SkyhookDM: Data Processing in Ceph with Programmable Storage

JEFF LEFEVRE AND CARLOS MALTZAHN

Jeff LeFevre is an Assistant Adjunct Professor for Computer Science and Engineering at UC Santa Cruz. He currently leads the SkyhookDM project, and his research interests are in cloud databases, database physical design, and storage systems. Dr. LeFevre joined the CSE faculty in 2018 and has previously worked on the Vertica database for HP. jlefevre@ucsc.edu

Carlos Maltzahn is an Adjunct Professor for Computer Science and Engineering at UC Santa Cruz. He is the founder and director of Center for Research in Open Source Software (cross.ucsc.edu) and a co-founder of the Systems Research Lab, known for its cutting-edge work on programmable storage systems, big data storage and processing, scalable data management, distributed system performance management, and practical replicable evaluation of computer systems. In 2005 he co-founded and became a key mentor on Sage Weil's Ceph project. Dr. Maltzahn joined the CSE faculty in 2008, has graduated nine PhD students since, and has previously worked on storage for NetApp. carlosm@ucsc.edu

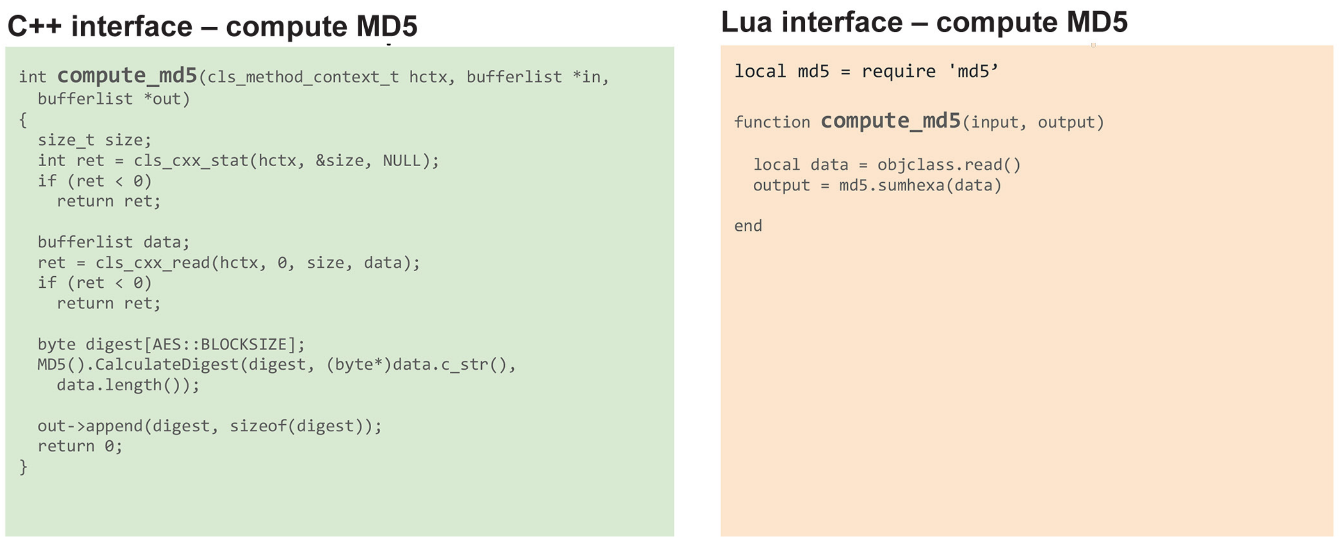
With ever larger data sets and cloud-based storage systems, it becomes increasingly more attractive to move computation to data, a common principle in big data systems. Historically, data management systems have pushed computation nearest to the data in order to reduce data moving through query execution pipelines. Computational storage approaches address the problem of both data reduction nearest the source as well as offloading some processing to the storage layer.

As storage systems become more disaggregated from client applications, such as distributed object storage (e.g., S3, Swift, and Ceph), there is new interest in computational storage disaggregated over networks [7]. There has also been a long line of efforts toward computational storage, including custom hardware and software for intelligent disks and active storage [5, 6, 13, 15], commercial appliances, and middleware approaches in the cloud [1, 2].

Recent research on *programmable storage systems* [4, 9–12, 14, 16] takes the approach of exposing, augmenting, or combining existing functionality already present in the storage system to increase storage capabilities, performance, or provide new storage APIs and services to clients. There are several benefits to this approach, including (1) building upon a trusted, production quality storage system rather than starting from scratch or building a one-off solution; (2) requiring no additional system or hardware (e.g., Zookeeper or specialized disks) to be installed to provide these new functions; and (3) avoiding the need for each client to reimplement available functionality on a per-use-case basis by simply accessing newly available storage services as they become available.

The Skyhook Data Management project (skyhookdm.com) [8, 9] utilizes programmable storage methods to extend Ceph with data processing and management capabilities. Our methods are applied directly to objects or across groups of objects by the storage system itself. These capabilities are implemented as extensions to Ceph's through its existing `cls` mechanism. This mechanism allows users to install custom functions that can be applied to objects in addition to `read()` or `write()`. Our approach that includes custom extensions along with data partitioning and structured data storage enables storage servers to semantically interpret object data in order to execute functions such as SQL SELECT, PROJECT, and AGGREGATE. We also developed extensions for data management functions that perform physical design tasks such as indexing, data repartitioning, and formatting. Partitioning and formatting can be especially useful in the context of row versus column-oriented formats for workload processing.

The immediate benefits of this approach are I/O and CPU scalability (for certain functions) that grows or shrinks along with the number of storage servers. Since objects are semantically self-contained (i.e., a database partition) and are the entities that custom functions operate on, and since the storage system automatically rebalances objects across available servers—our approach, using I/O and compute elasticity, can directly benefit any storage client application that is able to take advantage of these methods.



```
int compute_md5(cls_method_context_t hctx, bufferlist *in,
  bufferlist *out)
{
  size_t size;
  int ret = cls_cxx_stat(hctx, &size, NULL);
  if (ret < 0)
    return ret;

  bufferlist data;
  ret = cls_cxx_read(hctx, 0, size, data);
  if (ret < 0)
    return ret;

  byte digest[AES::BLOCKSIZE];
  MD5().CalculateDigest(digest, (byte*)data.c_str(),
    data.length());

  out->append(digest, sizeof(digest));
  return 0;
}
```

```
local md5 = require 'md5'

function compute_md5(input, output)

  local data = objclass.read()
  output = md5.sumhexa(data)

end
```

Figure 1: Example Ceph custom object class method to compute MD5 sum

The key insight of SkyhookDM is to simplify data management and minimize data movement by enabling the storage system to semantically interpret, manage, and process data. This can dramatically reduce the complexity of coordination and resources required higher up the software stack at the application layer. For example, applying a filter, building an index, or reformatting data can happen in parallel remotely on individual objects. This is because the necessary context for many common data processing and management tasks resides with the data, which makes data movement to establish computational contexts entirely unnecessary. In our work, this context includes the data semantics and the processing functions that are included in our formatted data within objects and our shared library extensions available within the storage servers, respectively.

For example, a single node database such as PostgreSQL can push query operations into the storage layer through its external table interface (foreign data wrapper), which can invoke these functions on objects and, hence, distribute computation across many storage nodes. For file interfaces, a similar mechanism is available in the scientific file format HDF5 with its Virtual Object Layer (VOL) that enables HDF5 to be mapped to non-POSIX storage back ends. Hence, similar to partitioning a database table, a large file can be "extended" by HDF5 functions into smaller objects across many storage nodes. SkyhookDM also provides a Python client using the Pandas DataFrame abstraction. In these ways, our methods can be used to scale out a database or another data-intensive application designed to run only on a single node.

Our approach to extend storage with data management tasks has several significant benefits:

- Increased query performance when pushing down computation directly to objects across many storage servers.
- Reduced network I/O and host interconnect bandwidth for computations that result in data reduction (e.g., SELECT, PROJECT, AGGREGATE).
- Reduced CPU at clients due to offloading and reduced CPU both at clients and servers due to creating, sending, and receiving fewer packets for data reduction queries.
- Reduced application complexity and resources for data management tasks such as indexing, re-partitioning data, or converting between formats (e.g., row to column).
- Support for operating on multiple data formats in storage, and the capability to extend support for other formats.
- Fewer application-level storage system assumptions of (possibly out-of-date) "storage-friendly" access patterns and more intelligent storage systems adapting to new devices.

Next we provide a short background, our architecture and methodology, and a few experiments to evaluate performance, scalability, and overhead of our approach to in-storage processing.

Background

Ceph is a widely used open source distributed object storage system created by Sage Weil at UC Santa Cruz and backed by Red Hat, IBM, and many other large corporations. It has no single point of failure, is self-healing, and scales to very large installations of 100's petabytes of data. It provides file, block, and object interfaces. New object methods can be created using cls extensions that are registered as READ and/or WRITE methods and then compiled into shared libraries within the Ceph source tree or via an SDK. These new shared libraries are installed on storage servers (also known as OSDs) in a directory well known to Ceph, `/usr/lib64/rados-classes/`.

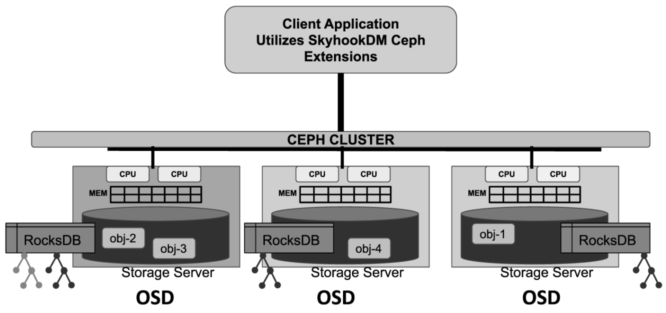


Figure 2: SkyhookDM architecture showing a three-node Ceph cluster with four objects

Ceph object classes currently support C++ and Lua interfaces, and an example implementation of an object method would be to compute a checksum, or perhaps generate a thumbnail image as part of a custom read or write method as shown in Figure 1. Critically, partial reads and writes of objects are also possible in Ceph, which is useful to reduce disk I/O for certain queries such as PROJECT a subset of all columns.

Apache Arrow and Google FlatBuffers are fast, in-memory serialization libraries. Arrow was developed for columnar processing and sharing data over the wire, and supports compression and interacts well with other formats, especially Parquet. FlatBuffers was developed for gaming, with a row-based abstraction; is very flexible, including a schema-free interface called FlexBuffers; and supports user-defined structs. They both offer a highest level abstraction of a table. We include these libraries within our shared library code to locally interpret and process object data.

SkyhookDM Architecture

Figure 2 shows our architecture, with a client application connected to a standard Ceph cluster with SkyhookDM cls extensions installed. The client application connects to Ceph using the standard librados library which makes the extensions available to the client. In this way databases such as PostgreSQL can invoke these extensions via their foreign data wrapper.

Figure 2 depicts a Ceph cluster of three Object Storage Devices (OSDs), each with its own CPU and memory resources that are utilized by our extensions for data processing. Each OSD stores a collection of objects and also has a local RocksDB instance that we exploit as an indexing mechanism.

SkyhookDM extensions are present as a shared library on each OSD, and these extensions can be applied to any local object for customized local processing. During processing, results can be returned to the client in a different format than the internal storage format, e.g., Arrow table, or PostgreSQL binary format from an object with FlatBuffer data format. Since our shared libraries are present on each OSD, they are immediately available to objects stored on newly added OSDs—for instance, when adding nodes to a Ceph cluster.

```
table FB_Meta {
  blob_format      : int32;    // enum SkyFormatType of data stored in blob
  blob_data        : [ubyte];  // formatted data (any supported format)
  blob_size        : uint64;   // number of bytes in data blob
  blob_deleted     : bool;     // is deleted?
  blob_orig_off    : uint64=0; // optional: offset of blob data in orig file
  blob_orig_len    : uint64=0; // optional: num bytes in orig file
  blob_compression : int=0;    // optional: populated by enum {none, lzw, ...}
}
```

Figure 3: Per object metadata wrapper regarding the serialized data partition (blob data) stored within

Methodology

Our work exploits Ceph's cls extension interface by first writing structured data to objects and then adding access methods implementing common SQL operations. We store structured data using popular and very efficient data serialization libraries such as Apache Arrow and Google FlatBuffers and use their APIs to implement new cls access methods. For greater flexibility to support multiple data formats, the structured data includes metadata about itself that expresses higher level information such as the data's current layout, whether or not it is compressed, and the data's length. Figure 3 shows this information, which is itself defined as a Flatbuffer wrapper. This helps to abstract away data layout information from the higher level client applications, creating flexibility to store and process data in various formats as well as reduce the need for client applications to keep track of data formats or compression on a per-object basis.

Physical and Logical Data Alignment

Physical and logical data alignment can be crucial for good performance, including with big data processing frameworks such as MapReduce [3]. In our work, physical and logical alignment is required such that when partitions are stored in objects of structures data, a given object contains a complete logical subset of the original data in order to interpret the data's semantics and perform any meaningful processing. For example, a database table partition could be stored in an object, resulting in a collection of complete rows that can be operated upon. In contrast, byte-aligned physical partitions (e.g., 64 MB) can result in incomplete rows, with part of a row stored across two different objects. Any processing for such rows would need to be performed at a higher layer and first perform a collect or gather operation across perhaps multiple storage servers. This would render object-local processing useless and result in unnecessary network I/O.

Data Partitioning and Format

In our work, we consider row and column-based partitions. Partitions are formatted using fast, in-memory data serialization formats: Google FlatBuffers for row partitions or Apache Arrow for column partitions. Both of these formats allow us to encode the data schema within, which allows the structured data to be interpreted by our cls methods.

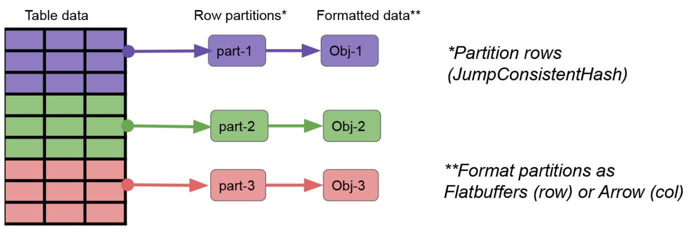


Table data
Row partitions*
Formatted data**
part-1
Obj-1
part-2
Obj-2
part-3
Obj-3
*Partition rows (JumpConsistentHash)
**Format partitions as Flatbuffers (row) or Arrow (col)

Figure 4: Data partitioning, formatting, and objects in SkyhookDM

For either row or column storage, data is partitioned, formatted, and named using conventions such as *table_name.partition_num*, resulting in a collection of objects per table where each object represents a logical data partition that is self-contained, with metadata indicating its semantics such as the table schema. Figure 4 depicts how data is partitioned and formatted for a database table. Data semantics are included within the format, which is then wrapped with our metadata wrapper, serialized as binary object data, and stored in Ceph. Data placement is handled by Ceph's pseudo-random data distribution function (CRUSH).

Rather than looking up all object names of a table, our object-naming convention includes content information, such as the table or column name. This results in constant-size metadata per table that includes only a name generator function and a few constants such as total number of objects. Further content-based information is possible to encode in the naming as well, such as row ranges for range-based row partitions (e.g., month). Such content-based object names and generator functions can then also be used for partition pruning during query plan generation.

This partitioning and formatting method achieves logical and physical alignment, embeds data semantics locally within each partition, and, along with the serialization format APIs and custom object classes, allows us to perform processing independently on each partition. Creating many objects (i.e., partitions) and scaling out the number of storage servers can enable a high degree of parallelism for data processing.

Data Layout and Physical Design

Within an object, there are several options to consider for laying out the data, either as a set of byte stream extents in a chunk store, as a set of entries in a key/value store, or combinations of both. The key-value store is a local instance of RocksDB, used by Ceph for managing the local collection of objects on the OSD. Object methods can also access RocksDB via Ceph's `object_map` or `omap` interface. SkyhookDM uses `omap` to store both logical information (data content) and physical information (data offsets). For instance, the logical information includes the row number of a particular value within an objects formatted data (e.g., row *i*) to provide quick lookups for point or range queries. The physical information includes the offsets and lengths of the sequence of

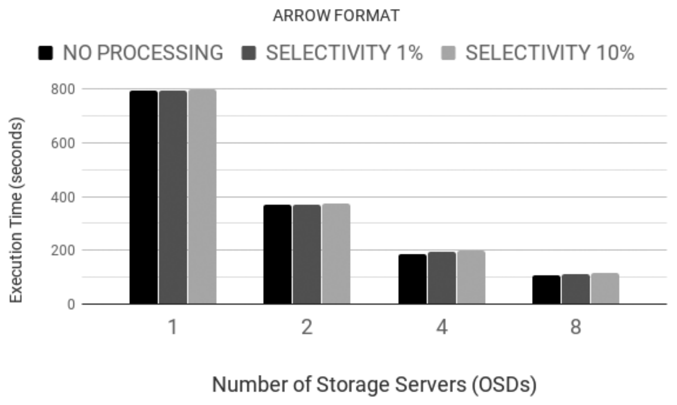


Figure 5: Query execution time as the number of storage servers is scaled out

data structures within an object. Both indexing and data layout within an object is a consideration for physical design [4] management, such as potentially storing each column of a table as a separate data structure in order to minimize the amount of disk I/O required to retrieve a given column. This helps to improve the performance of PROJECT queries, for example.

Evaluation

We executed all experiments on Cloudlab, an NSF bare-metal-as-a-service infrastructure. We used machine profile c220-g5 for all nodes, 2x Intel Xeon Silver 2.2 GHz, 192 GB RAM, 1 TB HDD, and 10 GbE. Our data set was the LINEITEM table from the standard TPC-H database benchmark, with one billion rows. We partition this table, format, and store into 10,000 objects of an equal number of rows. The objects are distributed by Ceph across all storage nodes. Data is formatted as FlatBuffer or Arrow as indicated.

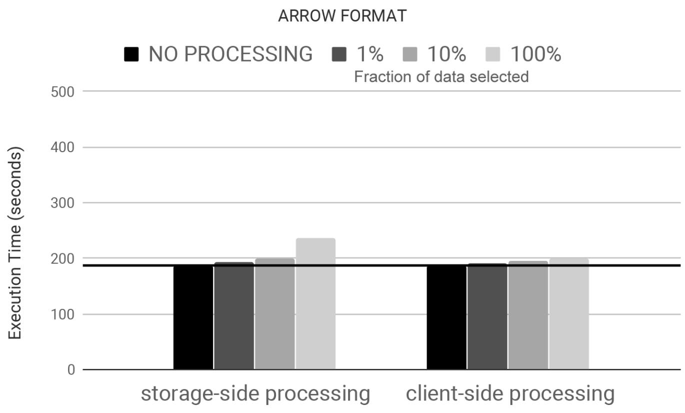


Figure 6: Query execution time for storage-side processing (offloading) versus processing on client machine with a four-node Ceph cluster

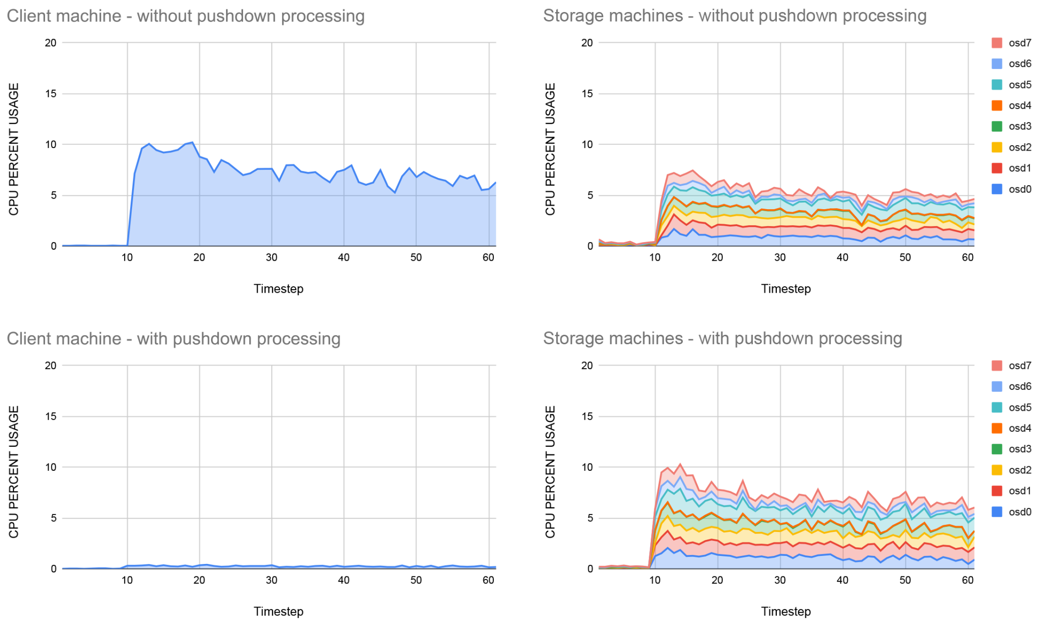


Figure 7: CPU usage during first 60 seconds for client machine (left) and eight storage servers (right, stacked) for 1% selectivity file query *without* pushdown processing (top) and *with* pushdown (bottom)

Performance Improvement with Pushdown Query Processing

Figure 5 shows query latency is reduced as the number of storage servers increases. The no-processing bar is a standard read of all the data, representing I/O scale out. The other bars show offloading of a select query for 1% and 10% of data rows. This represents I/O and CPU scale out. The offloading result tracks very closely to the I/O scale out, with little extra overhead for the storage servers to read and process versus only read. This highlights how the computation is distributed across all storage servers. While execution time is not reduced in this case, likely due to many cores and very fast network, the overhead to apply computation in Ceph is low, and CPU usage on the client is dramatically reduced, as we show next.

Overhead of Data Processing Libraries in Ceph

Figure 6 shows query execution time when processing data with all storage machines or on the single client machine. We first execute a standard read (no processing) as a baseline. Then we execute a query that selects 1%, 10%, or 100% of rows. In both cases there is little overhead to apply the data processing in storage except the case when selecting all data. This is due to the extra cost to both filter and then repackage and return each object when all rows pass the filter. This indicates the need for a statistics-based query optimizer to make wise decisions about what computations to offload.

CPU Usage with and without Offloading

Figure 7 shows that without offloading (no pushdown processing), the client spends over 5% of CPU usage to receive packets and apply SELECT (top left). With offloading the client CPU, usage is near zero (bottom left). This is because the client receives only 1% of the original data packets and does not have to apply SELECT. The processing work is shifted to the storage servers (bottom right), showing a small corresponding increase in the stacked total CPU usage that is distributed across all storage servers (bottom right). However, now the work done by storage servers is actually useful for data processing, whereas the work done by storage servers in the without pushdown case (top right) is simply creating and sending packets containing 99% unnecessary data.

Conclusion

SkyhookDM extends Ceph with object classes and fast serialization libraries to offload computation and data management tasks to storage. We have shown our approach has minimal overhead and scales with the number of storage servers. Our design is highly flexible, utilizing row or column-oriented data formats as well as the ability to dynamically convert between them directly in storage, eliminating the need to bring data into the client for processing or data management tasks. Extending our programmable storage approach to support custom data formats and more complex processing is another goal, and we are currently working on extensions for high energy physics data that uses the ROOT file format.

Acknowledgments
Support provided by the Center for Research in Open Source Software at UC Santa Cruz (cross.ucsc.edu), NSF Grant OAC-1836650, CNS-1764102, CNS-1705021. We would also like to thank everyone who has contributed to the open source SkyhookDM project.

References

[1] Amazon Redshift Spectrum documentation: https://docs.aws.amazon.com/redshift/latest/dg/c-using-spectrum.html.

[2] Swift storelet engine overview documentation: https://docs.openstack.org/storlets/latest/storlet_engine_overview.html.

[3] J. B. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis, and S. Brandt, "SciHadoop: Array-Based Query Processing in Hadoop," in *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC11)*, November 12–18, 2011, Seattle, WA.

[4] K. Dahlgren, J. LeFevre, A. Shirwadkar, K. Iizawa, A. Montana, P. Alvaro, and C. Maltzahn, "Towards Physical Design Management in Storage Systems," in *Proceedings of the 2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, November 18, 2019, Denver, CO.

[5] K. Keeton, D. A. Patterson, and J. M. Hellerstein, "A Case for Intelligent Disks (IDISKs)," *ACM SIGMOD Record*, vol. 27, no. 3 (September 1998), pp. 42–52.

[6] S. Kim, H. Oh, C. Park, S. Cho, S.-W. Lee, and B. Moon, "In-Storage Processing of Database Scans and Joins," *Information Sciences*, vol. 327 (January 10, 2016), pp. 183–200.

[7] P. Kufeldt, C. Maltzahn, T. Feldman, C. Green, G. Mackey, and S. Tanaka, "Eusocial Storage Devices: Offloading Data Management to Storage Devices that Can Act Collectively," *;login:*, vol. 43, no. 2 (Summer 2018), pp. 16–22.

[8] J. LeFevre and C. Maltzahn, "Scaling Databases and File APIs with Programmable Ceph Object Storage," 2020 Linux Storage and Filesystems Conference (Vault '20), February 24–25, 2020, Santa Clara, CA.

[9] J. LeFevre and N. Watkins, "Skyhook: Programmable Storage for Databases," 2019 Linux Storage and Filesystems Conference (Vault '19), February 25–26, 2019, Boston, MA.

[10] M. A. Sevilla, I. Jimenez, N. Watkins, J. LeFevre, P. Alvaro, S. Finkelstein, P. Donnelly, and C. Maltzahn, "Cudele: An API and Framework for Programmable Consistency and Durability in a Global Namespace," in *Proceedings of 32nd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2018)*, May 21–25, 2018, Vancouver, BC, Canada.

[11] M. A. Sevilla, R. Nasirigerdeh, C. Maltzahn, J. LeFevre, N. Watkins, P. Alvaro, M. Lawson, J. Lofstead, and J. Pivarski, "Tintenfisch: File System Namespace Schemas and Generators," 10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '18), July 9–10, 2018, Boston, MA.

[12] M. A. Sevilla, N. Watkins, I. Jimenez, P. Alvaro, S. Finkelstein, J. LeFevre, and C. Maltzahn, "Malacology: A Programmable Storage System," in *Proceedings of the 12th European Conference on Computer Systems (EuroSys '17)*, April 23–26, 2017, Belgrade, Serbia.

[13] M. Sivathanu, L. N. Bairavasundaram, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Database-Aware Semantically-Smart Storage," in *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST '05)*, December 13–16, 2005, San Francisco, CA.

[14] N. Watkins, M. A. Sevilla, I. Jimenez, K. Dahlgren, P. Alvaro, S. Finkelstein, and C. Maltzahn, "Declstore: Layering Is for the Faint of Heart," 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '17), July 10–11, 2017, Santa Clara, CA.

[15] L. Woods, Z. István, and G. Alonso, "Ibex: An Intelligent Storage Engine with Support for Advanced SQL Offloading," in *Proceedings of the VLDB Endowment*, vol. 7, no. 11 (July 2014), pp. 963–974.

[16] M. Sevilla, N. Watkins, C. Maltzahn, I. Nassi, S. Brandt, S. Weil, G.Farnum, and S. Fineberg, "Mantle: A Programmable Metadata Load Balancer for the Ceph File System," in *Proceedings of the 2015 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC15)*, November 15–20, 2015, Austin, TX.