H2020-JTI-EuroHPC-2019-1

Project no. 956748

ADAPTIVE MULTI-TIER INTELLIGENT DATA MANAGER FOR EXASCALE

D2.3
Short-lived ad-hoc storage systems

Version 1.0

Date: March 31, 2023

Type: Deliverable
WP number: WP2

Editor: Ramon Nou
Institution: BSC

Project co-funded by the European Union Horizon 2020 JTI-EuroHPC research and innovation programme and Spain, Germany, France, Italy, Poland, and Sweden		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Change Log

Rev.	Date	Who	Site	What
1	02/02/23	Ramon Nou	BSC	Document Structure.
2	17/02/23	Ramon Nou	BSC	Updated changelog for GekkoFS.
3	23/02/23	Ramon Nou	BSC	Added Spack instructions for GekkoFS.
4	20/02/23	Marc-André Vef	JGU	Modified introduction
5	21/03/23	Javier Garcia-Blas	UC3M	Added Spack instructions for Hercules.
6	21/03/23	Marc-André Vef	JGU	Modified main ad-hoc storage system section
7	21/03/23	Marc-André Vef	JGU	Modified GekkoFS Updates and Spack sections, and added the ADMIRE integration section
8	22/03/23	Marc-André Vef	JGU	Added GekkoFS integration and malleability sections
9	22/03/23	Marc-André Vef	JGU	Added activities chapter and MetaWBC section
10	22/03/23	Marc-André Vef	JGU	Added GekkoFS application integration
11	23/03/23	Felix Garcia-Carballeira	UC3M	Added Expand related information
12	23/03/23	Alejandro Calderon-Mateos	UC3M	Added Expand related information
13	23/03/23	Diego Camarmas-Alonso	UC3M	Added Expand related information
14	24/03/23	Marc-André Vef	JGU	Document revisions
15	26/03/23	Nafiseh Moti	JGU	Added I/O Tracing initiative
16	24/03/23	Marc-André Vef	JGU	Further document revisions
17	27/03/23	Hamid Fard	TUD	Review of deliverable
18	27/03/23	Adalberto Perez	KTH	Review of deliverable
19	30/03/23	Marc-André Vef	JGU	Final document revision

Executive Summary

Ad-hoc storage systems represent a dynamic component within the ADMIRE project to accelerate I/O performance of scientific applications and significantly reduce I/O traffic to the general-purpose storage backends, i.e., shared *parallel file systems*, of *High-Performance Computing* (HPC) clusters. Ad-hoc storage systems are often ephemeral and live within a particular application context (e.g., an HPC compute job), usually using node-local storage devices (e.g., SSDs). To achieve better performance in HPC systems within the ADMIRE project, ad-hoc storage systems are integrated and controlled by the *I/O scheduler* (respecting the malleable decisions taken by the *malleability manager*). Moreover, ad-hoc storage systems should be malleable themselves and support longer-running workflows, e.g., *job campaigns*. Overall, malleability allows such storage systems to optimize the I/O behavior of a particular application or the HPC system.

In Deliverable 2.1, we provided a detailed analysis of the ad-hoc storage systems, their interfaces to the ADMIRE ecosystem, and an initial analysis of the ADMIRE applications. In Deliverable 2.2, we focused on the deployment of each ad-hoc-storage system. This deliverable will present the up-to-date updates from all the ad-hoc storage systems, our actions towards KPIs 4, 5, and 6, and details on further activities.

Contents

1 Introduction

Work package 2 considers several ad-hoc storage systems to efficiently use node-local storage technologies, e.g., SSDs or future *non-volatile memories* (NVMs), to reduce the pressure on the main backend storage systems used in HPC systems, e.g., Lustre or GPFS.

In general, WP2 focuses on three main tasks:

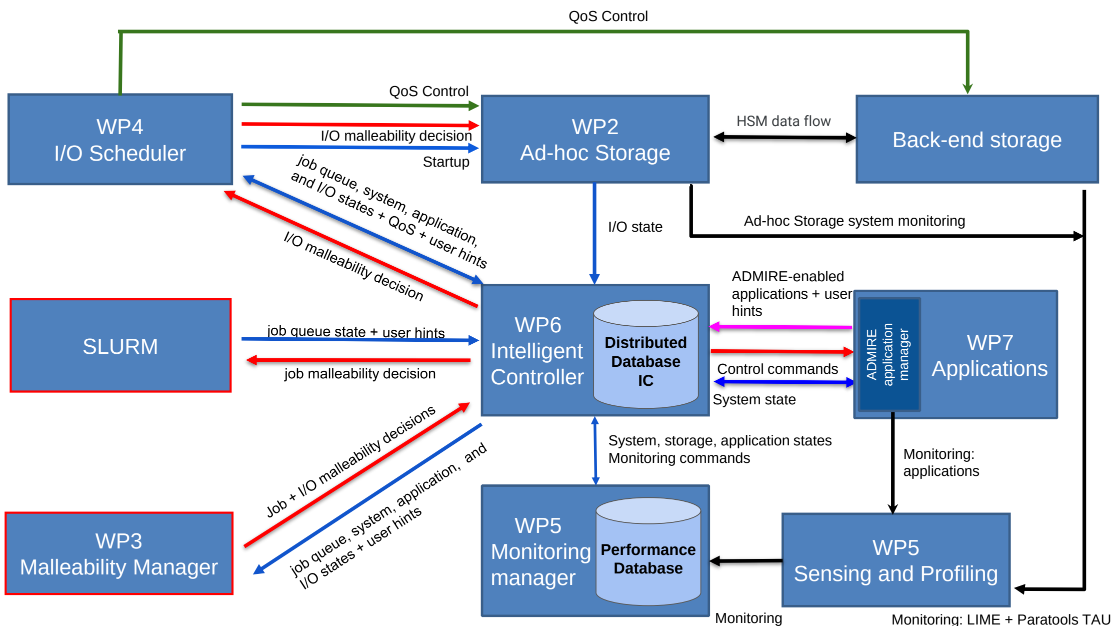
1. The use case applications in ADMIRE are analysed to understand how ad-hoc storage systems should be modified so that applications can benefit from a malleable and dynamic storage system that can react to specific I/O requirements during runtime (see D2.1 for details).

2. The ad-hoc storage systems are further developed and integrated into the ADMIRE ecosystem adding support for fast storage technologies and new data placement algorithms that fit how an application accesses its data, e.g., local data placement but global metadata placement.

3. Resilience mechanisms are added to the ad-hoc storage systems to run for longer periods, such as in application workflows consisting of multiple consecutive compute jobs that can use the same shared namespace without moving data.

Figure 1.1 presents an overview of the ADMIRE architecture with the ad-hoc storage system being connected to the following components:

1. the *I/O scheduler* (WP4), which controls and manages the ad-hoc storage system instances, including the data movement between the parallel file system and the ad-hoc storage system;



QoS Control
WP4
I/O Scheduler
QoS Control
I/O malleability decision
Startup
WP2
Ad-hoc Storage
HSM data flow
Back-end storage
job queue, system, application, and I/O states + QoS + user hints
I/O malleability decision
I/O state
Ad-hoc Storage system monitoring
SLURM
job queue state + user hints
job malleability decision
WP6
Intelligent
Controller
Distributed
Database
IC
ADMIRE-enabled applications + user hints
Control commands
System state
ADMIRE application manager
WP7
Applications
Job + I/O malleability decisions
job queue, system, application, and I/O states + user hints
System, storage, application states
Monitoring commands
Monitoring: applications
WP3
Malleability Manager
WP5
Monitoring
manager
Performance
Database
WP5
Sensing and Profiling
Monitoring
Monitoring: LIME + Paratools TAU

Figure 1.1: ADMIRE architecture overview. Each component developed in the project's scope has included its related work package (WP) label.

2. the *sensing and profiling* module (WP5), which receives the current ad-hoc storage system state and performance counters;

3. the *intelligent controller* (WP6), which receives malleability confirmations after forwarding malleability decisions from the *malleability manager* (WP3) to the ad-hoc storage system via the I/O scheduler;

4. the *backend storage system*, e.g., Lustre, in which an ad-hoc storage system is used within the *hierarchical storage management* (HSM).

This prototype deliverable is organized as follows. First, Chapter 2 presents the updates and extensions, since D2.2 developed in the ADMIRE project on all ad-hoc storage systems, including *GekkoFS*, *dataClay*, *Expand*, and *Hercules IMSS*. This chapter will also discuss the integration and malleability of ad-hoc storage systems. Chapter 3 provides new information on the application usage and benefits when used with ad-hoc storage systems. Further, this chapter discusses ADMIRE collaboration's with the IO-SEA project in this context. Next, Chapter 4 presents the current state of other activities: the I/O tracing initiative across several EuroHPC projects and MetaWBC, a recently published paper targeting metadata improvement in distributed file systems. Finally, Chapter 5 concludes this deliverable.

2 Ad-hoc storage systems

This section will discuss all ad-hoc storage systems that are used in the ADMIRE project and their software updates since the previous deliverable (D3.2). Refer to Deliverable 2.2 for detailed information on each ad-hoc storage system's design, use cases, deployment, and usage information. In the following, we will discuss the new features of each ad-hoc storage system as well as in the context of ADMIRE integration and I/O malleability.

As part of the integration efforts with other ADMIRE components, all project partners (including other work packages) agreed to make their software available via the Spack package manager. This allows all partners to easily install the corresponding software components on the dedicated cluster launched at the University of Torino to test and integrate all ADMIRE software components. Therefore, we outline the necessary steps to install the ad-hoc storage system with Spack for each ad-hoc storage system.

Finally, we will discuss the malleability support of each ad-hoc storage system. In particular, we have focused on the main malleable mechanism to control I/O bandwidth within a compute job: expanding and shrinking ad-hoc storage systems while the application is running. Hence, while the ADMIRE applications are becoming flexible, e.g., by using FlexMPI (as part of WP3), the ad-hoc storage system can also benefit from expanding or shrinking a job. We will also discuss other topics in the realm of malleability and the plans for the remaining year.

2.1 GekkoFS

GekkoFS [8, 10, 11] is a highly scalable distributed file system for HPC clusters that runs entirely in user space. GekkoFS can aggregate the local I/O capacity and performance of compute nodes to produce a high-performance storage space for applications. Using GekkoFS, HPC applications can run isolated from each other regarding I/O, which reduces interferences and improves performance. Further, GekkoFS has been designed with configurability in mind and allows users to fine-tune several of the default POSIX file system semantics, e.g., support for symbolic links, strict bookkeeping of file access timestamps and other metadata, or even modifying entire file system protocols. The overall design of GekkoFS takes previous studies on the behaviour of HPC applications into account [4] to optimise the most used file system operations. Further information is available in Deliverables 2.1 and 2.2.

GekkoFS is open-source and available at `https://storage.bsc.es/gitlab/hpc/gekkofs` and a documentation wiki can be found at `https://storage.bsc.es/projects/gekkofs/documentation`. The wiki is continuously extended with advanced information, such as file system architecture, consistency information, installation and running details, code reference based on Doxygen, and more.

2.1.1 Updates

The following list includes the updates and progress since the last deliverable in April 2022 for GekkoFS version v0.9.2 which is close to release. The current release candidate for v0.9.2 is currently in testing and awaits final updates. To install the release candidate, clone the latest GekkoFS version 0.9.2-rc1[1]:

[1]The current deliverable presents the current state of the release candidate of GekkoFS on the 30th of March of 2023 with its corresponding tag *v0.9.2-rc1* which can be directly accessed at `https://storage.bsc.es/gitlab/hpc/gekkofs/-/tags/v0.9.2_rc1`.

```
git clone --recurse-submodules --branch v0.9.2_rc1
↪  https://storage.bsc.es/gitlab/hpc/gekkofs.git
```

New

1. Added additional tests to increase code coverage
2. `GKFS_ENABLE_UNUSED_FUNCTIONS` added to disable code to increase code coverage
3. Updated Parallax version to new API (parallax option needs `kv_format.parallax` in the path and the database in a device with `O_DIRECT`)
4. Supported for increasing file size via `truncate()` added
5. Added PowerPC support
6. `GKFS_RENAME_SUPPORT` added to support renaming files. This specifically targets the use case for opened files using an existing file descriptor
7. Added `FLOCK` and `fcntl` functions for locks to interception albeit not supported by GekkoFS and returning the corresponding error code
8. Added ARM64 support
9. Added support for CMake presets to simplify build configurations
10. Added (parallel) append support for consecutive writes with file descriptor opened with O_APPEND
11. Added support for Spack so that it can be used to install GekkoFS
12. Dependency management is now handled more consistently: system dependencies are found using `find_package()`, whereas source-only dependencies are found using `include_from_source()`. This new function integrates a dependency provided its source code is available at `{PROJECT_ROOT}/external`. If it's not, it will try to automatically download it from its git repository using CMake's `FetchContent()`.
13. More consistent use of targets (we are closer to 100% modern CMake).
14. Added the `gkfs_feature_summary()` to allow printing a summary of all GekkoFS configuration options and their values. This should help users when building to precisely see how a GekkoFS instance has been configured.

Changed

1. Supported parallelism for path resolution tests
2. Supported parallelism for symlink tests
3. Updated Parallax release (PARALLAX-exp)
4. Improved and simplified coverage generation procedures for developers with specific CMake targets

Fixed

1. Updated daemon log level for tests
2. Using unlink now fails if it is a directory unless the AT_REMOVEDIR flag is used (POSIX compliance)
3. fchdir generate a SIGSEV in debug mode (due to log)
4. Supported glibc-2.34 or newer with syscall_intercept

5. Fixed segfault in sfind/gfind
6. Fixed fstatat to be able to understand AT_EMPTY_PATH flag used in coreutils (cat ...)
7. Updated CI pipeline for Gitlab version 15.1.1
8. Fixed a compilation error for testing
9. Fixed a bug which caused GekkoFS tests to fail
10. Fixed RocksDB compilation for future GCC versions
11. Fixed an issue where GekkoFS would halt when applications would create child processes, e.g., with fork, clone, pthread_create
12. Fixed an issue where compilation of syscall_intercept would fail for newer kernels using the clone3() system call
13. Updated code formatter to support the most recent clang-format-15 version
14. Refactored update file size during write operations

Outstanding issues will be part of the full v0.9.2 release.

2.1.2 Spack installation

The installation of GekkoFS with Spack is outlined in GekkoFS's readme and wiki pages. We confirm that it has been and can be installed at the Torino cluster by any user. The following bash commands outline the necessary steps (and the corresponding output) to install GekkoFS:

1. If not available, Spack needs to be installed first:

```
git clone https://github.com/spack/spack.git
. spack/share/spack/setup-env.sh
```

2. GekkoFS needs to be added to the Spack's repository

```
spack repo add gekkofs/scripts/spack
```

3. (optional) check the GekkoFS package and its options:

```
$ spack info gekkofs
CMakePackage:   gekkofs

Description:
    GekkoFS is a distributed burst buffer file system in user space. It is
    capable of aggregating the local I/O capacity and performance of each
    compute node in a HPC cluster to produce a high-performance storage
    space that can be accessed in a distributed manner. This storage space
    allows HPC applications and simulations to run in isolation from each
    other with regards to I/O, which reduces interferences and improves
    performance.

Homepage: https://storage.bsc.es/gitlab/hpc/gekkofs

Preferred version:
    0.9.1     https://storage.bsc.es/projects/gekkofs/releases/gekkofs-v0.9.1.tar.gz

Safe versions:
    0.9.1     https://storage.bsc.es/projects/gekkofs/releases/gekkofs-v0.9.1.tar.gz
    latest    [git] https://storage.bsc.es/gitlab/hpc/gekkofs.git on branch master

Deprecated versions:
    0.9.0     https://storage.bsc.es/projects/gekkofs/releases/gekkofs-v0.9.0.tar.gz

Variants:
Name [Default]            Allowed values          Description
======================== ====================    ==============================

agios [off]               on, off                 Enables the AGIOS scheduler
                                                  for the forwarding mode.
build_system [cmake]      cmake                   Build systems supported
                                                  by the package
build_type [Release]      Debug, Release,         CMake build type
                          RelWithDebInfo
compile [x86]             x86, powerpc, arm       Architecture to compile
                                                  syscall intercept.
dedicated_psm2 [off]      on, off                 Use dedicated _non-system_
                                                  opa-psm2 version 11.2.185.
forwarding [off]          on, off                 Enables the GekkoFS I/O
                                                  forwarding mode.
guided_distributor [off] on, off                  Enables the guided distributor.
ipo [off]                 on, off                 CMake interprocedural optimization

parallax [off]            on, off                 Enables Parallax key-value database.
prometheus [off]          on, off                 Enables Prometheus support
                                                  for statistics.
rename [off]              on, off                 Enables experimental
                                                       rename support.

Build Dependencies:
    agios  argobots  cmake  date  libfabric  lz4  mercury  mochi-margo
    ninja  opa-psm2 parallax  prometheus-cpp  python  rocksdb  syscall-intercept

Link Dependencies:
    agios  argobots  date  libfabric  lz4  mercury  mochi-margo  opa-psm2
    parallax  prometheus-cpp python  rocksdb  syscall-intercept

Run Dependencies:
    None
```

4. GekkoFS can be installed. GekkoFS supports multiple configurations and options (including several CPU architectures: x86_64, Powerpc, and ARM64), which can be added to the installation.

```
spack install gekkofs
# for installing tests dependencies and running tests
spack install -v --test=root gekkofs
```

5. The GekkoFS module can be loaded:

```
spack load gekkofs
```

6. Run GekkoFS as described in previous deliverables via its launch script that uses the Slurm environment as part of the API between WP4 and WP2. Alternatively, it can be started manually, as shown below. Note `$GKFS_CLIENT` is set when loading GekkoFS.

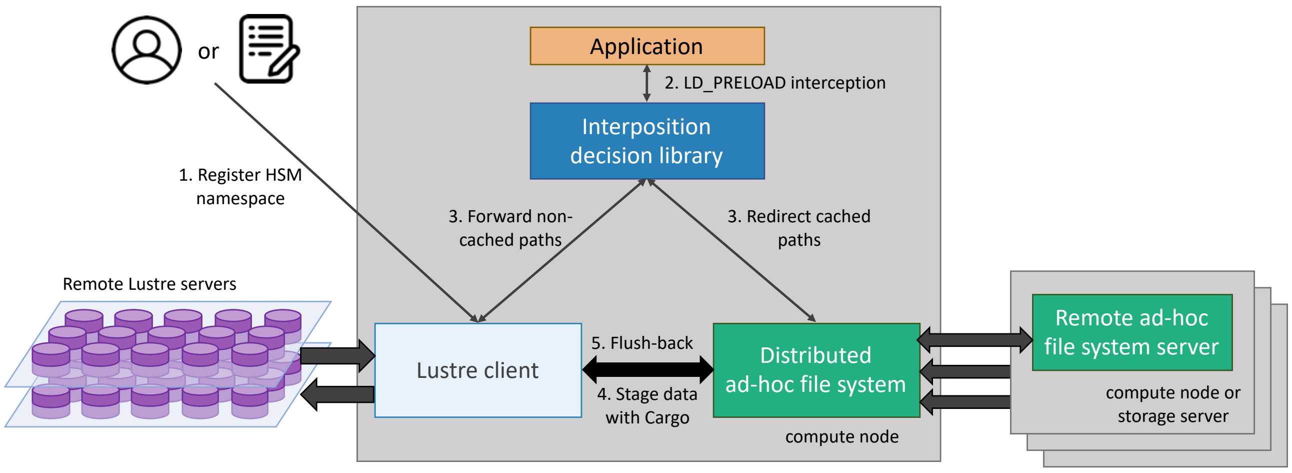

or
1. Register HSM namespace
Application
2. LD_PRELOAD interception
Interposition decision library
3. Forward non-cached paths
3. Redirect cached paths
Remote Lustre servers
Lustre client
5. Flush-back
4. Stage data with Cargo
Distributed ad-hoc file system
compute node
Remote ad-hoc file system server
compute node or storage server

Figure 2.1: Proof-of-concept of transparently coupling ad-hoc file systems with Lustre.

```
gkfs_daemon -r /tmp/gkfs_rootdir -m /tmp/gkfs_mountdir &
LD_PRELOAD=$GKFS_CLIENT ls -l /tmp/gkfs_mountdir
LD_PRELOAD=$GKFS_CLIENT touch /tmp/gkfs_mountdir/foo
LD_PRELOAD=$GKFS_CLIENT ls -l /tmp/gkfs_mountdir
```

7. Unload GekkoFS when done.

```
spack unload gekkofs
```

2.1.3 ADMIRE integration

In the context of GekkoFS's integration into the ADMIRE software stack, GekkoFS offers Spack integration (see above) as well as a statistics module to communicate with WP5 (see Deliverable 2.2). Further, GekkoFS can be controlled by the I/O scheduler (WP4) with GekkoFS's startup API, as demonstrated in the video demonstration at the midterm review[2]. In collaboration with the I/O scheduler, we are currently actively working on integrating with the I/O scheduler *Scord* (WP4) and the data scheduler *Cargo* (WP4) used for data movement between the parallel file system and GekkoFS, so that the application can access its data when needed.

Moreover, we are currently integrating GekkoFS transparently with the Lustre parallel file system using its Hierarchical Storage Management (HSM) mechanism. This effort is in collaboration with the partner DDN and was presented at *Lustre Administrators and Developers* workshop in September 2022 in Paris. The goal of this undertaking has been to allow users to transparently use ad-hoc storage systems on top of an existing parallel file system. Without this component, users need to specify their data input and output directory so that the data scheduler can move data between the namespaces of the ad-hoc file system and the parallel file system. In addition, the application needs to operate on another path. By integrating the namespace of the ad-hoc storage system into Lustre, this is no longer a requirement (but can still be helpful as hints to ADMIRE). Specifically, we propose a new design for this integration for all ad-hoc file systems (exemplarily shown with GekkoFS) leveraging Lustre's HSM mechanisms as shown by Figure 2.1. With this approach, we build on existing and robust Lustre mechanisms while avoiding extensive Lustre modifications. Further, other Lustre policy management and prefetching algorithms can be reused for maintaining high I/O performance.

2.1.4 Malleability

Since GekkoFS, as a user space file system, is not restricted by the kernel, it can offer configuration options to modify their behaviour. For instance, GekkoFS offers options such as selecting a specific metadata backend,

[2] https://drive.google.com/file/d/1Tqf7rqMXGcSxnWQ_VCptemYb9tZnd3hW/view

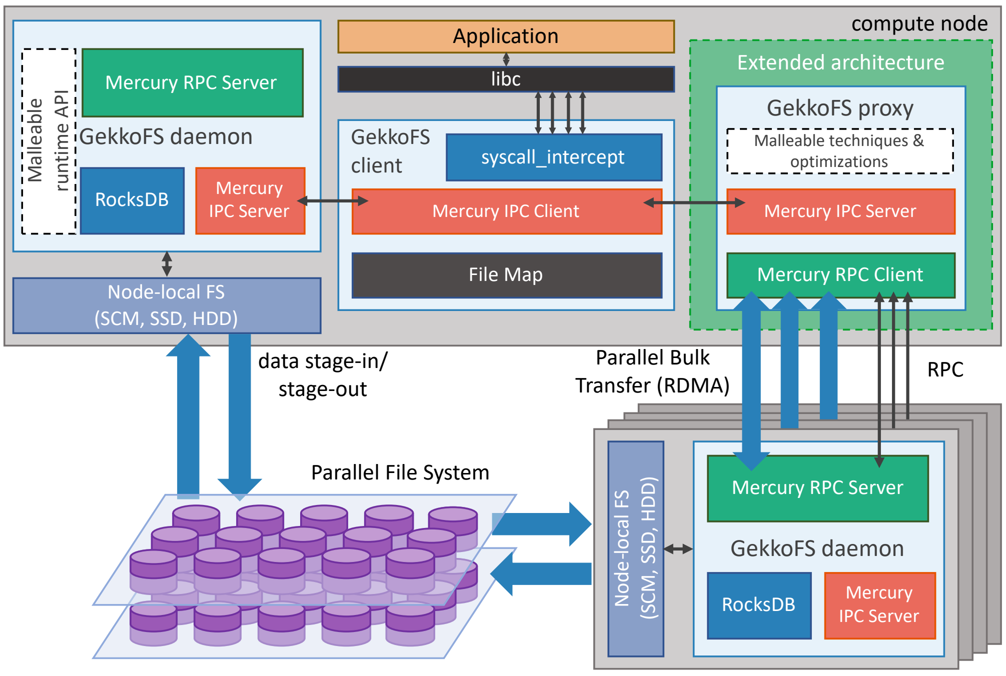


Figure 2.2: An extended GekkoFS architecture featuring the proxy.

choosing the data distribution algorithm, enabling symlink support, and altering file system protocols among others. Nonetheless, before support for malleability, such configurations must be set before the file system is launched and cannot be changed afterward, remaining static for the application's lifetime. Therefore, we have worked on a revised architecture that considers this limitation and attempts to offer a malleable platform for user space file systems based on an interception mechanism, such as GekkoFS.

To support malleability, GekkoFS's I/O servers are now equipped to support malleable mechanisms to a certain extent. Due to GekkoFS's decoupled architecture, adding mechanisms such as increasing or removing server nodes during runtime is relatively straightforward (and is currently in progress). GekkoFS's design supports this feature without any issues as long as the file system is empty. Changing the number of server nodes, however, would work but in an inefficient manner. Since a file's data and metadata server node are computed by generating hash keys, these keys would change and most of the data in the file system would need to be redistributed. This feature is already supported, and we are working to offer a more efficient data distribution, such as proposed by Miranda et al. [6].

On the other hand, the GekkoFS client design was unsuitable for supporting a broad set of malleability techniques. Since GekkoFS clients are restricted to an application process's lifespan (because the client's interception mechanism works within the process), it is not possible to implement relaxed cache consistency protocols and align them more closely with NFSv4 semantics [2] on a node-level, for example. Temporarily relaxing cache consistency guarantees, such as during the write burst of bulk-synchronous applications [1], could considerably improve performance, particularly for small I/O requests that are latency-sensitive. Overall, a GekkoFS client design that supports these use cases would complement a minimal *syscall_intercept* or `libc` interposition library by outsourcing more complex client code to another client process that operates on node-granularity. Therefore, we added a new file system component – the *GekkoFS proxy* – presented in Figure 2.2.

The details for this topic and the corresponding scalability evaluation have been submitted to the *2nd International Workshop on Malleability Techniques Applications in High-Performance Computing* (HPCMALL23) with the title *"From Static to Malleable: Improving Flexibility and Compatibility in Burst Buffer File Systems"*, and it is pending review. Overall, in this paper, we have discussed the critical issues of `LD_PRELOAD` as an interception mechanism and have proposed a new architecture for GekkoFS that overcomes these challenges, including presenting a path forward to broad malleability support in burst buffer file systems that can be deployed ad-hoc. We have evaluated the new architecture for latency- and throughput-sensitive operations and have concluded that the proxy only induces minor overheads, maintaining most of the file system's performance while offering new ways to support malleability and other optimisation techniques in user space file systems using an interception mechanism.

2.2 dataClay

dataClay [5] is a distributed object store with active capabilities. It is designed to hide distribution details while taking advantage of the underlying infrastructure, e.g., an HPC cluster or a highly distributed environment such as edge-to-cloud. Objects in dataClay are enriched with semantics, including the possibility to attach arbitrary user code to them. In this way, dataClay enables applications to store and access objects in the same format they have in memory, also allowing them to execute object methods within the store to exploit data locality. In this way, only the results of the computation are transferred to the application, instead of the whole object.

dataClay is implemented at user-level, so it is visible to applications using its client library. dataClay can be deployed in two different ways: as a service or as an ephemeral storage system, for which is currently being optimised. In the first case, it is deployed as a long-lived service in a dedicated set of nodes, and objects can be shared by multiple jobs or applications. In the second case, dataClay runs in the compute nodes assigned for a job, such that the data, which it contains, must be persisted after completing the job if needed. In both cases, the active capabilities (in-store method execution) of dataClay minimise data transfers and copies, either between nodes when the application and dataClay run independently, or between the application and the dataClay processes when both of them live in the same node. Additionally, regardless of the kind of deployment, disk accesses are avoided as much as possible by means of an object cache, where objects are already instantiated and ready to serve execution requests.

2.2.1 Updates

The following list includes the updates and progress in dataClay since the last deliverable in April 2022. The updates go in the direction of distributing the object store metadata to solve stage-in and stage-out performance issues, optimizing startup deployment by simplifying the registration mechanism, and improving support for Python.

New

1. Integrated functional tests with pytest and pytest-docker and migrate from AppVeyor to GitHub Actions for automated testing and deployment.
2. Implementation of a new metadata service based on Python, with improved throughput, latency and multi-threading, and support for both Redis and ETCD databases.
3. Added OpenTelemetry support for tracing between services.
4. Added functionality for object movement between backends.
5. Added rebalancing script for redistributing dataClay objects between backends.
6. Added Ansible scripts for HPC orchestration.
7. Added support for Spack.
8. Updated client API to allow simultaneous sessions.

Changed

1. Migrated Java services and runtime to Python codebase.
2. Improved serialization speed using built-in Pickle.
3. Simplified code sharing using PyPI imports.

4. Migrated from multi-repo to monorepo for easier maintainability and higher visibility of the project, and added new formatting standard.

5. Updated custom code annotations with Python built-in annotations.

2.2.2 Spack installation

Once Spack is available in the cluster, the dataClay repository should be added. This step will change once all the ADMIRE packages are in the same place, and eventually available on the official repository.

1. dataClay needs to be added to the Spack's repository

```
spack repo add dataclay/orchestration/spack
```

2. (optional) Once the previous step is done, DataClay is available:

```
spack info py-dataclay
```

3. To install dataClay:

```
spack install py-dataclay
```

5. The dataClay module can be loaded:

```
spack load py-dataclay
```

7. Unload dataClay when done.

```
spack unload py-dataclay
```

2.3 Expand

Expand is a parallel file system based on standard servers, as described in Deliverable D2.1. This section describes the work developed to convert Expand into an ad hoc parallel file system.

This section summarizes the updates from the last deliverable, the spack support and initial support for malleability.

2.3.1 Updates

The following list includes the updates and progress since the last deliverable in April 2022 for Expand:

- **New**
  1. Malleability initial support added.
  2. Spack support added.
  3. Simplified user experience: now it is easier to start and stop expanding ad-hoc servers using a script.
  4. New environment variables to configure XPN execution (threads, session and locality).

5. Initial Torino cluster support.

- **Changed**
  1. Improved threading support.
  2. Improved error returned in read syscall when the related file descriptor is a directory.
  3. Improved bypass library to intercept 64-bits related syscalls.
- **Fixed**
  1. stat syscall fixed.
  2. Unify the maximum path length using PATH_MAX.

2.3.2 Spack installation

Once Spack is available in the cluster, the Expand repository should be added. This step will change once all the ADMIRE packages are in the same place, and eventually available on the official repository.

1. Expand needs to be added to the Spack's repository

```
spack repo add xpn/scripts/spack
```

2. (optional) check the Expand package and its options:

```
AutotoolsPackage:   xpn

Description:
    The Expand Parallel File System (XPN)

Homepage: https://github.com/xpn-arcos/xpn/

Preferred version:
    2.1.0     https://github.com/dcamarmas/xpn/archive/refs/tags/v2.1.0.tar.gz

Safe versions:
    2.1.0     https://github.com/dcamarmas/xpn/archive/refs/tags/v2.1.0.tar.gz
    latest    [git] https://github.com/xpn-arcos/xpn.git on branch master

Deprecated versions:
    None

Variants:
    Name [Default]             When    Allowed values    Description
    ========================   ====    ==============    ====================
    build_system [autotools]   --      autotools         Build systems supported
                                                         by the package
    mpich [off]                --      on, off           Use MPICH

Build Dependencies:
    autoconf  automake  gnuconfig  libtool  mpich  mxml

Link Dependencies:
    mpich  mxml

Run Dependencies:
    None
```

3. Expand can be installed:

```
spack install xpn
```

4. The Expand module can be loaded:

```
spack load xpn
```

5. Unload Expand when done.

```
spack unload xpn
```

2.3.3 Malleability support in Expand

Expand allows dynamically reconfiguring a partition by adding new server nodes to it or removing server nodes. We have designed two strategies for adding new nodes to a partition:

1. Partition reconstruction.
2. Dynamic reconfiguration without reconstruction.

The current prototype includes the first of the designed techniques. Both schemes can be seen in figure 2.3. The only problem introduced by the addition of a new node to existing participation is the location of the metadata subfile of an Expand file in the new partition. Indeed, because the location of the master node of a file in the current prototype is based on the file name and the number of servers in participation, when this number changes, the location of the metadata also changes. The only operation necessary to keep the partition consistent is to relocate the metadata to their new master node.

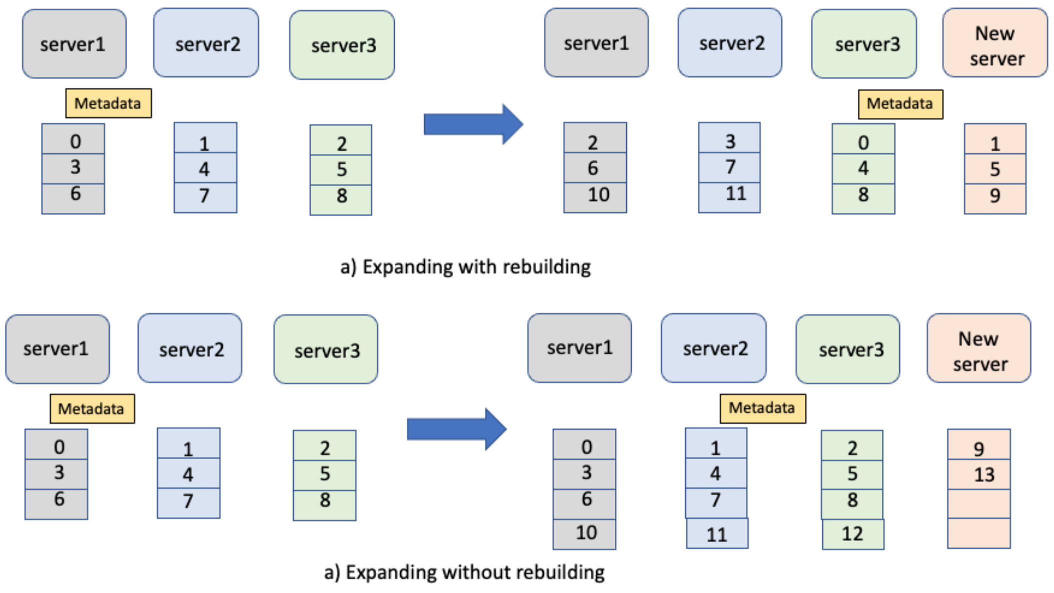


Figure 2.3: Malleability support in Expand

Figure 2.3-b shows the effect of adding a new node to a partition. The file in the old partition is composed of three nodes with a metadata subfile on server 2. When a new node is added, the hash function computation results in server 3, on which the metadata subfile does not reside. To keep this information consistent the metadata subfile is moved from server 2 to server 3.

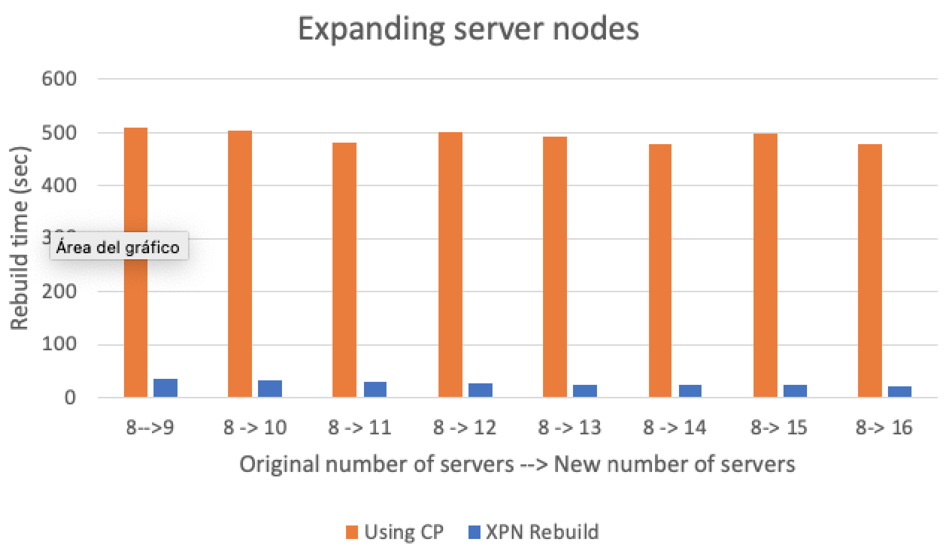


Figure 2.4: Time for expanding a partition

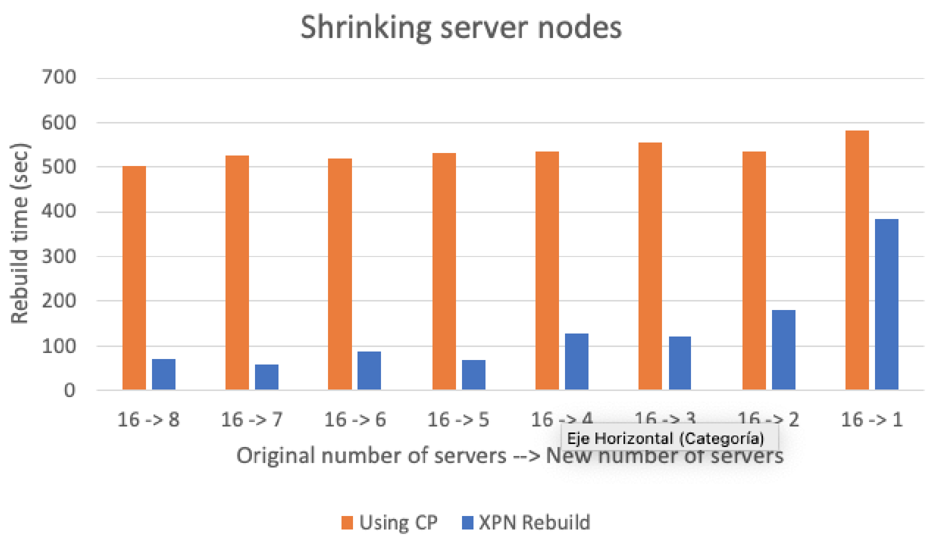


Figure 2.5: Time for shrinking a partition

In the case of deleting nodes from a partition, the only possibility is to rebuild the partition in the same way as shown in figure 2.3-a.

Figure 2.4 and Figure 2.5 shows a result of a first evaluation of the cost of adding or removing nodes to an existing partition. The figures show the performance of the reconstruction of a partition when adding or removing nodes and the time required to rebuild a 128 GB file to a new partition and compare it to the time required to make a traditional copy.

2.4 Hercules IMSS (in-memory storage system)

In this section, we highlight the new communication model included in Hercules, which takes advantage of the Unified Communication X framework (UCX) [9]. This solution leverages the capabilities of RDMA protocols, including Infiniband, Onmipath, shared memory, and zero-copy transfers.

In Figure 2.6, we show the communication model provided by Hercules, based on the Unified Communication X (UCX). The main components of the communication layer are the UCX workers. A UCX worker abstracts an instance of network resources such as a host, network interface, or multiple resources such as multiple network interfaces. UCX workers also represent virtual communication resources that can aggregate multiple devices, allowing Hercules to take advantage of cross-transport multi-rail communications by delivering data in multiple network interfaces in parallel (network bounding), without the need for any special tuning.

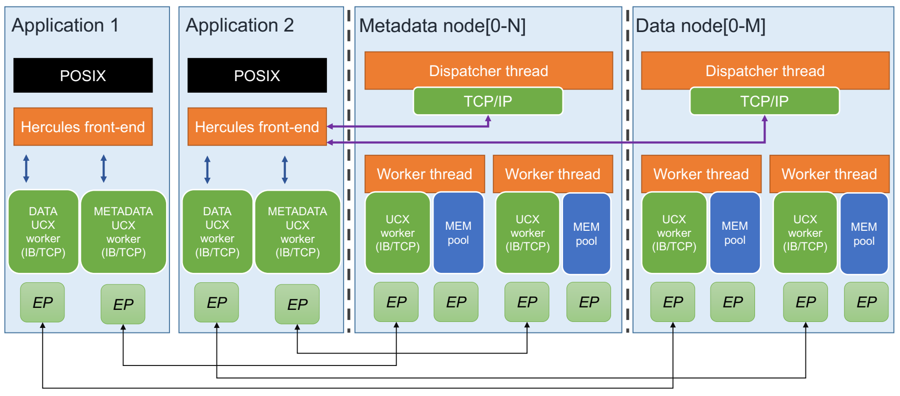


Figure 2.6: Architecture of a Hercules deployment.

The frontend layer relays on two independent UCX workers to enable point-to-point communication with data and metadata servers. This mechanism guarantees isolated transmission by using different communication queues. At initialization, the client library requests to the backend dispatcher thread the UCX worker address. This address is employed for the creation of endpoints at both sides that represent a connection from a local worker to a remote worker. All *get* and *set* requests are sent through these endpoints.

The development of the present work was strictly conditioned by a set of well-defined objectives. Firstly, Hercules provides flexibility in terms of deployment. To achieve this, the Hercules API provides a set of deployment methods where the number of servers conforming to the instance, as well as their locations, buffer sizes, and their coupled or decoupled nature, can be tuned. Second, parallelism should be maximized. To achieve this, Hercules follows a multi-threaded design architecture. Each server conforming an instance counts with a dispatcher thread and a pool of worker threads. The dispatcher thread distributes the incoming workload between the worker threads with the aim of balancing the workload in a multi-threaded scenario. The main entities conforming to the architectural design are Hercules clients (frontend), Hercules data (1 to M) and metadata (1 to N) servers (backend).

2.4.1 Updates

The following list includes the updates and progress since the last deliverable in April 2022 for Hercules IMSS:

New

1. Spack support added.
2. Portable support of deployment by using UCX.
3. Use of setup file for tuning configuration variables.
4. Added a memory pool for memory allocation of blocks at initialization.
5. Added a pool of endpoints on the backend side for increasing communication scalability.
6. Tested support for both Mellanox and Intel Omnipath networks.
7. Smart log system based on *slog*.
8. Moved from stream endpoints to tag-based messages.

Changed

1. Reduced number of internal memory copies in case large datasets.

Fixed

1. Tested and revised metadata calls.
2. Fixed support of system calls under IOR benchmark.
3. Fixed support of system calls under MDbench benchmark.

2.4.2 Spack installation

Once Spack is available in the cluster, the Hercules repository should be added. This step will change once all the ADMIRE packages are in the same place, and eventually available on the official repository.

1. Hercules needs to be added to the Spack's repository

```
spack repo add hercules/scripts/spack
```

2. (optional) check the Hercules package and its options:

```
CMakePackage:   hercules

Description:
    The Hercules file system

Homepage: https://gitlab.arcos.inf.uc3m.es/admire/imss/

Preferred version:
    1.0     https://gitlab.arcos.inf.uc3m.es/admire/imss/-/raw/master/releases/download/v1.0/imss.tgz

Safe versions:
    1.0     https://gitlab.arcos.inf.uc3m.es/admire/imss/-/raw/master/releases/download/v1.0/imss.tgz

Deprecated versions:
    None

Variants:
    Name [Default]              Allowed values          Description
    ===========================    ====================    =====================

    build_system [cmake]        cmake                   Build systems supported
                                                        by the package
    build_type [RelWithDebInfo] Debug, Release,         CMake build type
                                RelWithDebInfo,
                                MinSizeRel
    ipo [off]                   on, off                 CMake interprocedural
                                                        optimization


Build Dependencies:
    cmake  glib  ninja  pcre  ucx

Link Dependencies:
    glib  pcre  ucx

Run Dependencies:
    None
```

3. Hercules can be installed:

```
spack install hercules
```

4. The Hercules module can be loaded:

```
spack load hercules
```

5. Unload Hercules when done.

```
spack unload hercules
```

2.4.3 Shrink-expand initial support

We have included some malleability techniques in Hercules to facilitate dynamic modifications of the number of data nodes on runtime. Following the application's I/O needs, Hercules can expand or shrink the number of data nodes to increase or reduce the I/O throughput of the application. In any ad-hoc deployment of Hercules, when a dataset is created, we store in the metadata of the dataset the list of servers storing data for that dataset. That way, we can always know where Hercules should write/read the blocks for each dataset by applying the data distribution policy defined for that ad-hoc deployment. In addition to the list of servers, we store the number of servers storing the dataset to avoid extra operations with the list of nodes.

More information related to the implemented heuristics is included in Deliverable 3.3.

3 Applications and ad-hoc storage systems

This section will discuss new insights into the applications in the ADMIRE project and how they can use ad-hoc storage systems. Note that for the file systems, there is no direct integration necessary since the underlying standardised interfaces, i.e., POSIX and MPI I/O, are directly supported by them. However, there may be compatibility challenges because both GekkoFS and Hercules as user space file systems are using interception libraries to process the application I/O calls.

3.1 GekkoFS

GekkoFS's original design mainly focused on use cases found in HPC applications based on various studies [3, 12, 13]. Nevertheless, ADMIRE's Software Heritage application actively uses the `rename()` system call, which GekkoFS did not support. Support for this operation and hence the Software Heritage application was added in version 0.9.2. GekkoFS can be used by the Nek5000 application (turbulence simulation) without issues and, depending on the workload and configuration, can improve application runtime significantly. Since the interval of step and statistics information that are written from the application are essentially checkpoints, this shows WP2's benefits towards KPI 6. This use case was presented at a demonstration at the midterm review[1]. Further, GekkoFS has been shown to support Tensorflow applications found in the remote sensing and life sciences applications [8]. Quantum Espresso (molecule simulation) Wacomm++ (environment application) are under review.

Finally, the IO-SEA and ADMIRE projects are actively collaborating regarding using burst buffer file systems to improve I/O in HPC use cases.

3.2 dataClay

The new dataClay version featuring metadata distribution to optimize stage-in and stage-out has just been released. At this point, dataClay has been evaluated on a set of popular ML and data analytics kernels exhibiting different access patterns and workloads. Benchmarked algorithms include, among others, k-means, histogram, Cascade SVM, k-nearest neighbors or matrix multiplication, which is a basic operation in several ML and DL algorithms.

The next steps are to evaluate dataClay within the context of the selected ADMIRE applications, prioritizing those, such as the Software Heritage application, that can take advantage of the in-situ data processing capabilities provided by dataClay, and that are written in Python, as bindings have been optimized for this programming language.

3.3 Expand

XPN has been evaluated on different platforms with several benchmarks (IOR, MdTest, and IO500).

XPN has also been tested on the execution of programs written in Python, Fortran and C that make input/output syscalls to check the correct interception of system calls in those programs.

[1] https://drive.google.com/file/d/1Tqf7rqMXGcSxnWQ_VCptemYb9tZnd3hW/view

3.4 Hercules

Hercules has been tested over multiple applications and benchmarks. On the one hand, we have studied the feasibility of using Hercules in the ADMIRE use case applications. Hercules intercepts all the needed I/O system calls exposed by the POSIX interface. It is important to highlight that, as an in-memory ad-hoc file system, we need to include in the execution workflow two additional tasks to upload and download temporal data. Intercepted paths determine data location, so applications doing check-pointing (i.e., Nek5000) only request an output path. Data flushing is done once the application finalizes.

On the other hand, we have employed several benchmarks and microbenchmarks to verify the consistency of data. we have employed benchmarks such as IOR and MDBenchs. IOR has been employed to validate the throughput performance in multiple realistic scenarios: single and shared files, weak and strong scalability, and sequential and striped file accesses. MDBench evaluates a large number of metadata operations related to the data paths. Selected results have been included in a paper submitted to the conference Euro-Par 2023.

4 Further activities

In the following, we list additional activities achieved within the time frame since the last deliverable.

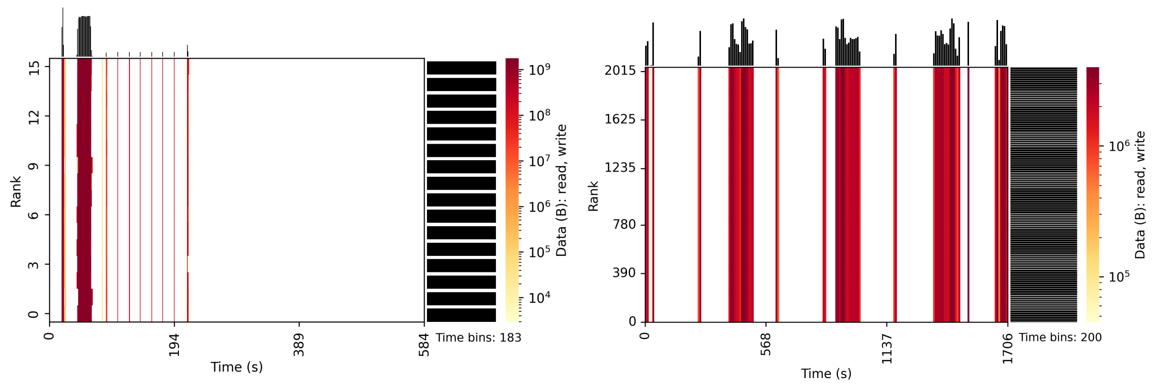
4.1 I/O tracing initiative

A collaboration for I/O tracing and analysis has been established between EuroHPC projects. The collaboration initially began between ADMIRE and IO-SEA projects. It aimed to create an open-access database of I/O performance data for applications running on different parallel I/O libraries and in different layers of the I/O stack. The database will help the scientific community analyse and identify I/O bottlenecks and design more efficient system software. In addition, this initiative provides a unique opportunity to study the I/O behaviour of representative workloads running on European clusters.

Understanding the I/O characteristics and requirements of various HPC applications is crucial for designing efficient storage and I/O solutions. However, the complexity and heterogeneity of today's scientific and HPC applications make this study challenging. In addition, existing analyses are limited to a few hardware settings and a specific family of applications due to the different hardware availability and application-specific instrumentation efforts.

To achieve more unified results, individual partners run applications using commonly available tools and settings. Our study collects various I/O performance data, including traces and profiles, using profiling and tracing tools selected to require minimal modification to the running setup and be readily applicable in different environmental settings with minimal overhead.

To gain unified results, JGU provided the initial version of the recipe containing the instructions for running the traces. It was distributed and tested internally at first within the ADMIRE project. The selected tool for profiling the workloads is Darshan[1]. Darshan records statistics about the I/O operations performed by an application, including the number of bytes read or written, the number of I/O operations, and the time taken to perform each operation.



(a) Remote sensing application

(b) Nek5000 application

Figure 4.1: Darshan Heatmap illustration of WP7 applications

[1] https://www.mcs.anl.gov/research/projects/darshan/

Figure 4.1 shows the illustrated heatmap of remote sensing and Nek5000 applications. *Heatmap* is a visualisation tool included with Darshan. It provides a graphical representation of the I/O behaviour of an application over time. The heatmap displays a matrix of rectangles, where each row represents a process in the application, and each column represents a time interval. The colour of each rectangle represents the amount of I/O activity performed by the process during the corresponding time interval. The darker the colour, the more I/O activity occurred during that time interval. This allows users to identify patterns in the I/O behaviour of their applications quickly and helps application developers, as well as system researchers, to identify performance bottlenecks and optimise the I/O behaviour of their designs.

Aside from the provided instructions, JGU has also initiated a mailing list and a website to collect the traces and metadata information of the submitted applications. The metadata contains information about datasets, job summaries, such as the number of MPI ranks and nodes, and the basic characteristics of the workload that is often lost in similar studies. JGU has already presented the initiative, instructions, and the initial website in a joint meeting with other application representatives of IO-SEA and DEEP projects. We have already received contributions and feedback from IO-SEA. The next steps include extending the study and support to cover the major workloads running on European clusters as well as leveraging the knowledge of applications into the ad-hoc storage systems' malleability mechanisms.

4.2 POSIX-compliant write-back caching for distributed file systems

In collaboration with DDN and the Whamcloud division, JGU worked on the paper titled *"MetaWBC: POSIX-compliant write-back caching for distributed file systems"* [7], which was published and presented at the A* conference *"The International Conference for High Performance Computing, Networking, Storage, and Analysis"* (SC22) in November 2022.

In summary, the paper proposes a novel metadata write-back caching (MetaWBC) mechanism to improve the performance of metadata operations in distributed environments. To achieve extreme metadata performance, the solution includes a fast, lightweight, and POSIX-compatible memory file system as a metadata cache. Further, it defines a file caching state machine and includes other performance optimisations. MetaWBC was coupled with Lustre and evaluated showing that MetaWBC can outperform the native parallel file system by up to 8x for metadata-intensive benchmarks, and up to 7x for realistic workloads in throughput.

Although MetaWBC was coupled only with Lustre, the descriptions are general enough to be used by other distributed file systems as well. In fact, the new insights provided by the paper will be extremely beneficial developing file system optimisations in the GekkoFS proxy and we aim to include some of the features proposed by MetaWBC into GekkoFS in the remaining year in ADMIRE.

5 Conclusion

In this prototype deliverable, we have presented the last updates of the four ad-hoc storage systems available in ADMIRE project. In this context, we have discussed integration w.r.t. Spack and malleability support, e.g., expanding and shrinking the ad-hoc file system. Further, we have described the applicability of the updates over the use cases in ADMIRE and other applications. Finally, we have presented further activities regarding publications as well as updates on the WP2-led EuroHPC tracing initiative.

The next steps will be to apply resilience over the different ad-hoc storage systems, completing task 2.3 and a final set of experiments demonstrating the benefits that the ad-hoc storage systems offer to the ADMIRE project and their use cases.

A Terminology

- Ad hoc storage system, ephemeral storage system that only exists in a determined period, i.e. during a job's execution.
- API, Application Programming Interface, a mechanism that enables an application or service to access a resource within another application or service. The application or service doing the accessing is called the client, and the application or service containing the resource is called the server.
- CLI, command line interface.
- DRAM, dynamic random-access memory.
- Ephemeral storage, file systems which are making persistent (surviving across system reboot) but which are designed to be deployed and destroyed over a limited period of time, from few hours up to few months.
- In situ data, processing the data where it is originated.
- In transit data, processing the data when it is moved.
- Node-local Storage, ability for a compute server to store persistent data on physically local storage devices.
- PFS, Parallel File System, type of distributed file system supporting a global namespace and spread across multiple storage servers.
- POSIX, Portable Operating System Interface, family of standardized functions.
- QoS, Quality of Service.
- RDMA, remote direct memory access.
- RPC, remote procedure call.
- Slurm, job submission system widely used.
- SSD, solid state drive.

Bibliography

[1] André Brinkmann, Kathryn Mohror, Weikuan Yu, Philip H. Carns, Toni Cortes, Scott Klasky, Alberto Miranda, Franz-Josef Pfreundt, Robert B. Ross, and Marc-Andre Vef. Ad hoc file systems for high-performance computing. *J. Comput. Sci. Technol.*, 35(1):4–26, 2020.

[2] Thomas Haynes. Network file system (NFS) version 4 minor version 2 protocol. *RFC*, 7862, 2016.

[3] Paul Hermann Lensing, Toni Cortes, and André Brinkmann. Direct lookup and hash-based metadata placement for local file systems. In *6th Annual International Systems and Storage Conference, SYSTOR '13*. ACM, 2013.

[4] Paul Hermann Lensing, Toni Cortes, Jim Hughes, and André Brinkmann. File system scalability with highly decentralized metadata on independent storage devices. In *IEEE/ACM 16th International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Colombia, May 16-19*, pages 366–375, 2016.

[5] Jonathan Martí, Anna Queralt, Daniel Gasull, Alex Barceló, Juan José Costa, and Toni Cortes. Dataclay: A distributed data store for effective inter-player data sharing. *Journal of Systems and Software*, 131:129–145, 2017.

[6] Alberto Miranda, Sascha Effert, Yangwook Kang, Ethan L. Miller, André Brinkmann, and Toni Cortes. Reliable and randomized data distribution strategies for large scale storage systems. In *18th International Conference on High Performance Computing, HiPC 2011, Bengaluru, India, December 18-21, 2011*, pages 1–10. IEEE Computer Society, 2011.

[7] Yingjin Qian, Wen Cheng, Lingfang Zeng, Marc-André Vef, Oleg Drokin, Andreas Dilger, Shuichi Ihara, Wusheng Zhang, Yang Wang, and André Brinkmann. MetaWBC: POSIX-Compliant metadata write-back caching for distributed file systems. In *SC '2: The International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, Texas, USA, 2022*. IEEE, 2022.

[8] Frederic Schimmelpfennig, Marc-André Vef, Reza Salkhordeh, Alberto Miranda, Ramon Nou, and André Brinkmann. Streamlining distributed deep learning I/O with ad hoc file systems. In *IEEE International Conference on Cluster Computing, CLUSTER 2021, Portland, USA, September 07-10, 2021*. IEEE, 2021. (Accepted for publication).

[9] Pavel Shamis, Manjunath Gorentla Venkata, M Graham Lopez, Matthew B Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L Graham, Liran Liss, et al. UCX: an open source framework for HPC network APIs and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43. IEEE, 2015.

[10] Marc-Andre Vef, Nafiseh Moti, Tim Süß, Markus Tacke, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann. GekkoFS - A temporary burst buffer file system for HPC applications. *J. Comput. Sci. Technol.*, 35(1):72–91, 2020.

[11] Marc-Andre Vef, Nafiseh Moti, Tim Süß, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann. GekkoFS - A temporary distributed file system for HPC applications. In *IEEE*

International Conference on Cluster Computing, CLUSTER 2018, Belfast, UK, September 10-13, 2018, pages 319–324. IEEE Computer Society, 2018.

[12] Chen Wang. Detecting data races on relaxed systems using recorder, 2022.

[13] Chen Wang, Kathryn Mohror, and Marc Snir. File system semantics requirements of HPC applications. In Erwin Laure, Stefano Markidis, Ana Lucia Verbanescu, and Jay F. Lofstead, editors, *HPDC '21: The 30th International Symposium on High-Performance Parallel and Distributed Computing, Virtual Event, Sweden, June 21-25, 2021*, pages 19–30. ACM, 2021.