HPC software, with novel algorithms, codes and tools optimised for future generations of supercomputers;
j) put in place large-scale industrial pilot test-beds and platforms for HPC and data applications and services in key industrial sectors;

This deliverable has contributed towards maintaining a world class data infrastructure through innovate storage solutions (a). The innovative storage technologies that are being assessed (from DDN and Seagate) contribute to (f). Also, we hope that the appliances and technologies provided in this deliverable will support the development of larger scale industrial test beds leveraging the storage technologies provided here, and hence the deliverable contributes to (j). Technologies from DDN and Seagate also provide APIs to support data ingest from the cloud as part of large scientific workflows. This also helps to provision these storage resources as part of larger cloud federations that are very much applicable to initiatives such as EOSC (European Open Science Cloud)[4] and potentially the HPC side of GAIA-X[5]. Hence it contributes to (e) and (f). Technologies developed in ESiWACE2 such as CORTX Motr are used in EuroHPC projects such as IO-SEA[6], helping to position Europe towards Exascale – especially in the area of storage and I/O. Hence the work contributes to (c). The design effort for Active Storage is directly linked to a reduced energy footprint for scientific data management and addresses (g).

8. Sustainability

The work in this deliverable is very synergistic to the IO-SEA project, in which CORTX Motr is being taken forward to work with other applications and use cases through novel usage scenarios. Lessons learnt from the Weather and Climate use cases are hence more broadly applied in IO-SEA. The storage system based on CORTX Motr will also be provided to RED-SEA[7] and DEEP-SEA[8], which are part of the SEA projects.

A key aspect of the work performed in ESiWACE is the importance of I/O libraries. The ability to decouple applications from the lower-level of the storage stack without performance impact is an important step toward sustainability. Scientific applications are made more maintainable, and similarly storage frameworks can evolve as long as the API with the I/O library is enforced. This shifts the focus toward careful API definition and standardisation. The importance of decoupled software architectures and API design has been addressed thoroughly in the ADMIRE EuroHPC project. On this aspect ESiWACE has been instrumental for DDN. The design effort made for Active Storage is also part of a long-term R&D effort for DDN regarding future storage product and their articulation with scientific applications.

This work is also highly synergistic and indeed a continuation of the work in D4.1 of ESiWACE2.

Seagate has learnt that Object stores are indeed viable for HPC use cases in the weather and climate realm and also to other use cases in HPC. However, one the challenges to note is that the CORTX Motr API to work with Motr Object store is very low level. This needs to be exposed through higher level interfaces (such as ESDM and S3) to make it more suitable.

DDN has learnt a lot about weather and climate applications. DDN had the opportunity to invest in netcdf and ESDM, which are respectively established and emerging standards for I/O management. Furthermore,

[4] https://eosc-portal.eu
[5] https://gaia-x.eu
[6] https://iosea-project.eu/
[7] https://redsea-project.eu/
[8] https://www.deep-projects.eu/

and this should be noted, the project has been an opportunity to collaborate with other industrial partners (i.e. Seagate), which we believe is important and fruitful for the community as a whole.

9. Dissemination, Engagement and Uptake of Results

a. Target audience

As indicated in the Description of the Action, the audience for this deliverable is:

X	The general public (PU)
	The project partners, including the Commission services (PP)
	A group specified by the consortium, including the Commission services (RE)
	This report is confidential, only for members of the consortium, including the Commission services (CO)

b. Record of dissemination/engagement activities linked to this deliverable

Type of dissemination and communication activities	Details	Date and location of the event	Type of audience	Zenodo Link	Estimated number of persons reached
Dissemination of CORTX Motr+ESDM work by Seagate - PLANNED	CORTX Meet an Architect Event	October 2022	Storage technologists and administrators	TBD	50
Discussion at Emerging Open Storage Systems and Solutions Workshop'23 (Organised through IO-SEA Project) - PLANNED	EMOSS'23 Workshop	June 2023	Storage Technologies and administrators, HPC application experts,	TBD	50

c. Publications in preparation OR submitted

In preparation OR submitted?	Title	All authors	Title of the periodical or the series	Is/Will ***open access*** be provided to this publication?
In Preparation	Earth Systems Data Middleware with CORTX Motr	Ganesan Umanesan, Sai Narasimhamurthy	White Paper (In preparation) – initially disseminated through CORTX Open-Source channels	Yes

d. Intellectual property rights resulting from this deliverable

Seagate – None
DDN – None. All codes produced are fully open source.