Authors

Sai Narasimhamurthy, Ganesan Umanesan, Nikita Danilov, Jean-Philippe Bernardy, Mathieu Boespflug, Nathan Rutman, Giuseppe Congiu, Arnaud Spiwack, Tim Williams, Malcolm Muggeridge and Sining Wu (All Seagate, UK), Dirk Pleiter, Nicolas Vandenbergen & Salem El Sayed (All Juelich Supercomputing Center, Germany), Philippe Deniel, Jacques-Charles Lafoucriere & Thomas Leibovici (All Commissariat à l’énergie atomique et aux énergies alternatives, France), Clemens Lutz (Deutsches Forschungszentrum für Künstliche Intelligenz, Germany), Stefano Markidis & Sergio Rivas Gomez (Kungliga Tekniska högskolan, Sweden), Keeran Brabazon (Allinea, UK), Herve Gloaguen (ATOS, France), Shaun De-Witt (Culham Center for Fusion Energy, UK), Alison Packer (Science & Technology Facilities Council, UK) and Dave Bond (Diamond Light Source, UK)

Copyright

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

Acknowledgements

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

More Information

All public deliverables of SAGE can be found at http://www.sagestorage.eu
Table of Contents

1. Executive Summary.................................................................................................................. 5
2. Introduction .............................................................................................................................. 6
3. Platform Requirements for SAGE .......................................................................................... 10
4. SAGE System Architecture ................................................................................................. 17
5. SAGE Platform Components ............................................................................................... 20
6. SAGE Ecosystem Components ............................................................................................. 31
7. Conclusion .............................................................................................................................. 40
8. References .............................................................................................................................. 41
9. Glossary of Acronyms ............................................................................................................ 43

Lists of Figures

Figure 1: Research areas within the SAGE Project ........................................................................ 7
Figure 2: Co-Design in SAGE: Deriving the SAGE Platform ....................................................... 9
Figure 3: BDEC/HPDA Workflow at Extreme Scales .................................................................... 11
Figure 4: Formal I/O Characterisation for Each Use Case for the SAGE Platform - Snapshot13
Figure 5: Storage Architecture Analysis for Each Use Case in SAGE for Mero - Snapshot ......... 14
Figure 6: Data Retention Characterisation (iPIC3D and NEST Examples) - Snapshot .............. 15
Figure 7: SAGE Conceptual Architecture .................................................................................... 17
Figure 8: SAGE System Stack ..................................................................................................... 18
Figure 9: Initial Mero Vision ........................................................................................................ 20
Figure 10: (A) Clovis Basic Functionalities, (B) Examples of Clovis on Top of Mero .................. 22
Figure 11: SAGE Racks - Front View ......................................................................................... 30
Figure 12: Integration of MPI I/O With Object Storage Support Using the Clovis API ............... 33
Figure 13: Illustration of a Process Reading/Writing Data Through PGAS I/O in SAGE ........... 35
Figure 14: Flow Diagram of an MPI Window Allocation Using Storage .................................... 35

© Seagate and other partners of the SAGE consortium 2015-2018
Lists of Tables

Table 1: Gathering Co-Design Inputs for the SAGE Platform ........................................12
Table 2: SAGE Tiers and Mapping to Physical Hardware ...........................................30
1. Executive Summary

The StorAgE for Exascale Data Centric Computing (SAGE) system, researched and built as part of the SAGE project, aims to implement a Big Data/Extreme Computing (BDEC) and High Performance Data Analytics (HPDA) capable infrastructure suitable for Extreme scales - including Exascale and beyond. Increasingly, overlaps occur between Big Data Analysis and High Performance Computing (HPC), caused by the proliferation of massive data sources, such as large, dispersed scientific instruments, sensors, and social media data, whose data needs to be processed, analysed and integrated into computational simulations to derive scientific and innovative insights. The SAGE storage system, will be capable of efficiently storing and retrieving immense volumes of data at Extreme scales, with the added functionality of “Percipience” or the ability to accept and perform user defined computations integral to the storage system. The SAGE system will be built around the Mero object storage software platform and its supporting ecosystem of tools and techniques, that will work together to provide the required functionalities and scaling desired by Extreme scale workflows. The SAGE system will seamlessly integrate a new generation of storage device technologies, including non-volatile memories as they become available. The SAGE system will also offer a very flexible API and a powerful software framework suitable for easy extensibility by third parties.

This white paper provides a technical overview of the SAGE system and describes its key component pieces and the extended capabilities and tools being created to support it.

Index Terms: HPC (High Performance Computing), Exascale, HPDA, Extreme Scale, Object Store, Non Volatile Memories, Storage system, API (Application Programming Interface), Data Centric Computing, Storage Class Memory SCM, Percipient Storage, Mero
2. Introduction

In this section, we describe the overall objectives of the SAGE system and discuss the methodology used to derive its architecture.

2.1. Objectives

Exascale computing is characterised by the availability of an infrastructure to support computational capability that exceeds current petaflop/petascale systems by 2-3 orders of magnitude. Currently, this definition is broadly understood to include storage and processing of an Exabyte of data as part of a scientific workflow or a simulation. We anticipate that Exascale computing infrastructures, capable of being exploited by applications and workflows for science and technological innovation, will be available in the 2022 timeframe.

Innovations in computing infrastructure have been driven by Moore’s law and the development of heterogeneity with multi-core and many-core processing. However, I/O and storage performance have lagged far behind compute capability. Although compute performance at Exascale will have increased 1000s of times, as compared to early Petaflop machines, storage performance in the same time period is predicted to improve by only 100 times, according to estimates provided by Ross, et al. [Ross2013] In fact, the performance of disk drives per unit capacity is actually decreasing with new very high capacity disk drives on the horizon. Simultaneously, the landscape for storage is changing with the emergence of new storage device technologies, such as flash (available today) and the promise of non-volatile memory technologies available in the near future. The optimised use of these devices in the I/O hierarchy, combined with existing disk technology, is just beginning to be explored in the HPC realm.[Los-Alamos] SAGE proposes hardware that supports a multi-tiered I/O hierarchy with associated intelligent management software, and provides a demonstrable path towards Exascale. Further, SAGE proposes a novel architectural approach in Extreme scale HPC, by moving computations, typically done in the compute cluster, to the storage system, which we term as “Percipience”;¹ dramatically reducing the energy footprint of the overall system.[EnergySavings] SAGE’s design will facilitate the Performance/Watt goals of Exascale class systems.[ASC-Report]

¹ Providing the storage system the ability to “percieve” data
The architecture of SAGE is built with future scaling in mind - Exascale and beyond ("Extreme scale"). Hence, in this paper, we use the terms Exascale and Extreme scale interchangeably.

The primary objective of this white paper is to describe the high-level architecture of the SAGE system. As a project, SAGE addresses many research areas pertinent to building an Extreme scale data-centric computing system as shown in the following pictorial. These areas are covered in later sections of this white paper (Figure 1).

**Figure 1: Research areas within the SAGE Project**
2.2. Methodology

The following methodology has been used to derive the SAGE system architecture:

1. Incorporating fundamental assumptions of and understanding the needs of:
   a. Exascale class I/O intensive applications
   b. HPDA\(^2\) use cases
   c. BDEC use cases

   These use cases generate, consume or have a need to process enormous volumes of data that are currently not seen in the Petascale realm and are loosely defined as Exascale Data Centric Applications.

2. Incorporating co-design inputs from SAGE use cases.

In the SAGE project, we have carefully selected use cases that reflect these data-centric applications. The use cases provide specific inputs that are designed to fine tune/modify the framework for the SAGE architecture, which is derived from fundamental assumptions specified in (1) above. Once the architecture is derived, the use cases (and their supporting tools) are designed to optimally adapt or exploit such an infrastructure. The overall methodology for deriving the SAGE system is depicted in Figure 2, below. The use cases are the primary drivers to provide requirements for the SAGE platform. The platform’s requirements apply both to the use cases as well as the “ecosystem” components assisting the use cases.

The SAGE system consists of the SAGE platform and the SAGE ecosystem components. The SAGE platform consists of Mero, the object storage software, Clovis, Mero’s API, and associated hardware.

\(^2\) High Performance Data Analytics (HPDA) is defined as the use of an HPC infrastructure for the needs of applications using Big Data Analytics.
In the next sections, we discuss requirements for the SAGE system, followed by specific inputs from the SAGE use cases, present a technical overview of SAGE and provide a description of individual system components.
3. Platform Requirements for SAGE

The general storage requirements gathered within SAGE are derived from Extreme scale data-centric computing use cases. These requirements will be addressed by the SAGE platform, alongside the specific co-design inputs described in Section 3.2.

3.1. Exascale Storage System Requirements

Our Exascale storage system requirements are primarily driven by:

- Inputs from the BDEC community and the US Department of Energy labs,[BDEC Workshops],[ASC-Report],
- Data needs for big science, as exemplified by the Square Kilometer Array[SKA] and the Human Brain Project[HBP]
- Extreme scale I/O requirements drafted by the European Technology Platform[ETP4HPC-SRA],
- Extreme scale data needs highlighted by the HPDA community[IDC-HPDA]

The following top-level objectives for an Exascale-capable storage system have been defined:

1. Ability to store and retrieve extreme volumes of data approaching orders of ~Exabyte for a given problem.
2. Ability to store and retrieve data at exceptionally high I/O rates[Ross13],[ETP4HPC-SRA]
3. Ability to ingest data from a variety of data sources.
4. Ability to manage workflows that include data from simulations and instruments.
5. Ability to run data processing/analyses as part of the workflow (in parallel with data gathering and scientific simulations), termed Percipience.
   a. Data processing that includes pre/post processing, data transformation, visualization, etc.
   b. Data analytics that includes Predictive Analytics, data mining, etc.
   c. Ability to feed insights from data analyses back into running simulations.
6. Ability to manage regularly occurring failures within the application and the infrastructure.
7. Ability to provide data integrity at Extreme scales.
Figure 3 depicts a workflow for a BDEC/HPDA storage system.

![Workflow Diagram]

**Figure 3: BDEC/HPDA Workflow at Extreme Scales**

In 2012/13, a series of Quality Attribute Workshops (QAW)s, coordinated by Xyratex (prior to acquisition by Seagate), were conducted to discuss the requirements of Exascale storage software (primarily object storage software). The workshops combined HPC data centre administrators, users and application developers with I/O and storage architects. These experts collaborated to identify the storage limits, bottlenecks, and functional incompatibilities of existing storage systems resulting from advanced scientific workflows.[PDSW – WIP Session]

Fundamental insights from BDEC and the QAW workshops were then validated with more specific BDEC use case inputs from the real world as part of the SAGE project co-design, leading to the platform architecture for SAGE.
3.2. SAGE Use Case Inputs

Table 1 shows diverse use cases that have been studied and used in the SAGE project to gather co-design inputs for the SAGE platform. The use cases cover a broad range of domains, including data from some of the world’s largest scientific experiments (including one of the world’s largest nuclear fusion facilities and one of the largest synchrotrons in Europe), aside from extremely data-centric HPC codes.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Domain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angelia</td>
<td>Big Data Analytics</td>
<td>Benchmarking Framework for Apache Flink</td>
</tr>
<tr>
<td>CCFE/ALF</td>
<td>Nuclear Fusion</td>
<td>Applications to perform analytics on data consumption log files</td>
</tr>
<tr>
<td>CCFE/EFIT++</td>
<td>Nuclear Fusion</td>
<td>Plasma equilibrium fitting code</td>
</tr>
<tr>
<td>CCFE/Spectre</td>
<td>Nuclear Fusion</td>
<td>Visualisation tool providing near real-time feedback on plasma and other operational conditions</td>
</tr>
<tr>
<td>iPIC3D</td>
<td>Space Weather</td>
<td>Particle-in-Cell code for simulations of space plasma</td>
</tr>
<tr>
<td>JURASSIC</td>
<td>Climate</td>
<td>Fast radiative transfer model simulation code for the mid-infrared spectral region</td>
</tr>
<tr>
<td>NEST</td>
<td>Life Sciences</td>
<td>Models for Mammalian and Human Brains</td>
</tr>
<tr>
<td>Ray</td>
<td>Bio-informatics</td>
<td>Distributed de novo assembly of metagenome</td>
</tr>
<tr>
<td>Savu</td>
<td>Synchrotron Experiments</td>
<td>Tomography reconstruction and processing pipeline</td>
</tr>
</tbody>
</table>

**Table 1: Gathering Co-Design Inputs for the SAGE Platform**

SAGE gathered the first formal list of inputs from all of the specified use cases. This phase included gathering inputs on formal I/O characterisation (Figure 4), SAGE architecture analysis (Figure 5), data retention characterisation (Figure 6) and data scaling analysis, which was an analytical study of how data and I/O requirements of the use cases would scale on a future basis.
Formal I/O Characterisation Criteria

<table>
<thead>
<tr>
<th>Characterisation of the I/O patterns for a suitable use case/workload (as defined above)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Total amount of data read/written</td>
</tr>
<tr>
<td>- Total amount of IOPs</td>
</tr>
<tr>
<td>- Maximum main memory footprint</td>
</tr>
<tr>
<td>- Archive requirements</td>
</tr>
<tr>
<td>- I/O intensity: time spent in I/O vs. total execution time</td>
</tr>
<tr>
<td>- Fraction of small and large I/O requests (meaning of small and large to be specified)</td>
</tr>
<tr>
<td>- Access patterns dominantly sequential or random</td>
</tr>
<tr>
<td>- I/O parallelism (I/O from single or multiple processes)</td>
</tr>
<tr>
<td>If using parallel I/O, is file/object access shared</td>
</tr>
<tr>
<td>- For shared access, what kind of inter-process I/O consistency is assumed? (commit-on-close, POSIX semantics, etc.)?</td>
</tr>
</tbody>
</table>

Metadata

<table>
<thead>
<tr>
<th>Characterisation of a workload with respect to file system metadata requirements:</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Extended file attributes are file system features that enable users to associate computer files with metadata not interpreted by the file system.</td>
</tr>
<tr>
<td>- Total number of file/object creations</td>
</tr>
<tr>
<td>- Maximum and minimum file/object size</td>
</tr>
<tr>
<td>- Total number of directories</td>
</tr>
<tr>
<td>- Directory tree depth</td>
</tr>
<tr>
<td>- Maximum number of files in a directory</td>
</tr>
<tr>
<td>Are extended attributes used?</td>
</tr>
<tr>
<td>- Is the file system used to store other application metadata</td>
</tr>
</tbody>
</table>

Fault Tolerance

<table>
<thead>
<tr>
<th>Supported fault-tolerance mechanisms (check-pointing, transactions, etc.) and what APIs are used?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault-tolerance related mechanisms needed from the underlying system (fsync, ACID transactions, HA, etc.)?</td>
</tr>
<tr>
<td>- Is there a logical concept of storage transaction within the application? If so, what ACID properties these transactions need?</td>
</tr>
<tr>
<td>- Does the application have a failure model (formal or informal)?</td>
</tr>
<tr>
<td>- Availability requirements (allowed down-time per second, lost data per byte or per object, etc.).</td>
</tr>
</tbody>
</table>

Figure 4: Formal I/O Characterisation for Each Use Case for the SAGE Platform - Snapshot
Storage Architecture Analysis

Mapping to Storage Hierarchy

Description of how application data could be distributed to different levels of the storage hierarchy addressing the following aspects:
- Data capacity requirements at a given level
- Bandwidth requirements
- Persistence requirements

In-Storage Processing

Description of opportunities for performing part of the data processing within the storage, including e.g. data pre- and post-processing. Try to address the following aspects:
- Indicate specific example functions that could be shipped/handled to/within storage
- Comment on infrastructure requirements (processing, memory, etc.) for offloaded functions and/or data analytics tasks

I/O and Compute Phase

Is there a distinction between I/O and compute phases in the application? General notes to compute and I/O relationship as in fully coupled, partially coupled, etc.

Opportunities for Pre-Load / Pre-Stage of Input Data and Asynchronous I/O

Description of opportunities to perform independent I/O by pre-loading input data and performing asynchronous I/O

Exploiting Mero Container Feature

Analyse whether it is desirable to partition data sets based on performance requirements and/or data formats using Mero’s data object container feature. This also makes it possible to independently define failure handling, etc. for these containers. Please provide any comments.

Mero Resilience Features

How resilient should intermediate and final data products be? Please provide any comments, for instance, higher tiers need lower resiliency, etc. Note that PD-RAID is the default in Mero, but other RAID schemes based on compromise between resiliency and data volumes could be provided.

Mero Transactions Support

Is it desirable to support storage transactions in the application? This makes it possible to easily roll back to consistent data states when there are failures.

Support for Data Compression

Please provide information how workload could benefit from data compression.

Other SAGE “Wish List”

Description of other opportunities or problems that can be addressed within SAGE for the workload described above. You may consider aspects like improvements on performance issues observed on existing systems (please describe which system), wish for new I/O interfaces or tools for analysing I/O behaviour and performance.

Figure 5: Storage Architecture Analysis for Each Use Case in SAGE for Mero - Snapshot
Figure 6: Data Retention Characterisation (iPIC3D and NEST Examples) - Snapshot
The APEX collaboration\cite{APEX} suggested analysing HPC workloads by classifying data objects that are created during such a workflow, according to retention time. The considered workloads include workflows for which multiple large-scale scientific simulations are coupled. During these simulations, data sets are produced (or consumed) which have different lifetimes, i.e. different retention times. Such a data retention time analysis (shown in Figure 6) indicated that applications in SAGE would generally benefit from fast storage layers that allow short-term and transient data sets to be held. This model includes cases in which data is replicated at multiple tiers, which needs to be managed by the object storage infrastructure.

Scaling analyses of I/O and storage requirements were systematically pursued, not just with simple estimates, but also through a detailed mathematical analysis of the application behaviour.\cite{D1} The scaling analyses performed for SAGE use cases showed that I/O capabilities had to scale proportionally (linear as well as quadratic dependence to key problem variables) to processing capabilities. Although higher data needs are (intuitively) nothing new, the precise details derived from this exercise were extremely insightful and instructive for the SAGE platform.

Thus, the co-design process yielded a very rich set of data points and information on top of the BDEC and QAW requirements.
4. SAGE System Architecture

We next describe the high-level design of the SAGE Percipient storage system and provide details of the SAGE platform architecture, based on the platform requirements discussed in Section 3.

We justify our high-level architecture with a I/O hierarchy, including in-storage compute capabilities, driven by an object storage infrastructure. Figure 7 summarises this concept:
Based on the fundamental requirements highlighted in Section 3 and the additional co-design inputs described in Section 3.2, we present the following detailed architecture of the SAGE system (shown in Figure 8).

The SAGE system is built on multiple tiers of storage device hardware technology. SAGE does not require a specific type of storage device technology, but typically it would include at least one NVRAM tier (Intel 3DxPoint technology is a strong contender at the moment), at least one flash tier and at least one disk tier. Together, these tiers are housed in standard form-factor enclosures and provide their own compute capability, enabled by standard x86 embedded processing components. Moving up the system stack, compute capability increases for faster, lower latency devices.
Mero, the object storage software, is layered on top of this hardware stack, providing fundamental management of object I/O and storage across tiers. Essentially, Mero forms the core of the SAGE system. Mero is presented to users through the Clovis API. Everything above Clovis forms the SAGE ecosystem components.

Layered on top of the Clovis API are various tools (debugging, performance analysis, HSM, etc.) that support the use cases accessing Mero (known as Use Case Access Methods) through pNFS[^Pnfs] data format I/O, [^NetCDF] and data analytics I/O, [Apache Flink] Future use case support is researched through adaptations of existing programming models (MPI and PGAS) to efficiently exploit the SAGE architecture.

SAGE will also support special HPC Runtimes enabling post-processing and data caching across the I/O hierarchy, including support for the I/O needs of visualisation utilities.

We next describe the SAGE platform, termed the Percipient storage system, which consists of Mero and Clovis, followed by the ecosystem components.
5. SAGE Platform Components

We next discuss the individual components of the SAGE Percipient storage platform, that provide the building blocks for the overall SAGE infrastructure.

5.1. Mero and Clovis

Mero provides the storage infrastructure that drives and controls data in the storage hardware. Mero is an Exascale-capable object storage framework developed by SAGE partner Seagate. Figure 9 shows the initial vision for Mero at the start of the SAGE project.

As depicted in Figure 9, Mero can be described through a “core” which can be accessed through the ‘Clovis’ API and the File Data Manipulation Interface (FDMI) API (now a part of SAGE).

---

5 Analytics is provided by Apache Flink\cite{Apache-Flink} in SAGE
Clovis), known as the extension interface for building "add on" data management applications. The vision for Mero is to be able to run a wide variety of workload types, including data analytics, HPC and cloud-based, powered by a core object store that is easily extensible by third parties through the FDMI.

5.1.1. Mero Core

The Mero core consists of a set of components and services that together, provide the capability to build scalable, distributed storage solutions. Other than the operating system and device drivers, this core does not depend on any software (local file systems, RAID layers, web services or HA systems). The core provides availability, performance, scalability, observability and storage efficiency. Mero’s core components are enumerated in the following list. A small subset of these components were available at the start of the SAGE project, with most of these features being architected, designed and implemented within SAGE:

- Storage Object (stob) - Module that abstracts details of the underlying bulk storage.
- Key-Value Store (KVS) - Module that implements local transactional key-value store for small data (e.g. metadata).
- Distributed Transaction Manager (DTM) - Provides consistency in the face of transient network and node failure.
- Resource Manager (RM) - Uniformly controls and distributes arbitrary types of resources and arbitrates conflicts.
- Loom - Performs internal distributed cooperative data movement and transformation, including recovery from permanent storage failures.
- Layout Manager - Specifies how logical entities (objects and metadata structures) are mapped to lower-level storage.
- Server Network Striping (SNS) - Network-distributed erasure coding scheme,
- Non-Blocking Availability (NBA) - Automatic storage redirection for writes.
- Container Manager - Organizes data and metadata for migration and placement.
- Analytics and Diagnostics Database (ADDB) - Collects information about system behaviour for analysis and monitoring.
- File Operation Machine (FOM) - Non-blocking state machine that executes storage operations.
- File Operation Log (FOL) - Log that records executed operations.
- High-Availability system (called “Halon”) - Quorum-based, consistent view of cluster state.

These core services will also be utilised to develop ubiquitous and peer-peer caching features in the project.

### 5.1.2. Clovis API

Within Mero, we have defined a rich, transactional storage API, Clovis, that can be used directly by user applications and also can be layered with traditional interfaces, such as POSIX and RESTful APIs,[REST](http://example.com) in a similar way to libRados, the interface upon which the CephFS (POSIX), RadosGW (S3), and RBD (block device) interfaces are built.[Ceph](http://example.com) Hence, Clovis exposes Mero to a variety of workloads from Big Data analytics, the cloud and HPC. Clovis’ interface definitions have been refined during the SAGE definition phase. Figure 10(a) shows a basic block schematic of Clovis functionalities in the context of the bit picture of SAGE and Figure 10(b) shows examples of Mero usage through Clovis. The pNFS gateway will be the focus of the SAGE project.

![Clovis Schematic](http://example.com)

---

**Figure 10**: (A) Clovis Basic Functionalities, (B) Examples of Clovis on Top of Mero

Clovis consists of an *access interface* that provides access to objects (including through various gateway stacks), specifies “containers” for objects and provides transactional semantics. Clovis contains a “management” interface that accesses ADDB telemetry records on system performance that can be fed into external system data analysis tools. Clovis also contains an extension interface that will be used to extend the features and functionalities of Mero. This interface, known as FDMI, is described in the following section.
Additional data management plug-ins can easily be built on top of the core through the FDMI interface (part of the Clovis “extension” interface). Hierarchical Storage Management (HSM) and information lifecycle management, file system Integrity checking, data indexing, data compression are examples of third-party plug-ins that, utilising the API, could be developed for Mero. FDMI plug-ins use the FOLs from Mero.

5.2. Mero Components in SAGE

The following sections describe the software components of Mero that will be built in the SAGE project.

5.2.1. Container Management System

Mero is an object based storage system. An object encapsulates data, attributes and actions corresponding to a single entity. Mero provides containers as an additional level of abstraction and encapsulates data, attributes and actions. In other words, containers can be used for grouping objects, performing transactions (or a combination of both) and performing actions that are common to all items in a given container. Grouping large numbers of similar objects in a container and performing one-shot actions on containers will improve Mero’s performance and Clovis’ application usability.

5.2.2. Resource Management System

The Resource Manager provides a scalable, coherent caching infrastructure. It allows applications (as well as internal Mero components) to cache resources locally in a consistent manner. In the RM system, a resource is anything with a well-defined notion of ownership: part of a data object, a key-value record, a fraction of network of storage traffic, node memory, etc. The Resource Manager provides common functionality to locate resources, transfer their ownership, and notify owners about conflicting access requests.

In the context of SAGE, the Resource Manager is the component that allows multiple users to concurrently access data that is stored (and moving) in a multi-tiered storage system.
5.2.3. Non-Blocking Availability System

The Non-Blocking Availability (NBA) system provides users with the ability to access data, when the storage is degraded in some manner. Consider a storage pool, in which data are stored redundantly (e.g. a network RAID). When such a pool experiences a failure, its data can still be accessed (thanks to redundancy), but such access is less efficient than in a failure free environment (e.g. in the case of network RAID, some data must be reconstructed from erasure codes). NBA redirects accesses from a degraded pool to a new pool, thus avoiding the inefficiency of reconstruction and also eliminating the interference between user accesses and the ongoing background process of reconstruction.

In the case of Exascale systems, NBA-like access methods are a necessity, because as a system grows, failures become more frequent, until (well before Exascale) the system may enter a state in which failures are never completely resolved. Correct and efficient operation in the event of failures, which are no longer rare events or exceptions, is critical for scalability.

5.2.4. High Availability System

Component reliability is not expected to improve in the future. Yet data has shown that the number of component failures scales proportionally with respect to certain units, such as the amount of RAM, the number of cores, the number of NICs, etc. As a result, we must plan for high failure rates at Exascale, in addition to software failures that may result (potentially) in crashed nodes. To maintain service availability in the face of expected failures, the global state (or configuration) of the cluster may need to be modified, by means of a repair procedure.

The HA subsystem for SAGE will coordinate automated repair activities in response to failures. The subsystem monitors for failure events (inputs) throughout the cluster. Then, on the basis of the collected events, the HA subsystem decides whether to take action. It does not consider events in isolation but quantifies, over the recent history of the cluster, quasi-ordered sets of events to determine which repair procedure (output) to engage, if any.

One challenge presented by this mechanism is that system availability is predicated, in part, upon the availability of the HA subsystem. When HA functionality is unavailable, then no repair is undertaken, reducing availability in the rest of the system. We propose to address this challenge via aggressive replication of the HA state.
5.2.5. Layout Management

A layout determines how a storage entity (an object, a key-value index, a container, etc.) is mapped to the available storage. Layouts determine performance and fault-tolerant properties of storage entities. Mero supports layouts as first-class objects, which can be defined by applications.

In the context of SAGE, two applications of layouts are especially important:

- Composite layouts, which determine how an object is distributed across multiple tiers of a deep storage hierarchy
- Clerk files, which provide a model for off-loading computation to storage (function shipping)

5.2.6. Distributed Transaction Management

Distributed transactions are groups of updates to the storage system that are guaranteed to be atomic with respect to failures. For a long time, transactions have been recognised as a necessary component of a generic storage solution. On the other hand, traditional Relational Data Base Management Systems-style transactions are known not to scale. To address this problem, Mero separates transaction control from other issues that are usually linked with it, concurrency control and isolation. The resulting transaction mechanism is scalable and efficient.

5.2.7. Function Shipping

The function shipping component will provide the ability to run data-centric, distributed computations directly on the storage nodes where the data resides. Additionally, the computations offloaded to the storage cluster will be resilient to errors. Therefore, the function shipping framework not only supports the shipping of functions per se, but also a runtime system capable of performing distributed computation. In return, the shipped functions must conform to a restrictive programming model.

To summarise, the outputs of the SAGE project, with regard to function shipping, are:

- Programming model for parallel, data-local computations
- Distributed runtime system capable of running such computations
- System design to ship and run computations on the data path of a distributed file system
5.2.8. Ubiquitous and Peer-Peer Cache Management

Traditional HPC clusters exhibit a rigid separation between the "compute" and "storage" islands. To utilise advanced, low-latency storage devices, it is necessary to eliminate this divide and mix storage and compute on the same node. Peer-peer caching enables this objective by treating available storage (both persistent and volatile) in the compute part of the cluster in the same manner as "server-side storage".

By replacing the classic client-server model with a more symmetrical peer-peer model, the SAGE platform will be able to efficiently utilise highly-performant layers of a deep storage hierarchy.

5.3. Clovis API Components

This section describes the capabilities and characteristics of the I/O, extension (FDMI) and management interface parameters of the Clovis API.

Core Mero functionality is extended to provide object data I/O-related functions to higher level client applications, which accesses object storage for I/O through the vertical I/O interface. Mero's functionality can be extended horizontally, using the extension interface, to provide plug-in support, as described earlier in this document. Plug-ins provide additional non-I/O functionality such as indexing, monitoring, backup, etc. We also describe the management interface that feeds ADDB records for diagnostics and monitoring.

5.3.1. Clovis I/O Interface

The I/O interface provides functions related to objects and indices for storing and retrieving data. Objects store traditional data. Indices store special data such as key-value pairs.

A Clovis object is an array of blocks. Blocks are of a power of two size bytes. Blocks are the same size for a particular object. The block size is selected when an object is created for the first time. Objects can be read from and written to at block-level granularity. Objects can be deleted at the end of their lifetime.

A Clovis index is provided through a key-value store. An index stores records in some order. Records are key-value pairs with the constraint that the keys are unique within an index. Clovis provides GET, PUT, DEL and NEXT operations on indices.

The GET operation returns matching records from an index for a given set of keys. The PUT operation writes/rewrites a given set of records. The DEL operation deletes all matching
records of an index for a given set of keys. The NEXT operation returns records corresponding to the set of next keys for a given set of keys.

5.3.2. **Clovis Extension Interface**

The Clovis extension interface provides APIs for extending the functionality of Mero. The FDMI is a distributed, publish-subscribe framework designed to generate, deliver and process events. It exposes two major interfaces: FDMI source and plug-in. This framework forms a foundation for extending Mero functionalities.

- Mero selectively exposes its internal events to its application via the FDMI framework. The richness of the events provides many opportunities to applications and can accommodate versatile functionalities.
- A stream of events is pushed to plug-ins in ‘near’ real time. This streaming mode of event delivery opens a dimension of design space for data management applications. Such applications are able to make timely decisions rather than rely on system scanning/polling to get entity states, which may have occurred in the past.
- Decoupling event generation from event processing allows applications to run as stand-alone "micro services". Interactions between applications and core Mero, via the FDMI event delivery mechanism, ease application development and also reduce interference to Mero.

A FDMI source is the entity generating events. The events can be triggered by any Mero subsystem on all or some Mero instances in a cluster. FDMI uses a “record” to represent an event. It is associated with rich runtime information, while an event is triggered, for example, by access parameters of an I/O request, performance metrics of different subsystems, updates on data and more. Examples of FDMI sources and their corresponding records include:

- File Operations Log (FOL)
- Analytics and Diagnostics DataBase (ADDB)
- Resource Manager (RM)

The FDMI framework is intended to offer a clean event source interface by hiding most of the details of event delivery and management and plug-in interactions. The FDMI source focuses on defining what and when to produce an event.
A FDMI plug-in is an event consumer that subscribes to one or multiple FDMI sources. Event records are pushed from a source, as a stream. When the records arrive in a plug-in, a plug-in handler is triggered to execute corresponding functions. The FDMI plug-in framework does not impose any restrictions on how an event is handled. An event can be processed immediately or temporarily saved into a "database" to enable the plug-in to process the event at its own pace. A plug-in handles this event, typically, by updating the cluster state through Clovis interfaces (access interface or management interface) or updating some state external to Mero (for example, as part of replication). Example plug-ins are listed below; several of them described in more detail in Section 3 (specifying plug-in requirements):

- Backup, migration, replication and HSM (already used in the project)
- Tier-management (burst buffer prefetching and de-staging), background compression, and de-duplication
- Online conversion of the pre-existing cluster state or data to a new format or a new metadata schema
- Audit and logging applications
- Preventive file system checking, RAID rebuilding and scrubbing applications
- Full-text indexing and searching applications

One feature of the FDMI framework is to allow users to subscribe to a subset of records specified by a filter. The introduction of filters sufficiently reduces the amount of data routed to a plug-in for post-processing. Mero uses a MapReduce-style algorithm to continuously monitor new records produced by sources across all Mero instances, collect records that match the filter, batch the filtered records and forward them to subscribed users. In other words, FDMI allows users to "listen" to interesting events occurring in the Mero system.

5.3.3. Clovis Management Interface

This interface will provide an API to control Mero’s cluster elements, including the Mero components, process, service, rack, enclosure, controller, etc. The API will be used to obtain ADDB records from these subsystems for diagnostics and performance analysis.
5.4. Hardware Platform

Table 2 describes the hardware components selected for the SAGE demonstration system. The SAGE strategy is to provide a very flexible architecture that incorporates emerging NVM technologies as they are developed, along with incumbent storage device tiers – to provide different “points” on the capacity-performance profile. The goal is to demonstrate all of the system’s new functionalities and capabilities rather than aim for specific performance metrics.

<table>
<thead>
<tr>
<th>Tier</th>
<th>Hardware Tier</th>
<th>Storage Technologies</th>
<th>Storage Interface</th>
<th>Storage Enclosure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local compute/GPU/visualisation</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>Bull RX</td>
</tr>
<tr>
<td>(Not part of the core SAGE storage system)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>1a</td>
<td>RAM</td>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>NVRAM 1</td>
<td>1b</td>
<td>NVRAM</td>
<td>CPU/PCIe NVMe</td>
<td>Bull Bullion S</td>
</tr>
<tr>
<td>(NVDIMM / new tech)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVRAM 2</td>
<td>1c</td>
<td>NVRAM - FLASH NVMe</td>
<td>PCIe NVMe</td>
<td>Seagate OneStor 2U24</td>
</tr>
<tr>
<td>2</td>
<td>NVRAM – FLASH 2.5” SSD</td>
<td>SAS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2: SAGE Tiers and Mapping to Physical Hardware

<table>
<thead>
<tr>
<th>Scratch or Archive (optional)</th>
<th>3 /4</th>
<th>Tier 3: Fast 3.5” SAS disk</th>
<th>Tier4: Cost optimised 3.5” SATA disk</th>
<th>SAS SATA (over SAS)</th>
<th>Seagate OneStor 5U84</th>
</tr>
</thead>
</table>

Note that Tier-0 is a physical part of the SAGE rack, but it is not a logical part of the SAGE system. Tier-0 is used to drive workloads and aid in visualisation. The Nytro NVMe solution from Seagate will be used as part of Tier-1 in the SAGE system (included in Tier1 ahead of 3DxPoint availability).

The SAGE hardware system will consist of components in two racks (shown in Figure 11). This system configuration will be initially tested at Seagate (leveraging InfiniBand between the Tiers) and connected to the compute cluster in the Juelich Supercomputing Center.

![Figure 11: SAGE Racks - Front View](image)
6. SAGE Ecosystem Components

We next discuss the ecosystem components that comprise the SAGE system and work on top of the Clovis API. The ecosystem components consist of Use Case Access Methods, Tools, Programming models and the SAGE Runtime environment (shown in Figure 7).

6.1. Use Case Access Methods

The following sections define three use case accesses for the Mero object store and Clovis API, namely, data analytics tools, parallel file system access and data format access.

6.1.1. Data Analytics Tools

Performing advanced data analytics on huge, heterogeneous data sets with high throughput and low latency is a challenge for applications used in science and in industry. In order to process data-centric computations in Exascale, researchers and analysts require novel, scalable software tools and solutions. Apache Flink[Apache-Flink] is an open-source, distributed dataflow framework for declarative analysis of Big Data using machine learning, graph analysis, signal processing, text mining and traditional database queries.

Through a new Mero storage connector, Flink obtains direct access to Mero objects. The physical co-location of Flink and Mero enables Flink's optimiser to automatically push computations close to data, thereby minimising data movements and optimising storage access. A low-latency Mero connection allows operations with very large working sets to overflow temporarily and continue normal execution.

Tightly integrating Apache Flink with Mero establishes a high-performance link between Flink's streaming and batch analytic capabilities and the data and compute-intensive applications running on Mero, which will enable novel workflows and use cases.
6.1.2. Parallel File System Access

Object storage has many advantages, but also presents one drawback: it is not the user’s “native language” for performing I/O. At this time, most HPC simulation codes use the legacy POSIX file system interface to access their data, as do many compute frameworks. Hence, maintaining a fully compliant POSIX interface to end users is critical. The parallel codes, providing performance enhancement, run on many nodes; hence, the file system interface needs to be parallel and scalable.

Many SAGE use cases do require the support of such POSIX-compliant storage access. This access is provided through the pNFS gateway built on top of Clovis, leveraging Mero’s Key Value Store infrastructure.

6.1.3. Data Format Libraries

In HPC, popular data formats, such as HDF5 and netCDF, provide scientists with dedicated libraries to describe and store their data. In these libraries, high-level data representations (data models) are translated into low-level representations (storage models) in the back-end storage system, using appropriate mapping schemas.

The typical storage model for data format libraries is the file (linear sequence of bytes organised following the POSIX file system representation, i.e. inodes and blocks). In this storage model representation, the user’s data structures are translated into a linear array of bytes in the file system. Since the data model can contain complex data hierarchies and attributes (as well as raw data) the final file will contain additional metadata that must be accessed using the POSIX-IO data interface (e.g. read() and write()) instead of the file metadata interface (e.g. stat(), lookup()). This layout makes the storage model of data format libraries incompatible with the typical parallel file system organisation, in which metadata and data are split apart and assigned to different services for optimal performance.

In the last few years, a new storage system paradigm has emerged in which files are organised in a flat namespace (e.g. object storage), removing the restrictions imposed by metadata operations, like namespace traversal, that are typical of POSIX file systems.

To better support the new storage models made available by object store systems, the HDF5 library now provides a new abstraction called the Virtual Object Layer (VOL). The VOL API allows more flexible and efficient mappings than the typical POSIX file.

SAGE directly exploits the VOL abstraction to provide a new storage model in which HDF5 data structures are mapped onto the Mero object store using the Clovis API. The new
storage model introduced in SAGE will allow for more efficient separation and management of data and metadata while keeping the same, familiar, HDF5 public API.

6.2. Programming Models

We next discuss the programming models that will be adapted to work on top of SAGE to prepare the object store for future use cases.

6.2.1. MPI I/O

MPI I/O provides high-performance parallel access to storage through versatile collective, non-blocking and even non-contiguous operations that aim to exploit the parallelism capabilities of modern HPC file systems. Its features have been widely adopted by the industry and research community. A good example of such adoption is the pHDF5 library, which is built on top of this API.

Nevertheless, although MPI I/O has proven successful enough to break the Petascale barrier over the past few years, within SAGE we consider the inclusion of new concepts and techniques that better suit the API to be necessary for the upcoming Exascale and Exabyte era. Without further changes, it is not clear whether the current MPI I/O specification will be fully capable of overcoming the performance constraints expected to arise in Extreme scale environments.
We will research the introduction of a potential layer of abstraction that may involve the specification of native object storage support within the MPI I/O standard. This layer will act in an equivalent form to traditional read and write operations for files, but consider new functionality that takes objects instead and allows for direct manipulation of metadata and data. This approach would enable high-level libraries to simplify their functionality by relying on the underlying implementation of the standard, which can be specifically optimised for the file system and provide native support for collective I/O operations (in this case, through the Clovis API of Mero (shown in Figure 12).

### 6.2.2. PGAS

Partitioned Global Address Space (PGAS) is a programming model in which the local memory of each process constitutes a shared global memory space accessible by any process, similar to accessing its local memory. In recent years, the PGAS approach has attracted increasing attention and even parallel programming languages; for instance, UPC and Chapel are fully based on the PGAS model. Furthermore, the approach has also been extended to the MPI standard through the so-called one-sided communication operations.

Recent systems, such as the one being built in SAGE, provide different tiers in the I/O system to cater for a variety of I/O requirements coming from the applications. These tiers can be characterised by their read/write access latency, the persistence capacity, the data retention period, the data volumes and other hardware-dependent metrics. However, despite previous relevant work in related topics [THoefferRSA, TrivediHPIO], most PGAS approaches do not consider extending the allocated global memory to any of these tiers, which could provide a considerable benefit to future data-centric applications.

In the SAGE programme, we are investigating the potential benefits of extending MPI one-sided communications to integrate storage support. This extended functionality would mean that different tiers of the I/O system might be exposed to the global memory space (shown in Figure 13), allowing the applications to direct different portions of the data to the tier with desirable characteristics. More specifically, we are developing a prototype that provides a seamless extension of MPI one-sided communications by exploiting the benefits of the Performance Hints interface of the standard. Our current prototype allows different types of MPI Window allocations, enabling processes to utilise storage-based allocations alongside traditional memory-based. This fact implies that processes could simultaneously combine both types of allocations as well, meaning hybrid persistent / non-persistent MPI Window objects within the same communicator.
The implications of such a novel approach are expected to be very relevant for current and future Exascale systems. In SAGE we are committed to determine its wide-range of possibilities, and applications such as tight process coupling, Big Data analytics, fault tolerance and more are currently being explored.

As a consequence, we are investigating in SAGE the potential benefits of extending MPI with shared global memory, targeting the desired storage tier (e.g. NVRAM 1 (e.g., STT RAM), NVRAM 2 (e.g., NAND Flash), High-Performance Scratch Storage or Archival Storage).

Recent systems, such as the one being built in SAGE, provide different tiers in the I/O system to cater for a variety of I/O requirements coming from the applications. These tiers of the I/O system might be exposed to the global memory space (Figure 12), allowing the operating system and the allocated address will be given to the MPI implementation to pass these hints to the specific MPI implementation and our prototype library will intercept the Window object. Lastly and independently of the type of allocation, an Allocation Attribute will be transparently attached to the returned object in order to later understand how to deallocate the associated Window.

Figure 13 depicts the simplified flow diagram when an MPI process (inside an application) allocates an MPI Window to be used following the MPI one-sided communications model. The example illustrates the use case in which this allocation is based on persistent storage, specified through the definition of the Performance Hints for the MPI implementation. This structure is critical, as it will determine where to allocate the MPI Window by a given hint targeting the desired storage tier (e.g. the NVRAM tier of the SAGE system). The process will pass these hints to the specific MPI implementation and our prototype library will intercept each allocation call to analyse its purpose. If the request is aiming for a conventional memory-based allocation, then the library will act as pass-through and rely completely on the MPI implementation; otherwise, a mapping in storage will be defined with the help of the operating system and the allocated address will be given to the MPI implementation to create the Window object. Lastly and independently of the type of allocation, an Allocation Attribute will be transparently attached to the returned object in order to later understand how to deallocate the associated Window.

Figure 14: Flow diagram of an MPI Window allocation using storage.
The implications of such a novel approach are expected to be very relevant for current and future Exascale systems. In SAGE, we are committed to determining the wide range of possibilities for this method and exploring applications such as tight process coupling, Big Data analytics, fault tolerance and more.

6.3. Tools

In the following sections, we discuss the tools that will leverage the SAGE system.

6.3.1. Allinea Performance Tools

Allinea performance tools are used to build time-dependent and summary profiles of program runs on large HPC systems. A programmer can use these profiles as part of an optimisation process to increase performance and reduce a program’s time to solution. The profiles also act as a record of how compiled code was executed on a particular system with specific hardware and software stacks. These records may also be used to investigate the performance of the underlying software and hardware components.

In SAGE, Allinea performance tools will be extended with a focus on recording data regarding the use of the memory hierarchy and the SAGE architecture. Users will be able to observe access patterns involving different levels in the memory hierarchy. Programmers will be able to use this data to target inefficient I/O patterns in their code, enabling faster memory technologies to be used. The generated profiles can be applied to evaluate performance increases between runs in which in-storage compute is in use and when it is not (i.e. when the SAGE architecture is fully utilised and when it is not).

The developed tools will directly benefit the evaluation of the SAGE project, and also allow programmers to make immediate use of the novel SAGE architecture, providing valuable insights for researchers aiming to scale their problems to ever larger systems and remain at the forefront of Extreme scale research.
6.3.2. HSM

In Exascale-class systems, the fastest storage technologies, like NVRAM, must be exploited to fill the increasing gap between compute power and traditional storage based on hard drives disks. However, fast media remain expensive, so they cannot be the only technologies considered when designing cost-effective storage solutions.

HSM makes it possible to design both high-performance and cost-effective storage architectures, by combining various storage media (NVRAM, SSDs, disks, etc.) and leveraging the strengths of each of these technologies. In such a system, the fastest technologies are used to provide applications with high throughputs and I/O rates, but they only represent a small storage capacity. Slower technologies, such as archive disks, are much cheaper, which allows huge capacities to be acquired. Multiple technologies can be combined, but not be at the expense of increased complexity for user applications.

HSM aims to provide transparent management of the different storage levels, by providing users with a storage system experience modelled on a fast, capacitive single technology. HSM directs application data to the most appropriate media technology and triggers data migration between tiers to ensure that space is always available in the fast level, despite limited capacity. Unused data is offloaded to capacitive storage levels until an application later requires fast access to it. In this case, the data is loaded back to a fast level to speed up subsequent application accesses.

Unlike existing HSM solutions, SAGE aims to manage deep storage hierarchies that consist of three or more tiers. It also brings innovations in the field of lifecycle management and policy engines, by enabling smart and auto-adaptive data management policies.
6.3.3. Integrity Checking

Data integrity refers to maintaining and assuring the accuracy and consistency of data over its entire lifecycle. Data integrity is a critical aspect to the design, implementation and usage of a system that stores, processes, or retrieves data. The term “data integrity” may have widely different meanings depending on the specific level in the distributed storage stack and its context: from disk drives at the lowest level to a local file system/database and distributed storage system.

- A single device is logically viewed as a collection of data units (blocks). The main purpose of data integrity for a device is to prevent any unintended change to any data block.
- A file system further assigns different roles to data blocks and the contents have varied semantics. Some blocks store file inodes, while others may contain directory information used to look up a file. The richness of data block semantics given by a file system is used to keep file system in a consistent state.
- Some distributed storage systems exploit networked de-clustered parity to organise data blocks across the cluster. Data integrity must guarantee those blocks in a parity group match via computing and comparing parity.

Many data integrity techniques have been proposed to tackle the data integrity issues at different levels of the storage stack. We identify key areas in the entire stack that are considered critical to maintain data integrity based on current support from the Mero implementation. A service to scan the storage system (runtime or offline) to ensure the consistency between metadata and object data is an important example.

Considering existing techniques in an Exascale context, those techniques fail to efficiently deliver object integrity checking. Existing file system and distributed storage system checkers (like Ceph scrubbing and Openstack Swift’s Auditor[Openstack-Swift]) require all storage devices to be scanned in one or more rounds. Such checkers may run extremely slowly and they recover a system in a timespan proportional to the system size. Unlike these checkers, we propose a runtime integrity checking service, which constantly maintains the necessary auxiliary information as part of normal operations to avoid time-consuming device scanning.
6.4. SAGE Runtimes

To further exploit SAGE’s in-storage processing capabilities, a runtime system is implemented that allows for flexible deployment of services into the storage. The goal is to offload efficient data management to the storage and to realise data-centric workflows.

This approach is strictly co-design driven, based on use cases that involve large data sets that have been chosen as representative of different classes of applications. The use cases differ in terms of the source of the data that is being processed. Data can, for instance, be produced by large-scale simulations running on an attached compute system and require post-processing before being archived. An opposite dataflow is found when processing observatory data that is first ingested into the archival tier for later processing by an HPC system. This is the case when processing data from earth monitoring satellites. During multi-year measurement campaigns, large amounts of data are accumulated and significant compute resources are required to process a vast numbers of data objects. Traditionally, processing is organised in a compute-centric manner in which data read requests are sent to the storage once compute resources are available. Within SAGE, a data-centric approach will be realised in which storage makes data available in a semi-persistent cache and compute node processes will process data once it is available in a fast storage tier.

A different approach is taken to enable access to large data volumes for visualisation facilities. By implementing a software-managed cache, hot data is moved to the fast storage tier allowing for processing steps to adjust the level of detail. This method will allow for a seamless memory hierarchy reaching from a SAGE archival storage tier, which is able to hold petabytes of data, to graphics processors with gigabytes of attached high-bandwidth memory.
7. Conclusion

When available, the SAGE system will be one of the very first storage systems with features designed using ground-up requirements gathering and co-design that specifically addresses the overlap of Extreme scale computing and Big Data. Through the Clovis API, the SAGE platform has already created the opportunity and, indeed, the provision of proof points, for the implementation of a wide repertoire of top-level solutions, as described in this paper.

SAGE has the potential to establish an exemplar blueprint for an effective storage platform for Exascale, even for applications that perform most of their processing in the compute nodes; an important consideration as I/O rate demands will continue to increase. Stakeholders will need to rethink the approach of existing parallel file systems, which were designed for sub-Petascale environments, and the current state of the art storage solutions, which will not be able to exploit the continuing evolution of storage device technologies.

SAGE can blur the lines between traditional memory and storage utilisation in future HPC infrastructures, leading to tangible benefits for many applications and users moving toward Exascale who, today, are severely memory- or I/O-bound.

Storage has always been seen as an isolated subsystem peripheral to a central compute engine. SAGE hopes to change that viewpoint and pave the way for infrastructures that will ultimately remove the boundaries between traditional compute and data storage, enabling more efficient and performant systems.

We intend the SAGE project to foster an ecosystem of tools and techniques required to support the future needs of BDEC, HPDA, and traditional HPC use cases, and extend further to better support the increasing overlap between the Big Data and HPC communities. We will endeavour to publish more information on SAGE’s milestones and progress in future forums, blogs, and technical conferences.
8. References


[EnergySavings] Steve Conway and Chirag DeKate(IDC), High Performance Data Analysis HPC meets Big Data, March 2013


### Bibliography

- **[REST]** Leonard Richardson and Sam Ruby, RESTful Web Services, O’Reilly Media, December 2008
- **[SKA]** C. Broekema et al., “Exascale High Performance Computing in the Square Kilometer Array”, AstroHPC’12, June 2012
## 9. Glossary of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDB</td>
<td>Analytics and Diagnostics DataBase</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BDEC</td>
<td>Big Data Extreme Compute</td>
</tr>
<tr>
<td>DTM</td>
<td>Distributed Transaction Manager</td>
</tr>
<tr>
<td>FDMI</td>
<td>File Data Manipulation Interface</td>
</tr>
<tr>
<td>FOL</td>
<td>File Operations Log</td>
</tr>
<tr>
<td>HA</td>
<td>High Availability</td>
</tr>
<tr>
<td>HPDA</td>
<td>High Performance Data Analytics</td>
</tr>
<tr>
<td>HSM</td>
<td>Hierarchical Storage Management</td>
</tr>
<tr>
<td>KVS</td>
<td>Key-Value Store</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface</td>
</tr>
<tr>
<td>NBA</td>
<td>Non-Blocking Availability</td>
</tr>
<tr>
<td>PGAS</td>
<td>Partitioned Global Address Space</td>
</tr>
<tr>
<td>RAID</td>
<td>Redundant Array of Independent Disks</td>
</tr>
<tr>
<td>RM</td>
<td>Resource Manager</td>
</tr>
<tr>
<td>SAGE</td>
<td>Percipient StorAGe for Exascale Data Centric Computing</td>
</tr>
<tr>
<td>SNS</td>
<td>Server Network Striping</td>
</tr>
<tr>
<td>STOB</td>
<td>Storage Object</td>
</tr>
<tr>
<td>VOL</td>
<td>Virtual Object Layer</td>
</tr>
</tbody>
</table>