Benchmarking DuckDB with Skyhook

Jayjeet Chakraborty
jayjeetc@ucsc.edu

March 10, 2022

1 Introduction

Skyhook [7] [4] is a programmable object storage system that allows offloading queries to the storage layer of Ceph [18]. It currently supports offloading only filter and projection queries provided in a Pythonic way. Skyhook does not have a native SQL interface to offload SQL queries, which severely limits Skyhook's usability. DuckDB [15], an in-memory SQL-based query engine, integrates with Skyhook using the Arrow Dataset API and allows executing SQL queries in Skyhook. We leverage this integration to offload SQL queries to the storage layer, which was impossible earlier. To benchmark this integration, we use TPC-H [9] with a scale factor of 100 and run all 22 queries. We report the results of our evaluations, the challenges we faced, and the future work that needs to be done. Overall, the contribution of our work are as follows:

- An integration of DuckDB and Skyhook in Python using the Arrow Dataset API. The code for our implementation can be found here.
- A performance evaluation of our integration over TPC-H benchmarks using a scale factor of 100.
- A brief description of the different technologies such as Skyhook, Ceph, Arrow, TPC-H, and DuckDB to set the context for the readers.

2 Problem

SQL, although quite old, is still a language of preference for data practitioners around the world. This is primarily because of its vast community, ease of use, and stature as a standard and robust data management language. Every popular data processing system such as Spark [22], Snowflake [8], Redshift [12], BigQuery [13], and Presto [17] support SQL queries. However, the Arrow `Dataset` API does not support SQL queries and can only execute filter and projection queries written in the form of Python code. Since Skyhook is exposed through the Arrow `Dataset` API, its interface is also restricted to what Arrow supports.

Since Skyhook did not have a SQL interface, the goal of this project is to make a SQL interface available for Skyhook users. We prefer not to modify existing systems; hence, we use a 3rd-party SQL-based database layer, DuckDB, that can be integrated seamlessly with Skyhook using the `Dataset` API. This integration allows us to benchmark the performance of Skyhook over industry-standard benchmarks such as TPC-H, which was not possible earlier. We believe that having a SQL layer on top of Skyhook is necessary to make Skyhook user-friendly and make it integrable with other SQL-based data management systems.

3 Background

This section discusses in detail the different systems and technologies that we integrate to build our implementation.

3.1 Apache Arrow

Apache Arrow is an in-memory columnar format optimized for efficient analytics operations on modern hardware. It describes a standard data format for exchanging structured data between different systems without serialization or deserialization. The Arrow format allows compute engines and query execution engines to maximize their efficiency when scanning and iterating large chunks of data. The contiguous columnar layout of Arrow enables vectorization using the latest SIMD operations on modern hardware. Besides being an in-memory format, it is also a collection of different data processing components that allow building parts of a data processing system. Some of the most-used components are Flight, a gRPC-based data transfer protocol [6]; Feather, A Arrow-based columnar persistent storage format [21]; Gandiva, An LLVM-based expression compiler [5]; Dataset API, An abstraction for realizing datasets over a directory of files [2]. Arrow is also language-independent as it has APIs in several different programming languages such as C++, Java, Python, Rust, R, JavaScript, and Julia. Several popular data processing systems such as Spark, Dask [16], Ray [14], Pandas, Parquet [1] have added support for Arrow data and Arrow data sources [7].

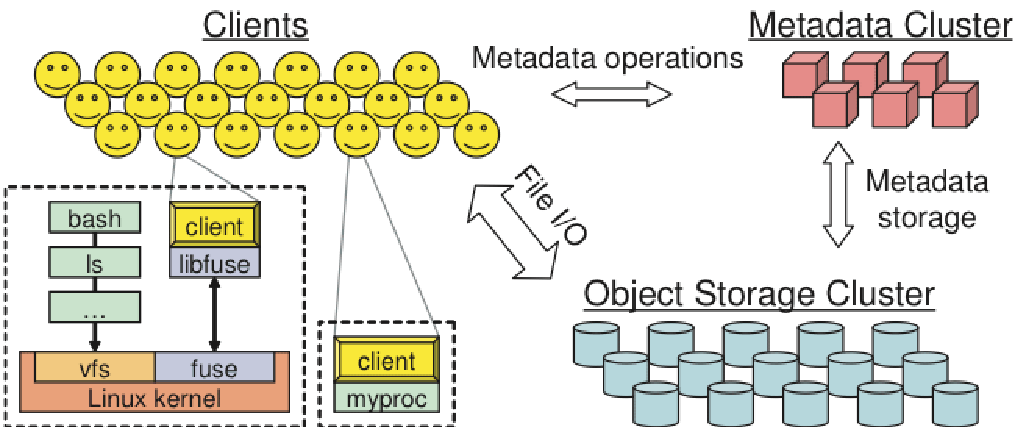

Figure 1: Architecture of Ceph [18].

3.2 Ceph

Ceph is a petabyte-scale distributed and programmable object storage system providing 3-in-1 interfaces for object, file, and block-level storage. Ceph was developed as a part of a Ph.D. thesis at UC Santa Cruz and was first published in 2006. Ceph does not use any other filesystems internally; rather, it manages the HDDs and SSDs directly with its custom storage backend, BlueStore, part of its RADOS [20] object store. Ceph was developed for commodity hardware, and hence it can replicate data across a cluster, employing several techniques such as erasure coding, replication, and snapshots. Ceph is unique as it does not have a single point of failure. This is because of the CRUSH [19] map feature that Ceph provides. CRUSH maps contain object-OSD mappings, which the Ceph client uses to calculate the location of an object in a Ceph cluster. After figuring out the location of an object, the client directly connects to a Ceph OSD and reads the object. Ceph also provides a plugin-based object store extension mechanism through its Object Class SDK [3]. This SDK allows writing embeddable plugins (in the form of shared libraries) in C++ and Lua containing logic to access and modify objects inside the storage servers within the RADOS I/O path. The SDK provides a subset of POSIX-like system calls such as `cls_cxx_read`, `cls_cxx_write`, and `cls_cxx_stat` to be able to read, write, and stat objects respectively. This SDK is heavily used by several Ceph components such as Ceph RBD (Rados Block Device) and CephFS (the Ceph filesystem). The architecture of Ceph is given in Figure 1.

3.3 DuckDB

DuckDB is an analytical in-memory database engine designed to be fast, reliable, and easy to use. It was developed by the CWI Database Architectures Group in the Netherlands and was first published in CIDR 2020. DuckDB is similar to SQLite in the sense that it is embeddable. It can be easily embedded in the form of shared libraries within host processes. DuckDB features modern Database features such as columnar data storage, ART-based indexing, vectorized execution, single-file storage, and multi-version concurrency control. DuckDB provides wrappers to different programming languages such as C, Java, Python, NodeJS, and R. DuckDB integrates seamlessly with Apache Arrow using the Arrow Dataset API, which allows scanning data formats and data sources supported by Arrow. DuckDB offloads a SQL query plan's filter and projection components into the Arrow Dataset API, which the Parquet reader then handles for row group and column pruning. DuckDB supports all 22 TPC-H queries and most of the TPC-DS queries.

3.4 Skyhook

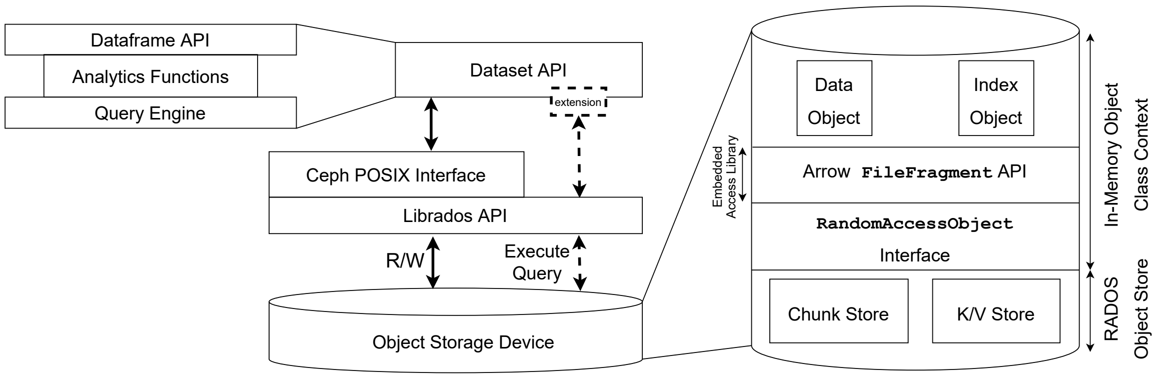

Figure 2: Architecture of Skyhook [7].

Skyhook is a programmable storage system built on top of Ceph that can offload query executions from the client to the storage layer. Skyhook supports scanning datasets containing files of different formats such as Parquet, Feather, CSV, JSON, or any other format as long as they are supported by Apache Arrow. Skyhook is built as a storage-side plugin for Ceph, which, when embedded inside Ceph OSDs in the form of shared libraries, allows executing queries within the Ceph storage nodes. Skyhook is exposed to the clients using a Arrow `FileFormat` API extension called the `SkyhookFileFormat` API. This API, when used with the Arrow `Dataset` API, allows offloading dataset scans instantly. Internally, `SkyhookFileFormat` leverages

Ceph filesystem metadata containing file striping information to map files in CephFS to RADOS objects and applies computation on those objects directly, bypassing the POSIX layer. In the storage plugin, we reuse the `ParquetFileFormat` of Arrow out-of-the-box to scan RADOS objects containing Parquet binary data. Generally, Arrow APIs cannot scan RADOS objects. However, we make this possible by creating a thin random-access filesystem shim called the `RandomAccessObject` API that wraps a RADOS object, keeps track of the file pointer, and provides a file-like view over the objects. This interface plugs into Arrow APIs seamlessly and allows scanning objects as files. One exception for Skyhook is that it requires files to be written in a specific way. To be able to scan Parquet files with Skyhook, they should be self-contained within a single RADOS object. This $1:1$ mapping is required to be able to translate filenames into object IDs easily and to let Arrow APIs scan a RADOS object as a complete Parquet file, as Arrow APIs cannot scan multiple physical files as a single logical file. To ensure every file goes into a single object while writing files to CephFS, we change the stripe unit of CephFS to match the size of the Parquet files being written. The architecture of Skyhook is given in Figure 2.

3.5 TPC-H

TPC-H is an industry-standard benchmark for evaluating decision support systems. Basically, it is used for benchmarking OLAP systems. TPC benchmarks usually come in different scale factors such as 1, 10, 100, 1000, 10000 where a particular scale factor refers to a corresponding gigabyte of data. For example, a scale factor of 100 implies a 100GB dataset with 1 billion rows in total. The TPC-H dataset contains 8 tables namely lineitem, orders, customer, supplier, part, partsupp, region, and national. Amongst these, lineitem is the largest table and consists of about 60% of the dataset. Each of these tables shares some common attribute, and hence, all of them can be joined. TPC-H provides a set of 22 queries which are all read queries. TPC-H does not have queries to update the database. Several of these queries consist of big joins and nested queries, joining up to 7 tables at once in a nested manner. Due to these enormous joins, some TPC-H queries generally need sufficient resources such as memory to run successfully.

4 Project Description

In this section we describe the architecture of the DuckDB-Skyhook integration and describe the workload used for the benchmarks. The architecture of the integration is given in Figure 3.

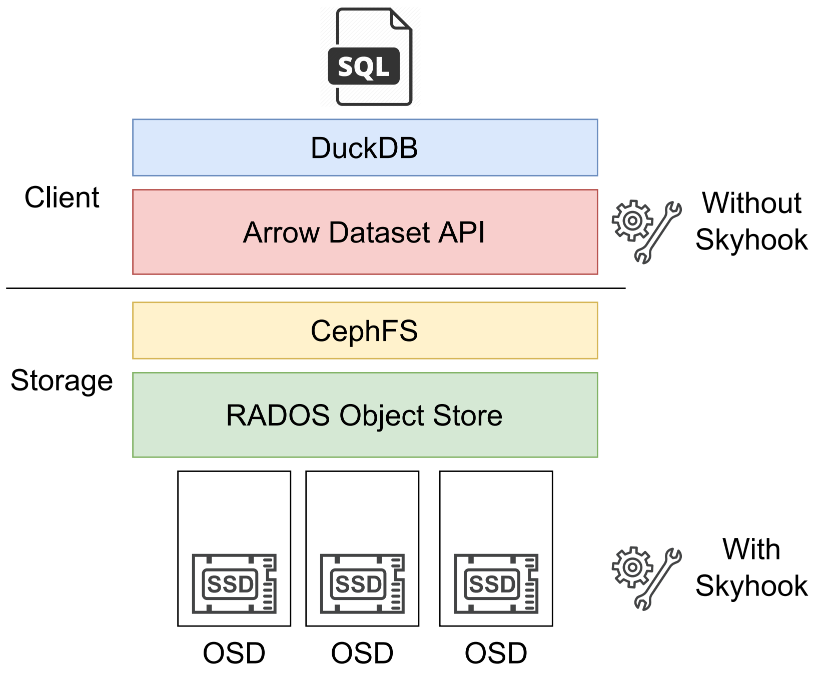

Figure 3: Architecture of the DuckDB-Skyhook integration.

4.1 Architecture

Skyhook is exposed to the user's through the `SkyhookFileFormat` API which is an extension of the `FileFormat` interface of the Arrow `Dataset` API. The `FileFormat` interface describes format-specific logic to scan files of different structured data formats. The `SkyhookFileFormat` API is a generic API that can offload query executions on datasets of different file formats such as Parquet, ORC, Feather, CSV, and JSON. The Arrow `Dataset` API takes the file format as an argument, where we pass "skyhook" instead of "parquet" to offload query executions to the Ceph object storage layer immediately. The `SkyhookFileFormat` API can be customized with a different Ceph configuration path and Ceph data pool as per the cluster deployment.

DuckDB allows querying Arrow Dataset's using the Arrow `Dataset` API. It uses the Arrow `Dataset` API to push down predicates and projections to the Arrow layer, which in turn pushes them down to the Parquet backend. We leverage this integration of DuckDB and `Dataset` API to replace Parquet with the Skyhook backend. In this way, the filter and projection parts of SQL-queries are seamlessly offloaded to Skyhook, where they are executed inside Ceph OSDs. Listing 1 shows a code sample of how the DuckDB and Arrow Dataset API integration works.

```python
import duckdb
import pyarrow.dataset as ds

dataset_ = ds.dataset("/path/to/dataset", format="parquet")
conn = duckdb.connect()
conn.execute("SELECT * FROM dataset_").fetchall()
conn.close()
```

Listing 1: DuckDB and Dataset API integration.

4.2 Dataset

We generated a TPC-H dataset of scale factor 100 using the `dbgen` utility from the TPC-H organization. We used `dbgen` in multi-threaded mode to speed up our dataset generation process. The multi-threaded generation of the 100GB TPC-H dataset took about 17 hours. Using `dbgen` in multi-threaded mode resulted in a partitioned TPC-H dataset. After the `dbgen` process finished execution, we converted the generated CSV files to Parquet format by applying table schemas collected from the TPC-H documentation. We then used Skyhook's `SplittedFileWriter` to write the Parquet files into CephFS by splitting them into 16MB chunks. We chose 16MB chunk size because Skyhook earlier showed better performance with object sizes of 16MB. The Parquet files we wrote were snappy compressed, the default compression strategy of Parquet. The generated CSV dataset was replicated 3 times across the Ceph cluster, and the total size before replication was about 109GB. After converting the files from CSV to Parquet format, the dataset size shrank to about 33GB. We plan to perform the same benchmarks with a TPC-H dataset of scale factor 10000 (1TB) in the future.

5 Evaluations

In this section, we describe the performance of offloading TPC-H queries to the storage layer of Ceph using Skyhook. We used bare-metal nodes from CloudLab [10], an NSF-funded bare-metal as a service infrastructure, to perform our experiments. The nodes we used were codenamed "c220g5" in CloudLab and had 2 Intel Xeon Silver 10-core CPU, 192 GB DRAM, 480GB SATA SSD, and a 10Gb ethernet card. We used a TPC-H dataset of scale factor 100 as our workload.

The Ceph cluster we used had 8 Ceph OSDs, 2 Ceph MDSs, and 1 Ceph MGR. Each of the OSDs was configured to use 8 threads internally to have enough parallelism while avoiding any lock contention due to excessive multi-threading. The Ceph OSDs used the SATA SSD on each node. The data pool in Ceph was replicated

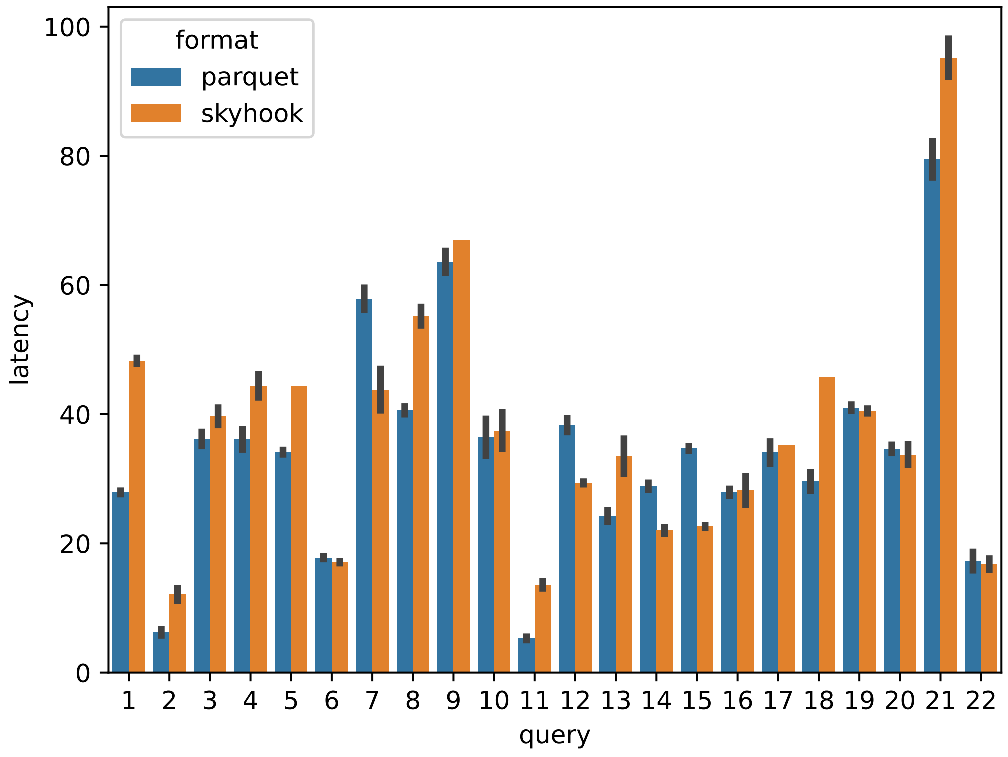

Figure 4: Query latency of running TPC-H at a scale factor of 100 with and without Skyhook.

3-way as in most Ceph configurations and we used 256 PGs (Placement Groups) for our data pool, which is enough for a Ceph cluster with 8 OSDs according to Ceph documentation.

We ran the benchmarks with every query executing with 40 concurrent threads to ensure enough intra-query parallelism. We did each run for 10 iterations to add some amount of statistical significance to our results. While performing our experiments, we found out that there was an occasional memory leak on the client, which only occurred when the DuckDB-Skyhook integration was used. The queries with the bigger joins, joining more than 2 tables at once, are most affected by this memory leak. The memory leak loads 2-3 times the amount of data that needs to be loaded typically and computes for a longer duration, proportional to the amount of memory loaded. We call this a memory leak because, after checking the Disk I/O performed during a leaky run, we found out that there was no extra I/O done. This points to the integration making useless copies of the data. For this project, we only report the numbers from the non-leaky runs to make an unbiased comparison of DuckDB performance with and without Skyhook. The result from the query latency experiment is given in Figure 4.

Skyhook outperforms Parquet in some queries such as 6, 7, 12, 14, 15, 19, 20, 22. We can attribute this performance improvement to the fact that Skyhook reduces data movement significantly, although in this case, the improvement is not very high, probably due to the very high serialization/deserialization overhead Skyhook has. In the rest of the queries, Skyhook performs worse than Parquet. The reason behind this might be that the queries do not have enough selectivity, in which case the serialization overhead of Skyhook exacerbates, as it is not traded off by reduced data movement. In the future, we plan to investigate this performance observation of Skyhook after fixing the memory leak issue that we currently face. We want to perform Flame Graph [11] and logging and tracing experiments to narrow down the root cause and hopefully fix the issue.

6 Conclusion

This paper presents the integration of a SQL-based in-memory database system, DuckDB with a programmable object storage system, Skyhook to allow offloading query executions from the client to the storage layer of object storage systems. We begin with a brief description of the different systems that participate in the integration such as Ceph, Arrow, and Skyhook. We describe our architecture and how we use the Arrow Dataset API as a common layer between DuckDB and Skyhook to offload filters and projections to the storage layer. We describe our dataset along with how we generate our workload to perform the benchmarking experiments. We performed evaluations to measure the query latency improvement by offloading queries to Skyhook and present the results. We used a industry-standard benchmark, TPC-H, with a scale factor of 100 for our evaluations. Irrespective of the performance achieved, we were able to run all the 22 TPC-H queries both with and without Skyhook. Currently, we did not get the best performance improvement due to various overheads and inefficiencies of Skyhook, but we plan to redo the experiment once those issues are fixed, ideally getting better results.

References

[1] Apache parquet. https://parquet.apache.org/.

[2] Arrow dataset api. https://arrow.apache.org/docs/python/dataset.html.

[3] Ceph object class sdk. https://docs.ceph.com/en/latest/rados/api/objclass-sdk/#sdk-for-ceph-object-classes.

[4] Skyhook: Bringing computation to storage with apache arrow. https://arrow.apache.org/blog/2022/01/31/skyhook-bringing-computation-to-storage-with-apache-arrow/.

[5] Gandiva: A llvm-based analytical expression compiler for apache arrow. https://arrow.apache.org/blog/2018/12/05/gandiva-donation/, 2018.

[6] Introducing apache arrow flight: A framework for fast data transport. https://arrow.apache.org/blog/2019/10/13/introducing-arrow-flight/, 2019.

[7] Jayjeet Chakraborty, Ivo Jimenez, Sebastiaan Alvarez Rodriguez, Alexandru Uta, Jeff LeFevre, and Carlos Maltzahn. Skyhook: Towards an arrow-native storage system. *CCGrid*, 2022.

[8] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, et al. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*, pages 215–226, 2016.

[9] Markus Dreseler, Martin Boissier, Tilmann Rabl, and Matthias Uflacker. Quantifying tpc-h choke points and their optimizations. *Proceedings of the VLDB Endowment*, 13(8):1206–1220, 2020.

[10] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[11] Brendan Gregg. The flame graph. *Communications of the ACM*, 59(6):48–57, 2016.

[12] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. Amazon redshift and the case for simpler data warehouses. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1917–1923, 2015.

[13] SPT Krishnan and Jose L Ugia Gonzalez. Google bigquery. In *Building Your Next Big Thing with Google Cloud Platform*, pages 235–253. Springer, 2015.

[14] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 561–577, USA, 2018. USENIX Association.

[15] Mark Raasveldt and Hannes Mühleisen. Data management for data science-towards embedded analytics. In *CIDR*, 2020.

[16] Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, volume 126. Citeseer, 2015.

[17] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, et al. Presto: Sql on everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1802–1813. IEEE, 2019.

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

[19] Sage A Weil, Scott A Brandt, Ethan L Miller, and Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 31–31. IEEE, 2006.

[20] Sage A Weil, Andrew W Leung, Scott A Brandt, and Carlos 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, 2007.

[21] Hadley Wickham. Feather: A fast on-disk format for data frames for r and python, powered by apache arrow. https://www.rstudio.com/blog/feather/.

[22] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.