IDIOMS: Index-powered Distributed Object-centric Metadata Search for Scientific Data Management

Wei Zhang[1], Houjun Tang[1], Suren Byna[2,1]
[1]Lawrence Berkeley National Laboratory, [2]The Ohio State University
zhangwei217245@lbl.gov, htang4@lbl.gov, byna.1@osu.edu

***Abstract*—Affix-oriented metadata search is one of the essential fuzzy search capabilities that allow users to find data of interest in their voluminous data set with incomplete query conditions. With the recent transition towards object-centric data management systems in the science community, there is a paramount need for the support of such features in distributed settings. However, existing metadata search solutions either do not support efficient affix-oriented metadata search or do not suit well in a distributed setting of object-centric data management systems. To bridge this gap, we introduce IDIOMS, a metadata search solution underpinned by a distributed metadata index, meticulously designed to enable high-performance affix-oriented metadata search for parallel object-centric storage. One of the standout features of IDIOMS is its efficiency in supporting four distinct types of highly demanded metadata queries. Furthermore, IDIOMS is flexibly catering to both independent and collective metadata search operations. Our experimental comparisons with SoMeta, a state-of-the-art metadata query method, demonstrate more than 400× performance boost for independent queries and up to 300× performance improvements for collective queries, while keeping a small index management overhead.**

***Index Terms*—Metadata Search, Metadata Management, Parallel Data Processing, Distributed Data Management, Scientific Data Management**

I. Introduction

To swiftly find the desired data in large scientific data sets [1]–[8], there is a critical need for an efficient metadata search mechanism to sift through large amounts of metadata. The sheer volume of data and vast amount of metadata introduce significant challenges, including uncertainties in search outcomes and the unpredictable nature of metadata attributes, making traditional exact-match searches cumbersome and time-consuming. To mitigate these issues, the affix-oriented metadata search feature proves beneficial for scientists who need to sift through data objects with incomplete query conditions. For instance, in a lattice light-sheet microscopy (LLSM) application [2], [9], if users would like to query all the data sets with stage coordinates of any dimension (e.g., StageX, StageY, StageZ, etc.) to be multiple of 100 (i.e., ending with 00), putting multiple conditions about all dimensions and iterating all possible coordinates that ends with "00" is an impossible task for scientists to perform. Rather, to achieve the same goal, they can issue a simpler query with a query condition partially matching the affixes of metadata attributes, such as "`Stage*=*00`". Such a query is called affix-oriented metadata search.

Recently, influenced by the need for high-performance I/O, scalable data access, and rich metadata management, the science community is witnessing a data management transition towards the object-centric data management systems, a.k.a., ODM systems, such as HEPnOS [10], DAOS [11], and PDC [12]. However, there are no efficient and scalable affix-oriented metadata search features in the current ODM systems. On the one hand, existing ODM systems mostly put their emphasis on the I/O performance, and some of them are still in their early ages. This results in insufficient attention on metadata search and hence accompanying underdevelopment of such features. For instance, in Proactive Data Containers (PDC) [12], the latest significant effort in ODM systems, metadata search requires scanning each metadata attribute on every server process and conducting basic pattern matching operations. This comprehensive scanning, both globally and locally against all attributes of all objects, makes searching for specific metadata attributes time-consuming, which significantly affects the efficiency of affix-oriented metadata queries.

On the other hand, existing metadata search solutions were proposed in the context of general data management, without complete compatibility to ODM systems. Several of these efforts use external or embedded database management systems (DBMS) as the backends of their metadata storage. Examples of these database systems include relational databases such as PostgreSQL [13] used in BIMM [14], SQLite [15] embedded in EMPRESS [16] and TagIt [17], NoSQL databases like MongoDB [18] used in JAMO [19] and SPOT Suite [20]. The major drawbacks of these database-powered solutions include 1) the need for a dedicated environment to host the database server; 2) a mismatch between the database schema and object-centric metadata schema that will enforce the metadata transformation from an ODM system to the targeting database (e.g., transform to tabular format in relational databases or JSON [21] format in MongoDB); 3) inefficient full-text search for affix-oriented metadata query. While other solutions [22], [23] were proposed to address the challenge in the context of self-describing formats, they do not consider the distributed nature of ODM systems where the clients access the data on the servers via network communications.

In addition, there is no solution to support efficient distributed infix metadata search. Although distributed data structures and corresponding algorithms like DHT [24] and DART [25] were proposed to address the distributed search

problem and can be used to facilitate distributed search, infix search on a DHT/DART-featured distributed system will still incur a complete scan against all distributed nodes, which can lead to significant performance deterioration.

Given the rising need for affix-oriented metadata search in ODM systems, we propose IDIOMS - an **I**NDEX-POWERED **D**ISTRIBUTED **O**BJECT-CENTRIC **M**ETADATA **S**EARCH solution. Our solution is designed to support affix-oriented metadata search in large-scale parallel applications, where distributed ODM systems are employed. In our design, we follow the classic client-server architecture that suits the ODM systems well, and we target both collective and independent metadata search operations for parallel applications. Our in-memory index on the server side is specifically crafted for efficiently serving each metadata query with affixes on both keys and values. Our client runtime is designed to ensure deterministic query routing against all servers. In addition, our design of suffix-tree mode in IDIOMS helps to unleash the power of infix metadata search, making it as efficient and smooth as exact metadata search.

We implemented IDIOMS and integrated it into PDC [12]. We also conducted an extensive evaluation on the Perlmutter supercomputer hosted at the National Energy Research Scientific Computing Center (NERSC). Our experiments with over 10 million metadata attributes from one million objects show that IDIOMS achieved at least 307× performance boost for the independent metadata queries and up to 303× performance improvement for the collective queries, as compared to SoMeta [26] and SQLite-powered metadata search method. Also, our experiment results show that IDIOMS only adds up to 52.57% memory overhead.

The contributions of this research are as follows:

- We identify the gaps and challenges of metadata search in ODM systems - the lack of efficient affix-oriented metadata search in existing ODM systems as well as the ignorance of the distributed nature of ODM systems in existing metadata search solutions.
- We propose IDIOMS - an Index-powered Distributed Object-centric Metadata Search solution which is designed to achieve efficient affix-oriented metadata search on top of ODM systems.
- Our metadata index is designed to handle affix-oriented queries against both metadata attribute keys and metadata attribute values.
- The client runtime of IDIOMS supports efficient and deterministic query routing in both independent query mode and collective query mode, for all four essential affix-oriented metadata search types.
- The index persistence and recovery mechanism is designed to ensure the adaptability towards server count changes among different application runs.
- We integrated IDIOMS into the PDC ODM framework. With the help of IDIOMS, the independent metadata search gets a 407× performance boost and the collective metadata search in PDC is now up to 300× faster than before.

The rest of the paper is organized as follows. In Section II, we review metadata management in object-centric data management. We describe the design of IDIOMS in Section III. We show our experimental results and provide corresponding discussions in Section IV. After briefly reviewing related works in section V, we finally conclude in Section VI.

II. Motivation and Challenges

Metadata search is a well-developed feature in most traditional file systems [27]. Typically, users can issue prefix, suffix, infix, and exact queries against file names. Such affix-oriented metadata queries are essential fuzzy search capabilities available to the users who may not exactly remember what the file names are. Similarly, such capabilities are also valuable tools to help users sift through the vast number of rich metadata attributes in an object-centric data management system. However, in most modern ODM systems, metadata search, especially affix-oriented metadata search, is still at its early stage and there is no particular metadata search solution specifically developed for ODM systems. For example, some recent metadata search solutions, like AMI [28], JAMO [19], SPOT Suite [20], and BIMM [14], use centralized database as the metadata store. The centralized database limits the scalability of these metadata management solutions and hence is not well-suited for ODM systems that aim to serve large-scale parallel applications. Some other solutions, like TagIt [17] and Empress [16], utilize embedded databases, in particular, SQLite, as the metadata store. While metadata search can be served by SQL queries on each distributed server process, each metadata query would still incur a complete scan on all distributed server processes, leading to huge communication overhead. Distributed adaptive radix tree (DART) [25] addresses the distributed affix-oriented keyword search problem, but it can only work on every single keyword instead of key-value formed metadata attributes and it only supports deterministic query routing for prefix, suffix, and exact queries, not for infix queries. Therefore, there is a pressing need for an efficient affix-oriented metadata search solution particularly crafted for ODM systems.

However, achieving efficient affix-oriented metadata search on top of ODM systems is a challenging task. The major challenges are as follows:

❶ ***Supporting Key-Value Search***: While metadata attributes in most ODM systems are usually in the form of key-value pairs, and the keys and values are not completely distinct, we need to support affix-oriented queries on both the key part and the value part of the metadata attributes. Although distributed query routing can be performed on metadata attribute keys, an appropriate local indexing structure must be designed to support efficient affix-oriented metadata queries on both metadata attribute keys and values on the server side.

❷ ***Minimizing Communication Cost***: Due to the distributed nature of data objects in ODM systems, without a proper global indexing technique, an affix-oriented metadata search operation would necessitate a time-consuming full scan over all object partitions in the ODM system due to the distributed

object and metadata model. Thus, the design of the distributed metadata indexing technique must put minimizing communication reduction in the first place.

❸ ***Supporting Two Query Modes***: Scientific applications are usually highly paralleled and mostly distributed. Depending on whether the application processes are working independently or following the same workflow, there are two query modes we need to support: 1) *Independent Query*: Some applications may use independent processes to handle different workflows simultaneously and hence may issue metadata queries independently to the server processes at any given time. Without a proper index, each affix-oriented metadata query must be sent to all servers to get the complete query result. This can cause significant network congestion when multiple clients issue queries simultaneously. 2) *Collective Query*: Another category of applications, may execute the same workflow among all client processes. A selected set of client processes can collectively send partial metadata search requests to the server processes to get partial results. Afterwards, clients must perform a result exchange to get the complete query result. Note that, in this case, all the clients, as a whole, can only send one query at a time, and the query response time largely relies on the longest turnaround time of all the partial requests. Therefore, it is necessary to minimize the server response time.

❹ ***Adaptability to Server Count Changes***: We consider that the applications may need to scale in and out, and hence the number of servers of an ODM system may need to be changed accordingly. Therefore, the distributed metadata indexing mechanism must be designed to cope with the potential server count change.

III. Methodology

A. System Overview

The goal of IDIOMS is to provide efficient affix-oriented metadata search capability to distributed ODM systems. Therefore, the design of IDIOMS naturally embraces a client-server based architecture where client processes (or "clients") and server processes (or "servers") communicate via remote procedure calls (RPC). In the design of IDIOMS, we consider achieving high performance parallel metadata query capability as our first priority. Therefore, there are two essential pitfalls we would like to avoid: 1) query broadcasting on all servers that may lead to excessive communication overhead, 2) slow response introduced by local complexities on each server that may lead to query congestion. Therefore, instead of providing affix-oriented metadata search directly on top of the distributed objects and their attached metadata attributes, we consider building an inverted index for metadata attributes and we partition such index onto all servers using our implementation of DART [25] - an index partitioning and query routing algorithm for affix-oriented metadata search.

As shown in Figure 1, IDIOMS contains three major parts:

① the **IDIOMS APIs** by which users can create, delete inverted metadata index records, and most importantly, perform affix-oriented metadata search;

② the **client-side runtime** which comprises 1) a load distributor that aims to distribute query workloads evenly among all clients, 2) a request router that routes each metadata search query to specific one or two servers, and 3) a response processor which performs result deduplication and optional result broadcasting. Note that these components exist across all clients of ODM systems;

③ the **server-side runtime** which consists of an index layer where the trie-based in-memory inverted index is maintained and a persistence layer where the persistence and recovery of the in-memory index is handled.

B. Metadata Index Model in IDIOMS

To address affix-oriented metadata search efficiently for ODM systems, in IDIOMS, we consider building an inverted metadata index. Conceptually, if we map all the inverted index records into a tabular view, we can consider the inverted metadata index to be a list of 3-element tuples. In each tuple, the first element corresponds to the metadata attribute key, the second element corresponds to the metadata attribute value, and the third corresponds to a list of object IDs. In IDIOMS, these index tuples are distributed across all servers based on the first element of the tuple, and, inspired by MIQS [22], are stored in a double-layered trie-based in-memory index locally on their destination servers, as shown in Figure 2. It is worth noting that, in addition to the "indexing by duplicating inverted strings" methods used in MIQS, we also designed a suffix-tree mode where these metadata attributes are indexed following a "indexing by iterating suffixes" approach, where we iterate all suffixes of keys for index record distribution while inserting all suffixes of keys and all suffixes of values into the first and second layer of the in-memory index tree, respectively. This eliminates the need to create an additional double-layered index structure specifically for suffixes, and also provides the capability for efficient infix search. This is because each infix search is now converted into a prefix search against a certain suffix of a metadata attribute key or value, which helps reduce the complexity from $\mathcal{O}(n*d)$ down to $\mathcal{O}(d)$ (where n denotes the total number of attribute keys or values, d denotes the height of the trie). Although iterating all suffixes seems to increase the number of index records on the trie and hence may cause significant memory overhead, in practice, the suffixes of a meaningful string like a metadata attribute key or a metadata attribute value may contain duplicated sequences and many of the trie nodes can be re-used instead of being created.

C. IDIOMS API

The API design of IDIOMS puts simplicity and effectiveness at its top priority. The `create_md_index` API can be used to create an index record when a new metadata attribute is attached to an object, and the `delete_md_index` API can be used when an existing metadata attribute is detached from an object. Also, when there is an update on the metadata attribute for a certain object, the `delete_md_index` API can be invoked first to delete the stale index record while the `create_md_index` API can be used to index the newly updated metadata attribute of the specified object. Note that there is no API for updating the metadata index since the

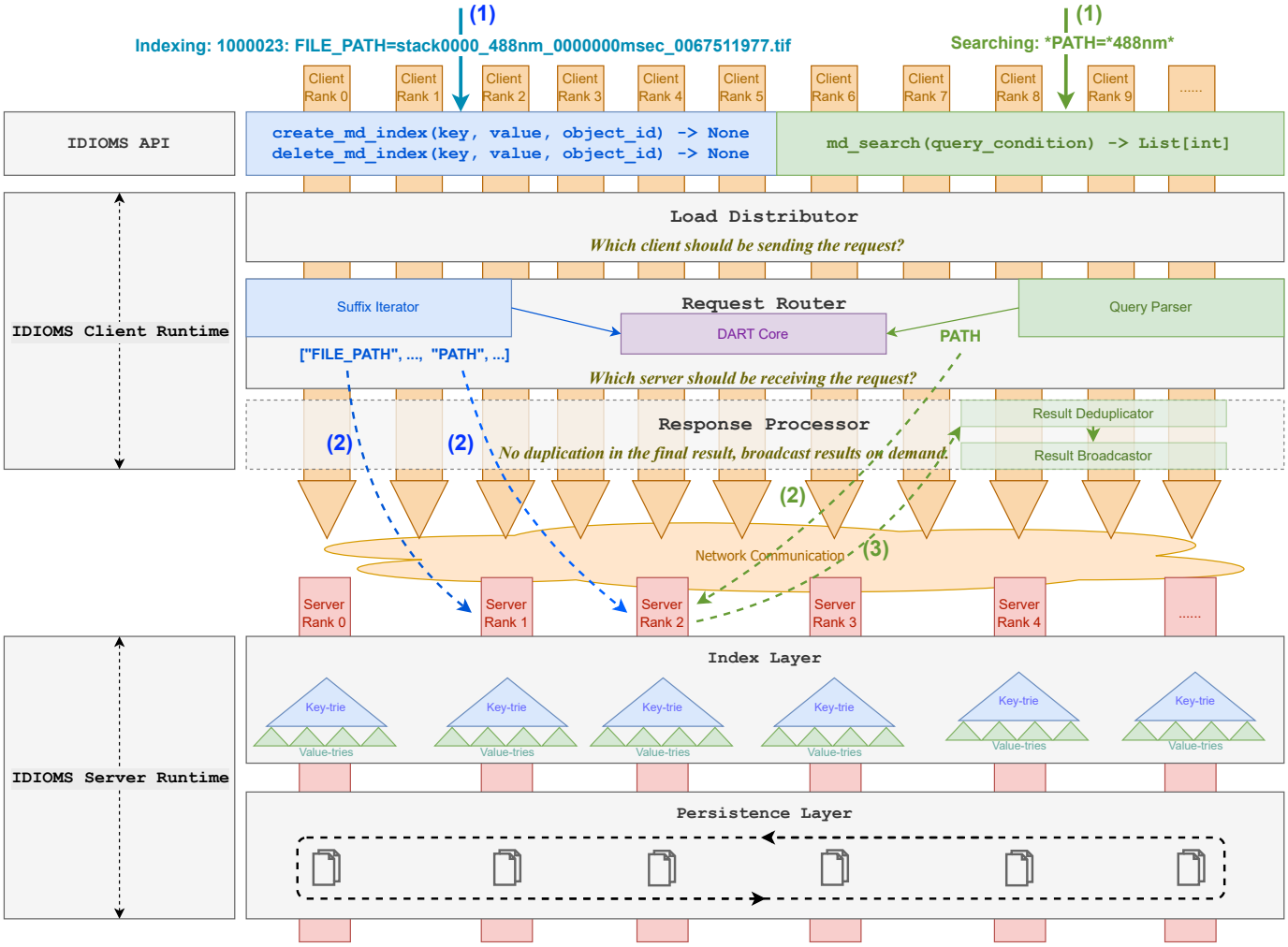

Figure 1: System Overview of IDIOMS. The metadata index operations are shown in blue color, and the metadata search operation is shown in green color. Note that the indexing-related components and querying-related components uniformly exist in all client processes, we put them side by side to clearly demonstrate these two different flows.

index records are scattered on different branches of the trie and the stale index record must be deleted from the trie first for any metadata attribute update so that the index can remain consistent with the actual change of the metadata attribute.

In addition to the metadata indexing APIs, the `md_search` API is the only API that IDIOMS provides to the end user. This API takes a query condition string which is in the form of "`:key=:value`", where "=" is the delimiter separating the key part "`:key`" and the value part "`:value`" of the query. Note that either the key part or the value part in the query condition string can be written in the form of a complete string or the form of an affix. This provides users with the greatest flexibility in writing their affix-oriented metadata search query conditions without the hassles of taking care of separate function parameters. Additionally, such an extensible design also opens up the possibility for unifying the user experience on new metadata search capabilities in the future.

In Figure 3, we take some sample metadata attributes and metadata queries from the LLSM application [2] to demonstrate the following types of queries supported by the IDIOMS query APIs:

- ***Prefix Search***: Locating data objects whose attribute keys or attribute values match with the given prefixes. If prefixes are given for both the key part and the value part of a query condition, we consider this type of query as a ***Complete Prefix Search***. For instance, in the figure, a complete prefix search with query condition `coordY*=12.*` is looking for data objects with Y coordinates within the range of `[12.0, 13.0)`, regardless of the version of the coordinates.

- ***Suffix Search***: Locating data objects where attribute keys or attribute values match with a given suffix. Similarly, for a query with suffixes on the key part and the value part of a query condition, we name it as a ***Complete Suffix Search***. For instance, a complete suffix search with query condition `*date=*-26` looks for data objects that were ever created, updated, or reviewed on the 26th day of every month.

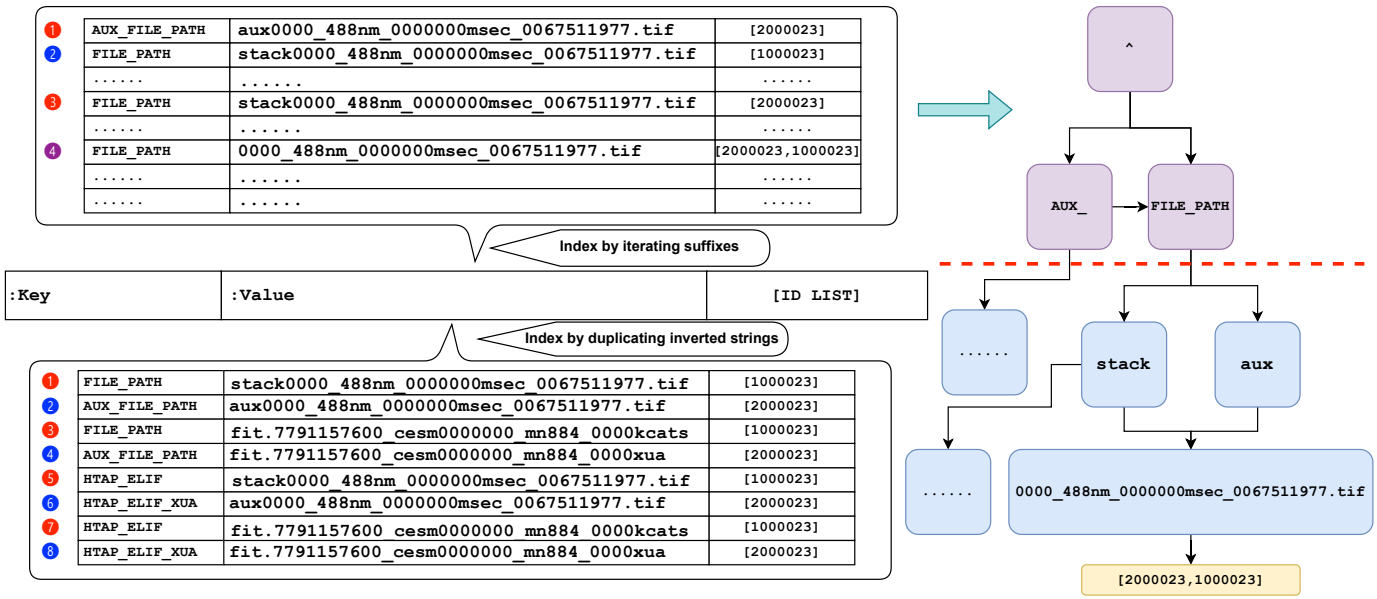

Figure 2: IDIOMS index model. The diagram shows an example of index records on two attributes marked with a red ❶ and a blue ❷ respectively. Note that indexing by duplicating inverted strings of both keys and values can lead to a quadruple of the index records in which rare trie nodes can be possibly shared. For instance, all the index records ❶-❽ in the "index by duplicating inverted string" table cannot share any trie node. In contrast, indexing by iterating suffixes may lead to a much larger number of index records, but in many cases, the trie nodes can be shared among these records due to the same ordering of the characters in the records and also the repetitive occurrence of characters in the metadata attribute key or metadata attribute value. For example, index record ❸, though coming from suffix interaction on the key of index record ❶, can share trie node for `FILE_PATH`. Also, index record ❹ may share some trie nodes on the value trie with the derivatives of index records ❶, ❷, for the first few zeros in the value part.

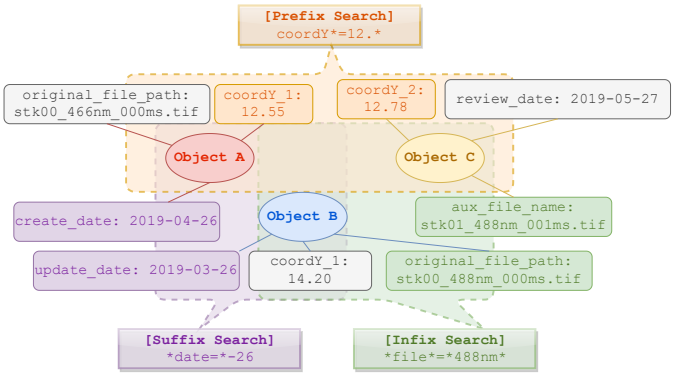

Figure 3: An overview of affix-oriented metadata search in object-centric metadata model, names of objects, and attributes are tailored for the ease of demonstration. Metadata attributes that do not match any given query condition are gray-colored.

- ***Infix Search***: Locating data objects with attribute keys or attribute values matching with given infix. A query with infixes on both the key part and the value part of the query condition is called a ***Complete Infix Search***. For instance, a complete infix search with query condition "`*file*=*488nm*`" looks for data objects that relate to any original file or auxiliary file with a file name containing the granularity target "488nm".

Note that, in a query condition, the key part and value part can be of any form - either an affix or a complete string. If both the key part and the value part of a query condition are in the form of an affix, we call those types of query to be ***Complete Affix Queries***. In addition to the above, we also support ***Exact Search***, where the keys or values in the query condition are given in their complete form.

We consider the single condition queries as the cornerstone of the metadata search capability, which can be combined to address more complex queries. Therefore, we do not aim for multi-condition queries for now.

D. Client-side Runtime

The client (or application) side metadata index handling APIs provide the metadata index creation and deletion operations. We consider supporting these two types of operations in two scenarios: 1) non-MPI based applications where all operations are independent between each client; 2) MPI-based applications where collective operations are needed. Therefore, for each type of operation, we provide support for independent mode and collective mode, as shown in Figure 4.

In independent mode, each client process works independently and a metadata index operation can be initialized from any client at any time. In this case, the load distributor is bypassed for all metadata indexing and searching operations. In contrast, for the collective mode, the load distributor will be utilized to distribute operation workloads among all clients first, to ensure that no operation congestion is formulated at a certain client via multiple times of collective operations. Since the independent mode and collective mode share the

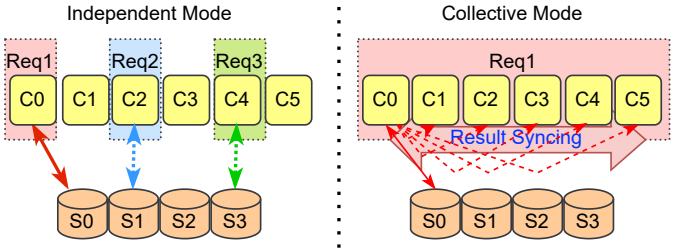

Figure 4: Two modes based on communication model: independent mode and collective mode. C0-C5 are client processes and S0-S3 are server processes. Note that clients may send different independent requests at the same time, like Req1, Req2, and Req3, while all clients can only send one collective query at a time.

most significant part of the metadata workflows, we now describe metadata index handling and metadata search operations separately in the collective mode. Note the only difference in independent mode is that all clients are acting independently and the load distributor is bypassed for any operation.

1) Metadata Index Handling on Client Side: In IDIOMS, the metadata index is partitioned using DART [25] algorithm and all metadata index operations will eventually be routed to 2 servers selected by DART. Therefore, in collective mode, there is no need to have all clients send the same requests and only one client is selected for performing the actual metadata index operation. To select such a client, we utilize the load distributor that resides in each client. In the load distributor of each client, we maintain an index operation counter for counting the number of collective calls on `create_md_index`, `delete_md_index`, and `md_search`. Each time a collective metadata index operation call is triggered, we first increase the counter by 1 at all clients, and then we utilize the load distributor to calculate the ID of the request sending client by performing a modulus operation with two parameters - the value of the counter and the total number of processes.

Once the client ID is selected, the client process with the selected ID will proceed to the suffix iterator where all suffixes of the metadata attribute key are iterated. Then, the client generates a list of requests with different suffixes of the metadata attribute key, the metadata attribute value, and the object ID given in the collective function call. With the help of the DART algorithm, each request in the request list will be attached with a virtual node ID tag. Additionally, to implement the DART replication mechanism on the metadata index, we also create additional requests for the replicas. With the request list, the request router will send these requests to different servers with the guidance of the DART algorithm.

2) Metadata Search on Client Side: Similar to the metadata indexing operations, a collective metadata search operation also starts with the load distributor, while the independent metadata search operations bypass it. At the request sending client, the metadata search query is firstly parsed so that the affix type and the actual affix in the key part of the query condition are extracted. For a given query condition "`:key=:value`", we first split the query string by delimiter "=", then we take the key part of the query condition and determine the query type. If there is only one "$\star$" in the key part and it appears the beginning of the key, we consider it as a suffix search against the key; if the "$\star$" appears only at the end of the key part, it is a prefix search on the key; if "$\star$" shows at both sides of the key part, it is an infix search against the key; no "$\star$" will be considered as an exact search on the key. Based on the DART algorithm, the request router will generate a destination list containing the IDs of the two servers where the corresponding metadata index record may exist. Note that if the key part is solely a "$\star$", the destination list will include all server IDs. With the destination list, the request sending client will send the query condition string to the destination servers. For example, query condition "`*PATH=*488nm*`" will be parsed as a suffix query on the key with the actual suffix token "`PATH`", and a destination list will be generated with the IDs of destination servers. As the example shown in Figure 1, the query condition "`*PATH=*488nm*`" is sent to server rank 2 where the index records with the suffix "`PATH`" of key "`FILE_PATH`" can be certainly found.

E. Server-side Runtime

The server-side runtime of IDIOMS comprises two layers: 1) the index layer where each server maintains an in-memory inverted index for efficient metadata search on the server side; 2) the persistence layer where each server process can dump its own in-memory index to an on-disk index file. Also, the index recovery based on these index files is also performed at this layer.

1) Index Layer: At this layer, instead of using any database, each server process maintains a partial inverted index in its runtime memory. The in-memory inverted index is a combination of two-level tries. The first level is a trie that corresponds to the metadata attribute keys, while the second level contains multiple tries connecting to the leaf nodes of the first-level trie, which correspond to the metadata attribute values of every indexed key. When indexing metadata attributes or deleting metadata index records with the suffix-tree mode, since the client side already iterates all the suffixes of the key in the request, the server side will directly insert the prepared suffixes of the key into the first-level trie. Then, for the value part, the server will also iterate all suffixes and insert them into the second-level tries that correspond to each suffix in the first-level trie. The corresponding object ID is then inserted at the leaf nodes of the second-level tries.

When executing a metadata query, once the server receives the request sent by the client, it will go through the in-memory index. Since all the suffixes of every indexed key and value are inserted into the trie-based in-memory index, any exact match or suffix match will definitely hit a leaf node on the first-level trie, and any prefix or infix matches will be accomplished by a scan over the trie branches radiating from the node that corresponds to the last character in the prefix or infix. Depending on where the match happens (either on the attribute key or on its value), the leaf node access or branch scan occurs either at the first-level trie or at a second-level trie.

2) Persistence Layer: The persistence layer at the server-side runtime of IDIOMS is designed to checkpoint the in-memory index onto disk. When checkpointing the in-memory index, one thing we would like to make sure is that, even if the ODM systems are redeployed with a different number of servers, the in-memory index can still be restored with uniform distribution across all servers. To this end, we consider reserving the DART virtual node ID along with each index record. Therefore, when the client-side runtime of IDIOMS figures out a server ID for each index request, we include the corresponding DART leaf virtual node ID in the index request. When creating the in-memory index on each server, we include this DART leaf virtual node ID in the corresponding leaf node of the first-level trie. With this mechanism, we can reserve the mapping between the key of each index record and its corresponding virtual node ID. This makes it possible to restore in-memory index records with a consistent mapping between the index records and the DART virtual nodes, even if the ODM system is redeployed with a different number of servers. Since the index records are distributed evenly across DART virtual nodes, a uniform distribution of the index records is guaranteed on all servers.

To ensure an efficient index recovery process, we designed an index persistence mechanism that makes index recovery independent and communication-free among all servers. Also, to maintain the index file in a compact way, we utilize the one-to-many mapping between index key and index values, and also the one-to-many mapping between each key-value pair and object IDs. We first group all index records by the DART virtual node IDs they are associated with. Then, in each group, we insert the key of each index record as the key of the hash table, and we insert a list of compiled structures as the value of the hash table. Each compiled structure contains a field storing the DART leaf virtual node ID and a field storing a list of 2-element tuples, where each tuple stores the value part of an index record and also the corresponding object IDs. Then we serialize this hash table into a file named with the virtual node ID of this group, for instance, "`idioms_{vnode_id}.idx`" where "`vnode_id`" is the virtual node ID of the group. When restarting all servers, we initialize the DART core at the persistence layer of server runtime. Each server will be able to get a list of all DART virtual node IDs it is mapped to, and will load the index files whose file names contain the corresponding virtual node IDs.

IV. Evaluation

A. Experimental Setup

1) Platform: We perform our evaluation on Perlmutter [29], the current flagship supercomputer at NERSC [30]. Perlmutter's CPU partition contains 3,072 CPU-only nodes and each of them is equipped with 2 AMD EPYC 7763 CPUs and 512GB DDR4 DRAM. The storage system features substantial local SSD scratch space and a sophisticated all-flash Perlmutter Scratch File System with 35 PB of disk. Software-wise, we build IDIOMS on top of Proactive Data Containers (PDC) [31] - an ODM system with metadata management over a distributed object-centric data model.

2) Evaluation Procedure: For our evaluation, we compare 4 different implementations including 1) "SoMeta" - the original PDC metadata search solution without an inverted metadata index; 2) SQLite - the implementation that employs SQLite as the metadata storage. The object-to-metadata mapping is stored in a 3-column table with the primary key (the first column) storing the object ID, the second column storing the key and the third column storing the value (Both key column and value column are indexed using SQLite indexing feature); 3) IDIOMS (no suffix tree) - the implementation where additional compound tries are used for indexing the inverse of key strings and value strings; 4) IDIOMS (suffix tree) - the implementation where all the suffixes of keys and values are indexed using the same set of compound tries. Throughout our IDIOMS evaluation, we take the default replication factor setting of DART, which is 10% of the total server count $\mathcal{N}$. Thus, each index record will be replicated for $floor(\mathcal{N}/10)$ times. Note that we compare with SQLite because of its adoption in some recent metadata search advances like TagIt [17] and EMPRESS [16], etc. We would like to show how IDIOMS can improve metadata search efficiency in an ODM system without proper metadata index, and how much IDIOMS outperforms database-powered solutions in terms of affix-oriented metadata search efficiency.

For clarity and predictability of the evaluation result, we created a synthetic data set for our evaluation based on the LLSM application [2], [3] metadata list. We extend the number of data set records to 1 million, and by following the format of the original metadata attributes, we extend the number of metadata attributes to 100 per data set record. Thus, we can create 1 million objects with a total of 100 million metadata attributes into our testbed, which can sufficiently pressurize our system.

We perform our test in four iterations with each at a different scale with a different server count setting (i.e., 16, 32, 64, and 128). For each iteration, after a clean up procedure, we create 1,000,000 objects in PDC and we attach 10 million metadata tags on a selection of 100 thousand created objects. Note that we evenly create 100 metadata tags on each of the selected objects. We then perform metadata search queries in two modes - the independent search mode and the collective search mode, as described previously in section III. When performing queries in each mode, we send four complete types of queries mentioned in Section III-C and Figure 3. For each query type, we send 100 queries with query conditions made from the 100 metadata tags we put on each of the 100 thousand objects. For example, we take the first 3 and last 3 characters of each key or value when making prefix and suffix respectively, and we truncate the leading and trailing characters when making infix. Note that, for collective query mode, both SoMeta and our SQLite variant are configured to use the same number of client processes as server processes. This setup allows for parallel execution of queries, where each client process independently retrieves a part of the overall result. Subsequently, both of them

employ `MPI_AllGather` to synchronize the partial results across all client processes. In contrast, IDIOMS only selects one client process to perform the actual query execution and then broadcasts the results to all the clients. At the test end of each scale, we checkpoint the IDIOMS index. Accordingly, at the beginning of each iteration, after the index is recovered, we first delete all the metadata tags during the clean up procedure and then create the metadata attributes and the corresponding index. To make sure we have metadata tags to be deleted at the first iteration, we run an initializing program before the first iteration to create the desired amount of objects and metadata attributes.

B. Performance Impact on Tag Creation and Tag Deletion

In figure 5, we show the total time for creating these tags and also the total time for deleting them.

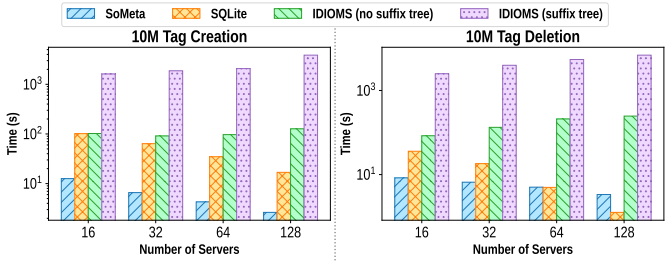

Figure 5: Tag Creation and Deletion Time of 10 Million Tags

As can be seen from the figure, in SoMeta where no indexing mechanism is applied, the total tag creation time for 10 million tags decreases from 12.5 seconds to 2.6 seconds as we increase the number of servers. With SQLite, we also see a decreasing trend for the total tag creation time, from 101 seconds at 16 servers to 16 seconds at 128 servers. The total tag creation time in IDIOMS does see a rising trend at a higher level for both cases (with/without suffix tree mode), with non-suffix-tree mode ranging from 102 to 126 seconds and the suffix tree mode ranging from 1613 to 3868 seconds. Such an escalation in indexing time is due to the double index created for prefix query and suffix query in non-suffix-tree mode and the iteration of all suffixes of each key and each value in the suffix-tree mode. Also, the ascending trend is because there are more replicas to be created by DART when the number of servers grows with a fixed replication factor - $floor(\mathcal{N}/10)$.

It is worth noting that the total tag creation time was measured in a single-threaded setting with all blocking calls subject to the current PDC implementation. In practice, there can be multiple optimization methods such as asynchronous client-side API calls and server-side multi-threading to further accelerate the client-side indexing performance. Additionally, the total tag creation time is measured during the initial tag creation stage. Considering the 100 million tags we created during our experiment, with IDIOMS, the average time for the initial creation operation of each tag is still between 0.01 to 0.38 milliseconds, which is within an acceptable range.

The tag deletion time sees a similar result as compared to the tag creation time. We do not go through the tag deletion performance in detail, since tag deletion and tag creation mostly follow a very similar procedure, and the only difference is that, in tag deletion, the index record is deleted and the corresponding memory is deallocated.

C. Index Persistence and Recovery

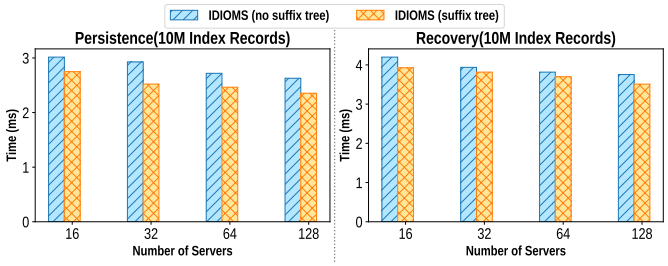

Figure 6: Index Persistence and Recovery Performance

Although IDIOMS may take a relatively long time to build the index for unindexed metadata attributes, the index persistence mechanism can help reconstruct the in-memory index much more efficiently. As shown in Figure 6, we can see that, in the worst case, it takes at most 3ms for IDIOMS to dump the in-memory index of 10M tags from all servers to disk (16 servers, no suffix tree mode). Also, in the worst case, it takes at most 4ms for all the servers to load and recover the in-memory index from disk. Given this result and the fact that we are able to perform our test at each scale after index recovery, we can see that IDIOMS is capable of recovering the index rapidly when the ODM servers need to be rebooted, even with a different server count setting. This verifies that IDIOMS is a practical and applicable solution in real applications.

D. Independent Query

The independent query mode in IDIOMS allows each client to perform metadata search independently. This query mode is ideal for applications where clients are still distributed but working without MPI support. Figure 7 shows the average latency per query. As can be seen in the figure, without any indexing mechanism, SoMeta takes more than 10 seconds to finish a single query, due to its full-scan over all distributed hash tables across all servers. With the help of SQLite, the query latency slightly drops down, but still remains higher than 10 seconds. We can see the benefit of local database adoption within each server process does not outweigh the drawback of global full-scan across all servers. In particular, as compared to exact queries, affix-oriented queries can take more time to finish on SQLite even with the database index on the key column and value column of the table. This shows that the index in the database does not help with the efficiency of affix-oriented metadata search.

In contrast, the performance of all queries in IDIOMS is immediately better. Without the suffix tree support, IDIOMS is able to achieve 4 - 14 ms for each exact, prefix, and suffix query and 47 - 78 ms for every infix query. With the help of suffix tree mode, IDIOMS is able to achieve an even lower

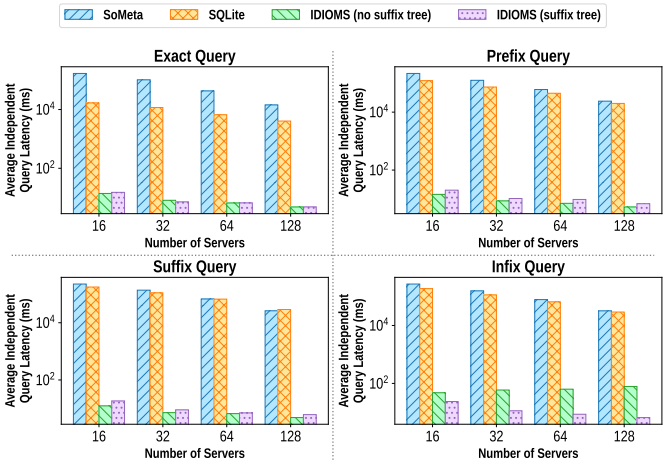

Figure 7: Independent Query Time Comparison

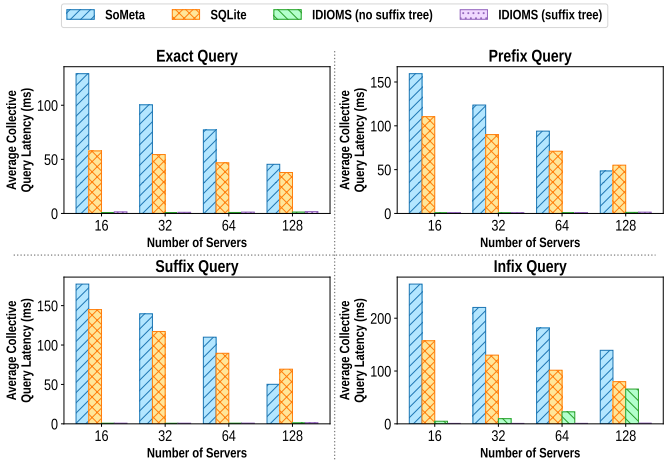

Figure 8: Collective Query Time Comparison

average latency for infix queries (ranging from 6 to 23 ms) with roughly the same performance as non-suffix-tree mode for exact, prefix, and suffix queries. This is because, firstly, the inverted index items are distributed across different servers by the prefix of the metadata key so that each exact query, prefix, and suffix query can be routed to a specific server rather than all servers, and secondly, every infix query is turned into a prefix query on a compound trie with the support of suffix tree mode. Notice that even without the help of suffix-tree mode, the performance of the infix query still beats SoMeta and the SQLite variant, due to the trie-based local in-memory index that helps to avoid complete examination on every metadata attribute of every object. Overall, IDIOMS achieves a 407-19065× performance boost as compared to SoMeta, and outperforms the SQLite variant by 370-15374×.

E. Collective Query

The collective query mode in IDIOMS is designed for MPI-powered applications. It allows a group of client processes to collectively issue the metadata search query and each one of them is supposed to get a complete search result. Figure 8 reports the latency of collective metadata search queries. As compared to independent queries where the full scan over all servers is performed in sequential order, the highest latency of collective queries in SoMeta and the SQLite variant declined to about 100 ms at its maximum scale, due to the parallel query server scanning in these two implementations. We can see that SoMeta is the slowest and the SQLite variant performs slightly better for the affix-oriented metadata queries.

When it comes to IDIOMS, we can see that, the exact, prefix/suffix queries are all finished within 0.8 - 1.7 ms. At the scale of 16 servers, IDIOMS is able to achieve up to 221× performance boost as compared to SoMeta and outperforms the SQLite variant by 180×. Even at the scale of 128 servers, IDIOMS is able to achieve up to 40× performance boost for all these queries. Moreover, with the help of suffix-tree mode, for infix queries, IDIOMS outperforms SoMeta by 100× to 300× and outperforms the SQLite variant by 57× to 178×.

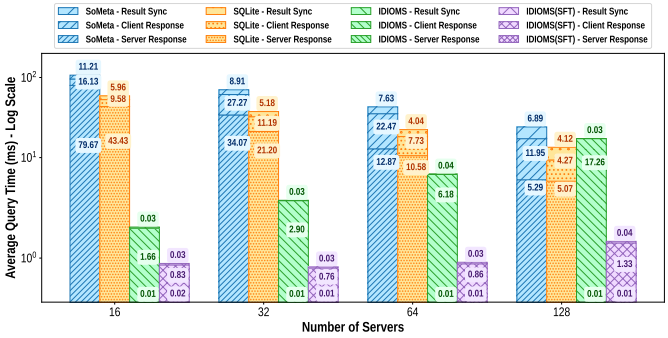

Figure 9: Average Query Time Breakdown for Collective Metadata Search. The bottom part of each stacked bar indicates the server response time, the middle part indicates the client-side response time, and the top part shows the result syncing time for all client processes to get complete results. "IDIOMS(SFT)" stands for IDIOMS with suffix tree mode, while "IDIOMS" stands for IDIOMS without suffix tree mode.

We provide a breakdown of collective metadata search in Figure 9. The time reported in this figure is the average time of all 400 queries of all 4 types we've issued in the test. We can see that, since the local full scan of all metadata attributes is inevitable in both SoMeta and the SQLite variant, the average server response time dominates the entire query time stack of each kind at every scale, ranging from 5 to 79 ms. For IDIOMS, we can see that the average server response time usually hits its bar at 0.01-0.02 ms. However, without the help of suffix-tree mode, the average client response time of all queries we have sent is following an increasing trend as the number of servers increases. This is largely because the selected query sending client still needs to scan all servers to get the infix query result. With the help of suffix-tree mode, though a very gentle upward trend is shown as our evaluation scales out, the average client response time is significantly reduced to less than 1.33 ms, leading to a significant drop in average query time at all scales. Moreover, since IDIOMS only takes one client to perform query execution, there will be no need

to perform complex `MPI_AllGather` operations, and result syncing is accomplished by more lightweight `MPI_Bcast` operations. As compared to SoMeta and the SQLite variant, we can see that, although not the most dominating factor for the query time reduction, the average result syncing time is even reduced to less than 0.04 ms.

F. Memory Overhead

TABLE I: Percentage of Additional Memory Introduced by IDIOMS (on the basis of 5.9GB metadata memory consumption)

Number of Servers	16	32	64	128
IDIOMS (suffix tree)	4.77%	11.19%	14.54%	20.63%
IDIOMS (no suffix tree)	1.09%	13.04%	25.06%	52.57%

In our evaluation, we also measured the total memory consumption of all servers. We take the total memory consumption of the metadata part in SoMeta as the basis of our evaluation (which is 5.918GB), and we show the percentage of additional memory introduced by IDIOMS at all scales in Table I. In the suffix tree mode, IDIOMS only introduced up to 20.64% additional memory. In contrast, as a result of using dedicated trie for indexing suffixes, which reduced the chance of trie node sharing among all indexed keys and values, the adoption of IDIOMS leads to at most 52.57% additional memory. Also, as the number of servers increases, the indexed keys and values tend to be more dispersed across all servers and hence have less chance to share a common trie node on the same server. This causes the increasing trend of memory consumption in both cases. But overall, IDIOMS does not introduce significant memory overhead considering the base memory consumption for all metadata attributes is only 5.9GB from all servers and the abundant 100GB+ memory space that each compute node of a modern supercomputer mostly has.

V. Related Work

Historically, scientific data is mainly stored on file systems, such as GFS [32], BeeGFS [32], [33], HDFS [34], Lustre [35], GlusterFS [36], CephFS [37], Spectrum Scale [38], etc. In these systems, fundamental metadata search capability is supported for users to locate files of interest based on metadata attributes of files, mostly file names. Given the limitations in traditional file systems (e.g. file name length, metadata types, hierarchical structure, etc), the object-centric data management (a.k.a ODM) gets more and more popularity. Some of the early efforts in object-centric data management either expose ODM features on top of the file system (e.g., Ceph-RADOS [39]) or store objects into files on file systems (e.g., HDF5, netCDF, ADIOS) and such practices makes the scalability of these systems be bound to that of the supporting file systems. Influenced by the software-defined storage paradigm, modern software-defined ODM systems like PDC [12], DAOS [11], HEPnOS [10], tend to provide flexible, scalable and parallel data management with user-level control.

Since the FAIR guiding principles were proposed in 2016 [40], as one important aspect of "Findability" defined in the FAIR principles, metadata search has been the subject of consistent focus in the scientific data management community and many metadata search solutions have been proposed. Some of them are using database systems as the store of metadata, like AMI [28], JAMO [19], SPOT Suite [20], and BIMM [14]. The centralized database may lead to scalability issues and single point of failure and hence may not suit the need of ODM systems. Solutions like TagIt [17] and Empress [16], are utilizing embedded databases, such as SQLite, as the metadata store. However, each affix-oriented metadata query in ODM systems would still incur a complete scan on all distributed server processes, which can cause network congestion. Distributed adaptive radix tree (DART) [25] addresses the distributed affix-based keyword search problem. However, metadata attributes are not a single keyword. Rather, they are often in the form of key-value tags in most modern ODM systems. Thus, DART cannot be directly applied to metadata search in ODM systems. Additionally, there is no support for efficient infix query in DART.

VI. Conclusion & Future Work

Affix-oriented metadata search capability is essential to the day-to-day work of scientists as well as large-scale scientific applications where finding data sets via metadata information is highly needed. To enable efficient affix-oriented metadata search on top of object-centric data management system is significant and imperative with the rapid transition towards ODM systems in the science community. In this study, we proposed an index-powered distributed object-centric metadata search solution, named IDIOMS. It offers high-performance affix-oriented metadata search on top of ODM systems, eliminates the need of scanning all distributed servers for affix-oriented metadata search and avoids the effort of encapsulating DBMS in the implementation of an ODM system, outperforms the DBMS-powered solution by at least 57-370$\times$ and achieves at least 40-407$\times$ performance boost as compared to the pure metadata scan approach in SoMeta. More importantly, the design of IDIOMS can be also applied to other distributed storage systems where affix-oriented metadata search is needed. In the future, we plan to expand IDIOMS' capability to support even more types of metadata search queries on even more types of metadata, and also we plan to provide more complex metadata search capability in the next version of IDIOMS.

Acknowledgment

The work is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. (Project: End-to-end object-centric data management, Program managers: Dr. Hal Finkel and Dr. Margaret Lentz). This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility. The work was the result of a collaboration between Lawrence Berkeley National Laboratory (LBNL) and The Ohio State University (Grant # GR130493, a subcontract from LBNL).

REFERENCES

[1] AdvancedBioimagingCenter, "LLSM5DTools: Tools for 5D Light Sheet Microscopy Data Processing," https://github.com/abcucberkeley/LLSM5DTools/tree/dev, 2023.

[2] X. Ruan and S. Upadhyayula, "LLSM5DTools: Tools for the analysis of 5D live images from lattice light-sheet microscopy (LLSM)," https://github.com/francois-a/llsmtools, 2020.

[3] X. Ruan, M. Mueller *et al.*, "Image processing tools for petabyte-scale light sheet microscopy data," *bioRxiv*, 2024. [Online]. Available: https://www.biorxiv.org/content/early/2024/01/01/2023.12.31.573734

[4] R. Gao, S. M. Asano *et al.*, "Cortical column and whole-brain imaging with molecular contrast and nanoscale resolution," *Science*, vol. 363, no. 6424, p. eaau8302, 2019.

[5] S. Habib, V. Morozov *et al.*, "The universe at extreme scale: multi-petaflop sky simulation on the BG/Q," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[6] A. S. Almgren, J. B. Bell *et al.*, "Nyx: A massively parallel amr code for computational cosmology," *The Astrophysical Journal*, vol. 765, no. 1, p. 39, 2013.

[7] J. Elmsheuser, C. Anastopoulos *et al.*, "Evolution of the ATLAS analysis model for Run-3 and prospects for HL-LHC," in *EPJ Web of Conferences*, vol. 245. EDP Sciences, 2020, p. 06014.

[8] B. Friesen, A. Almgren *et al.*, "In situ and in-transit analysis of cosmological simulations," *Computational astrophysics and cosmology*, vol. 3, no. 1, pp. 1–18, 2016.

[9] F. Aguet, S. Upadhyayula *et al.*, "Membrane dynamics of dividing cells imaged by lattice light-sheet microscopy," *Molecular biology of the cell*, vol. 27, no. 22, pp. 3418–3435, 2016.

[10] S. Ali, S. Calvez *et al.*, "HEPnOS: a Specialized Data Service for High Energy Physics Analysis," in *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2023 - Workshops, St. Petersburg, FL, USA, May 15-19, 2023*. IEEE, 2023, pp. 637–646. [Online]. Available: https://doi.org/10.1109/IPDPSW59300.2023.00108

[11] J. F. Lofstead, I. Jimenez *et al.*, "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 2016, Salt Lake City, UT, USA, November 13-18, 2016*, J. West and C. M. Pancake, Eds. IEEE Computer Society, 2016, pp. 585–596. [Online]. Available: https://doi.org/10.1109/SC.2016.49

[12] H. Tang, S. Byna *et al.*, "Toward Scalable and Asynchronous Object-Centric Data Management for HPC," in *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018, Washington, DC, USA, May 1-4, 2018*, E. El-Araby, D. K. Panda *et al.*, Eds. IEEE Computer Society, 2018, pp. 113–122. [Online]. Available: https://doi.org/10.1109/CCGRID.2018.00026

[13] PostgreSQL, "PostgreSQL," https://www.postgresql.org, 2022.

[14] D. Korenblum, D. L. Rubin *et al.*, "Managing Biomedical Image Metadata for Search and Retrieval of Similar Images," *J. Digital Imaging*, vol. 24, no. 4, pp. 739–748, 2011.

[15] sqlite.org, "SQLite," https://sqlite.org, 2022.

[16] M. Lawson, W. Gropp, and J. F. Lofstead, "EMPRESS: Accelerating Scientific Discovery through Descriptive Metadata Management," *ACM Trans. Storage*, vol. 18, no. 4, pp. 34:1–34:49, 2022. [Online]. Available: https://doi.org/10.1145/3523698

[17] H. Sim, A. Khan *et al.*, "An Integrated Indexing and Search Service for Distributed File Systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 31, no. 10, pp. 2375–2391, 2020. [Online]. Available: https://doi.org/10.1109/TPDS.2020.2990656

[18] "MongoDB," https://www.mongodb.com, 2022.

[19] Joint Genome Institute, "The JGI Archive and Metadata Organizer(JAMO)," http://cs.lbl.gov/news-media/news/2013/new-metadata-organizer-streamlines-jgi-data-management, 2013.

[20] T. Craig E., E. Abdelilah *et al.*, "The SPOT Suite project," http://spot.nersc.gov/, 2013.

[21] "Introducing JSON," https://www.json.org, 2018.

[22] W. Zhang, S. Byna *et al.*, "MIQS: Metadata Indexing and Querying Service for Self-Describing File Formats," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC'19, Denver, Colorado, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356146

[23] M. Kuhn and K. Duwe, "Coupling Storage Systems and Self-Describing Data Formats for Global Metadata Management," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2020, pp. 1224–1230.

[24] Y.-J. Joung, L.-W. Yang, and C.-T. Fang, "Keyword search in dht-based peer-to-peer networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, 2007.

[25] W. Zhang, H. Tang *et al.*, "DART: Distributed Adaptive Radix Tree for Efficient Affix-based Keyword Search on HPC Systems," in *Proceedings of The 27th International Conference on Parallel Architectures and Compilation Techniques (PACT'18)*, November 2018.

[26] H. Tang, S. Byna *et al.*, "SoMeta: Scalable Object-Centric Metadata Management for High Performance Computing," in *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 359–369.

[27] H. Dai, Y. Wang *et al.*, "The State of the Art of Metadata Managements in Large-Scale Distributed File Systems - Scalability, Performance and Availability," *IEEE Trans. Parallel Distributed Syst.*, vol. 33, no. 12, pp. 3850–3869, 2022. [Online]. Available: https://doi.org/10.1109/TPDS.2022.3170574

[28] J. Fulachier, J. Odier *et al.*, "ATLAS Metadata Interface (AMI), a generic metadata framework," in *Journal of Physics: Conference Series*, vol. 898, no. 6. IOP Publishing, 2017, p. 062001.

[29] National Energy Research Scientific Computing Center, "Perlmutter System Architecture," https://docs.nersc.gov/systems/perlmutter/architecture/, accessed: 2023-11-01.

[30] "National Energy Research Scientific Computing Center," https://www.nersc.gov/, accessed: 2023-11-01.

[31] S. Byna, B. Dong *et al.*, "Proactive Data Containers (PDC) v0.3," https://github.com/hpc-io/pdc/releases/tag/0.3, 5 2017. [Online]. Available: https://www.osti.gov/servlets/purl/1772576

[32] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, M. L. Scott and L. L. Peterson, Eds. ACM, 2003, pp. 29–43. [Online]. Available: https://doi.org/10.1145/945445.945450

[33] "BeeGFS Documentation 7.4.1," https://doc.beegfs.io/, accessed: 2023-09-01.

[34] D. Borthakur *et al.*, "HDFS architecture guide," *Hadoop apache project*, vol. 53, no. 1-13, p. 2, 2008.

[35] P. Schwan *et al.*, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux symposium*, vol. 2003, 2003, pp. 380–386.

[36] E. B. Boyer, M. C. Broomfield, and T. A. Perrotti, "Glusterfs one storage server to rule them all," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2012.

[37] G. Borges, S. Crosby, and L. Boland, "CephFS: a new generation storage platform for Australian high energy physics," in *Journal of Physics: Conference Series*, vol. 898, no. 6. IOP Publishing, 2017, p. 062015.

[38] D. Quintero, L. Bolinches *et al.*, *IBM Spectrum Scale (formerly GPFS)*. IBM Redbooks, 2017.

[39] S. A. Weil, A. W. Leung *et al.*, "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*, 2007, pp. 35–44.

[40] M. D. Wilkinson, M. Dumontier *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.