# The SAGE Project: a Storage Centric Approach for Exascale Computing

Invited Paper

Sai Narasimhamurthy
Seagate Systems UK

Nikita Danilov
Seagate Systems UK

Sining Wu
Seagate Systems UK

Ganesan Umanesan
Seagate Systems UK

Steven Wei Der Chien
KTH Royal Institute of Technology

Sergio Rivas-Gomez
KTH Royal Institute of Technology

Ivy Bo Peng
KTH Royal Institute of Technology

Erwin Laure
KTH Royal Institute of Technology

Shaun de Witt
Culham Center for Fusion Energy

Dirk Pleiter
Jülich Supercomputing Center

Stefano Markidis
KTH Royal Institute of Technology

## ABSTRACT

SAGE (Percipient **S**tor**AG**e for **E**xascale Data Centric Computing) is a European Commission funded project towards the era of Exascale computing. Its goal is to design and implement a Big Data/Extreme Computing (BDEC) capable infrastructure with associated software stack. The SAGE system follows a *storage centric* approach as it is capable of storing and processing large data volumes at the Exascale regime.

SAGE addresses the convergence of Big Data Analysis and HPC in an era of next-generation data centric computing. This convergence is driven by the proliferation of massive data sources, such as large, dispersed scientific instruments and sensors where data needs to be processed, analyzed and integrated into simulations to derive scientific and innovative insights. A first prototype of the SAGE system has been been implemented and installed at the Jülich Supercomputing Center. The SAGE storage system consists of multiple types of storage device technologies in a multi-tier I/O hierarchy, including flash, disk, and non-volatile memory technologies. The main SAGE software component is the Seagate Mero Object Storage that is accessible via the Clovis API and higher level interfaces. The SAGE project also includes scientific applications for the validation of the SAGE concepts.

The objective of this paper is to present the SAGE project concepts, the prototype of the SAGE platform and discuss the software architecture of the SAGE system.

## KEYWORDS

SAGE Project, Storage Centric Computing, Exascale Computing

*CF '18, May 8–10, 2018, Ischia, Italy*
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5761-6/18/05…$15.00
https://doi.org/10.1145/3203217.3205341

**ACM Reference Format:**
Sai Narasimhamurthy, Nikita Danilov, Sining Wu, Ganesan Umanesan, Steven Wei Der Chien, Sergio Rivas-Gomez, Ivy Bo Peng, Erwin Laure, Shaun de Witt, Dirk Pleiter, and Stefano Markidis. 2018. The SAGE Project: a Storage Centric Approach for Exascale Computing: Invited Paper. In *CF '18: CF '18: Computing Frontiers Conference, May 8–10, 2018, Ischia, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3203217.3205341

## 1 INTRODUCTION

The next-generation Exascale supercomputers will be available to the community sometime in the 2021-2023 timeframe and it will be capable of delivering up to $10^{18}$ floating point operations per seconds to support traditional HPC compute-intensive applications. With the emergence of new HPC data centric applications, such as workflows and data analytics workloads, the definition of Exascale computing is now broadening to include storage and processing of an order of an exabyte of data. In fact, Big Data Analysis and HPC are converging as massive data sets, such as very large volumes of data from scientific instruments and sensors, needs to be processed, analyzed and integrated into simulations. For this reason, we envision that Exascale supercomputers will be capable to be exploited by applications and workflows for science and technological innovation.

Computing infrastructure innovation has been driven by Moore's law and the development of even more parallelism with multi-core, many-core and accelerator processing to accommodate the increasing performance requirements of Exascale. However I/O and storage have lagged far behind in computational power improvement. Storage performance in the same time period is predicted to have improved for only 100 times, according to early estimates provided by Vetter et al. [28]. In fact, at the time of publication of this work, the performance of disk drives per unit capacity is actually decreasing with new very high capacity disk drives on the horizon [28]. 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 [15, 19]. The optimal use of these devices (starting with flash) in the I/O hierarchy, combined with existing disk technology, is only beginning to be explored in HPC [4] with burst buffers [10].

The SAGE project is a European Commission funded project to investigate Exascale computing [14]. It supports a storage centric view of Exascale computing by proposing hardwares to support multi-tiered I/O hierarchies and associated software. They provide a demonstrable path towards Exascale. Further, SAGE proposes a radical approach in extreme scale HPC by moving traditional computations, typically done in the compute cluster, to the storage system. This has the potential of significantly reducing the energy footprint of the overall system [22].

The primary objective of this paper is to present the initial hardware and software architecture of the SAGE system. The paper is organized as follows. Section 2 presents the initial development of the SAGE platform. Section 3 describes the SAGE platform architecture and software stack. Section 5 discusses the related work. Finally, Section 6 summarizes the paper and outlines the future work.

## 2 A STORAGE CENTRIC ARCHITECTURE

The SAGE storage system developed by the SAGE consortium provides a unique paradigm to store, access and process data in the realm of extreme-scale data centric computing.

The SAGE platform consists of multiple tiers of storage device technologies. At the bottom of the stack is the *Unified Object-Based Storage Infrastructure*. The system does not require any specific storage device technology type and accommodates upcoming NVRAM, existing flash and disk tiers. For the NVRAM tier, we are using Intel 3D XPoint technology [2] in our *Tier-1*. We will also use emulated NVDIMMs (Non-Volatile DIMMs) in Tier-1 because of the lack of NVDIMM availability in vendor roadmaps. We are using Flash based solid state drives in *Tier-2*. Serial Attached SCSI high performance drives are contained in *Tier-3* and archival grade, high capacity, slow disks ( based on Serial ATA and Shingled Magnetic Recording) are contained in *Tier-4*. These tiers are all housed in standard form factor enclosures that provide their own computing capability through standard x86 embedded processing components, which are connected through an FDR Infiniband network. Moving up the system stack, compute capability increases for faster and lower latency device tiers. The storage system is also capable to perform computation in storage (either through a function shipping interface or a run-time supporting, e.g.,pre-/post-processing of data) on behalf of the applications. This avoids the need to move data back and forth between compute subsystems and storage subsystems as in a typical HPC cluster.

A SAGE prototype system has been developed by Seagate and other SAGE consortium partners and is installed at the at the Jülich Supercomputing Center. Figure 3 shows the SAGE prototype consisting of the four tiers:

- Tier-1: PCIe-attached NVMe SSDs based on NAND Flash or 3D XPoint memory
- Tier-2: SAS-attached SSDs based on NAND Flash
- Tier-3: High performance disks
- Tier-4: Archival grade disks.

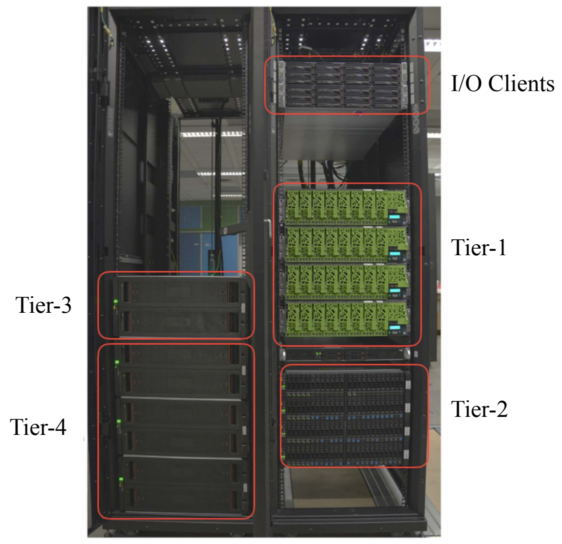

**Figure 1: SAGE supercomputer prototype that has been installed at the Jülich Supercomputing Center in 2017.**

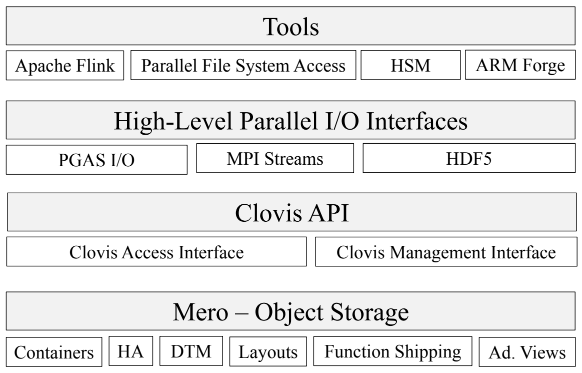

**Figure 2: SAGE Software Stack.**

## 3 SAGE SOFTWARE STACK

Together with the development of SAGE storage centric prototype platform, we designed and developed software capable of taking advantage of the SAGE infrastructure. A simplified diagram of the SAGE software stack is presented in Figure 2.

### 3.1 Mero

Mero is at the base of the SAGE software stack, providing the Exascale-capable object storage infrastructure that drives the storage hardware [3]. Mero is a distributed object storage platform. Lustre file system, NFSv4 and database technology are the the main sources of inspiration for the Mero design. An emerging consensus is that traditional file system properties (hierarchical directory namespace, strong POSIX consistency guarantees, etc) pose a major obstacle for the performance and scalability of Exascale systems.

Mero controls a cluster of nodes connected by a network. Some nodes have persistent storage attached to them. Mero distinguishes various types of persistent stores including rotational drives, SAS-attached non-volatile memory devices and PCIe-/memory bus attached non-volatile memory devices. Mero presents a collection of services to applications.

Mero Object store has a "core" providing - scalable re-writable fault-tolerant data objects, Index store with scalable key-value indices, and, resource management capabilities for caches, locks, extents, etc.

*Extreme scale* features such as containers and function shipping are built on top of the core. Mero also features a Distributed transaction manager, which makes it possible to use other services in a consistent manner in the face of hardware and software failures, which is also an extreme scale feature on top of the core.

Mero also has an Extension mechanism which allows the addition of new functionalities without modification to the core, called the FDMI, or File Data Manipulation Interface. New services such as Information lifecycle management (ILM), indexing, search, etc can hence be built as extensions by third parties leveraging the core.

A Mero application can be a traditional user space application, running standalone on a node, a large MPI job running on multiple nodes, a cluster management utility monitoring the state of the system, or a NFS/CIFS daemon (a front-end) exporting Mero objects to non-Mero clients, etc.

**Container abstraction.** Containers are the main way of grouping objects in various ways. They virtualize the object name space by providing the capability to label objects in various ways. There can be containers based on data formats (for example, HDF5 containers, etc) and containers based just on performance (for example, high performance containers which are mapped to objects in higher tiers, etc). It is possible to do operations such as function shipping, pre/post processing on a given containers.

**High Availability (HA) System.** Existing systems and data [25] indicate that we can expect many hardware failures per second at Exascale, in addition to software failures resulting in crashed nodes. To maintain service availability in the face of expected failures, a global state (or configuration) of the cluster is maintained. This may need to be modified by the means of repair procedures in response to failure events. The HA subsystem for SAGE will perform such automated repair activities within storage device tiers. The HA subsystem thus monitors failure events (inputs) throughout the storage tiers and then decides to take action based on collected events.

**Distributed Transaction Management (DTM).** In Mero, all I/O and metadata operations are, ultimately, organized into transactions. Transactions are atomic with respect to failures. In other words, either all or none of the updates corresponding to a transaction are visible to other users. This property is known as atomicity. A related property is failure atomicity, i.e. either all or none of the updates corresponding to a transaction survive a failure. A group of updates that must atomically survive a failure is called a distributed transaction.

Mero implements a Distributed Transaction Manager (DTM) that guarantees efficient management of system state consistency in an environment in which dependent data and metadata are scattered over multiple nodes to provide fault tolerance and scalability. DTM provides the necessary interface to group a collection of object store updates into a distributed transaction and guarantees that, in the event of a server node failure and restart, the effects of distributed transactions that have updates for the affected server are either completely restored after restart or completely eliminated.

**Layouts.** A layout is a mapping of different parts or regions of an object to storage tiers. Each object has a layout attribute that defines how the object is stored in the cluster. Mero provides a flexible, hierarchical description to map object subcomponents to physical locations, or storage tiers. This mapping allows for compact formulaic expressions, as well as data transformations, such as erasure coding, de-duplication, encryption and compression. Layouts also describe data redundancy models, like simple replication or Server Network Striping. As an example of a layout, an object can have some extents mapped to Tier-1, other extents mapped to Tier-2 and a few others mapped to Tier-3. Further, each of set of extents mapped to a certain tier can have its own "sub-layout".

**Function Shipping.** Function shipping in Mero provides the ability to run application functions directly on storage nodes. This addresses one of the big bottlenecks foreseen for Exascale systems, which is the overhead of moving data to computations. Indeed moving very large quantities of data from storage to compute is extremely energy intensive and energy is one of the prime candidates to address for Exascale systems. Well defined functions within the use cases are registered on the storage nodes and are invoked by the use cases using remote procedure calls. Function shipping is accomplished through extensions of the Mero Clovis API (see next section).

**Advanced Views.** Modern storage systems are increasingly heterogeneous. This means not only multitude of data sources and data formats, but also the multitude of applications accessing shared data sets with different access patterns, multitude of various storage policies (retention, access control, provenance, etc.) and multitude of conflicting goals that the storage system must balance (latency vs. throughput, different storage hardware). Historically, providing an interface that allowed different applications to access shared data often resulted in great benefits both for application and system developers, the two most famous examples being UNIX "everything-is-a-file" and relational models. Data stored in Mero are accessible to multiple applications using various existing storage interfaces (e.g., POSIX, pNFS, S3, HDF5). A component of Mero called *Lingua Franca* (LF) implements common meta-data formats and interfaces that enables interoperability between multiple external interfaces and internal meta-data users.

LF is a mechanism to share the same sets of storage entities (objects, indices and containers) between multiple applications with different access interfaces. Its straightforward use is to provide interoperability between different front-ends. Together with other Mero features, like containers and function shipping, it can be used to implement a very flexible access arrangement to the data.

## 3.2 Clovis

Mero's application programming interface (API), known as Clovis, provides a library of functions that applications and front-end programs can use for accessing and manipulating storage resources

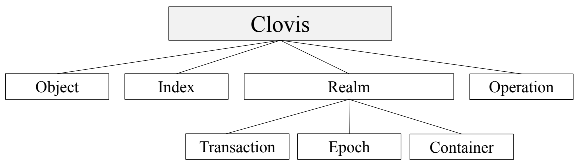

**Figure 3: Clovis abstractions.**

in Mero. Access to storage resources by outside applications is strictly controlled via Clovis; no other interfaces exist. The Clovis API contains optimized functions to manage performance and scalability for modern extreme scale computing applications as well as legacy applications. We expect higher-level applications, as part of development work related to accessing Mero storage resources, to build their own APIs on top of Clovis. In other words, we do not expect computational scientists to directly interface with the Clovis interface. Higher-level interfaces include HDF5, POSIX via pNFS, and others.

The Clovis API is implemented in the C programming language, but equivalent versions of the API are planned in other popular programming languages such as Java, Python, etc. The Clovis API supports asynchronous transactions, i.e. an application can continue after starting a transaction and only later check for completion of the transaction. For clarity, the API is divided into three sub-APIs: the Clovis Access API, the Clovis Management API and the Clovis Extension API.

Clovis provides the following abstractions, shown in Figure 3. Short descriptions of the abstractions are provided below:

- Object is an array of fixed size blocks of data
- Index is a key-value store
- An entity is an object or an index
- Realm is a spatial and temporal part of the system with a prescribed access discipline
- Operation is a process of querying and/or updating the system state
- Objects, indices and operations live in realms
- Transaction is a collection of operations that are atomic in the face of failure
- Epoch is a collection of operations done by an application that moves the system from one application-consistent state to another
- Container, in the Mero context, is a collection of objects used by a particular application or group of applications

The Clovis Access API handles all I/O related functions for applications. In addition to standard initialization and finalization functions related to running a Clovis instance, the API provides create(), write(), read(), and free() functions that enable applications to create Mero objects, transfer data to and from them, and then delete them (completely freeing the resources the objects use).

The Clovis Management API handles all management-related functions in Mero, including system configuration, management of services, and analytics as well as diagnostics functions.

The Clovis Extension API provides a list of compute functions that support the development of third-party Mero plug-ins without modifying the core. The FDMI (File Data Manipulation Interface) is an example for how to use this feature.

## 3.3 Higher-Level I/O Interfaces

At the top of the software stack, we further develop widely-used HPC legacy APIs, such as MPI and HDF5, to exploit the SAGE architecture.

**PGAS I/O.** The goal of the Partitioned Global Address Space (PGAS) programming model is to provide processes with a global view of the memory and storage space during the execution of a parallel application. This is similar to what a Shared Memory model provides in a multithreaded local environment. In the PGAS approach, remote processes from different nodes can easily collaborate and access memory addresses through load / store operations that do not necessarily belong to their own physical memory space. In SAGE, we propose an extension to the MPI one-sided communication model to support window allocations in storage: MPI storage windows [23, 24]. Our objective is to define a seamless extension to MPI to support current and future storage technologies without changing the MPI standard, allowing to target either files (i.e., for local and remote storage through a parallel file system) or alternatively address block devices directly (i.e., as in DRAM). We propose a novel use of MPI windows, a part of the MPI process memory that is exposed to other MPI remote processes, to simplify the programming interface and to support high-performance parallel I/O without requiring the use of MPI I/O. Files on storage devices appear to users as MPI windows (MPI storage windows) and seamlessly accessed through familiar PUT and GET operations. Details about the semantics of operations on MPI storage windows and the implementation are provided in Ref. [23].

**MPI Streams for Post-Processing and Parallel I/O.** While PGAS I/O library addresses the challenge of heterogenous storage and memory, streams can be used to support function-shipping for post-processing and highly scalable parallel I/O. *Streams* are a continuous sequence of fine-grained data structures that move from a set of processes, called data *producers*, to another set of processes, called data *consumers*. These fine-grained data structures are often small in size and in a uniform format, called a *stream element*. A set of computations, such as post-processing and I/O operations, can be *attached* to a data stream. Stream elements in a stream are processed *online* such that they are discarded as soon as they are *consumed* by the attached computation.

In particular, our work in SAGE focuses on *parallel streams*, where data producers and consumers are distributed among processes that require communication to move data. To achieve this, we have developed a stream library, called MPIStream, to support post-processing and parallel I/O operations on MPI consumer processes [13, 16, 17]. More details about MPI streams operation semantics and MPIStream implementation are provided in Ref. [18].

**HDF5.** Typically, data formats in HPC provide their own libraries to describe data structures and their relations (including I/O semantics). The HDF5 data format needs to be supported in SAGE, and is layered directly on top of Clovis. The HDF5 will use the Virtual Object Layer Infrastructure within HDF5 (used to interface HDF5 with various object formats), to interface with Clovis.

### 3.4 Tools

A following are a set of tools for I/O profiling and optimized data movement across different SAGE platform tiers at the top of the SAGE software stack.

**Data Analytics Tools.** Apache Flink is a framework for data analytics workloads. Flink connectors for Clovis are currently under development to enable the deployment of data analytics jobs on top of Mero.

**Parallel File System Access.** Parallel file system access is the traditional method of accessing storage in HPC. Many of the SAGE applications and use cases need the support of POSIX compliant storage access. This access is provided through the pNFS gateway built on top of Clovis. However, pNFS requires some POSIX semantics to be developed by leveraging Mero's KVS. For instance, an abstraction of namespaces on top of Mero objects is needed.

**Hierarchical storage management and Data Integrity.** In SAGE, an Hierarchical Storage Management (HSM) is used to control the movement of data in the SAGE hierarchies based on data usage. Advanced integrity checking overcomes some of the drawbacks of well known and widely used file system consistency checking schemes.

**ARM Forge.** ADDB telemetry records from the Clovis management interface are directly fed to ARM Forge performance report tools that reports overall system performance for SAGE.

## 4 VALIDATION WITH APPLICATIONS

As seen in the previous section, the SAGE platform supports appropriate scientific computing data formats and legacy application interfaces such as parallel file systems and POSIX. SAGE also needs to interface with emerging big data analytics applications (on top of the API) to access the rich features of these tools, and the Volumes, Velocity and Variety (potentially) of data coming from sensors, instruments and simulations. We have created a portfolio of scientific data-centric applications that have been used to provide requirements to the development of the SAGE system and to validate the developments in the projects. The applications we chose for the SAGE project are:

- iPIC3D is a parallel Particle-in-Cell Code for space physics simulations in support of NASA and ESA missions [12, 20, 21].
- NEST is a spiking neural network simulator to study brain science [7].
- Ray is a parallel meta-genome assembler [1].
- JURASSIC is a fast radiative transfer model simulation code for the mid-infrared spectral region [8].
- EFIT++ is a plasma equilibrium fitting code with application to nuclear fusion [11].
- The ALF code performs analytics on data consumption log files
- Spectre is a visualization tool providing near real time feedback on plasma and other operational conditions in fusion devices.
- Savu is a code for tomography reconstruction and processing pipeline [29].

## 5 RELATED WORK

To the best of our knowledge SAGE is the first HPC-enabled storage system to implement new NVRAM tiers, flash and disk drive tiers as part of a single unified storage system. The SAGE Architecture progresses the state of the art from Blue Gene Active Storage [6] and Dash [9], which use flash for data staging. It also progresses the state of the art from Burst Buffer technologies as discussed earlier.

When compared to the FastForward Project [4], SAGE highly simplifies storage, developing a solution for deep I/O hierarchies, including NVRAM technologies. A major difference between SAGE and FastForward is the FastForward solution is evolutionary as it tries to make use of an existing storage solution, namely, Lustre [26] used for the last 20 years or so. Lustre was really designed for the previous era, when use cases and architectural assumptions were different. On the hand, SAGE and Mero are the product of a complete redesign in consideration of the new requirements arising out of the Extreme scale computing community.

Mero, the object store in SAGE, extends the state of the art in existing object storage software such as Ceph [30] and Open Stack Swift [27] by building Exascale components required for extreme scale computing. While Ceph and Openstack swift are designed for supporting mainly cloud storage, Mero is built to meet the needs of the extreme scale computing community.

## 6 CONCLUSIONS

The SAGE project objective is to design and implement an I/O system capable of supporting I/O workloads of Exascale supercomputers. The SAGE platform has been recently installed at Jülich Computing center. It supports a multi-tiered I/O hierarchy and associated software stack, to provide a demonstrable path towards Big Data analytics and HPC convergence. The SAGE software stack consists of four main software layers: the Seagate Mero object-storage, the Clovis API, high-level interfaces and tools.

Current ongoing work focuses on the performance characterization of various new NVRAM device technologies. We also currently investigating lower level software and Operating System (OS) infrastructure requirements to exploit these new devices types, below Mero in the SAGE stack. We clearly recognize that various NVRAM technologies have their own performance characteristics and limitations. New NVRAM technologies can be part of the SAGE hardware tiers based on where they ultimately are on the performance and capacity curve. The SAGE stack and Mero indeed is designed to be agnostic of storage device types as long as adaptations are in place within the OS.

The next steps will be to quantify the benefits of the various features of the SAGE stack on the SAGE prototype system currently installed at Jülich Supercomputing Center, with focus on providing results for the remaining SAGE components and the SAGE architecture as a whole. As a part of this external organizations outside of the SAGE consortium ( eg: from Climate and Weather, Astronomy, etc) will soon be granted access to study how their codes and workflows can exploit the features of the SAGE platform. We will then look at extrapolation studies of the benefits of the various SAGE features at Exascale through analytical and simulation models. These will be discussed separately. Porting of the SAGE stack across other sites and extensions of the SAGE prototype is also planned. We are targeting SAGE work to be a part of European Extreme Scale Demonstrators [5] which will be pre-Exascale prototypes.

## ACKNOWLEDGEMENTS

The authors acknowledge that the SAGE work is being performed by a consortium of members consisting of Seagate (UK), Bull ATOS (France), ARM (UK), KTH (Sweden), STFC (UK), CCFE (UK), Diamond (UK), DFKI (Germany), Forchungszentrum Jülich (Germany) and CEA (France) - which is being represented by the Authors in this paper.

Funding for the work is received from the European Commission H2020 program, Grant Agreement No. 671500 (SAGE).

## REFERENCES

[1] Sébastien Boisvert, Frédéric Raymond, Élénie Godzaridis, François Laviolette, and Jacques Corbeil. 2012. Ray Meta: scalable de novo metagenome assembly and profiling. *Genome biology* 13, 12 (2012), R122.

[2] Katherine Bourzac. 2017. Has Intel created a universal memory technology?[News]. *IEEE Spectrum* 54, 5 (2017), 9–10.

[3] Nikita Danilov, Nathan Rutman, Sai Narasimhamurthy, and John Bent. 2016. Mero: Co-Designing an Object Store for Extreme Scale. In *First Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS)*.

[4] DoE. 2014. DoE Fast Forward Program Documents. https://wiki.hpdd.intel.com/display/PUB/Fast+Forward+Storage+and+IO+Program+Documents. (2014). [Online; accessed Nov 2017].

[5] ETP4HPC. 2018. Extreme Scale Demonstrators. http://www.etp4hpc.eu/esds.html. (2018). [Online; accessed Feb 2018].

[6] Blake G Fitch, A Rayshubskiy, MPTJ Chris Ward, B Metzler, HJ Schick, B Krill, P Morjan, and RS Germain. 2010. Blue gene active storage. *Proc. HEC FSIO* (2010).

[7] Marc-Oliver Gewaltig and Markus Diesmann. 2007. Nest (neural simulation tool). *Scholarpedia* 2, 4 (2007), 1430.

[8] Sabine Griessbach, Lars Hoffmann, Michael Höpfner, Martin Riese, and Reinhold Spang. 2013. Scattering in infrared radiative transfer: A comparison between the spectrally averaging model JURASSIC and the line-by-line model KOPRA. *Journal of Quantitative Spectroscopy and Radiative Transfer* 127 (2013), 102–118.

[9] Jiahua He, Arun Jagatheesan, Sandeep Gupta, Jeffrey Bennett, and Allan Snavely. 2010. Dash: a recipe for a flash-based data intensive supercomputer. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 1–11.

[10] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. 2012. On the role of burst buffers in leadership-class storage systems. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 1–11.

[11] I Lupelli. 2015. The Efit++ Equilibrium Code: Recent Upgrades And Applications To Air-Core And Iron-Core Machines. In *1st EPS conference on Plasma Diagnostics (ECPD2015)*.

[12] Stefano Markidis, Giovanni Lapenta, and Rizwan-uddin. 2010. Multi-scale simulations of plasma with iPIC3D. *Mathematics and Computers in Simulation* 80, 7 (2010), 1509–1519.

[13] Stefano Markidis, Ivy Bo Peng, Roman Iakymchuk, Erwin Laure, Gokcen Kestor, and Roberto Gioiosa. 2016. A performance characterization of streaming computing on supercomputers. *Procedia Computer Science* 80 (2016), 98–107.

[14] Sai Narasimhamurthy, Nikita Danilov, Sining Wu, Ganesan Umanesan, Stefano Markidis, Sergio Rivas-Gomez, Ivy Bo Peng, Erwin Laure, Dirk Pleiter, and Shaun de Witt. 2018. SAGE: Percipient Storage for Exascale Data Centric Computing. *Parallel Comput.* (2018).

[15] Ivy Bo Peng, Roberto Gioiosa, Gokcen Kestor, Pietro Cicotti, Erwin Laure, and Stefano Markidis. 2017. Exploring the performance benefit of hybrid memory system on HPC environments. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*. IEEE, 683–692.

[16] Ivy Bo Peng, Roberto Gioiosa, Gokcen Kestor, Erwin Laure, and Stefano Markidis. 2017. Preparing HPC Applications for the Exascale Era: A Decoupling Strategy. In *Parallel Processing (ICPP), 2017 46th International Conference on*. IEEE, 1–10.

[17] Ivy Bo Peng, Stefano Markidis, Roberto Gioiosa, Gokcen Kestor, and Erwin Laure. 2017. MPI Streams for HPC Applications. *New Frontiers in High Performance Computing and Big Data* 30 (2017), 75.

[18] Ivy Bo Peng, Stefano Markidis, Erwin Laure, Daniel Holmes, and Mark Bull. 2015. A data streaming model in MPI. In *Proceedings of the 3rd Workshop on Exascale MPI*. ACM, 2.

[19] Ivy Bo Peng, Stefano Markidis, Erwin Laure, Gokcen Kestor, and Roberto Gioiosa. 2016. Exploring application performance on emerging hybrid-memory supercomputers. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*. IEEE, 473–480.

[20] Ivy Bo Peng, Stefano Markidis, Andris Vaivads, Juris Vencels, Jorge Amaya, Andrey Divin, Erwin Laure, and Giovanni Lapenta. 2015. The formation of a magnetosphere with implicit particle-in-cell simulations. *Procedia Computer Science* 51 (2015), 1178–1187.

[21] Ivy Bo Peng, Juris Vencels, Giovanni Lapenta, Andrey Divin, Andris Vaivads, Erwin Laure, and Stefano Markidis. 2015. Energetic particles in magnetotail reconnection. *Journal of Plasma Physics* 81, 2 (2015).

[22] Daniel A Reed and Jack Dongarra. 2015. Exascale computing and big data. *Commun. ACM* 58, 7 (2015), 56–68.

[23] Sergio Rivas-Gomez, Roberto Gioiosa, Ivy Bo Peng, Gokcen Kestor, Sai Narasimhamurthy, Erwin Laure, and Stefano Markidis. 2017. MPI windows on storage for HPC applications. In *Proceedings of the 24th European MPI Users' Group Meeting*. ACM, 15.

[24] Sergio Rivas-Gomez, Stefano Markidis, Ivy Bo Peng, Erwin Laure, Gokcen Kestor, and Roberto Gioiosa. 2017. Extending Message Passing Interface Windows to Storage. In *Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on*. IEEE, 727–730.

[25] B. a. Schroeder. 2010. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing* (2010).

[26] Philip Schwan et al. 2003. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux symposium*, Vol. 2003. 380–386.

[27] Open Stack. 2017. Open Stack Swift. https://docs.openstack.org/swift/latest/. (2017). [Online; accessed Nov 2017].

[28] Jeffrey S Vetter, Vinod Tipparaju, Weikuan Yu, and Philip C Roth. 2009. HPC interconnection networks: The key to exascale computing. *Advances in Parallel Computing: High Speed and Large Scale Scientific Computing* (2009), 95–106.

[29] Nicola Wadeson and Mark Basham. 2016. Savu: A Python-based, MPI Framework for Simultaneous Processing of Multiple, N-dimensional, Large Tomography Datasets. *arXiv preprint arXiv:1610.08015* (2016).

[30] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 307–320.