Performance Analysis of Data Processing in Distributed File Systems with Near Data Processing

Shiyue Hou*, Nathan R. Tallent†, Li Wang*, Ningfang Mi*
*Department of Electrical and Computer Engineering, Northeastern University, MA USA
†Advanced Computations and Maths Division, Pacific Northwest National Laboratory, Richland, WA, USA

2024 International Symposium on Networks, Computers and Communications (ISNCC) | 979-8-3503-6491-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISNCC62547.2024.10758994

***Abstract*—In the era of big data, the escalating volume and velocity of data generation pose significant challenges in data processing. Traditional systems like Spark [1] and Hadoop [2] manage the increasing amount and velocity of data by improving data placement and processing speeds. However, they face inherent limitations due to the essential data movement required for processing. In this paper, we explore the Skyhook framework, a novel extension of the Ceph distributed system, which significantly reduces the need for data movement. We present an extensive case study using the Skyhook framework, applying it with the TPC-H and K-means clustering algorithms. More specifically, we leverage the TPC-H benchmark to distinguish between CPU-intensive and I/O-intensive tasks. We explore the integration of K-means clustering into SQL, coupled with a near-data processing system to offload the computational burden of the K-means clustering algorithm to storage nodes. We conduct a comprehensive performance evaluation of distributed data processing applications across three processing approaches: traditional layout (baseline), optimized layout, and near-data processing. Additionally, we introduce the use of the FIO tool to simulate real-world system workloads, enabling the measurement of performance metrics such as average latency and CPU utilization. Our research is a significant advance in understanding how to optimize data processing systems to meet the demands of the modern data landscape.**

***Index Terms*—Near data processing, system performance analysis, distributed system, optimized layout**

I. Introduction

In the contemporary information era, the exponential surge in data across various domains presents both opportunities and challenges. While efforts have been made to bring computation closer to data, exemplified by platforms like Spark [1] and Hadoop [2], each system approaches this task differently. Spark's in-memory computing paradigm ensures that data is cached in memory whenever possible, reducing the need for disk I/O and enabling near-data processing. This capability allows Spark to execute iterative algorithms and queries efficiently with low latency. On the other hand, Hadoop's MapReduce framework employs optimization techniques such as data compression, combiners, and custom partitioning to minimize data movement during the map and reduce phases, enhancing data locality and processing efficiency.

However, despite these optimizations, both Spark and Hadoop still involve data movement. Spark primarily processes data in memory, necessitating the loading of data from disk into the memory of worker nodes for computation. Similarly, while MapReduce tasks are scheduled on nodes where data blocks are stored, the actual processing occurs away from the data location. Recognizing the ongoing challenge of reducing data movement and exploring new avenues for improvement, researchers are investigating various techniques. These include moving processing closer to data, proposing new data layouts, and enhancing the parallelism of data processing. A promising direction involves pairing near-data processing with innovative data layouts.

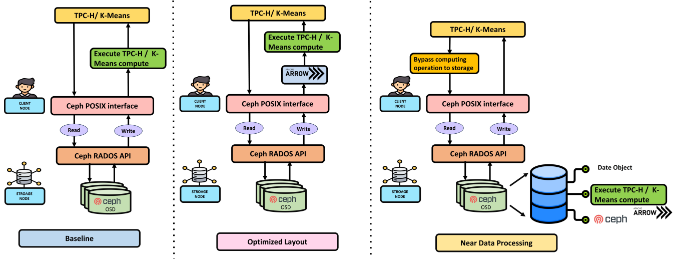

Fig. 1: Architecture overview for traditional layout (left), optimized layout (middle), and near data processing (right).

One such advancement is exemplified by Skyhook [3], a programmable object storage system built upon the Ceph [4] distributed system, shown as the right plot (Near Data Processing) in Fig. 1. Skyhook allows computation to be offloaded directly to the storage layer, eliminating the need for extensive data transfers between storage and processing nodes. By executing computations closer to the data, Skyhook minimizes data movement over the network. Moreover, Skyhook utilizes a columnar representation of data akin to formats like Apache Parquet [7] and Apache Arrow. This approach enhances compression ratios and enables selective data retrieval, further reducing the volume of data transferred across the network for processing.

We also create an optimized layout system and a traditional layout system (as the baseline), utilizing Apache Arrow and without it, respectively, depicted in Fig. 1. The optimized layout system significantly improves performance by minimizing the need for data serialization and deserialization. By leveraging a columnar memory format specifically optimized for analytic operations, this system minimizes serialization requirements on the client node and reduces memory consumption compared to traditional row-based formats. In contrast, the baseline system involves copying all data from the storage node into client memory and necessitates serialization and deserialization processes on the client node. This approach incurs maximal data copying overhead and diminishes overall system resource utilization.

In this paper, we present a case study of applying the Skyhook framework to a new use case involving TPC-H [5] and K-means [6] cluster algorithms. Additionally, we introduce the utilization of the FIO tool to simulate real-world system workloads, enabling the measurement of near-data processing, optimized layout system, and baseline system metrics such as average latency and CPU utilization. Through this study, we aim to shed light on the efficacy of near-data processing techniques in modern data processing workflows. The contributions of our work are summarized as follows:

- Analyzing workload characteristics of the near-data processing system, we leverage the TPC-H benchmark to distinguish between CPU-intensive and I/O-intensive tasks.
- This paper explores the integration of K-means clustering into SQL, coupled with a near-data processing system to offload the computational burden of the K-means clustering algorithm to storage nodes.
- This study conducts a comprehensive performance evaluation of distributed data processing applications (e.g., TPC-H and K-means clustering) on three alternative processing approaches: traditional local processing (baseline), optimized layout, and near-data processing, and understands system behaviors and performance benefits under various workloads and node conditions.

In the remainder of the paper, we present the background of our performance analysis in Section II. Sections III and IV elaborate on our experimental setup and evaluation, employing TPC-H, the K-means clustering algorithm, and FIO. Finally, our findings are summarized in Section V.

II. Background

A. Ceph

Ceph [4] is an advanced open-source storage platform that offers high scalability and reliability in a distributed network. It uses Object Storage Daemons (OSD) for managing data storage, replication, and recovery. With no single point of failure, Ceph ensures robustness via Monitors that track the cluster's state. It supports block, object, and file storage in one system. The CRUSH algorithm boosts performance by efficiently distributing data, making Ceph suitable for high-availability storage in cloud services and large data centers.

B. Apache Parquet

Apache Parquet [7] is a columnar storage file format optimized for big data frameworks like Hadoop and Spark, enhancing data compression and encoding. By organizing data by columns instead of rows, Parquet improves compression and read speeds, especially for column-specific analytics. It utilizes compression codecs such as Snappy and GZIP and encoding methods like dictionary and run-length encoding to boost storage efficiency and query performance. Ideal for handling complex nested data and schema evolution, Parquet also supports predicate pushdown to improve query efficiency by omitting irrelevant data blocks, making it well-suited for high-performance data applications.

C. Skyhook

Skyhook [3] is a programmable storage system that enhances Ceph by offloading query executions from the client to the storage layer. It supports scanning datasets in various formats including Parquet, Feather, CSV, and JSON, among others, provided they are compatible with Apache Arrow. Implemented as a storage-side plugin within Ceph, Skyhook operates through shared libraries embedded in Ceph OSDs, enabling query execution directly on the storage nodes. The system interfaces with clients via an Arrow FileFormat API extension known as the SkyhookFileFormat API. This API, in conjunction with the Arrow Dataset API, facilitates immediate offloading of dataset scans.

Internally, the SkyhookFileFormat uses Ceph [14] filesystem metadata, which includes file striping information, to map files in CephFS to RADOS objects and directly process these objects, circumventing the POSIX layer. The plugin reuses Arrow's ParquetFileFormat to scan RADOS objects that contain Parquet binary data—a function not typically supported by Arrow APIs. To overcome this, Skyhook introduces a RandomAccessObject API—a filesystem shim that provides a file-like view over RADOS objects, maintains the file pointer, and integrates seamlessly with Arrow APIs to allow for scanning objects as if they were files.

A specific requirement of Skyhook is that for effective scanning of Parquet files, each file must be self-contained within a single RADOS object. This one-to-one mapping simplifies the translation from filenames to object IDs, enabling Arrow APIs to interpret a RADOS object as a complete Parquet file. To ensure that each file is stored as a single object, the stripe unit of CephFS is adjusted to align with the size of the Parquet files being written.

D. Data Processing Applications

In this paper, we consider the following two representative data processing applications for performance evaluation.

TPC-H [5] is a benchmark suite for measuring the performance of database management systems (DBMS) using a decision support system (DSS) workload. It is designed to evaluate both query processing capabilities and data management efficiency of relational databases in handling complex queries, typically associated with large volumes of data. The benchmark consists of a set of business-oriented ad-hoc queries and concurrent data modifications designed to emulate real-world decision support systems. TPC-H includes a database schema with eight tables representing typical business data, such as customer, order, and part information, and it supports queries involving various SQL features like joins, group by and nested subqueries.

K-means clustering [6] is an unsupervised machine learning technique used to partition a set of data points into K distinct non-overlapping clusters, where each data point belongs to the cluster with the nearest mean. The process begins by initializing K centroids randomly, which are used as the starting points for each cluster, and then iteratively performs two steps: assignment and update. In the assignment step, each data point is assigned to the nearest centroid based on a distance metric, typically Euclidean distance. In the update step, the centroids are recalculated as the mean of the data points assigned to each cluster. These steps are repeated until the centroids no longer move significantly, indicating convergence, or until a specified number of iterations is reached. K-means is favored for its simplicity and efficiency in processing large data sets, but it requires the number of clusters to be specified beforehand and can be sensitive to initial centroid placement, affecting outcomes.

III. The Proposed Methods

This section introduces our three design paradigms and provides an in-depth description of the platform implementation. In addition, we integrate the TPC-H benchmark and K-means clustering algorithm to evaluate system performance.

A. Near Data Processing

We incorporate Skyhook [3] to enable the near-data processing capability, which is a query-based in-storage data processing system developed on top of the Ceph distributed system. Skyhook introduces an API named 'SkyhookFileFormat,' designed to extend the FileFormat of the Apache Arrow Dataset API. It supports Parquet file formats, significantly reducing read I/O requests to the data store. Moreover, all data is stored at the Ceph storage nodes in Parquet files. Refer to Fig. 1 for the architecture.

Moreover, Skyhook facilitates the offloading of queries to the storage nodes. This allows client nodes to bypass compute operations by delegating them to the storage nodes, where the storage nodes retrieve Parquet data, load it into memory for processing, and subsequently return the computed results to the client nodes. This approach minimizes data movement and enhances storage memory locality, although it involves serialization to the client node. We have further expanded Skyhook's capabilities by integrating the K-means clustering algorithm within SQL, using both Skyhook and DuckDB to foster near-data processing. Consequently, the K-means algorithm can be executed directly on Skyhook with computational tasks offloaded to the storage node.

B. Optimized Layout

We have developed an optimized storage layout tailored for the Ceph distributed system. This solution involves converting data into the Parquet format prior to storage within Ceph nodes. Adopting Parquet not only reduces the requisite physical storage space but also enhances the speed of read/write operations due to the smaller data footprint. By leveraging Apache Arrow's capabilities, we efficiently transfer Parquet data from the storage nodes directly into client memory, as depicted in Fig. 1. Apache Arrow's zero-copy columnar in-memory format eliminates the need for serialization and deserialization, thereby alleviating the associated performance drawbacks.

Expanding on this infrastructure, we have integrated DuckDB [13] to leverage Arrow so that queries have efficient access to the optimized columnar data format. This integration not only improves performance by enhancing memory locality at the client node but also minimizes the need for data serialization, resulting in more efficient query processing and reduced latency.

C. Baseline

In our baseline scenario, we utilize a traditional data format layout. The baseline approach involves reading data from the storage node and subsequently transferring it to the client node. Following this, we execute the TPC-H and K-means algorithms directly on the client node. This method suffers from poor memory locality on the client and necessitates the introduction of serialization and deserialization processes, which can impact performance adversely, as shown in Fig. 1.

IV. Experiments

In this section, we present our experiments designed to compare average application latency and CPU utilizations across client and storage nodes for the TPC-H [5] and K-means clustering algorithms [6]. These experiments are conducted on a private computing cluster equipped with 16 Intel Xeon W-2245 CPUs, 62GB of RAM, and running Ubuntu 20.04. We build three storage nodes, each with a single Ceph OSD attached to an NVMe drive, and one client node equipped with a Ceph Monitor (MON) and a Ceph OSD to oversee the Ceph storage cluster. We mount a CephFS interface in user mode via the cephfuse utility on a 3-way replicated pool.

For the TPC-H benchmark, we work with 1GB of data converted to uncompressed Parquet format. Direct reading from RADOS allows us to avoid any cache-related performance discrepancies by replicating the same file under consistent conditions. Additionally, we generate 1GB of synthetic data consisting of 50,000,000 samples organized into distinct clusters, using the sklearn.datasets API [9]. Our experimental datasets are configured to compare near-data processing, optimized layout, and baseline conditions. We utilize the Python version of the Arrow Dataset API for these tests and employ Python's ThreadPoolExecutor to launch parallel scans, adhering to an asynchronous I/O model. The experiments measure both end-to-end latency and CPU utilization under near-data processing, optimized layout, and baseline design paradigms.

A. Average Latency

1) TPC-H workload: In our study of TPC-H queries and their resource usage, we draw upon insights from previous research [10] to categorize these queries based on their resource intensity, as detailed in Table I. Specifically, we distinguish

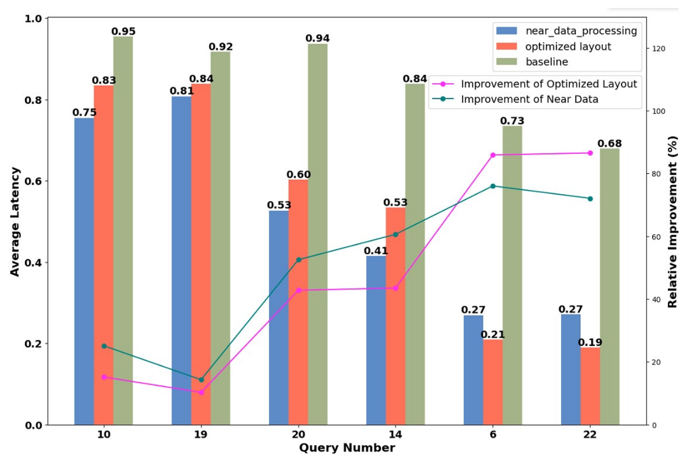

Fig. 2: TPC-H query latency for near data processing, optimized layout, and baseline with relative improvement.

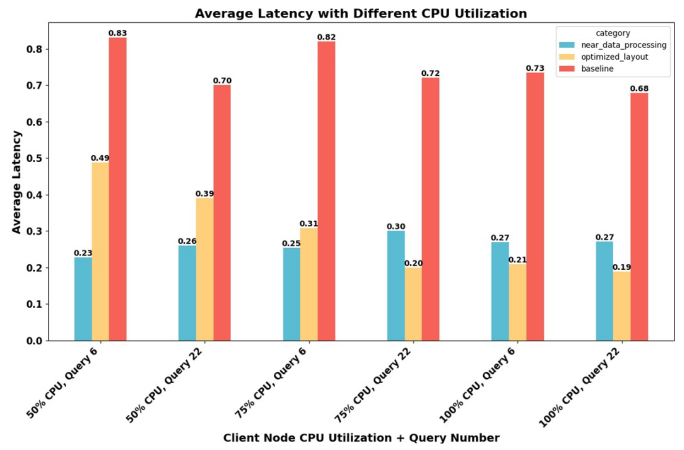

Fig. 3: TPC-H query latency for near data processing, optimized layout, and baseline start from 50%, 75%, and 100% available CPU capacity at the client node.

between CPU-intensive queries, which typically involve sorting and filtering operations, and I/O-intensive queries, which primarily require the loading of large datasets into memory or the execution of operations across multiple datasets, such as joins. Notably, certain queries, like Q14, and Q20, exhibit dual characteristics, showing both CPU and I/O intensity.

To assess the enhancement in the performance of near data processing and optimized layout system over the baseline, we also introduce a metric termed relative improvement, defined by Formula 1. Relative improvement denoted as $\mathcal{I}_{Relative}$, quantifies the disparity between the average latency of the baseline $\overline{L}_{\text{baseline}}$ and the average latency of each platform, $\overline{L}_{\text{platform}}$. It is computed as the difference between the baseline average latency and the average latency of each platform, divided by the baseline average latency.

$$\mathcal{I}_{Relative} = \frac{\overline{L}_{\text{baseline}} - \overline{L}_{\text{platform}}}{\overline{L}_{\text{baseline}}} \tag{1}$$

Our comparative analysis of selected queries under the near-data processing, optimized layout system, and baseline approaches reveals varying performance outcomes. As illustrated

TABLE I: TPC-H query classfication

	I/O Intensive	CPU intensive
Query Number	Q10, Q19, Q20, Q14	Q6, Q22, Q20, Q14
Workload Size (MB)	581MB, 518MB, 518MB, 449MB	115MB, 141MB, 518MB, 449MB

in Fig. 2, the latency of the mixed-intensity queries (e.g., Q14 and Q20 in Table I) is lower with near-data processing compared to both the optimized layout system and the baseline. This suggests that near-data processing is particularly advantageous for CPU and I/O-intensive workloads. By offloading computation to storage nodes, near-data processing reduces unnecessary data movement and alleviates CPU bottlenecks on client nodes.

Furthermore, for I/O-intensive workloads (e.g., Q10 and Q19), near-data processing shows better relative improvement compared to the optimized layout system, shown in Fig. 2. Even though these are I/O-intensive, they still consume some CPU resources. Near data processing offloads computation to the storage node, benefiting the client node. In the case of CPU-intensive workloads (e.g., Q6 and Q22), the advantages of near-data processing should be even more pronounced. However, the optimized layout system achieves better improvements for specific queries like Q6 and Q22 than near-data processing. The reason is that the total workload for these queries is relatively small, and near-data processing incurs overheads from serialization/deserialization [3], which reduces its effectiveness.

To closely simulate the behavior of client nodes in real-world systems, which often manage multiple workloads concurrently, we utilize FIO [11], a versatile I/O testing tool, to generate additional I/O workloads tailored to our needs. Specifically, we configure FIO to intensify I/O operations while keeping available CPU capacity at predefined levels. Our targets are 75% and 50% available CPU capacity, achieved through FIO's random write workload. Preliminary tests indicate that executing a 1GB random write operation with FIO allows the client node to reach 75% available CPU capacity, while a 5GB operation results in approximately 50% available CPU capacity.

Fig. 3 depicts the average latency of two CPU-intensive queries (i.e., Q6 and Q22) at these CPU capacity levels for a client node engaged in near-data processing, optimized layout system, and baseline scenarios. These figures show that near-data processing performance improves with increased client node workload. For example, as shown in Fig. 3, in scenarios with 50% available CPU capacity, near-data processing exhibits a superior improvement of the optimized layout system. This advantage is because, at 50% available CPU capacity, the client node processes a heavier workload, whereas the optimized layout system still relies on the local CPU for computations. In contrast, near-data processing offloads tasks to the storage node, thereby alleviating the local node's CPU load and enhancing overall system performance. However, at 75% available CPU capacity, the optimized layout system shows better improvement than near-data processing for query Q22. Our analysis reveals that although Q22 involves joining two tables, each table's size is manageable. Thus, even at

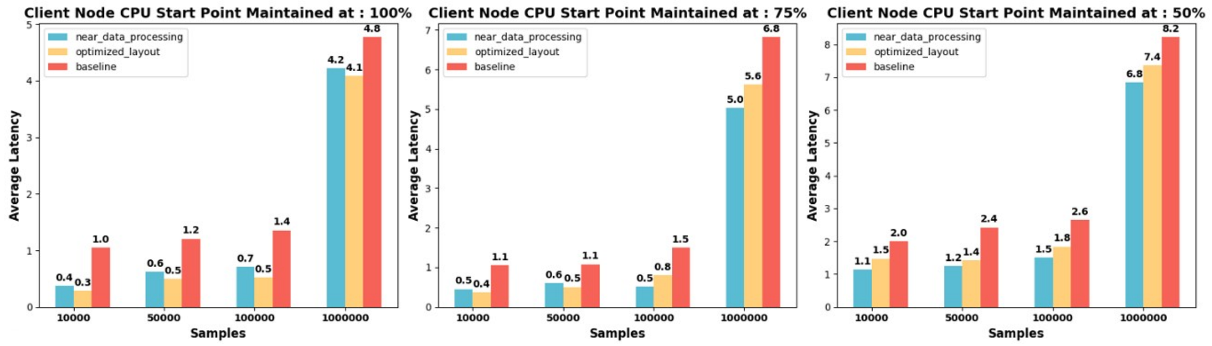

Fig. 4: Latency of K-Means across various sample sizes for near-data processing, optimized layout, and baseline scenarios, initiated at 100%, 75%, and 50% available CPU capacity at the client node. The number of clusters is set as 10.

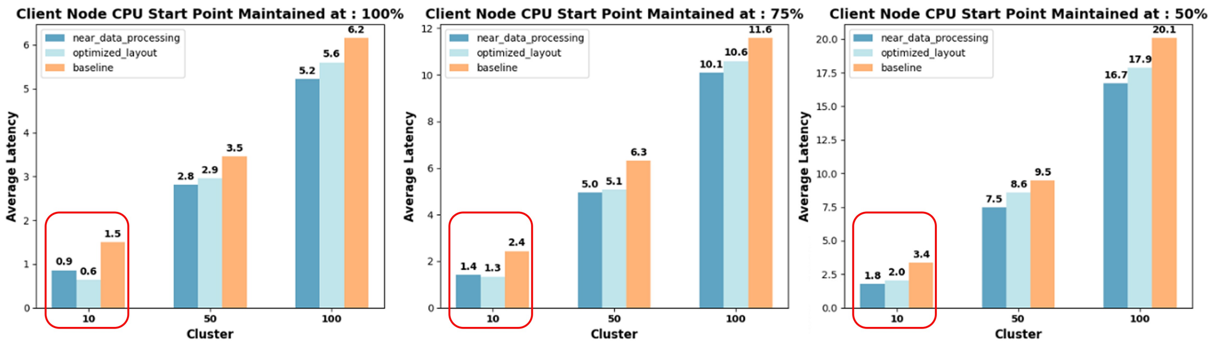

Fig. 5: Latency of K-Means across different cluster sizes for near-data processing, optimized layout, and baseline scenarios, start from 100%, 75%, and 50% available CPU capacity at the client node. The sample size is 100K.

75% available CPU capacity, the optimized layout system outperforms near-data processing.

2) K-means workload: We further conduct experiments to explore the sensitivity of input data size and the number of clusters for the K-means algorithm. We compare the performance of K-means clustering across different available CPU capacity start levels (e.g., 100%, 75%, and 50%) at the client node for near data processing, optimized layout system, and the baseline. In this set of experiments, we set the number of clusters as 10 and change the number of data samples ranging from 10K to 1000K. Fig. 4 illustrates the average latency of K-means under near data processing, optimized layout system and the baseline. We obverse that when the client node starts with 100% available CPU capacity, the optimized layout system outperforms near-data processing and the baseline. This is because the optimized layout system provides efficient memory allocation and management for hardware architectures, thereby minimizing memory fragmentation and overhead. However, as we employ FIO to increase workload thereby decreasing the CPU capacity of the client node and increasing workload size, the advantages of near-data processing become increasingly apparent.

In another series of experiments, we set the sample size to 100K and vary the cluster size from 10 to 100. Fig. 5 shows the average latency of K-means under near-data processing, optimized layout system, and the baseline. We observe that when the cluster size is set to 50 and 100, the performance of near-data processing outperforms the optimized layout system and the baseline. This phenomenon arises from the fact that an increase in cluster size results in a greater number of cluster centers that must be initialized, updated, and optimized throughout the iterative process of the algorithm. Consequently, the computational complexity of the K-means algorithm escalates with the number of clusters. Additionally, a larger cluster value entails the calculation of distances between data points and cluster centers for a larger number of clusters, further augmenting the computational load.

Referring to Formula 1, we determine the improvement for near-data processing and optimized layout system based on K-means, as shown in Table II. It is evident that when the cluster size equals 10, there is less computational demand for K-means, resulting in better improvement for the optimized layout system compared to near-data processing. However, as the cluster size increases, the computational intensity of the system escalates. At this juncture, near-data processing exhibits a superior relative improvement over the optimized layout system. Thus, near-data processing emerges as the preferred approach for large-scale data computation. By offloading computational operations to the storage node, near-data processing alleviates workload and CPU bottlenecks on the client node, and reduces data movement.

B. CPU Utilization

Fig. 6 depicts the CPU utilization over time at both the client and the storage nodes during the execution of TPC-H

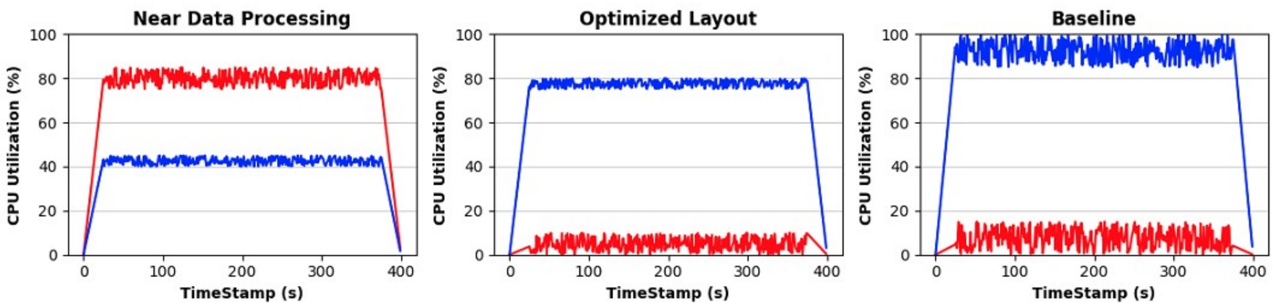

Fig. 6: CPU utilization under near-data processing, optimized layout system, and the baseline. The red line shows the CPU utilization of the storage node, and the blue line shows the CPU utilization of the client node.

TABLE II: The improvement of near data processing and optimized layout system compared with baseline for k means

Method			Near data processing improvement			Optimized layout improvement		
			100%	75%	50%	100%	75%	50%
k-means	Cluster = 10	Sample = 10k	60%	54.5%	45%	70%	64%	25%
		Sample = 50k	50%	45%	50%	58%	55%	42%
		Sample = 100k	50%	67%	42%	57%	47%	30%
		Sample = 1000k	12.5%	26%	17%	14.5%	17%	9%
	Sample=100k	Cluster = 10	40%	41%	47%	60%	46%	41%
		Cluster = 50	20%	21%	21%	21%	19%	9%
		Cluster = 100	16%	13%	17%	10%	9%	11%

queries on a cluster with 4 nodes. We utilize Dstat [12] to monitor and record the CPU utilization.

We observe that the TPC-H [5] queries nearly exhaust the client node's CPU resources. Even in the case of an optimized layout system platform, the TPC-H [5] queries strain up to 80% of the available resources. This scenario implies that the client node would be incapable of undertaking any additional processing work, resulting in a bottleneck in query performance due to the client node's CPU limitations.

Conversely, in the near-data processing platform, ample CPU resources remain available at the client node. This disparity arises because near-data processing offloads tasks to the storage node for processing, thereby minimizing data movement. Consequently, with an abundance of CPU resources on the client side, it becomes feasible to launch more asynchronous threads to enhance parallelism or to allocate the client node to other processing tasks.

V. Conclusion

In this paper, we present an extensive case study on the performance of near-data processing using Skyhook [3]. By integrating the TPC-H and K-means clustering algorithms, we explored CPU-intensive and I/O-intensive tasks and investigated how K-means clustering can be integrated into SQL environments to offload computational burdens to storage nodes through a near-data processing system. Furthermore, we introduced the use of the FIO tool to simulate real-world system workloads, enabling the measurement of key performance metrics such as average latency and relative improvement across various processing approaches: traditional local processing, the optimized layout system, and near-data processing. Our comprehensive performance evaluation revealed distinct advantages of near data processing and optimized layout over traditional processing methods, particularly in handling large-scale data sets by minimizing data movement. Our research advances the understanding of optimizing distributed data systems in the era of big data and suggests potential avenues for further enhancing data processing efficiency in future studies.

Acknowledgments

This work was partially supported by National Science Foundation Award CNS-2008072.

This research is partly supported by the U.S. Department of Energy (DOE) through the Office of Advanced Scientific Computing Research's "Advanced Memory to Support Artificial Intelligence for Science" and "Orchestration for Distributed & Data-Intensive Scientific Exploration".

References

[1] Zaharia, Matei, et al. "Spark: Cluster computing with working sets." 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10). 2010.
[2] White, Tom. Hadoop: The definitive guide." O'Reilly Media, Inc.",2012.
[3] https://arrow.apache.org/blog/2022/01/31/skyhook-bringing-computation-to-storage-with-apache-arrow/
[4] https://docs.ceph.com/en/reef/
[5] Dreseler, Markus, et al. "Quantifying TPC-H choke points and their optimizations." Proceedings of the VLDB Endowment 13.8 (2020): 1206-1220.
[6] Kodinariya, Trupti M., and Prashant R. Makwana. "Review on determining number of Cluster in K-Means Clustering." International Journal 1.6 (2013): 90-95.
[7] Vohra, Deepak, and Deepak Vohra. "Apache parquet." Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools (2016): 325-335.
[8] Raasveldt, Mark, and Hannes Mühleisen. "Duckdb: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data. 2019.
[9] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html
[10] Hou, S., & Kong, Z. (2024). FedLDCS: Adaptive Divergence-Based Client Selection for Federated Learning. In FedKDD: International Joint Workshop on Federated Learning for Data Mining and Graph Analytics.
[11] https://github.com/axboe/fio
[12] https://linux.die.net/man/1/dstat
[13] Raasveldt, Mark, and Hannes Mühleisen. "Duckdb: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data. 2019.
[14] Weil, Sage, et al. "Ceph: A scalable, high-performance distributed file system." Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI'06). 2006.