Mero: Co-Designing an Object Store for Extreme Scale

Nikita Danilov (nikita.danilov@seagate.com) [Seagate]
Nathan Rutman (nathan.rutman@seagate.com) [Seagate]
Sai Narasimhamurthy (sai.narasimhamurthy@seagate.com) [Seagate]
John Bent (john.bent@seagategov.com) [Seagate Government Solutions]

Abstract

Within the HPC community, there is consensus that Exascale[1] computing will be plagued with issues related to data I/O performance and data storage infrastructure reliability, caused primarily by the growing gap between compute and storage performance, and the ever increasing volumes of data generated by scientific simulations, instruments and sensors. The architectural assumptions for extreme computing are now changing to accommodate these extreme volumes of data as they transit through scientific workflows. Historically, however, there has been a disconnect between HPC users of extreme-scale storage systems and designers of such systems, as designers make architectural decisions based on an incomplete picture of the use cases they are required to address and users construct individualized workarounds to adjust as necessary. This paper presents the co-design process for deriving an Exascale architecture and presents one such exemplar system: Mero, an object storage software solution specifically architected for BDEC.[2] We systematically gather co-design application requirements from the extreme scale HPC and I/O community and then corroborate those inputs through real world extreme scale use cases, to derive the architecture for Mero.

Keywords

Exascale, Object Store, HPC I/O, Hierarchical Storage Systems, BDEC

1. Introduction

Exascale computing is characterised by the availability of an infrastructure to support computational capability in the order of an Exaflop. Currently, this definition is broadly understood to include storage and analysis of an Exabyte or more of data as part of a scientific workflow or a simulation. Based on recent estimates, we anticipate that Exascale computing infrastructures will be available in the 2022 timeframe. As relevant to Exascale, the landscape for HPC-related storage is changing with innovations in new device technologies, such as Flash, and other forthcoming non-volatile memory technologies. The optimal use of these devices in the I/O hierarchy, in concert with existing disk technology, is just beginning to be explored in the HPC realm [1][2].

The inclusion of proliferating quantities of scientific and instrumented data into scientific workflows further exacerbates storage challenges in HPC deployments, the implications of which are being discussed by the BDEC community. As an example, when fully operational, the Square Kilometre Array [3] experiment will generate one Exabyte of raw data every day data that needs to be reduced, processed and analysed.

Mero is a next generation object storage software solution, built from the ground up, to cater to data-centric Exascale computing use cases. Mero can suitably exploit new storage devices in the I/O hierarchy, enable the use of compute capability in the I/O stack for applications and provide much needed capabilities for I/O performance scaling and efficiency in BDEC-type use cases.

The primary objective of this paper is to describe a systematic co-design exercise that consists of gathering architectural inputs from the HPC I/O community and corroborating them through real-world BDEC-type use cases in the SAGE EU project [4]. These descriptions then inform the top-level software architectural considerations of Mero.

2. Background and Related Work

The process of designing Mero for Exascale has been different from what has been done for other HPC-level parallel file and object storage systems, such as Lustre [5] and Ceph [6]. Although tremendously successful for early Petascale infrastructures and cloud storage use cases, they used a relatively ad-hoc design process of architecting for Exascale layered on a foundation of older architectural assumptions.

When Lustre was first designed in 1999, the hardware assumptions about concurrencies, heterogeneity and parallelism were very different. Extremely disruptive innovations in hardware architectures, such as many core processors and heterogeneous processing units such as GPUs, occurred years later. Hence, the process of repurposing Lustre for Exascale computing I/O stacks involves the introduction of additional functionality layered around the file system, which results in heavy overheads.

[1] All of our arguments apply for solutions beyond Exascale, which can be generalized as extreme scale. The terms "extreme scale" and "Exascale" are used interchangeably in this paper.

[2] Big Data Extreme Compute – A term used to indicate data centric extreme computing.

One such example is the Fast Forward [7] I/O stack which, as proposed, would have needed many layers to support Lustre, as compared to the relatively straightforward design of Mero, shown in Figure 1:

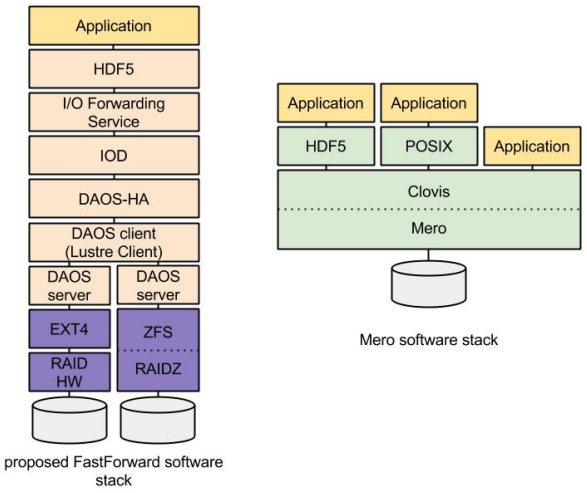

Figure 1: Extreme Scale I/O Stacks: Lustre in the FastForward Project vs. Mero

A fundamental problem with incumbent parallel file systems in HPC is that they are layered on top of other software abstractions, such as local file systems and local RAID systems, which were designed with entirely different software stacks and access patterns in mind. This design introduces the possibility of latency-inducing context switches and memory copies, the possibility of redundant functionalities, such as compression, incompatible interfaces and unnecessarily excessive amounts of metadata duplicated across multiple layers. Further, the cores of the software kernels in these parallel file systems need to be heavily patched to get them Exascale-ready, resulting in continued problems. For example, consider the plethora of *external* middleware and underlying storage layers that have been developed to bring parallel file systems forward from Petascale to Exascale. Examples include Zest [8], Giga^{+} [9], PLFS [10], SCR [11], and the ZFS [12] backend to Lustre. Additional efforts, such as Lustre DNE Phases 1 and 2, have addressed these issues *internally,* within parallel file systems.

Considering the challenges with traditional parallel file systems, there has been a case to move towards using object stores to build extreme scale HPC storage systems [13][14]. However, object stores were primarily designed for cloud environments where, again, assumptions regarding parallelism and usage models are entirely different. Object stores fall short of meeting many of the I/O requirements of Extreme Scale Computing and need to be heavily re-architected and patched to be used in Exascale environments. Taking the example of Ceph, the following issues are already apparent:

- Architectural performance constraints: Synchronous writes in Ceph mean that applications are blocked from progressing until data is committed to disk [15]. These latencies need to be hidden from applications through additional tiers, leading to some of the same challenges of working with a many-layered software stack discussed earlier, regarding the Fast Forward project proposed for Lustre.
- Not built for multi-tier hierarchies: Ceph does not yet support ecosystem components, such as HSM, although this feature is starting to be addressed [16][17]. Clearly, mechanisms such as HSM in Ceph seem to be built for very limited tiering conditions (Scratch, Archive, etc.), and may still require additional work to support deeper I/O hierarchies as required for Exascale storage.
- Extensibility features in Ceph supporting additional services are not systematically covered: Although work has been done to incorporate several specific features via plug-ins (e.g. erasure coding [18]) other missing functionality is being added by layering services on top of Ceph [19][20]. In contrast, support for additional services and robust extensibility through plug-ins is one of the core tenets of the architecture of Mero. This architectural design will be essential to provide a path for smooth evolution and the addition of new features on top of a well-defined core.
- HPC-style access: Insufficiently flexible core API (libRados)[21] to provide HPC-friendly access methods such as asynchronous transactional collective bulk-I/O.

These issues reflect development of incumbent file systems that lack a well-defined co-design process, resulting in designs that are heavily biased towards the assumptions and backgrounds of storage architects, rather than those based on HPC community user and stakeholder inputs.

3. Mero Co-Design Inputs: Quality Attribute Workshops

Mero's inception can be traced to 2010 when a group of Lustre community architects (including the file system's founders and early contributors to the project) gathered to systematically analyse several limitations of Lustre and consider a base design for an Exascale-capable I/O storage technology. This initial meeting was held in Paris in 2009. The group recognised a compelling need to work on a bottom-up design with significant, sustained involvement of the HPC application and user community.

Accordingly, in 2011, the Exascale I/O Workgroup (EIOW) was formed to gather requirements for general I/O middleware for Exascale. The EIOW was adopted as a supporting initiative of the European Open File System (EOFS) [22] organisation. Under the auspices of EOFS/EIOW, the principal participants decided to host

multiple Quality Attribute Workshops (QAWs), to gather comprehensive inputs from HPC application developers, users, data centre administrators and storage architects regarding the new architecture. Three such workshops were held in 2012 (in Munich, Portland and Tokyo), with participation from 35 leading HPC-focussed organisations.

The QAWs were part of the process defined by the Architecture Centric Engineering (ACE) initiative from the Software Engineering Institute (SEI) at Carnegie Mellon University [23] by which requirements are gathered after an introductory presentation about the system architecture, its business objectives and its architectural drivers. At each QAW, participants suggest Quality Attribute Scenarios (QASs) that are discussed and added, in summary form, to a master list. QASs are similar to use cases; they systematically describe the system or user events that cause the specific scenario. Duplicate QASs are removed and the audience votes on relative primacy and importance of the remaining QASs.

Summaries of the three QAWs, held in 2012, after the initial meeting in Paris, are described below, including the highest-ranking QASs identified at each workshop.

a. Munich QAW (Feb 2012)

The Munich workshop collected more than 40 Exascale QASs in the areas of scalability, performance, availability and data integrity. After the workshop participants voted, the following qualities emerged:[3]

1. API Hints (guided mechanisms): This quality is the capacity to implement actions and behaviours based on data usage hints provided by applications and workflows. These hints could indicate near future access, for example, to enable the storage system to make data available at the right tier at the right time. Also, it was agreed that hints support should be *integrated* in the software stack, that is, hints should be passed through all software layers and all layers should be able to analyse and modify hints.

2. Tiered Storage Management: This is the capacity to automatically handle data in tiered storage with Hierarchical Storage Management (HSM) tools.

3. Data Layouts: With the arrival of new, non-volatile memory technologies, there is a very strong case to leverage the benefits of these innovations in the HPC I/O stack. As each storage device technology (NVRAM, Flash, disk, etc.) has its own performance and data retention characteristics, it would make sense to distribute the appropriate pieces of data, based on its usage characteristics, to the right tier. This distribution of data that spreads a single object across multiple tiers is called a "layout". A data distribution formula, unique for each object, or classes of objects, can essentially drive these layouts. Further, it is possible for this data to be compressed, encrypted or deduplicated across these tiers, leading to the potential for "compressed layouts", "encrypted layouts", "deduplicated layouts", etc.

4. Data Compression: It was agreed that there should be mechanisms to implement *lossy* compression.

5. Plug-ins: A whole host of data management tools were considered by workshop participants, such as Information Lifecycle Management, Replication, Migration, File System Checking, Indexing, In-Storage-Compute, etc. Tracking these many features within the core system adds massive complexity that may not be generally useful and thus, justifies their implementation as plug-ins.

6. Storage System Telemetry and Simulation: The group discussed existing methods to debug and analyse systems, such as scouring through extremely large amounts of unstructured logs looking for performance hotspots, etc. and agreed that it would be highly undesirable and not sustainable to use these techniques at Exascale. Lack of observability in existing systems was deemed a critical obstacle for practical scalability. There was consensus on a requirement to collect very structured telemetry data from different subsystems in a format that could be easily analysed and processed. Further, the participants discussed feeding these records into a simulator to analyse "what if" scenarios.

b. Portland Quality Attribute Workshop (Apr 2012)

After careful review of the outcomes from the first QAW, the Portland workshop defined 30 QASs and identified the following key qualities:

1. Data Containers: Containers are, essentially, virtualised abstractions of the storage system, that offer multiple possible consolidations of data and infrastructure as needed by applications and workflows. Initial discussions on containers centered on the need to group related workflow data structures within a container for easy accessibility and manageability. Participants discussed containers based on data formats such as HDF5, POSIX, etc. However, the group also considered that it would be possible to define containers based on performance aspects, for example, a high performance container that only exposes higher tiers or a volatile cache on a diskless client, a persistent cache on a storage server, etc.

2. Experimental Data Handling: Participants discussed examples such as the Square Kilometer Array. Additionally, there could be cosmological simulations that need to work with near real-time data received from the telescopes as well as previously collected data—simulations requiring data to be filtered, analysed and pre-processed before being used. This is a classic example of scientific insights derived not just from simulations, but real instrumented data that is collected as part of the scientific workflow. Many such examples, including a few simulations from CERN, were discussed.

[3] Because of space constraints in this paper, we have omitted detailed QAS descriptions.

Thus, the storage system and its API need to be capable of handling simulation I/O and also have the ability to directly ingest (and process, in-storage) very large data sets from external sources. This capability is extremely difficult to achieve with incumbent parallel file systems, which are primarily designed to work only with simulation I/O. Extending the system's capabilities to work with real scientific data, as described above, will overly complicate the design of the I/O stack.

3. Distributed Transactions and Epochs: The group agreed that storage systems need to get away from POSIX consistency models and have the ability to roll back to previous system states. Epochs and transactions effectively eliminate the global barrier needed for checkpoints, a problem that is exacerbated at Exascale [25] with, potentially, billions of threads doing I/O. In the model of distributed transactions and epochs, each thread associates all writes made by it as part of an epoch entity and closes the epoch once the I/O phase is completed. Multiple epochs can be open in the system at the same time. In the event of failure, the application threads can start from the last epoch that was committed to persistent storage. By assuming a reasonable level of support from the application (in the form of the ability to report failures and to restart from a past epoch boundary), epochs provide a lightweight and scalable alternative to traditional ACID[4] or POSIX transactions.

c. Tokyo Quality Attribute Workshop (May 2012)

The Tokyo workshop identified the following qualities:

1. Lazy Commit and Roll-back: Participants agreed that the storage system must be able to move from snapshot to snapshot and be capable of rolling back to previous states.

2. Plug-ins: There was consensus on the need for data management plug-ins (e.g. Data Indexing) through the previously introduced concept of a File Data Manipulation Interface and the capability to support File Operation Logs.

3. ls-l Command: The group concurred that ls-l command function is the Achilles heel of parallel file systems and there is a need for adequate methods to enumerate distributed containers.

4. Lightweight mechanisms to deal with the most frequent recovery scenarios and build an HA system based on similar concepts.

The QAW outcomes, including requirements and attributes identified by the workshop participants, formed the core architectural principles to build Mero and its API, Clovis. Coincident to when the QAWs were held, initial BDEC discussions started, led by several laboratories in the U.S. Department of Energy. It was an opportune time to corroborate the Mero architecture with proposed BDEC-type use cases.

4. Mero Co-Design inputs: Extreme Scale Use Cases

As discussed above, the QAWs provided key architectural considerations to design Mero. Once built, the Mero architecture was further validated against data-centric, extreme computing use cases from the EU's SAGE project [4] which implements a hierarchical storage system (using many tiers of storage devices and in-built compute capability) driven by Mero. The SAGE programme's D1.1 deliverable [24] presents diverse use cases in the domains of Energy (Nuclear Fusion), Synchrotron experiments, Climate and Weather Science, Bio-informatics and Big Data Analytics that have been used to gather co-design inputs for the project's Mero-based platform. The crucial point of the co-design exercise is that the specified use cases are BDEC-type rather than classic HPC style.

The co-design exercise for Mero included formal I/O characterisation, SAGE architecture analysis, data retention characterisation (use of multiple tiers driven by the object store) and data scaling analysis (a detailed mathematical analysis of I/O requirements) [24]. The formal I/O characterisation exercise, for instance, included information on data volumes, data intensity, data sizes, I/O parallelism, inter-process I/O consistency, metadata parameters (such as directory tree depth) and fault tolerance semantics.

Hence, the co-design process yielded a rich set of data points and information, including very specific use case inputs that were applied to cross check against and build on architectural assumptions for Mero derived from the QAWs.

5. Deriving the Mero Architecture

Inputs from the QAWs and the co-design process in the SAGE project have yielded the following key considerations for the Mero architecture. We reference these attributes to the components shown in the high level architectural view of Mero (Figure 2).

- Scalability (Horizontal and Vertical)
    - Containerisation of data (as described in the QAW outputs) [Component: Realm]
- Availability, reliability and fault tolerance
    - Transaction management to move the storage system from one stable epoch to another and the ability to atomically group I/O requests to achieve the same purpose [Component: DTM]
    - Continuous availability for storage applications in the face of constant storage and hardware failures inherent at Exascale [Components: NBA and Loom]

[4] Atomicity, Consistency, Isolation and Durability

- Elimination of the overhead of RAID rebuilds by employing parity declustered RAID techniques/Server Network Striping across the cluster [Component: SNS]

- System Observability
  - Availability of well defined, structured analytics, diagnostics and telemetry data (user configurable) [Component: FOL, includes the Analysis and Diagnostics DataBase (ADDB) subcomponent]

- Quality of Implementation
  - Address problems related to constantly patching core elements of the storage software code by having a relatively stable core on top of which additional plug-ins and gateways can be easily added
  - Address implementation aspects of scalability (for example, thread per request scalability issues inherent in server architectures) [Component: FOM, which is a threadless, non-blocking state machine implementation]
  - Address implementation aspects of scalability by having a symmetric client server architecture, with no distinction between clients and servers; any node can either be a client requesting access or a server serving up its storage resources. This is a powerful paradigm, especially to expose compute node local NVRAM and Flash resources.

- Extensibility or the capability of the system to track evolving requirements; adding new features and capabilities in a timely manner without sacrificing system stability. Several example plug-ins are shown in Figure 2.
  - Ability to easily extend the core storage system to support new features, such as indexing, ILM, HSM, backup, migration, etc.
  - Ability to add third-party gateways on top of the storage system to support various access protocols such as POSIX, S3, etc.
  - Ability of the storage system to exploit local compute resources to provide capability for in-storage compute
  - Plug-ins to accept guided I/O hints from applications

- Data locality and hardware awareness
  - Ability of each object data layout to span multiple storage hardware tiers with the assumption there will be three or more tiers (NVRAM, Flash and disk) [Component: Layout]
  - Layouts also need to define data compression (compressed layouts), encryption (encrypted layouts), Parity Declustered RAID, etc.

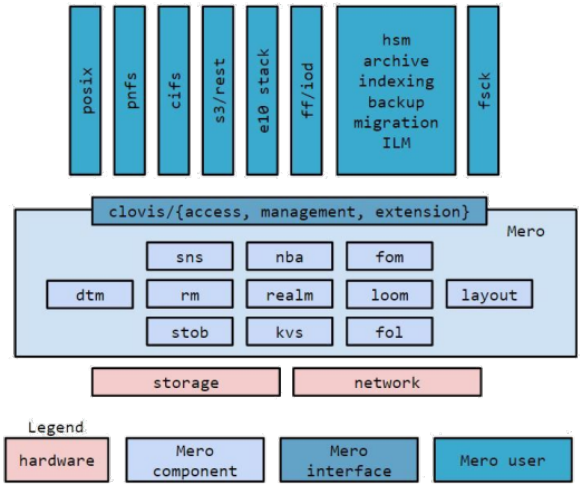

Figure 2: Mero Top-Level Architecture Based on Co-Design and QAWs

The Clovis API provides access to Mero through an access interface, a management interface (providing telemetry records) and an extension interface (providing operation logs to build third-party plug-ins). Access gateways and third-party plug-ins that can be added on top of Clovis are shown in Figure 2. The Resource Manager (RM) controls coherent caching and replication of storage resources such as nodes, caches, object extents, quotas, locks, etc. by external applications as well as Mero's internal processes. The Key Value Store (KVS) provides the basic key value store for objects and stob abstracts underlying storage.

6. Conclusion and Next Steps

In this paper, we have described the co-design process and provided a brief summary of the user inputs and key considerations driving the architecture of Mero, a BDEC storage software platform. As far as we know, Mero is the first extreme scale I/O architecture developed from the outset with significant community stakeholder involvement and input. Our next steps are to present the detailed Mero architecture and provide performance results for individual software components and the entire Mero system as we move towards Exascale.

7. Acknowledgements

We thank Juelich Supercomputing Center (Germany), the Diamond Light Source (UK), the Culham Center for Fusion Energy (UK), the Science and Technology Facilities Council (UK), the German Research Center for Artificial Intelligence (Germany) and the Royal Institute of Technology (Sweden) for providing use case inputs through the SAGE project and Dirk Pleiter (from Juelich) for coordinating these inputs.

8. References

[1] Liu, N.; Cope, J.; Carns, P. H.; Carothers, C. D.; Ross, R. B.; Grider, G.; Crume, A. & Maltzahn, C. (2012), On the Role of Burst Buffers in Leadership-Class Storage Systems, in 'MSST', IEEE Computer Society, pp. 1-11.

[2] Bent, J.; Faibish, S.; Ahrens, J.; Grider, G.; Patchett, J.; Tzelnic, P. & Woodring, J. (2012), Jitter-Free Co-Processing on a Prototype Exascale Storage Stack, in 'MSST' , IEEE Computer Society

[3] SKA, https://www.skatelescope.org/sdp/, Accessed Sept'16

[4] SAGE, http://www.sagestorage.eu, Accessed Sept'16

[5] Lustre, https://www.lustre.org, Accessed Sept'16

[6] Ceph, https://www.ceph.com, Accessed Sept'16

[7] FastForward, https://wiki.hpdd.intel.com/display/PUB/Fast+Forward+Storage+and+IO+Program+Documents, Accessed Sept'16

[8] P. Nowoczynski, N. Stone, J. Yanovich, and J. Sommerfield. Zest: Checkpoint Storage System for Large Supercomputers. In Proceedings of the 3rd Petascale Data Storage Workshop, Austin, TX, November 2008.

[9] Giga+, https://www.usenix.org/legacy/event/fast11/tech/full_papers/PatilNew.pdf, Accessed Sept'16

[10] John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. PLFS: A Checkpoint Filesystem for Parallel Applications. In SC09, Portland, Oregon, November 2009

[11] SCR, http://computation.llnl.gov/projects/scalable-checkpoint-restart-for-mpi, Accessed Sept'16

[12] ZFS, https://wiki.archlinux.org/index.php/ZFS, Accessed Sept'16

[13] SWIFT, http://docs.openstack.org/developer/swift/overview_architecture.html, Accessed Sept'16

[14] Ceph Storage, http://ceph.com/ceph-storage/, Accessed Sept'16

[15] Librados Presentation, http://events.linuxfoundation.org/sites/events/files/slides/20150311 vault15 librados.pdf, Accessed Sept'16

[16] Ceph-Devel Discussion, http://www.spinics.net/lists/ceph-devel/msg17159.html, Accessed Sept'16

[17] Towards Ceph Cold Storage, http://tracker.ceph.com/projects/ceph/wiki/Towards_Ceph_Cold_Storage, Accessed Sept'16

[18] Ceph Erasure Coding, http://docs.ceph.com/docs/hammer/rados/operations/erasure-code-jerasure/ , Accessed Sept'16

[19] Collectd Github activity, https://github.com/rochaporto/collectd-ceph, Accessed Sept'16

[20] Ceph Dynamic Object Interfaces, http://ceph.com/rados/dynamic-object-interfaces-with-lua/, Accessed Sept'16

[21] Librados API Discussion http://docs.ceph.com/docs/master/rados/api/librados/, Accessed Sept'16

[22] EOFS, http://www.eofs.eu, Accessed Sept'16

[23] SEI Website, http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=28284, Accessed Sept'16

[24] SAGE Public Deliverables Page, http://sagestorage.eu/research/deliverables, Accessed Sept'16

[25] John Bent, Brad Settlemyer, Haiyun Bao, Sorin Faibish, Jeremy Sauer, and Jingwang Zhang. BAD-Check: Bulk Asynchronous Distributed Checkpointing. In Petascale Data Storage Workshop at SC15 (PDSW15), Austin, TX, November 2015.