An Exploration into Object Storage for Exascale Supercomputers

Raghunath Raja Chandrasekar, Lance Evans and Robert Wespetal
Cray Inc.
{*raghu, lance, rwespetal*}*@cray.com*

***Abstract*—The need for scalable, resilient, high performance storage is greater now than ever, in high performance computing. Exploratory research at Cray studies aspects of emerging storage hardware and software design for exascale-class supercomputers, analytics frameworks, and commodity clusters. Our outlook toward object storage and scalable database technologies is improving as trends, opportunities, and challenges of transitioning to them also evolve. Cray's prototype SAROJA (Scalable And Resilient ObJect storAge) library is presented as one example of our exploration, highlighting design principles guided by the I/O semantics of HPC codes and the characteristics of up-and-coming storage media. SAROJA is extensible I/O middleware that has been designed ground-up with object semantics exposed via APIs to applications, while supporting a variety of pluggable file and object backends. It decouples the metadata and data paths, allowing for independent implementation, management, and scaling of each. Initial functional and performance evaluations indicate there is both promise and plenty of opportunity for advancement.**

***Keywords*-storage; object-store; NoSQL; resilience; POSIX;**

I. Introduction

The need for highly scalable, resilient, and high-performance I/O and storage has never been greater in high performance computing. Sites planning for next-generation supercomputers are increasingly focused on advanced I/O and storage technologies capable of standing up to challenging scientific and analytics workflows. To satisfy these emerging requirements, Cray investigates and experiments with aspects of storage architecture for highly parallel computer platforms.

Herein we discuss current observations, architectural challenges, and design philosophies in several related areas: emerging forms of memory and storage media, upcoming low-latency high-bandwidth fabrics, tiered caching and movement for increasingly parallel systems, the overlap of HPC, cluster, and analytics storage, scalable and resilient object stores, smart metadata for current and future application interfaces, and storage software co-development in an increasingly open-source community.

We also describe a practical example of Cray's work in this area, the prototype SAROJA (Scalable And Resilient ObJect storAge) library. SAROJA is extensible I/O middleware used to study key aspects of storage software design for future storage software paradigms and fast media. The metadata and data paths are decoupled, allowing for independent management and scaling of each. Our initial prototype provides a POSIX or object interface using a NoSQL database for metadata and an object store for data.

II. Trends and Challenges

The HPC trend toward broad horizontal CPU scaling shows no sign of abating, presenting new challenges to I/O and storage designs that necessitate similar parallelism along with efficient, scalable software. These challenges arise not only from sheer system size, but from greater CPU channel and NIC throughput, larger datasets, increased failure probability with large device counts, and virtualization trends. All these factors are relevant in commercial and analytics sectors as well, which may not require extreme node counts but require increasingly concurrent processing and 24x7 fault tolerance. Fortunately storage and fabric hardware innovations are setting a brisk pace as well. Solid state storage devices achieve ever-lower latency and higher throughput with each generation, and fabrics continue doubling in speed while providing improved congestion management. The greatest unmet challenge is to create software adequate to realize the full potential of recent compute, fabric, and storage hardware advances.

A. Storage Media

Seek latency of mechanical disk media has been the dominant constraint in I/O subsystem design for decades. Heroic efforts are undertaken in HPC and analytics applications to compensate. From the HPC application through the entire I/O chain, we serialize, buffer, align, and coalesce data from thousands of clients to avoid inducing random disk seeks. Despite tuning, disks still typically deliver less than 20% of their potential streaming bandwidth to applications. In analytics, one of the core principles driving initial Hadoop design, was to distribute software specifically to maximize sequential disk scanning via a share-nothing approach. More recent analytics, cluster, and multi-tenant HPC environments thwart any attempt to coordinate I/O to shared storage. The emergence of flash has allayed many of these concerns.

It is difficult to overstate the impact solid state storage media has had on the computing industry. Wholesale adoption of flash in mobile and consumer markets, coupled with moves to flash by webscale cloud and social media providers, has driven flash vendors to innovate and compete for volume markets. This has resulted in plentiful flash with high reliability and performance, impressive density, and falling prices. Enterprise, HPC, and analytics markets are now beneficiaries of these sweeping trends. Flash devices are insensitive to random I/O and actually perform better as queues fill, operating at 100x lower random read latency and achieving 10x maximum throughput of streaming disk. They are available in higher capacity, consume less power, are lighter, and are insensitive to vibration. While cost/gigabyte of enterprise flash is still about 20x that of capacity enterprise disk, flash cost/gigabyte/second beats random I/O to disk by orders of magnitude. And as applications of all types scale horizontally, high thread counts make it nearly impossible to tune away random I/O. These reasons, coupled with constant price decreases, indicate a wholesale transition to all-flash primary storage is likely in all segments of the supercomputing market shortly after the end of this decade.

Newly emerging non-volatile media types suitable for storage applications continue to be elusive. Those that are available are cost-prohibitive, useful only as small caches. Most promising among them, however, is 3D-Xpoint, born of a collaboration between Intel and Micron. Both parties have rights to the medium; Intel will attach it both to the memory bus for byte-level access granularity, and to the peripheral bus for block. Public details are limited, as products are just being readied for market. Intel's initial claims of performance touted near DRAM speeds at the raw media, while industry speculation and recent press from Micron indicate that complexities of system integration will likely limit practical speeds to half DRAM or less when on the memory bus, and similar to industry-leading NVMe flash devices when on the PCIe bus. Even so, three clear advantages are apparent: its byte addressability, when on the memory controller; significantly higher endurance than flash, though less than DRAM; and consistently lower latency than flash, which yields maximum I/O rates and throughput with shallower queues. Sub-DRAM speeds are inadequate for working storage of most applications, and pricing is expected to be closer to DRAM than flash. So use cases are limited as well, initially for indexes, journals, and writeback caches, next full read-write caches fronting flash, then likely in-memory databases too large for actual DRAM.

B. Low-Latency High-Bandwidth Fabrics

Latency characteristics of current- and next-generation fabrics are a fraction of flash latencies, making disaggregation of flash from servers practical with less than 10% latency penalty. Latencies of typical enterprise SSDs are several times the single-digit microsecond latencies achievable with NVMe-over-Fabrics. This block sharing technique is becoming common for virtualization and containerization environments where fabric-attaching block storage to single compute nodes is desirable. Flash device throughputs continue rising, with fastest SSDs reaching parity with volume fabric bandwidths. For the foreseeable future, throughput parity with high-end fabrics such as InfiniBand, Omnipath, and Aries will require greater than 1:1 ratio of flash SSDs to fabric ports.

Intel Optane NVMe devices based on 3D-Xpoint are imminent for release to gamers and enthusiasts, with latency specified at 9us read and 30us write; future devices will likely be lower. At these storage speeds, the fabric and mediating I/O software become significant fractions of device latency. High-end persistent memory solutions are expected to again reach throughput parity with fabric ports even as speeds rise toward 25GB/s unidirectionally. When accessed remotely, minor fabric congestion issues and inefficient I/O setup can trap device performance and stall I/O transactions issued from expectant applications. These exposures are not as pronounced as those in DRAM-based MPI applications, but storage systems built from these components require similar design principles, plus additional fault tolerance and serviceability requirements expected of networked storage devices.

C. Scaling and Parallelism

Over the last decade, since the breakdown of Dennard scaling, HPC systems have relied on massive parallelism to meet insatiable computational demand presented by scientific applications. A resulting trend toward higher core counts and threading within CPUs has been coupled with ever-higher socket counts, resulting in creation of supercomputers of unprecedented size and complexity. Early exascale machines will exceed 128k CPU sockets, each of which will likely contain over 64 cores, 1TB of internal memory bandwidth, and 100GB/s of fabric I/O. The I/O and storage subsystem must be similarly equipped with unprecedented scale and parallelism.

The currently prevailing storage model for HPC application acceleration is the burst buffer, a flash-based file caching layer intervening between compute nodes and primary resilient storage, designed to absorb more challenging I/O volumes and patterns than current disk-based storage can handle. Previously, spindle count was increased beyond required capacity to attain bandwidth, while disks were short-stroked, wasting that capacity to achieve only modest IOPS improvements. Applications and file systems were tuned for decades to optimize sequential spindle I/O, further compensating for this weakest link in the I/O chain. Though still slightly expensive at HPC scales, flash-based burst buffers are proving to be an early answer to common limitations of spinning disk in servicing millions of concurrent I/O threads.

The next generation of HPC systems does not abandon this caching concept, instead migrating it onto compute nodes or near-node local switches when flash becomes economical enough to wholly replace disk as primary storage. Compute-node-local flash and/or persistent memory may be configured to cache up large volumes of data or absorb bursts of output from processors, later destaging it at throttled rates.

Industry is currently experimenting with various approaches to exploit these node-local caches in key use cases, via scalable checkpoint software and advanced client-side file caching.

Projecting from all solid-state primary storage on planned pre-exascale systems, the first exascale systems are likely to require approximately 1k storage nodes with 16k devices, achieving over 80TB/s throughput. Parallelism of access from the compute complex, enabled by intervening storage software, will be crucial in realizing that bandwidth. The potential for I/O operations in these systems will be in the billions, limited foremost by software. It should be noted this is still likely to be an order of magnitude less than on- or near-node local storage can sustain.

Conversely, analytics use cases require only tens to hundreds of nodes at present. This enables systems to be economically configured for each specific use case. Yet most tend to depend on a similar approach: highly parallel share-nothing I/O to node-local storage. Each analytics system's I/O is optimized around specific compute hardware concentrations (e.g. nodes full of GPUs), with application frameworks serving specific use cases. But most share this similar topology. Hadoop's original map/reduce model proved useful, and now dominates current analytics framework designs. Latest trends rely on RAM-heavy nodes, caching to local temporary flash that is then tiered to external, reliable cold storage via object interfaces. Machine learning use cases rely heavily on GPUs, but also generally couple node-local GPU RAM, CPU RAM, and flash caching, to external cold storage.

D. *Convergence of HPC and Analytics*

Complex workflows are emerging wherein customers would like to use the latest analytics and big data tools to either pre-process HPC inputs or post-process outputs.

In-situ analysis, visualization, and computational steering are rudimentary examples of this. Experimentation is ongoing, running analytics stacks on Cray supercomputers traditionally reserved for scaled jobs. And it is increasingly feasible in analytics use cases to exploit the power of a supercomputer as a co-processor dedicated to detecting unforeseen correlations in data, or performing machine learning operations at great scale. To entertain these interesting possibilities requires sharing of data between platforms, which typically requires extract-transform-load job steps in the workflow, and transferring data from one system to another.

Often, the data generated by computing systems is the most expensive asset an organization possesses. Opportunities exist to employ the latest big data tools in conjunction with those classically considered for HPC, to accelerate results and derive new insights from extant data. An opportunity to converge analytics with HPC hinges on more easily sharing an ever-increasing deluge of structured and unstructured data. HPC users and scientists want to pre-process data before job submission, analyze output during long runs, and consider implications after jobs complete. Applying advanced discovery algorithms to HPC data is new and fertile ground for analytics, while difficult analytics workflows are becoming an HPC-scale problem.

Workflows allowing scientists to make discoveries faster, require not only actively sharing the underlying storage infrastructure, but also the content. Analytics and HPC workflows can be linked today, but data must generally be moved between systems, often by flushing data wholesale to site-wide storage for later ingestion into each subsequent processing pipeline stage. And data generally must be transformed when moving between systems. Analytics frameworks now write rich formats like Spark's Resilient Distributed Datasets (RDDs)[36]. To facilitate convergence, such formats must ultimately be rationalized with traditionally HPC scientific formats like HDF5[31] and NetCDF[27].

Converged storage and data formats can enable much more fine-grained and subtle comparison of data using analytics tools. If the storage infrastructure is informed of the data's structure, correlations can be made without extracting entire container files, and metadata extracted so analysis might be conducted without even touching the data itself. To the extent that common formats can be established, indexing, synthetic namespaces, alternate presentations of the same data can be accomplished without the expensive operation of moving or transforming the data for downstream use.

E. *Infrastructure Trends and Challenges*

As articulated in the hardware scalability discussion, machine sizes and component ratios will vary but their I/O and storage infrastructures often employ similar topologies across application use cases. A three-tiered persistence model is emerging that serves the latest technologies to the breadth of most frequently used cluster, analytics, and HPC applications. Scale-out software-defined storage architectures are trending in each of these tiers, which include: a distributed volatile or persistent caching layer on or close to the compute nodes; a system-wide sharable, byte-addressable, high-capacity primary storage layer; and a site-wide bulk get-put cold store. Each must be independently operable so a system may be configured with a subset. Each of these layers has unique features, though all must offer storage-grade resiliency and serviceability options that scale down as far as they scale up. Storage needs to be available in small increments, but in large quantity with near-linear scaling characteristics.

Frequency of failures increases nonlinearly as interdependent component count rises. Yet continuous operation is an ever-higher priority for customers. With pre-exascale systems' Job Meantime To Interruption measured in hours, steps must be taken in preparation for exascale, to reduce or eliminate storage-related contributions to system interruption, data access loss, and data loss. Larger systems are more difficult to diagnose, necessitating automated recovery and directed service actions to minimize human error when administrator intervention is required. Management software that provides assistance with initial configuration, rolling upgrades, component presence, consensus, monitoring, automated recovery, and data redistribution are important to keeping the computer's work progressing efficiently.

And to facilitate data convergence, common underlying infrastructure is necessary. Developing and maintaining scalable fault tolerant storage systems is difficult and requires long-term investment. And sustaining more than one design for a given tier is generally impractical. Finding the right software design that enables sharing across all system and application types, while retaining the unique capabilities required for each application's requirements is challenging. In the exascale time frame it appears that regardless of system type, we will converge on common site-wide storage infrastructure based on spinning disk interoperable with the cloud and tape, a single primary storage infrastructure based on flash, and unique application-specific layers caching to node-local flash, persistent memory, and RAM. In this latter layer, common output formats into common primary storage is required, if data sharing across domains is to be feasible.

Finally, cloud interoperability is of growing import in analytics and HPC circles. An ability to replicate data to the cloud for resiliency or to support bursting is becoming a core requirement. The site-wide storage tier needs to be compatible with the cloud, so objects deposited there from any machine can be moved to the cloud as necessary. Employing cloud APIs for site-wide cold storage thus makes sense, so the same access methods may be used regardless of which application, or where the application is to be run.

F. Application interfaces

Parallel POSIX interface filesystems traditionally used to access primary HPC storage have had problems providing adequate metadata and data performance beyond a few thousand clients. And initial exascale node counts are projected to reach 6-10x that of currently running systems. Metadata and data coherency requirements of POSIX-compliant file APIs are unlikely to be practical at exascale client and server node counts. Yet it is unlikely that existing HPC applications using POSIX files will be adapted to a different, more scalable interface.

I/O forwarding schemes have been employed as stopgaps, fanning out parallel filesystem clients across the fabric to multiply supportable compute node quantities. However such approaches add fabric hops, increasing complexity and latency on both data and metadata paths. They also shift key client functionality off of compute nodes onto intermediate I/O forwarding nodes, forwarding simple filesystem API calls across the fabric that that would have been handled locally by full filesystem clients. While parallel filesystems with forwarding can be adequate for less latency sensitive sequential I/O patterns, this approach runs counter to the small-block random trend in HPC, multi-tenancy, and analytics application use cases. It also works against I/O efficiency increases necessary to fully realize the benefits of flash and persistent memory technologies expected in exascale systems.

Meanwhile, applications found outside of HPC increasingly use forms of primary storage without POSIX interfaces. Often relatively immature, they vary in granularity of access, performance, security, and scalability. Unfettered by constraints imposed by POSIX standards, proliferation of storage interfaces is widespread. In common use amongst web and analytics developers are a variety of No–SQL and K/V databases and various object storage systems. There are no standards, but some commonality and trending APIs set by convention amongst open-source developer communities and the marketplace. Little attention is paid to efficient file I/O beyond serialization of internally represented objects. However, these forms of storage hold great promise, as they have many characteristics attractive to developers and and to storage system designers. Coarse-grained data access and versioning reduce the amount of serialization typically required for concurrency control in POSIX storage systems. Without central brokers for coherency management, storage services can better be decentralized and scaled across commodity hardware.

Analytics frameworks consume domain-specific representations of data from applications via a layer above primary storage, expressed through library calls, sometimes into a distributed caching layer. These interfaces hold great promise as well, because the internal metadata of rich data structures describes the relationships between data elements, which can be relevant in later analysis, correlation, and reuse of data in unforeseen ways. The trending scientific HDF5 format, and the leading RDD analytics format, might serve data from a common underlying object and metadata store. When applications can not only share the capacity of common storage, but express data in common ways across use cases, complex data structures and metadata in these systems can be used to dynamically provide alternative views of data and to empower complex application workflows via extensible storage services.

Considering all these trends, challenges, and opportunities, storage software vendors are challenged to forward a means of transition that permits existing file-based applications to operate and scale within reasonable limits of traditional parallel file systems, while encouraging movement toward distributed objects. Tunable relaxation of POSIX files is necessary to ease scaling of file-based applications when they don't require the strict temporal or spatial coherency guarantees of POSIX. Application-level library interfaces are needed to enable domain-specific data structures to be expressed into caching infrastructure, and ultimately into the supporting primary storage system through its object interface. And it is this common primary storage system that enables sharing of data structures across platform types and use cases. An application should be able to store data via a domain-specific object API such as HDF5, RDDs, or POSIX files, then access or analyze it via another without restriction.

G. Evolutionary Steps

Success requires consistent, methodical innovation toward this aspirational goal, with practically applicable milestones along an evolutionary path. Backward compatibility must be preserved to the extent necessary in order to ease transition of applications from traditional file interfaces to objects, and from existing storage hardware and software infrastructure to new. And constraints governing technology evolution and industry software investment tolerance must be considered. This is no mean feat.

Initial steps are apparent and tractable to pursue. First, existing primary storage and parallel POSIX filesystem technology must be flash-enabled to the extent possible, while burst buffers are migrated toward persistent memory or flash file caches on or near compute nodes. I/O forwarding will remain the interim method of scaling storage in the HPC context. Traditional parallel filesystems will be employed tactically to propagate files as block devices into virtual machines and containers when non-traditional applications are run self-contained within an HPC context. Meanwhile analytics frameworks and underlying object storage infrastructures will continue to evolve toward de-facto interface standards, both for primary storage and for high-level caching layers.

Second, highly scalable primary object-level interfaces and the scale-out infrastructure that supports them must be pursued, considering to-date open-source work in these areas, requirements of the applications that will consume them, and requirements of emerging fabric and media technologies. Applications may consume these presented interfaces directly, or middleware caching layers may intervene to consume them on behalf of that higher level application software. While a single primary storage interface is desirable, two are more likely: one serving cold bulk storage needs, and another serving extremely fast byte-level changes to distributed transactional memories. These will be used in tandem when appropriate, with tiering software enabling automated movement and lifecycle management. Candidate interfaces for cold storage include S3, Swift, and RADOS, while the leading interface candidate for scalable byte-level transactional persistent memory is currently that of Intel's DAOS.

Third, application-level interfaces that express rich data structures into the storage must be pursued. These are rapidly evolving and will continue to do so over time, as will the caching frameworks that support them above the primary storage layer. Obviously interfaces relaxed from POSIX must remain, and will be adapted from existing mature parallel filesystems to retain the most backward compatibility possible. However more advanced high-level data interfaces are required. The leading candidate in the HPC space is HDF5, and the leading analytics candidate is RDDs. Care must be taken with the scalable primary storage interfaces, that they are able to support these high-level requirements.

And fourth, methods for scalably storing, managing, querying, and presenting the rich metadata associated with the applications' structures must be pursued. Whether stored data is containerized into some canonical format, laid out in POSIX directories and files, or expressed in its full internal richness to an application-level storage API, a manifest or database of the structures must be maintained to enable later search and presentation to applications downstream in the workflow. Scalable NoSQL database technology is an obvious candidate for housing this metadata, with several commercial and open-source technologies available to choose from. Linkages between the applications used to store data and such database infrastructure must be carefully written so as not to impede data flow into the underlying store, and to keep metadata in sync with changing content.

Over time, Cray and the community will endeavor to build the infrastructure required to support low- and high-level objects. Emerging industry and de-facto standards will be leveraged, as well as open-source software implementations where viable, propelling high performance computing toward a climate of scalable workflow convergence. Cray will rely on its strong community of customers and partner developers to collaboratively drive application-level code adaptations, transitioning gradually from classic POSIX interfaces toward emerging commercial and cloud object interfaces that fit. Toward these ends, Cray continues working in several relevant areas to meet our customers' immediate needs. Examples include Lustre stability and performance enhancements, burst buffer leadership, I/O forwarding with relaxed POSIX, and investigations into scalable primary object and metadata infrastructure for cold and flash storage. SAROJA is an example research effort exploring three areas: the potential for scalable rich metadata, software-scaled primary storage infrastructure on flash, and an extensible, scalable application interface to support both.

III. SAROJA

In an effort to validate the trends and challenges discussed in section II, we have developed *proof-of-concept* object storage middleware — SAROJA. Figure 1 illustrates the high-level architecture of a representative SAROJA cluster. At its core is the *libsaroja* client library, which is a shared-nothing library that can be linked to the address spaces of application processes directly if native object APIs are

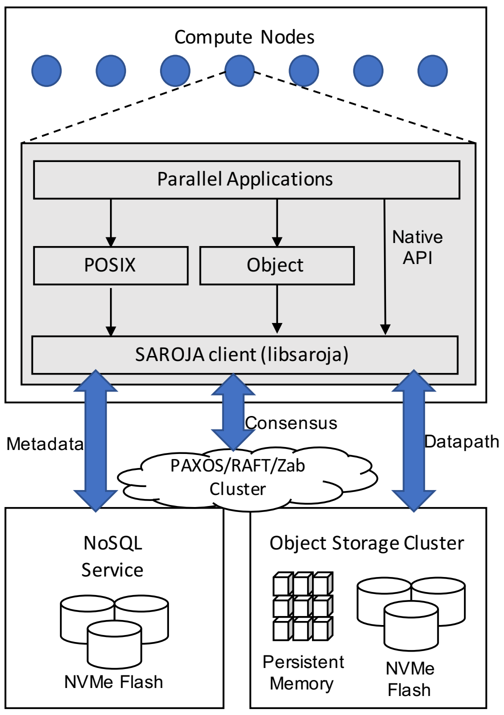

Figure 1. High-Level SAROJA Architecture

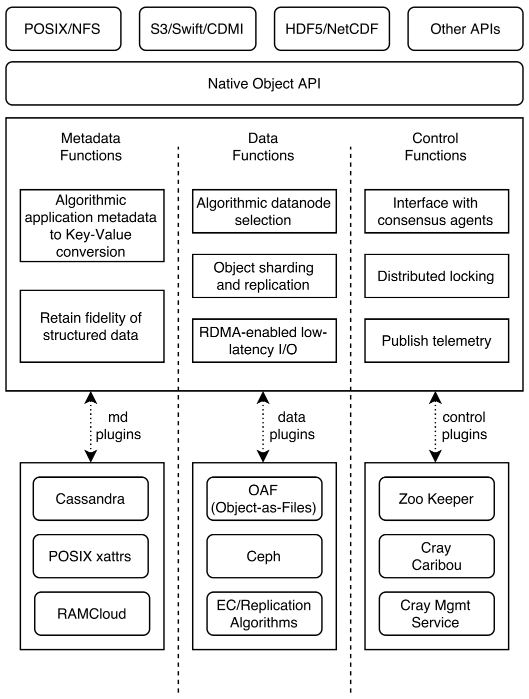

Figure 2. SAROJA Client Architecture

desired, or through one of the API emulation layers if interfaces like POSIX or HDF5 are desired.

Note that the implementation of libsaroja itself does not preclude the use of other key-value stores, object stores, or consensus protocols, support for which can be added using extensible plugins. We chose these specific technologies based on their merits, desirable features, and desire to study the different trends discussed in Section II. The metadata cluster and the object storage cluster are managed and scaled out independently, but can also be collocated physically. Each of these clusters are backed by NVMe flash devices. The object storage cluster, Ceph in this case, can also be configured to make use of available persistent memory devices to further reduce the latency of object access.

A. *Shared-nothing Clients*

The anatomy of the SAROJA client is illustrated in Figure 2. The client performs three classes of functions — metadata, data, and control. These functions are embodied in an interface-agnostic library, `libsaroja`, which forms the building block for implementing a multitude of application interfaces.

Metadata functions of the client primarily include the translation of application metadata, such as POSIX files, HDF5 containers, or general objects into SAROJA application objects and storage objects. While the semantics of the metadata vary based on the upper-level interface being emulated, it is critical to use a data model that works optimally for most of the interfaces. The metadata functions of the SAROJA client, in conjunction with the scalable metadata service, will also allow data to be accessed via multiple application interfaces and offer clients the ability to analyze and transform data using rich metadata queries. The SAROJA prototype is extensible to incrementally support the following interfaces: POSIX, a native object API, HDF5, NetCDF, pNFS, RESTful S3, and Swift.

The client can be extended with data plugins, including resiliency mechanisms (such as erasure-coding and replication), algorithmic data node selection (if not delegated to the object storage cluster), and supporting low-latency data path operations using RPC and bulk-synchronous RDMA techniques where possible. The client can be extended via control plugins, including distributed lock management (in cases where strict consistency semantics are desired by the upper-level interfaces), publishing performance telemetry and event streams to monitoring and management systems, and participating in presence and consensus management protocols for fault-tolerance.

Figure 2 also lists other representative plugins that one could design, such as a control plugin to interface with the Cray System Management Software[19], Cray's Caribou[17] storage performance and metrics monitoring framework, or a data plugin to implement different erasure-coding algorithms than the ones supported by the backing object store.

SAROJA's pluggable back-end interface supports multiple metadata and data stores. We have currently implemented:

a plugin that uses Apache Cassandra [15]; a *metadata-as-xattrs* plugin which can store the metadata as POSIX extended attributes in a distributed parallel filesystem such as Lustre; a Ceph [33] data plugin using the *librados* native Ceph API; an *Object-as-Files* data plugin that stores SAROJA objects as files in a parallel filesystem; and a control plugin using Apache Zookeeper [14] to enforce consistency in our POSIX emulation layer.

Our initial prototype is entirely in userspace: Our long term goal is to bypass the kernel for the latency and throughput benefits. However, studies [32, 25] have demonstrated that the performance degradation caused by the data indirection in FUSE can be highly-variable based on the workloads. It is likely that we would need to implement both kernel and user space versions of the SAROJA client to fully support POSIX applications.

B. Scalable Metadata Services

A key challenge in building a scale-out storage system for exascale-class machines is the design of a metadata service. It must support hundreds of thousands of nodes and support high object counts. Traditional storage metadata services are usually centralized, strongly consistent, and normalized. Yet to scale metadata, strict consistency guarantees held by traditional filesystems must be carefully relaxed in ways that applications can tolerate. Part of the SAROJA prototype is intended to explore ways in which NoSQL database designs align with the requirements of scaled metadata services.

NoSQL databases like Apache Cassandra, MongoDB, and Redis employ a variety of techniques to effectively distribute and store data, providing greater scalability. These include distributed hash tables for spreading data between multiple servers, and Log-Structured Merge Trees for storing the data. These characteristics favor NoSQL databases for the metadata of distributed storage frameworks, and suitable for adaptation to persistent memory and flash. However NoSQL databases typically sacrifice consistency for performance, instead providing eventual consistency guarantees. These types of databases also encourage denormalization, wherein multiple copies of data are maintained for query efficiency.

NoSQL databases have only recently begun undergoing tuning efforts to fully realize their performance potential on flash and persistent memory. We intend to follow developments as these databases evolve, running additional experiments opportunistically. We also will follow and potentially investigate the role NewSQL database technology might play in a scalable metadata service. The goal of NewSQL databases is to provide the consistency and transactional support of traditional SQL databases with the scale and performance of NoSQL databases.

Initial prototypes of SAROJA's metadata service have been focused on relaxed POSIX operations, since the majority of HPC applications today require a POSIX interface. Storing fully POSIX compliant metadata in a NoSQL database is challenging considering NoSQL design criteria. The main requirements for a prototype POSIX interface on a NoSQL database include: scalability beyond a single node; ability to remain consistent during node failures and network partition events; good performance with the chosen database; and ability to express POSIX metadata in the format of the database client. This section explores the challenges encountered when prototyping POSIX metadata within a NoSQL database.

1) Metadata models for POSIX: One possible approach to model POSIX metadata in a NoSQL database is to use the namespace's entire pathname as a key for each file. Values of the key store metadata such as directory permissions and extended attributes. Functions like `readdir()` scan the key space of the database using the directory as the prefix of the key, like "/a/b/*". The pathname-as-key data model is relatively easy to implement and will work with a simple key-value database. This model is optimized for creates and lookups but operations like `rename()` require a significant amount of deletions and creations to update the parent keys.

Another metadata model is a two-tier approach. In this model, all application metadata is represented as a collection. The key of a collection is its unique identifier, the value contains a list of related collections. For POSIX metadata the value contains the identifiers (like an inode number) for files in a directory. Symbolic names, like POSIX filenames, are stored separately from the hierarchy of collections. This metadata model fits within any NoSQL database due to its simple key/value format and can express non-POSIX metadata as well. Transaction latency and server hot-spots are a concern, as multiple queries are needed for a simple lookup operation, and servers that store the upper collections in a hierarchical set of collections will be accessed more frequently. Keeping the collection lists consistent as multiple clients update them is also a challenge.

IndexFS [26] uses a third approach that we have explored in-depth, as its primary goal was also to represent POSIX metadata in a NoSQL database. IndexFS is storage middleware that adds scalable metadata services to an existing parallel filesystem. It achieves this by storing metadata in a key-value database on the parallel filesystem. IndexFS splits directories in the filesystem between multiple IndexFS servers using giga+ [23], a dynamic subtree partitioning algorithm. The key contains the parent directory ID, the partition ID, and the hash of the name of the directory entry. The value contains the inode metadata and a pointer to the file on the parallel filesystem. This design experiences minimal crosstalk between the IndexFS servers. In addition, the developers have run with up to 128 metadata servers, which is impressive. However, it is prototype code that does not support renames across servers, and is not crash safe. Our initial SAROJA prototype uses a data model similar to IndexFS, to store metadata.

2) Metadata partitioning: In addition to how filesystem metadata is stored, it is critical to consider how it is distributed between multiple servers. Three different methods for distributing hierarchical metadata are outlined below. The ability to define a metadata partitioning model depends on the data model and database type used.

Dynamic subtree partitioning moves portions of a directory to other servers based on criteria like server load, number of files in a directory, or how often a directory or files within that directory are accessed. This partitioning scheme can adapt to changing workloads, and preserves metadata locality for workloads that can be handled by a single metadata server. However, dynamic subtree partitioning introduces complexity that can cause varied performance for similar workloads. Some examples of dynamic subtree partitioning include IndexFS with giga+, the current implementation of CephFS, and a planned improvement for CephFS called Mantle [29].

Static subtree partitioning replicates either the entire directory hierarchy or portions of the directory hierarchy between multiple metadata servers. Files that belong to a directory are distributed between the metadata servers via distributed hash tables. Static subtree partitioning is best for workloads where the hierarchy does not change frequently. Compared to dynamic subtree partitioning, no work is required to redistribute metadata, which provides consistent performance. However, this approach makes it difficult to change the directory hierarchy and can adversely affect metadata locality for some workloads. Some examples of static subtree partitioning include ShardFS [35], Lustre DNE [4], MarFS [5], and GlusterFS [1].

Random subtree partitioning distributes file and directory metadata between metadata servers using key ranges or hashing techniques. This model prevents directory hot-spots and is an approach that is typically used to partition data in NoSQL databases. However, random partitioning flattens the namespace, reducing metadata locality. The pathname-as-key data model discussed earlier could use this approach to distribute metadata.

3) Metadata Consistency: As previously mentioned, NoSQL databases typically sacrifice consistency for performance. Most metadata operations in POSIX require locking of the affected inodes or the entire filesystem: A NoSQL database will not provide these semantics. One approach that web-scale applications use with eventually consistent databases is Multi-Version Concurrency Control (MVCC). By marking versions of metadata as consistent, applications can roll back to a known good state when inconsistencies are detected, avoiding strict serialization via consensus. Some future filesystems, like Intel DAOS [18] use a similar technique to support a potential POSIX interface.

If the application requires strict serialization, consensus algorithms such as PAXOS [16] will be necessary. Hosts reach consensus by voting on proposals until an agreement is reached. Many database projects (e.g. Google Spanner, Google Megastore, Hadoop, etc) do implement PAXOS or PAXOS-like variants to enforce consistency. However, reaching consensus with hundreds of thousands of clients is impractical if not impossible. Finding ways to eliminate the need for consensus for most metadata operations is desirable.

C. The Data path

For our proof of concept, we used Ceph as the primary data store. Ceph is a software-based, scalable, distributed storage system that is inherently fault-tolerant. As mentioned earlier, the SAROJA architecture does not preclude the use of a different object store such as Scality RING or Gluster in place of Ceph.

The data path in an object storage system has several functions. As part of our research, we studied several of these functions in the context of large-scale systems. And as part of the SAROJA experiment, we chose to delegate other functions to off-the-shelf object storage that already provides these services. We have observed that there is ample opportunity for performance and scaling improvement, achievable by redesigning some of these functions: algorithmic mapping of object identifiers to storage servers hosting distributed object shards; object resilience by means of replication or erasure-coding; awareness of system failure domains that inform object placement and redundancy across servers; on-the-fly data recovery in the face of failures; data movement services between various tiers and data stores (cloud, archive, etc); and most importantly, efficient, highly granular use of underlying memory, storage, and network hardware to offer low-latency object access to applications.

Many of these capabilities are implemented as part of the RADOS (Reliable Autonomous Distributed Object Store)[34] layer in Ceph. We implemented a Ceph plugin to SAROJA using the *librados*[3] C API. Datapath plugins in SAROJA are agnostic to the top-level I/O interface used by applications. The stripes of a POSIX file, binary objects written to an S3 interface, and HDF5 files, are all stored as RADOS objects. It is key to understand the difference between these user-facing *application objects* and internal on-media persistent *storage objects*. *libsaroja* will control striping user application object data into multiple storage objects. The current implementation uses a static approach to track object extents (like Lustre or DVS[30]/Datawarp[13]), where clients track the object ID and a list of servers that contain stripe elements of that object. The client has the file layout as it is statically defined, and contacts the correct server in the list, for a given object ID and offset. It is possible (though metadata-heavy) to extend this design so the entire layout of a user file/object can be mapped by the client from metadata placed in the scalable metadata service, with each stripe a unique underlying object in the object store.

Ceph has support for various low-level Object Storage Daemon (OSD) back ends, each tuned to suit the hardware and access characteristics of the underlying media where the *storage objects* will eventually be persisted. Ceph's most mature back end is a module called FileStore, which depends on a local under-filesystem. There are several other upcoming and experimental back ends — BlueStore, MemStore, and KeyValueStore — that improve efficiency when NVMe flash devices and byte-addressable volatile or non-volatile memory devices are used.

BlueStore avoids write amplification issues encountered with FileStore, and reduces object access latency. It collocates user data, OSD metadata, and write-ahead logs on a given device. The user data objects are stored either directly on a raw block device without requiring an under-filesystem, or via the Intel SPDK engine that polls for completions from user address space to optimize for low latency NVMe devices. The OSD metadata and write-ahead logs are stored on the same device as the associated user data, in a RocksDB instance. RocksDB[7] leverages the Log-Structured Merge Tree (LSMT)[22] data structure to lay out data efficiently on flash devices.

As part of our research, we are also tracking potential viability of Open-Channel[6] NVMe SSDs in the Ceph data path. Open-Channel SSDs differ from traditional flash devices by exposing the internal parallelism of the SSD to the host, providing three benefits: I/O isolation, predictable latencies, and software-defined non-volatile memory. The LightNVM[8] stack provides enabling technologies in the kernel and in userspace that would allow LSM-Tree based key-value stores like RocksDB, LevelDB[2], and WiscKey[20], to directly benefit from Open-Channel SSDs.

For the Ceph-based datapath, we favor a two-tier datapath when both persistent memory and flash devices are available in the configuration. Ceph supports both replication and erasure coding at a *Placement Group* granularity. However, support for read-modify-write operations required for byte-level granularity in erasure coded pools, is still nascent. There is also a performance penalty in erasure coded pools due to the large granularity of erasure coded stripes, and the need for two-phase commit during object modification. With a two-tier model, libsaroja writes first to a replicated pool using librados. Ceph will then manage migrating data between the replicated pool and the erasure coded pool based on preset caching policies. We expect this to perform well since hot data will remain in the faster flash memory and avoid costly erasure coding computations until the data is cold enough to be de-staged. It is important to note that Ceph does not allow for direct reads from the erasure coded pool in this configuration, so data must be staged into the persistent memory first. We see opportunity for improvement here. At the time of this writing, RADOS clients are unable to manually override the caching policies to force data to be de-staged on demand. For a latency-optimized Ceph-based SAROJA datapath, a replicated pool might thus be stored in persistent memory, with the erasure coded pool stored on NVMe SSDs. A potential path forward is to use the BlueStore OSD back end on flash, and study RocksDB coupled with an adaptation of MemStore to create a latency-optimized, byte-granular PMStore for persistent memory devices.

All components of the SAROJA Ceph data path — including clients, OSDs, and MONs — communicate with each other using the Ceph Messenger protocol abstraction. Currently, three messenger types exist; a *Simple* messenger that sends synchronous messages, an *Async* messenger that sends messages asynchronously, and an *Xio* messenger that leverages the Accelio RPC acceleration library to send messages via RDMA over InfiniBand networks. The latter two are experimental and are not yet production-ready. Both the SimpleMessenger and the AsyncMessenger rely on traditional TCP/IP and support only Ethernet fabrics. With the SimpleMessenger implementation, an endpoint spawns a new thread to service each connection, degrading the performance of smaller-sized I/O operations. The AsyncMessenger addresses this by offloading the communication progress to a pool of threads and polling on them for completion. To allow the SAROJA data path to be portable to multiple system and fabric architectures, and to validate the fabric trends discussed in Section II, we intend to extend the Messenger abstraction to implement a *libfabric* plugin.

libfabric[12], designed by the OpenFabrics Interfaces Working Group (OFIWG), comprises a set of portable interfaces that enable a tight semantic map between applications and underlying fabric services. Specifically, libfabric software interfaces have been co-designed with fabric hardware providers and middleware developers, with a focus on the needs of HPC users. Cray has been leading the OFIWG's efforts along with our partners, and we have started prototyping use of libfabric across our product portfolios. There already exists an implementation of libfabric for the Cray XC^{TM} systems which has been shown to give initial good performance for MPI and SHMEM micro-benchmarks[10, 24, 28]. Having such a plugin to Ceph would allow future enhancements to libsaroja's data path to leverage zero-copy RDMA mechanisms for directly accessing data from a target storage server's NVMe SSD or persistent memory device.

D. Presence and Consensus

The amount of user interaction with SAROJA is intended to be minimal: The system should be able to adapt and repair itself without user intervention. For example, a storage device failure should cause the system to recover autonomously and rebalance affected data onto another device, notifying the user to replace a specific failed device. To that end, the inventory and configuration along with consensus and presence of SAROJA clients, servers, and devices must be automated. This work has not been undertaken. Pro-

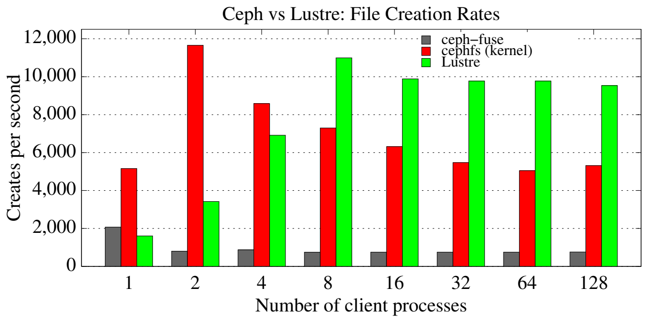

Figure 3. Ceph vs Lustre: File creation rates

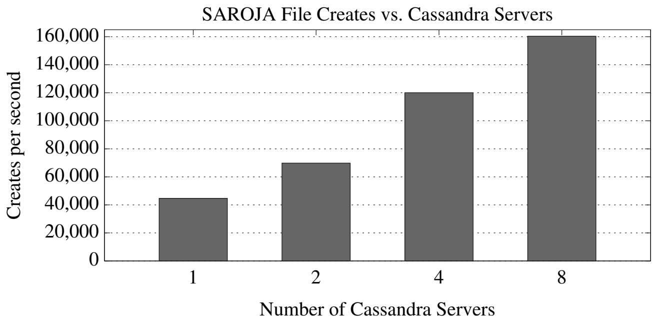

Figure 4. File creation rates using 4480 MPI ranks on 56 Cray XC nodes

visioning SAROJA clients and servers after discovery and configuration must also be automated. We have investigated the use of Apache ZooKeeper [14] for provisioning and device management. ZooKeeper provides resilient, strongly consistent configuration and synchronization for distributed applications. For example, we can use ZooKeeper to store the state and configuration of a server node and update it when a new node is added or a node leaves the cluster. Clients and other servers can watch the state of an item and will be notified when there are changes. We are skeptical that ZooKeeper itself will meet the scalability requirements of exascale, even when hierarchically implemented. But we believe that its simple yet elegant feature set will meet our needs for initial prototypes. Further work in this area is necessary to establish feasibility and design proposals.

IV. Preliminary Evaluation

While a system like SAROJA can be evaluated in multiple dimensions, as we intend to do in the future, this section presents our preliminary evaluations of the metadata and data paths.

A. Metadata Path Evaluation

As a first step, it is desirable to understand how the metadata services offered by distributed object stores' POSIX clients perform in comparison to incumbent HPC parallel filesystems. We used the *mdtest* benchmark, which is the defacto metadata evaluation benchmark in HPC, to compare Ceph and Lustre. The specification of the experimental platform we used is described in Table I.

Figure 3 shows the file creation rates of Ceph and Lustre. As the number of clients were scaled up, each process created 100 files within a common root directory. There are several takeaways from these results. As one would expect, the in-kernel Ceph client performs better than its FUSE counterpart, as it is written natively using the VFS interface and does not suffer the penalty of frequent user-to-kernel address space copies. When the MDSs are bottenecked, Lustre is able to better sustain the metadata operations as the number of clients increases, while Ceph shows degradation in performance. And although Ceph outperforms Lustre with lower client count, its file creation performance is significantly lower than Lustre with increasing clients. We have also observed that Ceph performs better than Lustre with read-intensive metadata workloads, wherein stat() performance reading multiple attributes of a file from the filesystem appears to be consistently better with Ceph. CephFS shows promise in its initial production releases, and work is actively being undertaken by the Ceph development community to improve metadata performance via multiple MDSs and dynamic namespace partition techniques[29]. Still, CephFS has a long way to go before reaching parity with more mature parallel filesystem implementations.

We have also started to validate the SAROJA metadata model with the prototype FUSE filesystem (described in Section III-A) that emulates POSIX on libsaroja, and we have taken a few initial measurements as a functional test.

Figure 4 shows our preliminary results benchmarking metadata operations with a fixed number of clients and tasks, while varying the number of Apache Cassandra servers. This experiment was run on 56 compute nodes of a Cray XC system, with 4480 MPI ranks in total (80 tasks per node) running the *mdtest* metadata benchmark. Each rank created 500 files in its own unique directory. All Cassandra client and server traffic was sent via TCP on the Aries high speed network. Cassandra version 3.10 was used with the default configuration, except the *concurrent_read and concurrent_write* parameters were increased to recommended values based on the number of cores. Re-purposed compute nodes were used for servers, so tmpfs was used for Cassandra data storage. Since we intended to study just the scaling trends, Cassandra data replication was not used, nor was Zookeeper-based distributed locking. Clients cached metadata to reduce the number of round trips required to create a file. We wanted to compare the results to a raw Cassandra write benchmark and also evaluate performance for a POSIX application that can tolerate eventual metadata consistency.

Although initial untuned benchmarks are underwhelming, we are encouraged by these results as strictly a functional

	Ceph	Lustre
Software Version	v11.0.0	v2.7.1
Object Servers	4	4
Number of SSDs	24	24
Replication Factor	1	N/A
Number of MDS	1	1
Storage Backend	BlueStore	1 OST-per-SSD
Fabric interface	IPoIB	IPoIB
Network driver	SimpleMessenger	sockets LND

Table I
DATA PATH EXPERIMENTAL SETUP

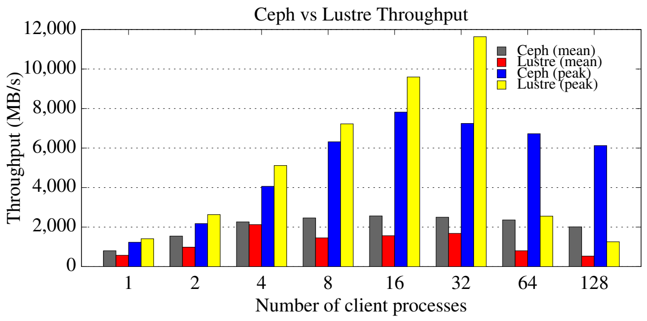

Figure 5. Ceph vs Lustre IOR throughput with varying number of clients

test of POSIX on NoSQL. 56 clients reached their maximum file creation rate with eight Cassandra servers. Running a raw Cassandra benchmark with *cassandra-stress* on the same configuration, we can achieve approximately 100,000 write operations per second per server with linear scale. The data model used in the SAROJA client was used for the raw Cassandra benchmark. We intend to retry this test with a version of mdtest that can access SAROJA directly rather than via FUSE, as this may be inhibiting performance. In addition, a thorough investigation of scalable database technologies is needed, along with each database's unique potential for optimization, for the higher transaction counts and consistency guarantees expected of a storage metadata service.

B. Data Path Evaluation

To evaluate the data path, we configured two small identical storage clusters, one running Ceph, and the other Lustre. Table I outlines the configuration used for the two clusters. While these clusters are not representative of real deployments by any measure to be able to evaluate scaling trends, they are sufficient to study the relative performance characteristics of each. Note that the replication factor of Ceph was set to 1 to allow for a comparable configuration.

We ran the widely-used IOR benchmark to study the I/O throughput, with client MPI processes ranging from 1 to 128. Figure 5 here depicts the mean and absolute peak throughput values for Ceph and Lustre across 10 iterations. From the data, it is evident that Lustre outperforms Ceph when considering the absolute peak throughput, and vice-versa on average. Lustre is able to better utilize the bandwidth provided by the 24 underlying SSDs. It is also clear that 32 client processes are able to saturate the bandwidth of the storage devices backing Ceph and Lustre. With over-subscription of clients, which is a common scenario in several large-scale deployments, the Ceph data path is relatively able to handle the load better. Nonetheless, the key takeaway from these results is that the Ceph data path has potential for use in future HPC storage systems. With the performance considerations and optimizations discussed in Section III-C, we believe that the gap in performance can be further reduced.

V. RELATED WORK

There is a large body of research in literature that studies the various aspects of storage software design for exascale systems. This section aims to summarize the efforts that are related to our work.

Scaling the metadata services of parallel filesystems has been a continuing topic of research for several research groups. The availability of high-performance databases in the enterprise world (such as HBase, Cassandra, CalvinDB, etc.) have paved the way for solutions like CalvinFS, CassandraFS, and GiraffaFS. These solutions represent a POSIX namespace by denormalizing the hierarchy and storing the entire pathname with each key (like the *Pathname-as-key* data model discussed in Section III-B). IndexFS[26] and ShardFS[35] are two related solutions from Carnegie Mellon University, which leverage node-local flash-optimized databases like LevelDB to store metadata and small files, while delegating large-file management to the underlying parallel file system. DeltaFS[37], which builds on the concepts of IndexFS, is storage middleware that explores a server-less filesystem design for exascale. More recently, HopFS[21] proposed by Niazi et al. studies the impact of using a NewSQL database as the metadata layer for HDFS.

Likewise, there have been quite a few efforts that study and propose the use of object storage technologies in the supercomputing storage hierarchy. The Distributed Application Object Storage (DAOS)[18] layer, the outcome of a joint effort between several US National laboratories and industry partners, is a persistent storage interface and translation layer between user-visible objects and the requirements of underlying storage hardware. Mero[11] is a next-generation object store developed by Seagate. MarFS[5] from researchers at LANL, much like SAROJA, decouples the metadata and data management, and promotes the use of object stores in the datapath. The MarFS authors have demonstrated[9] up to 800 million file creates per second using benchmarks that replicate the underlying MarFS protocols.

The goals of the above efforts and ours are well-aligned and complementary. We have learned from and utilized the findings of several of these efforts in our research and

SAROJA prototype design. We are also engaged with several of these groups to collaborate and progress the notion of making object storage technologies viable as the primary storage tier for large-scale supercomputing, cluster, and analytics systems.

VI. Summary and Conclusion

In summary, the scaling challenges of supercomputing, coupled with concurrency and enterprise reliability requirements of cluster and analytics storage, are being met with disruptive solid state storage device types and flat, scale-out software defined storage topologies. A trend across these use cases toward a 3-tier storage hierarchy comprising node-local cache, system-wide primary, and enterprise-wide cold storage is viable, perhaps likely. The opportunity brought about by these inflection points in storage system design could precipitate evolution beyond current parallel file systems toward converged, object-based system infrastructure and management, with the potential for diverse workflows to share common object APIs, formats, data, and semantic metadata access in the exascale timeframe.

Cray's SAROJA experiments represent one path in an ongoing quest for a smooth transition toward this convergence, based on open-source initiatives, big data tools, and newly minted object store features. Our user-level libsaroja prototype demonstrates key concepts, abstracting application object and search APIs from underlying file, storage object, and scalable metadata via an extensible, pluggable design. While our initial evaluations of Cassandra and Ceph lacked luster, and Lustre's maturity, they convinced us that with work and product advancements to leverage flash and persistent memory, NoSQL database and software-defined object store technology can, with effort, become staples of supercomputing.

We aspire to continue this effort, expanding libsaroja's capabilities to allow for a more comprehensive evaluation of object storage technologies. We also intend to evaluate the metadata and data paths at a larger scale to determine whether our assumptions and preliminary results can hold true. We are open to both feedback and collaboration from the community and our customers. While it is still early to postulate how the supercomputing storage landscape will evolve over the next decade, we hope to continue our efforts toward better understanding it and participating in community efforts to satisfy our industry's emerging storage requirements.

Acknowledgments

We would like to thank Ben Myers, Aamir Rashid, Kevan Rehm, and Leo Tucker, who were an integral part of the research efforts described in this paper.

References

[1] GlusterFS Documentation. http://gluster.readthedocs.org/en/latest/.

[2] LevelDB. http://leveldb.org/.

[3] librados. http://docs.ceph.com/docs/master/rados/api/librados/.

[4] Lustre DNE slides. http://cdn.opensfs.org/wp-content/uploads/2013/04/LUG-2013_DNE.pdf.

[5] MarFS Github. https://github.com/mar-file-system/marfs.

[6] Open-Channel SSD. https://openchannelssd.readthedocs.io/en/latest/.

[7] RocksDB. http://rocksdb.org/.

[8] M. Bjørling, J. Gonzalez, and P. Bonnet. Lightnvm: The linux open-channel ssd subsystem. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017.

[9] D. Bonnie, H. B. Chen, G. Grider, J. Inman, B. Kettering, and W. Vining. MarFS Metadata Scaling. In *First Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2016.

[10] S.-E. Choi, H. Pritchard, J. Shimek, J. Swaro, Z. Tiffany, and B. Turrubiates. An Implementation of OFI Libfabric in Support of Multithreaded PGAS Solutions. In *Proceedings of the 2015 9th International Conference on Partitioned Global Address Space Programming Models*, PGAS '15, 2015.

[11] N. Danilov, N. Rutman, S. Narasimhamurthy, and J. Bent. 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)*, 2016.

[12] P. Grun, S. Hefty, S. Sur, D. Goodell, R. Russell, H. Pritchard, and J. Squyres. A brief introduction to the OpenFabrics interfaces–A new network API for maximizing high performance application efficiency. In *Proc. 23rd Annual Symposium on High-Performance Interconnects*, Aug. 2015.

[13] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, and N. J. Wright. Architecture and Design of Cray DataWarp. In *Cray User Group Conference (CUG)*, 2016.

[14] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, 2010.

[15] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 2010.

[16] L. Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[17] P. Langer and C. Flaskerud. Caribou: Monitoring and

Metrics for Cray Sonexion Storage. In *Cray User Group Conference (CUG)*, 2017.

[18] J. Lofstead, I. Jimenez, C. Maltzahn, Q. Koziol, J. Bent, and E. Barton. DAOS and Friends: A Proposal for an Exascale Storage System. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, 2016.

[19] H. Longley. Cray Management System for XC Systems with SMW 8.0/CLE 6.0 (Tutorial). In *Cray User Group Conference (CUG)*, 2016.

[20] L. Lu, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. WiscKey: Separating Keys from Values in SSD-conscious Storage. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, 2016.

[21] S. Niazi, M. Ismail, S. Haridi, J. Dowling, S. Grohsschmiedt, and M. Ronström. Hopsfs: Scaling hierarchical file system metadata using newsql databases. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017.

[22] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. The Log-structured Merge-tree (LSM-tree). *Acta Informatica*, 1996.

[23] S. Patil and G. A. Gibson. Scale and concurrency of giga+: File system directories with millions of files. In *FAST*, volume 11, pages 13–13, 2011.

[24] H. Pritchard, E. Harvey, S.-E. Choi, J. Swaro, and Z. Tiffany. The GNI Provider Layer for OFI libfabric. In *Cray User Group Conference (CUG)*, 2016.

[25] R. Rajachandrasekar, A. Moody, K. Mohror, and D. K. Panda. A 1 PB/s File System to Checkpoint Three Million MPI Tasks. In *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, 2013.

[26] K. Ren, Q. Zheng, S. Patil, and G. Gibson. Indexfs: Scaling file system metadata performance with stateless caching and bulk insertion. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 237–248. IEEE, 2014.

[27] R. Rew and G. Davis. Netcdf: an interface for scientific data access. *IEEE Computer Graphics and Applications*, 1990.

[28] K. Seager, S. Choi, J. Dinan, H. Pritchard, and S. Sur. Design and implementation of openshmem using OFI on the aries interconnect. In *Proceedings of OpenSHMEM 2016*, Aug. 2016.

[29] M. A. Sevilla, N. Watkins, C. Maltzahn, I. Nassi, S. A. Brandt, S. A. Weil, G. Farnum, and S. Fineberg. Mantle: A programmable metadata load balancer for the ceph file system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 21. ACM, 2015.

[30] S. Sugiyama and D. Wallace. Cray DVS: Data Virtualization Service. In *Cray User Group Conference (CUG)*, 2008.

[31] The HDF Group. Hierarchical Data Format, version 5. http://www.hdfgroup.org/HDF5/.

[32] B. K. R. Vangoor, V. Tarasov, and E. Zadok. To FUSE or Not to FUSE: Performance of User-Space File Systems. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 2017.

[33] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A Scalable, High-performance Distributed File System. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, 2006.

[34] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn. Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07*, pages 35–44. ACM, 2007.

[35] L. Xiao, K. Ren, Q. Zheng, and G. A. Gibson. Shardfs vs. indexfs: Replication vs. caching strategies for distributed metadata management in cloud storage systems. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 236–249. ACM, 2015.

[36] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX, 2012.

[37] Q. Zheng, K. Ren, G. Gibson, B. W. Settlemyer, and G. Grider. Deltafs: Exascale file systems scale better without dedicated servers. In *SC15 The International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015.