Using Provenance to Efficiently Improve Metadata Searching Performance in Storage Systems

Jinjun Liu, Dan Feng*, Yu Hua, Bin Peng, Zhenhua Nie

Wuhan National Laboratory for Optoelectronics

School of Computer Science and Technology, Huazhong University of Science and Technology, WuHan, China

Abstract

In cloud storage systems, more than 50% of requests are metadata operations, and thus the file system metadata search performance has become increasingly important to different users. With the rapid growth of storage system scales in volume, traditional full-size index trees can not offer high-performance metadata search due to hierarchical indexing bottleneck. In order to alleviate the long latency and improve the quality-of-service(QoS) in cloud storage service, we proposed a novel provenance based metadata-search system, called PROMES. The metadata search in PROMES is split into three phases: i) leveraging correlation-aware metadata index tree to identify several files as seeds, most of which can satisfy the query requests, ii) using the seeds to find the remaining query results via relationship graph search, iii) continuing to refine and rerank the whole search results, and sending the final results to users. PROMES has the salient features of high query accuracy and low latency, due to files' tight and lightweight indexing in relationship graph coming from provenance's analysis. Compared with state-of-the-art metadata searching schemes, PROMES demonstrates its efficiency and efficacy in terms of query accuracy and response latency.

Keywords: Metadata search, provenance, cloud storage service

1. Introduction

Today file metadata operations account for more than 50% of all requests in cloud storage systems, which results in that the accurate and low-latency file metadata retrieving becomes a critical research problem[1]. Hundreds or thousands of common users want to use and analyze the data via sharing files in petabyte, even exabyte scale file systems. Users need to identify the interested data via searching file metadata. In HPC analytics applications [2], the scientists can reduce the input data and enormously decrease the overhead of computing and storing via file metadata search[3]. For example, the scientists run 1 ligand compound and need 10-million protein targets which are stored in 10-million files in the eScience applications. There are only 1000 protein targets needed for analyzing protein-ligand interactions via the simple file metadata searches. File metadata search can also help storage administrators understand the properties of the data being stored, and decide on their management policies to improve the QoS in cloud storage service[1]. With the rapid growth of storage system scales in volume, it is becoming more and more important to obtain the required data via file metadata search to different users.

There are, unfortunately, enormous size of big data in today's data centers. The large fraction of data in data centers are stored and infrequently accessed. Traditional file systems based on hierarchical namespace cant be efficient to high-performance metadata search due to these garbage data. Several file storage systems[4] use relational database systems to manage and retrieve metadata. Nevertheless, due to the limitation of requiring too many system resources and the lack of scalability, this technology is not able to provide high performance metadata searching and updating. There are some file/storage systems, such as, ZFS, Lustre, Drop[5], using files' path name hashing for the metadata searching. This approach works well for point query, but does not apply to advanced metadata queries, such as, Range query, Top k query, and approximate query.

Currently, the state-of-the-art studies on the metadata search focus on using high-performance metadata index trees to accelerate metadata searching and enhance the accuracy ratio of queries, such as Ceph[6], Spyglass[1], SmartStore[7], Security Aware Partitioning[8], GIGA+[9]. Popular multi-dimensional hierarchical indexing structures include the R-tree[10], the B-tree, the X-tree, the M-tree, the KD-tree[11] and their variants. These metadata index trees group metadata via single or multi file attributes(e.g., name space, creator, creation time, and update time).

*Corresponding autor. Tel.: 86-027-87792450.
Email addresses: `liujinjun@hust.edu.cn` (Jinjun Liu), `dfeng@hust.edu.cn` (Dan Feng), `csyhua@hust.edu.cn` (Yu Hua), `pengbin@hust.edu.cn` (Bin Peng), `niezhenhua@hust.edu.cn` (Zhenhua Nie)

The method of using metadata index trees is suitable for a small-scale storage system. With the sharp growth in the amount of files and indices, traditional full-size index trees cannot offer high-performance metadata search, and suffer from three major disadvantages. First, hierarchical indexing bottleneck is unavoidable. With the increase of dimensionality, the dimensions of disaster will come up in the tree-like index structures and the index tree will be not able to be kept entirely in memory and bring vast disk I/O. Second, traditional file attributes may be not helpful to find the requested files, which is suggested by some recent studies[12]. For example, concerning the size of a document, the common search criteria, 30.8% of the recalls are partial correct and 53.8% of the recalls are utterly incorrect. Traditional full-size index trees are often built based on these attributes. Therefore, depending only on these index trees, it is difficult to offer high-performance metadata search.

Third, through these traditional file attributes, we only get the current relationship between files, and ignore their historical relationships, which leads to the difficultly of pruning the unnecessary search space in metadata searching. For example, an Approximate Nearest Neighbor query can locate the nearest and the lowest MBR(Minimum Bounding Rectangle) in R-tree, which will be used to index the metadata. If the results in this MBR are not enough for users, the adjacent MBRs will be retrieved. Unfortunately, these MBRs are always divided into many branches, some of which are stored on disks, and accessing the disks will greatly reduce the efficiency of metadata search.

With the development of data analytics technologies, there are lots of new challenges that need to be addressed in a good metadata search system , such as energy conservation, real-time, versatility, and scalability.

To address these issues, we proposed a novel ***PRO***venance based ***ME***tadata-search ***S***ystem, called PROMES. PROMES leverages correlation-aware metadata index tree to identify several files as seeds. The operation of searching seeds is very rapid due to the small range of seeds and efficient search positioning in spatial index, and most of seeds can satisfy the query requests. Then PROMES uses seeds to find other query results via relationship graph searching, and finally sends completed and accurate query answers to users after refining and reranking the whole query results. We store the files relationships and discard the primitive provenance information, and PROMES can build a lightweight relationship graph coming from analyzing files' provenance. In order to describe the more effective relationships between files, the factor of time and the files' weight have been considered in computing the relevance score due to temporal locality in file search and various values of files in cloud storage services.

Some recent studies have focused on using provenance to organize and retrieval files. Keiko Yamamote et al. [13] use some words which were stored in provenance as the keys of file queries, such as "which files did I get from Tom yesterday?", "which files did I edit by MS PowerPoint last week?". This kind of file search is restricted to some specific operations and cannot be used in the complex application environment. Shah et al. [14] designed a context-enhanced search architecture, which can improve the hit rates via reordering and extending the results coming from the pure content-only search. They used the breadth-first reordering and extending mechanism to calculate the relevance score that only relate to the fraction of the outgoing edge weight and reliability weight, and the relevance score calculation will bring large overhead and inaccuracy.

Compared with state-of-the-art metadata searching scheme, PROMES demonstrates its efficiency and efficacy in terms of query accuracy and response latency. The average speed of a metadata query in our PROMES can be up to 1-2 orders of magnitude faster than traditional index tree structure. When the range of search predicates is bigger, The space consumption of metadata query for PROMES can be also down to 1-2 orders of magnitude lower than in our reference index tree. The overhead of building the relationship graph grows linearly with data volume and the relationship graph with 50 millions of files and files' relationship can be stored in about 800MB of main memory. PROMES ensures the query accuracy is more than 90% via experimental evaluation. Due to reducing the expensive accesses the disks, PROMES can provide the efficient metadata searching. Moreover, the traditional method has to compare vast amount of files' metadata in candidate subtrees to achieve better query accuracy, some of which might be stored on disk.

The contributions of this paper are threefold.

High-performance, low-cost searching structure. We propose a novel provenance based metadata-search system, called PROMES, which takes advantage of the correlation-aware metadata index tree to quickly identify seeds, and then uses seeds to find other query results via relationship graph searching. The index tree and the graph can be designed to be stored in memory. Then, all of query operations can be finished in memory, and finally PROMES effectively improves metadata searching performance.

Accurate relationship description. In order to obtain precise files' relationship and improve the query accuracy, we employ two parameters, namely, the relationship's time factor and the files' weight, to compute the relevance overall scores. The first parameter is to computer the tradeoff between historical relationship and current relationship, and the second one reflects the importance of files' weight in the relevance overall score. The experiment results demonstrate that choosing two suitable parameters can get a high hit rate of metadata queries.

Quantitative experimental evaluation. We evaluate a software implementation of PROMES, showing that the relationship graph coming from the analysis of provenance can exhibit and record the more precise files' relationship in modern applications. This relationship can effectively reduce the overhead of metadata searching.

The remainder of this paper is organized as follows.

Section II introduces the necessary background for our work. In Section III, we describe the architecture and workflow of PROMES. Section IV explains how our system manages and searches the relationship graph and calculates the relevance score. In Section V, we present and discuss the evaluation results. Section VI summarizes the related work. Finally, Section VII concludes our paper.

2. Background and Motivation

This section describes challenges of file metadata search, the usage of provenance in storage system and our motivation of this work.

2.1. File Metadata and Metadata Search

Traditional file metadata are divided into three kinds according to the sources of file metadata. The fist one is generated by users, such as file name, author, keywords, and note. Since these metadata are not uniquely distinguishable each other, it is difficult to exploit the relevance of files via analyzing those metadata[15]. For example, most of programmers like to name the applications' configuration file as "configuration.txt". The second one is generated by applications, such as filename extension, retention policy, and open mode. This kind of file metadata only records files' current settings and status. Thus it is the static metadata which ignores the file's relationship information created during the files' operations. The third kind of file metadata is generated by filesystems or storage systems, such as size, owner, and timestamps. Almost every file has such metadata and the traditional metadata index is built based on these metadata. The traditional metadata only record the files' static attributes and are deficient for mining the relevance of files. Some studies tried to use the files' dynamic attributions to index metadata, such as the access times, the amount of data being read or written. These attributes can be collected via tracing filesystem for analyzing the relevance of some files.

With the explosively growing size of data storage systems, some recent studies [16, 17] collect and analyze characteristics of various system workloads in practical application. From these studies, some new features of the file metadata are found. For example, the majority of file's attributes is power law distributed, the number of the hot files or the hot users and the times of the operations significantly exceed the Pareto principle, there are some files' attributions have null values. The features have brought new challenges in the research of metadata index and search optimization.

In HPC applications, the demand for complex queries is markedly increasing, such as ANN searching, Top k searching, and Skyline searching. With the growing size of storage volume, the traditional method of complex queries becomes increasingly difficult to support high-performance application. More and more researchers focus on using provenance information to optimize metadata searching. We need to find out the files' relevance via analyzing the various information of files as much as possible.

2.2. Usage of Provenance in File Searching

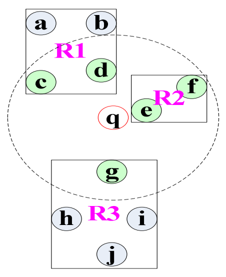
There are two methods of using provenance to optimize metadata searching. The first one uses provenance searching to optimize metadata searching. Recent studies[12] have shown that users can't remember and find the files through some traditional attributes. Due to the provenance recording all the information about the files' history and status, the users can finally find the files via searching provenance as long as they remember little clues about the files. For example, a user may forget a file's filename and size, even filetype, while, he remembers that he has received the file from *Tom*. According to this information, he can find the file via searching the provenance. This method is always used to improve query accuracy in personal filesystems and small-scale filesystems. In cloud storage system, the volume of provenance is increasing quickly, resulting in that the provenance search becomes more and more inefficient. Although the provenance information can be compressed, it does not scale with the rapid expansion of volume of storage systems. Hence, this kind of provenance approach is not suitable for large-scale storage system.

The second method uses the provenance to speed up metadata searching. In order to improve the query accuracy, the range of searching increases gradually. If we reduce the range, the query processing will be accelerated. Hence, how to cluster the relational files becomes a very important topic in metadata search. The provenance includes more files' relevance information than traditional file metadata. It will be a nice way to optimize metadata indexing and searching via analyzing the provenance to mine the files' relationship. For example, Figure 1 shows a snapshot of a video-editing provenance trace (generated in the PASS system[18]). We can get some relationships via the analysis of provenance trace. According to the level of collection, the provenance can be divided into application-level and system-level provenance. Since this paper focuses on the QoS of the storage systems[19], PROMES builds the relationship graph via system-level provenance. With the rapid growth in the amount of files and indices in cloud storage systems, some metadata index subtrees have to be stored on disk. When a query request was received by a storage system, the system would execute the query in some subtrees. If the system has an inefficient metadata index, the range of searching will become wide and include some subtrees which are stored on disk. Hence, the traditional full-size index trees can not offer high-performance metadata search any more.

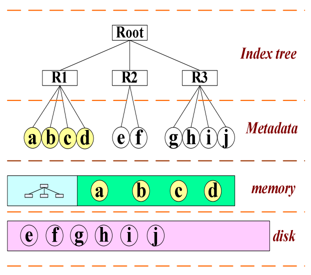
For example, as shown on Figure 2(a), there are ten file nodes, a, b, ..., i, j, and a Top 5 query request q. The file nodes within the dashed circle are the query results. If we use an R-tree to index the files' metadata, there are three MBRs, $R1$, $R2$ and $R3$, and the $R1$ includes four files a, b, c, and d, and the $R2$ includes tow

```
210.0 NAME /home/ffmpeg-2.1.1/ffmpeg
210.0 ARGV [ffmpeg][-i][concat][/home/ffmpeg-2.1.1/example/a.avi][/home/ffmpeg-2.1.1/example/b.avi][-c][copy][/home/ffmpeg-2.1.1/example/c.avi]
210.1 INPUT [ANC] 211.0
211.0 NAME /home/ffmpeg-2.1.1/example/c.avi
215.0 NAME /home/ffmpeg-2.1.1/ffmpeg
215.0 ARGV [ffmpeg][-ss][0:0:30][-t][1:10:20][-i][/home/ffmpeg-2.1.1/example/c.avi][-vcodec][copy][-acodec][copy][/home/ffmpeg-2.1.1/example/d.avi]
215.1 INPUT [ANC] 216.0
216.0 NAME /home/ffmpeg-2.1.1/example/d.avi
227.0 NAME /home/ffmpeg-2.1.1/ffmpeg
227.0 ARGV [ffmpeg][-i][/home/ffmpeg-2.1.1/example/d.avi][-ab][128][-ar][22050][-r][29.97][-qscale][6][-y][/home/ffmpeg-2.1.1/example/f.mp4]
227.1 INPUT [ANC] 228.0
228.0 NAME /home/ffmpeg-2.1.1/example/f.mp4
235.0 NAME /home/ffmpeg-2.1.1/ffmpeg
235.0 ARGV [ffmpeg][-i][/home/ffmpeg-2.1.1/example/e.wmv][-ab][128][-ar][22050][-r][29.97][-qscale][6][-y][/home/ffmpeg-2.1.1/example/f.mp4]
235.1 INPUT [ANC] 236.0
236.0 NAME /home/ffmpeg-2.1.1/example/f.mp4
244.0 NAME /home/ffmpeg-2.1.1/ffmpeg
244.0 ARGV [ffmpeg][-i][/home/ffmpeg-2.1.1/example/g.avi][-ab][128][-ar][22050][-r][29.97][-qscale][6][-y][/home/ffmpeg-2.1.1/example/h.mp4]
244.1 INPUT [ANC] 245.0
245.0 NAME /home/ffmpeg-2.1.1/example/h.mp4
246.0 NAME /home/ffmpeg-2.1.1/ffmpeg
246.0 ARGV [ffmpeg][-i][/home/ffmpeg-2.1.1/example/g.avi][-ab][128][-ar][22050][-r][29.97][-qscale][6][-y][/home/ffmpeg-2.1.1/example/i.wmv]
246.1 INPUT [ANC] 247.0
247.0 NAME /home/ffmpeg-2.1.1/example/i.wmv
```

Figure 1: Provenance trace. The notation "A NAME B" in the provenance trace means that the name of A is B which is a process or a file. If B is a file, the notation "A INPUT [ANC] B" means that a process A writes data into a file B. The notation "A ARGV B" means the parameter of a process A is B. We can get some relationships via the analysis of the provenance trace.

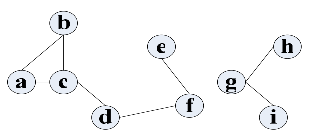


(a) File distribution.

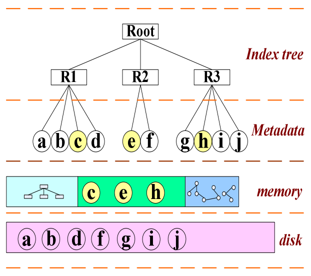


(b) Traditional index tree.

Figure 2: Traditional index tree



(a) Relationship graph.



(b) PROMES.

Figure 3: Index structure of PROMES

files e f, and other four files g, h, i, and j belong to the $R3$. The traditional index tree(R-tree) is displayed in Figure 2(b). According to the principle of locality of file access, the yellow file nodes, a, b, c, and d, are hot nodes and thus cloud be stored in the memory, and the other nodes have to be stored on disk. In order to finish the query q, comparison operations will be executed 14 times, $(q, Root), (q, R1), (q, R2), (q, R3), (q, a), ..., (q, j)$, and access six files' metadata from disk. Even when there is merely a result g in the $R3$, the metadata of the other three files which are included in the subtree $R3$ will be obtained from disk.

There are lots of new challenges that would need to be addressed in a good metadata search system with the development of data analytics technologies. Power conservation is one of the newest research focuses in the field of computer architecture research, and it is also becoming a new challenge in the field of metadata search. Because the data centers store petabyte, even exabyte files, a terrible metadata index structure will lead to large number of calculations, disk accesses and network transmissions, which may consume a large amount of energy. There are also other challenges for the metadata search, such as realtime, versatility, and scalability.

In order to efficiently improve metadata searching performance in cloud storage systems, we need a more compact structure to prune the invalid subtrees and file nodes as much as possible. From the foregoing description of provenance, we know that the relationship graph, coming from the analysis of provenance, can offer a more precise file correlation. We adopt the relationship graph to reduce the overhead of the query. Because this graph will be accessed frequently and has to be stored on the memory, the relationship graph will be small by only storing relationship information and few files' information. And because the graph will occupy some memory space, we need building a correlation-aware metadata index tree to find out the seeds. This tree would ensure that the seeds of each query can be found from the relevant sub-trees.

For example, the relationship graph, as shown on Figure 3(a), records the relationships between files, displayed on Figure 2(a), and this graph can be built by analyzing files' provenance which is shown in Figure 1, and will be also stored in the memory, as shown on Figure 3(b). In PROMES, the less yellow file nodes, c, e, and h, are stored in the memory due to the correlation-aware metadata index tree. Hence, for the query request q, the results c and e can be hit in RAM, and these two file nodes and the node h will become the seeds. According to the seeds and the conditions of query, we can search the relationship graph

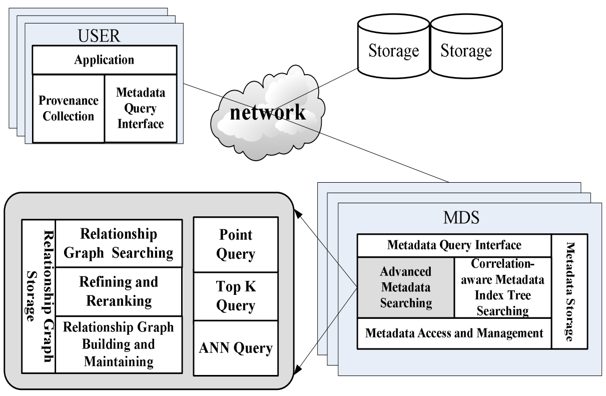

USER
Application
Provenance Collection
Metadata Query Interface
network
Storage
Storage
MDS
Metadata Query Interface
Advanced Metadata Searching
Correlation-aware Metadata Index Tree Searching
Metadata Access and Management
Metadata Storage
Relationship Graph Storage
Relationship Graph Searching
Refining and Reranking
Relationship Graph Building and Maintaining
Point Query
Top K Query
ANN Query

Figure 4: The system architecture of PROMES

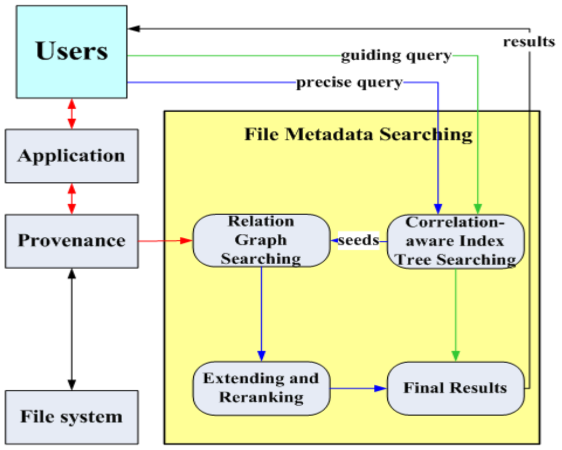


Figure 5: The workflow of file metadata searching

and discover that the remainder results are d, f, and g. Finally, we access the disk to get these three files' metadata. Comparing with six files' metadata via traditional index tree searching, there are only three files' metadata which need to be read from the disk.

3. The Design of PROMES

3.1. The System Architecture

In cloud storage systems, users can search files which are stored in the storage nodes through accessing the MDSs, and these MDSs manage and store files' metadata. As shown in Figure 4, the architecture of PROMES is divided into two parts, running on the client-side and on the MDS-side.

The first part, running on the client-side, includes the module of provenance collection and the interface of metadata query. The module of provenance collection collects the provenance of user's files and extracts the relationship of files, and then sends these relationship information to the MDSs. The metadata query interface gets various query requirements of applications or users, and receives and shows(returns) the query results.

The second part, running on the MDS-side, includes five modules. The metadata query interface on the MDS receives the query request from client-side and returns the query results to the client-side, the module of correction-aware metadata index tree searching can provide three kinds of metadata queries, the module of metadata storage is responsible for management of the metadata distribution on the disk, and the module of metadata access and management supports metadata to access, update, add, delete, move, and other operations.

Our work focuses on the module of advanced metadata searching, which includes five modules. In order to support the high query accuracy and low-latency, we leverage an in-memory non-relation database to store the relationship graph in the module of relationship graph storage. We will refine and rerank the whole search results for potentially improving query accuracy via the module of refining and reranking. The three query modules offer three kinds of frequently-used query, point query, top k query and ANN query. The module of relationship graph searching can rapidly find out the accurate results. The module of relationship graph building and maintaining can create and maintain the relationship graph after receiving the relationship of the files from the client-side.

3.2. The Workflow

In cloud storage systems, there are different characteristics of queries due to various search tasks which are driven by different users. According to the different QoS requirement in the process of user queries, there are commonly two steps[20]. Because the users have not any priori knowledge of the system and they firstly want to get some understanding about the system, or the users could not insure that the search criteria was suitable for themselves. They don't need the very precise and too many query results. They need only few search results quickly to guide the next query. This step can be done by searching correlation-aware metadata tree index. When users finally determine their query conditions, the system needs to provide users with the accurate query results. As shown in Figure 5, two kinds of search modes are designed in PROMES,.

The first kind of search mode, called guiding search, is correlation-aware metadata tree index search, the users can get few results through the guiding search, and this operation only consumes very little time. There are two reasons why we can quickly get these search results. The first reason is that we can quickly locate several branches closest to the query point in the index tree structure. The second one is that the metadata index tree used by us is vastly different from traditional full-size index trees, we do not store the whole index tree and only stored the frequent searched files' closest branches in memory. Recent studies[21, 22] have shown that data which have been queried may also be re-queried later, which means the query has a temporal locality and spatial locality. Thus we can consume a few memory spaces to store these query hotspots' adjacent index branches to achieve high performance index search.

In order to quickly provide more accurate answers, we develop the second search mode in PROMES, called precise search. There are three steps to finish this search mode. The first step identifies several files as seeds via the correlation-aware metadata tree index search. Because these seeds are most adjacent to the query node, most of the seeds are the query results. The second step uses the seeds to find the other query results via relationship graph searching. The last step is providing final results to user after refining and reranking the result set.

The relationship graph, stored on the MDS-side, is built from analyzing the files' provenance. We use a provenance collection tool to record the information about the files' usage on the client-side. These application operations include creating, reading, writing, copying, moving and removing. This work will be described clearly in the next section.

3.3. Refining and Reranking Search Results

Because there is a set of search result candidates which corresponds to every seed via the relationship graph search and the number of these candidates exceeds the needs, these search results need to be combined and refined. According to the type of search, we adopt the number or the minimum relevance threshold to refine the query results.

In order to search effectively, a search system needs to be actually able to help users to obtain the data they are looking for. According to some researches about search on the internet web, there are only top few ranked search results which are needed. Therefore, considering the energy consumption of a query operation, we use insertion sort to get and gradually send and show the final results in PROMES.

4. Relationship Graph Representation

4.1. Converting Provenance Graph to Relationship Graph

The provenance graph[23] can be created by analyzing the files' provenance from the client-side. In the Open Provenance Model[24], a provenance graph is defined as a directed acyclic graph. We can obtain the information about ancestry of any node via the provenance graph, and thus we get the files' lineage relationship (parent-child relationship) in the file provenance graphs.

As shown in Figure 7(a), we find its father *G2* and *G3* from *P2* along the blue arrows. However, it is not enough to get this longitudinal origin relationship between files and ignore nodes' transverse brotherly relations in the metadata search. For example, two files always are accessed by the same application together, which reflects that these two files have some relationships. We remove the direction attribute of the provenance graph via using the algorithm which is shown in Figure 6, and use an undirected graph, called the relationship graph, to describe the files' relationship. Figure 7(b) shows we not only find its father *G2* and *G3* from *P2*, but also find its child *F1*, its grandson *C1*, brother *P1* and brother's father *G1* via the relationship graph.

```
Convert_Graph(G, G')
Input: a provenance graph G=(V,E)
Output: a relationship graph G'=(V',E')
while ($E \neq$ NULL) do
    E → (u,v)
    (u,v) → E'
    (v,u) → E'
    v, u → V'
    /*The nodes $v$, $u$ are added to the relationship graph $G'$*/
end while
```

Figure 6: Converting the provenance graph into the relationship graph.

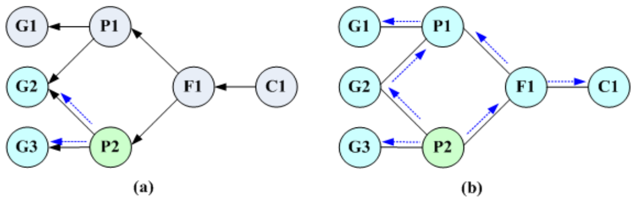


Figure 7: The ANN query in the provenance graph and the relationship graph

After we convert the provenance graph to the relationship graph, the multipath will be caused by using an undirected edge to describe a relationship. As shown in Figure 7(b), we can get the relationship between*P1* and *P2* via *G2* or *F1*. The difference between the two paths will determine whether *P1* is the query result. According to existing studies on graph query processing, the relationship can be superimposed, and the relationship between *P1* and *P2* is the sum of the relationships via these two paths.

4.2. Relationship Graph Management and Retrieval

In order to make PROMES having the salient features of high query accuracy and low latency, the relationship graph needs to be small enough to be fit in memory. After the seeds are identifed via the correlation-aware metadata index tree search, PROMES needs to quickly find out the seeds' adjacent nodes via the relationship graph searching.

In order to satisfy the above two performance requirements of the relationship graph, we customize the Berkeley DB to store and search this graph. There are two advantages of using Berkeley DB here to provide high-performance metadata searching. First, this database is not only easy and simple to use, but also a compact system. It only occupies tens of megabytes of memory space and can manage and retrieve 256 terabytes of data. Second, this database can provide high-performance concurrent retrieval of key/value pairs. To reduce the additional space overhead of PROMES, PROMES only stores the files' relationships extracting from the primitive provenance information.

In a common cloud object service, it would provide a REST-based architecture for accessing objects and uses a unique URI(Uniform Resource Identifier) to identify each object. PASSv2[25] can run in the cloud and use the URI to identify an object. Because an URI includes tens of or hundreds of chars, and the index of the relationship graph would be stored in the memory, we replace it with a unique ID, which is unsigned integer, to identify a file.

The relationship graph management and retrieval is the most important part in our design. This part includes three sub-modules, as shown in Figure 4,which are *the provenance collection module*, *the relationship graph building and maintaining module* and *the relationship graph searching module*.

The provenance collection module would be placed on the client-side to collect the files' provenance and extract the files' relationships. Many tools can be used for this work. TaskTracer[26] tracks files and applications include some high level operations, such as email, phone call, clipboard, etc. These provenances can be only used in knowledge work for interruption recovery and knowledge reuse. Provenance-Aware Storage System(PASS)[18] is a storage system which monitors files' operations in the operating system layer and generates a provenance graph automatically. There are many other tools to capture provenance at this level, such as ES3[27], ClearCase and Vesta[28]. In PROMES, we use PASSv2 to collect the provenance. PASSv2 provides passive monitoring in the operating system kernel and can potentially track system events at a much finer granularity. In order to minimize the time and space overhead for capturing the provenance, we only collect the information about the files' relationships, such as, the input files and the output files over the lifetime of one process, the source files and the destination files via a same $copy - paste$ operation. The parameters of the instruction and execution environment will be not collected.

The relationship graph building and maintaining module would be installed on the MDS-side. When the provenance collection module gets the relationships of files and transfer the relationships to the MDS, the MDS will perform the maintenance of the relationship graph. The files' information and the relationships will be stored into the relationship graph. The relationship graph searching module, also installed on the metadata server, will offer more accurate results. According to query conditions, the relationship graph searching module will search the seeds' adjacent nodes in the relationship graph. The detailed algorithms of these two modules' will be described in the next subsection.

4.3. Relevance Score

We define a threshold S. If the relationship score between two metadata nodes is less than S, this relationship will be deleted. There is longer lengthen from a node to another node and this relationship is less meaningful. We can improve the search effect via reducing some unnecessary relationships.

In order to obtain more precise files' relationships and improve the query accuracy on the relationship graph, we bring two parameters, the time factor and the files' weight, to compute the relevance overall scores. We use the first parameter to trade-off between historical relationship and current relationship, and utilize the second one to reflect a file's prestige.

4.3.1. Time Factor in the Relationship

In our design, we lay an emphasis on the time factor in the relationship description. There are two reasons. Firstly, this time term records the last time that two files were operated by the same application together. This time reflect not only the activeness of files and files' value, but also the probability of the next producing associated. For example, with listing bookmarks in a backward-chronological order, "del.icio.us" emphasizes on timeliness, and let users to follow the latest trends. Secondly, this time helps users to obtain the requested files. For example, a user expects to find a paper written by two authors on DBLP, while he doesn't achieve. Then he can try to look for the most recent papers written by them to know the possible relational work about this paper.

PROMES adopts the exponential decay function for attenuation relationship value over time, like PROPHET routing. In the relationship graph, an edge shows a relationship between two files. So, if two files have created relationship at time T and the relevance score is defined $e_{(T)}$, then after Δt, they have been operated together again, at this time the relevance score is $e_{\left(\mathrm{T}+\Delta t_i\right)}$,which is defined by under formula.

$$e_{\left(\mathrm{T}+\Delta t_i\right)} = e_{(T)} * e^{-\Delta t_i/\tau} + C_i \qquad (i = 1, 2, ..., n) \quad (1)$$

The τ is the residence time. If τ is 2.5 years, it means that a relevance score will drop to 37% (1/e) of its initial value after 2.5 years. The C_i is the relationship strength at the time of the i-th producing transmission. Values of these two parameters reflect the tradeoff between the new relationship and the old relationship.

4.3.2. Files' Weight in the Relationship

In cloud storage systems, the value of the files varies greatly. When we search the metadata, the files' weight need to be included. On the relationship graph of PROMES, the nodes'(files') prestige will be included to the relevance score. If there are some nodes $A_k (k = 1, 2, ..., n)$ directly connect with the node P. The w_k is the weight of A_k. The relevance score between P and A_k is defined $e'_{\left(\mathrm{P},A_k\right)}$,which is defined by under formula.

$$e'_{\left(\mathrm{P},A_k,w_k\right)} = e_{\left(\mathrm{P},A_k\right)} * w_k^{\lambda} \qquad (k = 1, 2, ..., n) \quad (2)$$

In the formula(2), the parameter λ helps adjust the importance of edge and node scores, it is always 0.2 in web graph search. It is similar that the relationship edge is

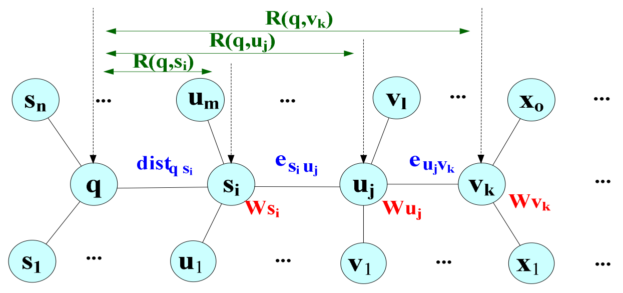


Figure 8: The procedure of computing relevance scores.

more important and more reliable to reflect which node is more important. We will test different λ to find out a nice choice in the experiment.

4.3.3. Relevance Score Computing

After receiving a request with some conditions, which can identify a node in metadata space and named the query node q, the MDS can obtain a set of seeds via searching the metadata index tree, which defined set $S = \{s_1, s_2, ..., s_n\}$. We define $R_{(q,s_i)}$ as the relevance score between the query node q and the i-th seed s_i $(i \le n)$. If the q has k-dimensions attributes, defined $C_q = \{c_{q,1}, c_{q,2}, ..., c_{q,k}\}$, the ith seed s_i has k-dimensions attribute, $C_{s_i} = \{c_{s_i,1}, c_{s_i,2}, ..., c_{s_i,k}\}$ $(i \le n)$,in metadata space,we defined $dist\left(q, s_i\right)(i \le n)$ as the distance between the query node and the seed.

Because there are correlations between these attribute dimensions, we use the Mahalanobis distance, which has a multivariate effect size and differs from Euclidean distance in that it takes into account the correlations of the data set and is scale-invariant, to computer the $dist\left(q, s_i\right)(i \le n)$. S is the covariance matrix of the matrix $\{s_1, s_2, ..., s_n, q\}$.
$$dist\left(q, s_i\right) = \sqrt{\left(C_q - C_{s_i}\right) S^{-1} \left(C_q - C_{s_i}\right)}\,(i \le n) \quad (3)$$

As show in Figure 8, when the server receives a metadata search request,$s_1, s_2, ..., s_i, ..., s_n$ are the seeds for the query node q, $u_1, u_2, ..., u_j, ..., u_m$ are connected to the seed s_i,$v_1, v_2, ..., v_k, ..., u_l$ are connected to the seed u_j, $x_1, v_2, ..., u_o$ are connected to the seed v_k. We can compute the relevance score $e_{s_i u_j}$ between s_i and u_j via operating formula(1), the $e_{u_j v_k}$ is same as $e_{s_i u_j}$. We define that e_{ps_i} is equal to $dist_{qs_i}$ via operating formula(3) and the overall score $R_{(q,u)}$ is the relevance score between the node q and u. In consideration of the node prestige score and edge weight, combining with formula (3), we can define the formula (4).

$$R\left(qs_i\right) = \frac{dist_{qs_i} * w_{s_i}^{\lambda}}{\sum_{i=1}^{n} dist_{qs_i} * w_{s_i}^{\lambda}} \qquad (k = 1, 2, ..., n) \quad (4)$$

When we get the overall score $R\left(qs_i\right)$, the overall score $R\left(qu_j\right)$ can be computed via operating the formula (5) similar to the formula (4).

$$R\left(qu_j\right) = R\left(qs_i\right) * \frac{e_{s_i u_j} * w_{u_j}^{\lambda}}{\sum_{j=1}^{n} e_{s_i u_j} * w_{u_j}^{\lambda}} \quad (k = 1, 2, ..., n) \quad (5)$$

Table 1: Notation used in relationship graph searching and maintaining algorithms

q	a query node
u, v	the file node u , the file node v in the relationship graph
$R_{u,q}$	the relevance score of $q$and u
s_i	the i-th seed
S	Set of s_i
$dist_{q,s_i}$	the distence from q to s_i in traditional index space
d_i	the i-th diffused file node
D	Set of diffused file nodes
AD	Set of the file nodes awaiting for being diffused
ad_{head}	the head node with the max $R_{q,u}$ in AD
$e_{u,v}$	the weight of the edge $\langle u, v\rangle$
w_u	the node prestige score of the file node u

Step by step, we can compute all overall scores between the file node u and the query node q, and then we can rerank the results via comparing these overall scores.

4.4. Relationship Graph searching and maintaining Algorithms

Table 1 shows the data structures used by these algorithms. A file node is a diffused file node, which means that the searching for its neighbored file nodes have been done, and means there are more than two paths link the seeds and this file node.

4.4.1. Relationship Graph Maintaining Algorithm

In PROMES, when the client closes a process or an application, the monitor program can collect the provenance of the files used by the process or the application. Then, the analysis program, running on the client-side, extracts the relationships between these files and sends the relevance information to MDS.

When a MDS receives a section of relationship information, commonly including tens to hundreds of files, it will activate the relationship graph maintaining module to process the relationship information. In the view of the relationship graph, each line in the information is an edge, so, the program firstly judges whether the two nodes on the edge is a new file node or not. If at least one node is a new node, the edge will be a new edge, and there is a new relationship between these two file nodes. If two nodes are existed, it is uncertain and thus it needs to search this edge. The pseudo-code for the relationship graph maintaining algorithm is shown in Figure 11.

In the algorithm, we use Node_Table to store the information of files, such as files' URI and files' ID defined by MDS, and use Edge_Table to store the information about edge, that is the relationship information. If users have not stored the files' ID on the client-side, the MDS will have to search the files' ID via files' URI in Node_Table. So, the Node_Table needs very high performance. Because the URI is unique globally, we can use key/value pairs to store files' ID and files' URIs.

```
Processing_Two_Old_Nodes(u,v)
if In_Edge_Table(u,v) then
    /*If the edge(u,v) is exsting*/
    Update_Edge_Table(u,v,time)
    /*update the information of edge(u,v) via the formula(1)
    */
else
    Insert_New_Edge(u,v,time)
    u.connection ++
    v.connection ++
    Update_Node_Table(u,u.connection)
    /*we use the connection of the node u to compute the
    node prestige score */
    Update_Node_Table(v,v.connection)
end if
```

Figure 9: Algorithm for processing the relationship between two old nodes.

```
Processing_Two_New_Nodes(u,v)
/*the node u and v are two new nodes*/
Inseart_Node_Table(u,u.connection=1)
Inseart_Node_Table(v,v.connection=1)
Insert_New_Edge(u,v,time)
```

Figure 10: Algorithm for processing the relationship between two new nodes.

```
Relationship_Into_Graph(E)
while (E $\neq$ NULL ) do
    E $\rightarrow$ (u,v,time)
    /*the file node u and v produce relationship at time*/
    if In_Node_Table(u) then
        /*If the node u is exsting*/
        if In_Node_Table(v) then
            Processing_Two_Old_Nodes(u,v)
        else
            Inseart_Node_Table(v,v.connection=1)
            Insert_New_Edge(u,v,time)
        end if
    else
        if In_Node_Table(v) then
            Inseart_Node_Table(u,u.connection=1)
            Insert_New_Edge(u,v,time)
            v. connection ++
            Update_Node_Table(v,v.connection)
        else
            /*the node u and v all are new nodes*/
            Processing_Two_New_Nodes(u,v)
        end if
    end if
end while
```

Figure 11: Algorithm for processing the relationship come from the client-side.

4.4.2. Relationship Graph Searching Algorithm

The query process is actually to spread around from some seed nodes on the relationship graph. The pseudocode for the ANN searching algorithm is shown in Figure 13. We create two sets, one of the sets records the file nodes which have been searched for the neighbored file nodes, and other one records the file nodes which are the result candidates and do not know whether their neighbored file nodes are. Because the searching begins from the file node which is the most related to query node, the second set will be sorted frequently.

5. Performance Evaluation

In order to verify our algorithm in some widely accepted application scenarios, we evaluated our prototype PROMES on some representative filesystem access traces, *HP* traces [29] and some other traces [30] collected via Karma [31]. Because the collection of provenance is not the focus of this paper, we adopt these widely used traces to verify the performance of our system. Although these traces are not specialized provenance traces, we can extract provenance information on them. In current applications, an MDS can store and manage millions of files' metadata, these traces don't record so many files' metadata information, and we reasonably scale up these traces.

Our PROMES prototype is implemented on an MDS cluster with 50 servers. Each server includes an Intel Core 2 Duo CPU, 2GB of main memory, 250GB ST3250310AS disk, and a high-bandwidth network, the access speed ratio of memory and disk is about 20-50. In order to store more metadata on memory, we reduce some fields of metadata and the size of every file metadata is 200 bytes. Each MDS can manage and store $100 - 500$ millions of files' metadata and there are about one percent of files' metadata will be stored in memory.

By analyzing the trace files with a range of long time, some correlation-aware metadata, including some static attributions and some dynamic attributions, can be obtained, such as, files' size, files' last update time, the amount of files' data to be read, the number of operations. The correlation-aware metadata index tree can be built via R-tree from a single MDS upper to an MDS cluster. There is a relationship graph in every MDS's memory. Each graph records the relaionships of files which are stored in the local MDS. Since there are few query requests in these traces, we execute some read or write operations as the query commands from the later traces. We compare PROMES with the metadata search system with a traditional index tree. This traditional index method uses an R-tree structure to index the metadata and adopts a LRU algorithm to replace the metadata stored in memory, which is commonly used by cloud storage systems.

5.1. Overhead of Relationship Graph Building

Overhead of Relationship Graph Building. In our PROMES, we use two tables which are called *Node_Table* and *Edge_Table*, to store the relationship graph in the memory. From Figure 14(a), we can discover that the time

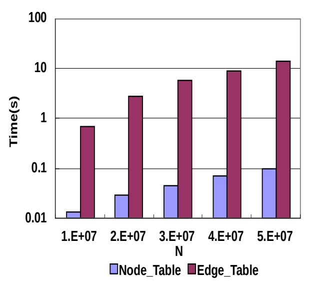
```
ANN_Relationship_Graph_Search(ad_i,query_time,min_dist)
  open Edge_Table
  create the set of results $X$ = NULL
  while (In_Edge_Table(u,ad_i )) do
    /*When there are a node u connected to ad_i,it will test whither the node u is a result node*/
    Compute_Edge_Weight(u,ad_i,query_time)
    /*according to the formula(1)*/
    $R(q\,u)$ = Compute_Overall_Score(q,u)
    /*according to the formula(5)*/
    if ($R(q\,u) \leq$ min_dist ) then
      u join $X$
    end if
  end while
  return $X$
```

Figure 12: Algorithm for ANN search on the relationship graph.

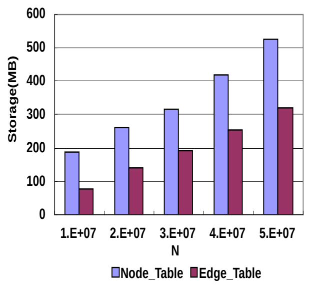
```
ANN_Query(S, min_dist, query_time)
  create $D$ and $AD$
  S join $AD$
  while ($AD \neq$ NULL and $ad_{head} \leq$ min_dist ) do
    create result set $X$
    $X$=ANN_Relationship_Graph_Search($ad_{head}$,query_time,min_dist)

    while ($X \neq$ NULL) do
      $x_i \in X$
      if ($x_i == d_j$ and $d_j \in D$ ) then
        /*If the $x_i$ had been searching for neighborhood nodes*/
        $R\left(q\,d_j\right)=R\left(q\,d_j\right) + R\left(q\,x_i\right)$
      else
        $x_i$ join $AD$
      end if
    end while
    $ad_{head}$ join $D$
    Rank($AD$)
    /*Sorting $AD$ is for the relevance score between $ad_{head}$ and $q$ is biggest*/
    Rank($D$)
    /*Sorting $D$ is for rerank the set of results*/
  end while
```

Figure 13: Algorithm for processing ANN search request with some seeds on the relationship graph.



(a) Time overhead.



(b) Space overhead.

Figure 14: The overhead of building the relationship graph

overhead of building these two tables grows linearly with the number of files or edges. Because we adopt key/value method, adding file nodes or edges of relationship graph is very rapid. It is acceptable that the tens of millions of file nodes' information or edges' information can be added to the relationship graph in less than 20 seconds.

According to the algorithm shown in Figure 11, the operation of adding an edge needs some comparisons and is more complex than the operation of adding a file node. Because the *Node_Table* stores the mapping relationship between the file' URI and the file' ID and the URI maybe are hundreds of characters, the size of *Node_Table* is larger than *Edge_Table* as shown on Figure 14(b). However, the storage overhead of building these two tables also grows linearly with the number of files or edges like time overhead. The relationship graph with 50 millions of files and edges can be stored in about 800MB of main memory, so, this low-cost index structure can improve high-performance services.

System Scalability We evaluate the scalability of our PROMES through *HP* traces and some traces collected via Karma. As shown in Table 2, we find that different application scenarios have different average number of edges, and the gap between them is enormous. The average number of edges per node in *HP* traces is about 50 due to the processes operate lots of files in few seconds. With the increasing number of nodes, the average number of edges per node is gradually stabilized in *HP* traces. In the other traces, the average number of edges per node is no more than 3, and some of them show a downward trend with the number of nodes increases. The major reason is that the applications of files have very strong spatial locality in cloud storage services, and this feature ensures that our system owns good scalability.

5.2. Overhead of Metadata Searching

The R-tree has been the most popular spatial index structure, and it is more suitable for indexing multi-dimensional attributes than B-tree [7, 17]. X-tree is a variant of the R-tree. When a node needs to be split, the X-tree increases the capacity of the node instead of splitting it. The highly skewed distributions of metadata values make supernodes not suitable for indexing file metadata. M-tree is a kind of tree data structures that are similar to R-trees and B-trees. KD-tree has been used to index metadata in Spyglass [1]. Hence, we compare PROMES with two baseline systems. The first one is a simple R-tree-based index structure that organizes each file based on its multi-dimensional attributes, denoted as R-tree. The second one is a popular index structure that uses a KD-tree to index each metadata attribute, denoted as KD-tree. Those two systems do not take into account index structure optimization.

Top k Query. Figure 15(a) shows the average speed of a top k query for *HP* traces in our PROMES can be up to 1-2 orders of magnitude faster than searching in R-tree and

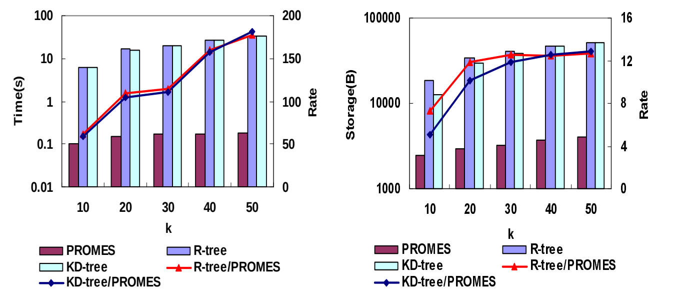
Table 2: The average number of edges per file node

NO.files	hp	animation-real	motif-real	ncfs-real	nam-wrf-real
1.0E+07	23.52	1.58	2.61	1.81	2.1
2.0E+07	48.51	1.385	2.609	1.79	2.05
3.0E+07	49.25	1.377	2.608	1.70	2.03
4.0E+07	49.387	1.463	2.448	1.61	2.02
5.0E+07	49.8	1.52	2.48	1.54	2

KD-tree. In the experiment, we find that the traditional index tree searching can hit more results on memory than PROMES, while, for the better query accuracy, this kind of traditional method has to compare vast amount of files' metadata in candidate subtrees, some of which might be stored on disk. PROMES doesn't need to access the disk until needing to return some of the final results stored on disk to the clients.

The Figure 15(b) shows the average space overhead of a top k query for *HP* traces. Space usage of top k query in our PROMES is 5×-12× lower than in the traditional index tree searching because the traditional index method requires more space to store the files' whole metadata of candidate subtrees. In PROMES, it is unnecessary to access the metadata of the files involved in the relationship graph searching. The final results can be obtained via comparing some edges in the relationship graph and do not need these files' metadata at all.

From Figure 15, we can find that, with the growth of k, the performance of PROMES become better and better. With increasing size of the search range, the traditional index will have to access more and more candidate subtrees. Therefore, our PROMES is more suitable for large k query.

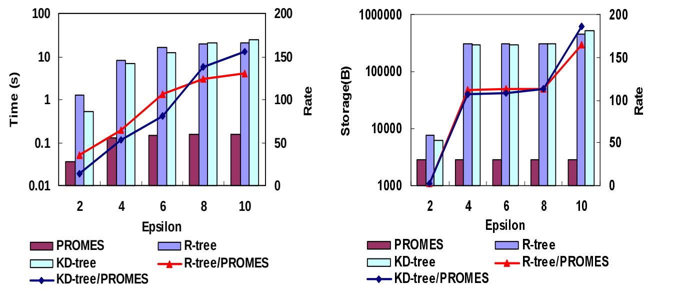


(a) Time overhead. (b) Space overhead.

Figure 15: The overhead of top k query

ANN Query. The Figure16(a) shows that the average speed of an ANN query for *HP* traces in our PROMES can be also up to 35×-130× times faster than searching in R-tree, and 14×-155× times faster than in KD-tree. This speedup rate is lower than top k query due to the distribution of file nodes around the query points, which is always very dense. When the value of ε becomes bigger, the ANN query needs to compare more subtrees and file nodes than the top k query. We also can observe that R-tree is more suitable for big ε than KD-tree.

The Figure 16(b) shows the space consumption of ANN query in our PROMES is 2×-185× lower than traditional index tree searching, the reason of this better speedup is same as top k query. When the value of ε is small, obtaining the file nodes of adjacent subtrees in traditional index tree is effortless. With the increasing of ε , obtaining the adjacent file nodes will become more and more difficult due to some of the nodes may be stored on the disk. PROMES only needs to accesses few file nodes on the disk. Hence, PROMES is more suitable for big ε.

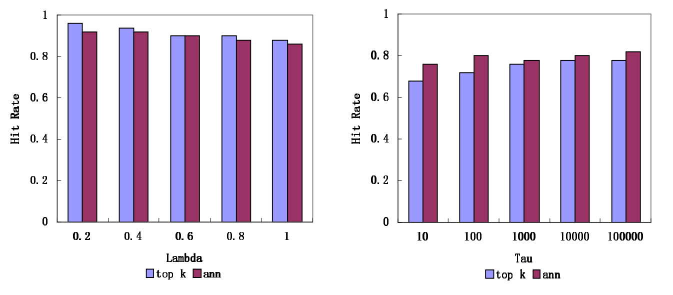


(a) Time overhead. (b) Space overhead.

Figure 16: The overhead of ANN query

5.3. *Query Accuracy and Two Parameters*

Because the two parameters, the time factor and the files' weight, are used to computer the relevance overall scores, we finally examine the relationship between query accuracy and these two parameters. The λ reflects the importance of nodes' weight in relevance overall score. As shown on Figure 17(a), PROMES ensures the query accuracy is more than 90%. With the increasing value of λ, the accuracy of queries is decreased. Hence, using $\lambda = 0.2$ is a good choice. The τ is a tradeoff between historical relationship and current relationship. From Figure 17(b), we can find that the hit rate is increased with the growth of τ, and choosing a bigger τ can obtain a higher hit rate. This characteristic also reflects that using provenance can efficiently improve metadata searching performance in cloud storage services.



(a) Query accuracy and Lambda. (b) Query accuracy and Tau.

Figure 17: Query accuracy.

6. Related Work

Currently, the state-of-the-art studies on the metadata search focus on using a high-performance metadata index tree to accelerate metadata search. Many researchers use popular data structure and grouping techniques on traditional multi-dimensional metadata via mining the relationship between files to organize metadata storage and to optimize the metadata search. Spyglass[1] adopts static subtree-based partitioning to divide total file system's namespace with versional index, and use KD-tree to take advantage of the skewed distributions of metadata. It focuses on how to create indexes with fast update. PROMES, unlike Spyglass, focuses on how to implement metadata searching with high accuracy and low latency. Smartstore[7] mines metadata semantic correlations which are not only from static attributions but also from dynamic attributions and uses R-trees to cluster metadata via semantic grouping scheme and imports Bloom filter [32] to accelerate metadata query. PROMES mines the files' correlation not only from the files' present attributions, but also from the files' attributions of history.

In order to provide high degree of concurrency with a high rate of file creation in one large directory, GIGA+[9] proposes a distributed indexing scheme. The characteristics of the index are asynchrony and eventual consistency, and it has a good scalability. This kind of distributed indexing has the inherent drawback of a single point of failure. In order to provide a better scalability, VSFS [3] uses consistent hashing to divide the whole index into small partitions, and uses the same way to place these partitions into the various index severs. The index type of every partition can be different, and it has high versatility and high compatibility. It is a feasible method to combine PROMES with GIGA+ or VSFS for improving metadata search performance. The concrete methods are using the GIGA+ or VSFS to index metadata and find seeds, and using PROMES's relationship graph searching to find the final query results.

In small-scale filesystem or personal filesystem, the use of provenance can be helpful to improve file searching performance. Through using a relationship graph made from some provenance, Shah et al.[14]designs a context-enhanced search architecture, which can add hit rates by reordering and extending the query results from the content-only search. Because they only collect the provenance from the local computer, the design is limited to optimize file search in the personal filesystem. In cloud storage services, this causality is not all-inclusive, and some potential causality and the temporal locality would be introduced into the causality in the relationship graph. PROMES not only improves the hit rate via searching the relationship graph, but also reduces the overhead of searching because PROMES doesn't store full-size index tree and only searches few index substrees to identify several files as seeds.

7. Conclusions

Metadata query becomes more and more important to improve the storage systems' QoS in current cloud environment. Conventional full-size index trees are not suitable for efficient metadata searching in cloud storage services. In this paper, we proposed a novel provenance based metadata-search system, called PROMES. PROMES finds the final results and doesn't need to access the disk except that the required metadata are stored on the disks. PROMES demonstrates the significant performance improvements in terms of metadata search, which can be up to 1-2 orders of magnitude than traditional index tree structure. Experimental results demonstrate that the use of provenance is efficient to improve metadata searching performance.

8. Acknowledgments

We are very grateful to anonymous reviewers for their valuable comments and suggestions. This work was supported in part by National Basic Research 973 Program of China under Grant 2011CB302301, and National Natural Science Foundation of China (NSFC) under Grant 61025008, 61173043 and 61232004, and National High Technology Research Program of China under Grant No.2013AA013203.

References

[1] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, E. L. Miller, Spyglass: fast, scalable metadata search for large-scale storage systems, in: Proccedings of the 7th conference on File and storage technologies, FAST'09, USENIX, Berkeley,CA,USA, 2009, pp. 153–166.

[2] T. S. Somasundaram, K. Govindarajan, Cloudrb: A framework for scheduling and managing high-performance computing (hpc) applications in science cloud, Future Generation Computer Systems 34 (2014) 47–65.

[3] L. Xu, Z. Huang, H. Jiang, L. Tian, D. Swanson, Vsfs: A versatile searchable file system for hpc analytics, Tech. rep., University of Nebraska-Lincoln (2013).

[4] A. Leung, M. Shao, T. Bisson, S. Pasupathy, E. Miller, High-performance metadata indexing and search in petascale data storage systems, Journal of Physics: Conference Series 125 (2008) 012069.

[5] Q. Xu, R. V. Arumugam, K. L. Yong, S. Mahadevan, Drop: Facilitating distributed metadata management in eb-scale storage systems, in: Proccedings of the 29th Symposium on Mass Storage Systems and Technologies, MSST'13, IEEE, Long Beach,CA,USA, 2013, pp. 1–10.

[6] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, C. Maltzahn, Ceph: A scalable, high-performance distributed file system, in: Proccedings of the 7th Symposium on Operating Systems Design and Implementation,OSDI'06, USENIX, Long Beach,CA,USA, 2006, pp. 307–320.

[7] Y. Hua, H. Jiang, Y. Zhu, D. Feng, L. Tian, Smartstore: a new metadata organization paradigm with semantic-awareness for next-generation file systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'09, IEEE, Portland,OR,USA, 2009, pp. 1–12.

[8] A. Parker-Wood, C. Strong, E. L. Miller, D. D. Long, Security aware partitioning for efficient file system search, in: Proccedings of the 26th Symposium on Mass Storage Systems and Technologies, MSST'10, IEEE, Lake Tahoe,NV,USA, 2010, pp. 1–14.

[9] S. Patil, G. A. Gibson, Scale and concurrency of giga+: File system directories with millions of files, in: Proccedings of the 7th conference on File and storage technologies, FAST'11, USENIX, San Jose,CA,USA, 2011, pp. 177–190.

[10] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: Proccedings of the Special Interest Group On

Management Of Data, SIGMOD'84, ACM, Boston,MA,USA, 1984, pp. 47–57.

[11] K. Zhou, Q. Hou, R. Wang, B. Guo, Real-time kd-tree construction on graphics hardware, ACM Transactions on Graphics(TOG) 27 (126) (2008) 126.

[12] T. Blanc-Brude, D. L. Scapin, What do people recall about their documents?: implications for desktop search tools, in: Proceedings of the 12th international conference on Intelligent user interfaces,IUI'07, ACM, Honolulu,HI,USA, 2007, pp. 102–111.

[13] K. Yamamoto, T. Kuriyama, H. Shigemori, I. Kuramoto, Y. Tsujino, M. Minakuchi, Provenance based retrieval: File retrieval system using history of moving and editing in user experience, in: Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference,COMPSAC'11, IEEE, Munich,Germany, 2011, pp. 618–625.

[14] S. Shah, C. A. Soules, G. R. Ganger, B. D. Noble, Using provenance to aid in personal file search, in: Proceedings of the 2007 USENIX Annual Technical Conference,ATC'07, USENIX, Santa Clara,CA,USA, 2007, pp. 171–184.

[15] A. Parker-Wood, D. D. E. Long, E. L. Miller, M. Seltzer, D. Tunkelang, Making sense of file systems through provenance and rich metadata, Tech. Rep. UCSC-SSRC-12-01, University of California, Santa Cruz (Mar. 2012).

[16] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, W. Hsu, Characteristics of backup workloads in production systems, in: Proceedings of the 10th USENIX Conference on File and Storage Technologies,FAST'12, USENIX, San Jose,CA,USA, 2012, pp. 33–48.

[17] A. Parker-Wood, B. MADDEN, M. McThrow, D. D. Long, Examining extended and scientific metadata for scalable index designs, Tech. rep., UCSC-SSRC-12-07 (2012).

[18] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, M. I. Seltzer, Provenance-aware storage systems, in: Proceedings of the 2006 USENIX Annual Technical Conference,ATC'06, USENIX, Boston,MA,USA, 2006, pp. 43–56.

[19] K. Skałkowski, R. Słota, D. Król, J. Kitowski, Qos-based storage resources provisioning for grid applications, Future Generation Computer Systems 29 (3) (2013) 713–727.

[20] J. Kim, A. Can, Characterizing queries in different search tasks, in: Proceedings of the 45th Hawaii International Conference on Systems Science,HICSS'12, IEEE, Grand Wailea,Maui,HI,USA, 2012, pp. 1697–1706.

[21] S. Jiang, X. Ding, F. Chen, E. Tan, X. Zhang, Dulo: an effective buffer cache management scheme to exploit both temporal and spatial locality, in: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies,FAST'05, USENIX, San Francisco,CA,USA, 2005, pp. 101–114.

[22] D. Zhan, H. Jiang, S. C. Seth, CLU: Co-optimizing Locality and Utility in Thread-Aware Capacity Management for Shared Last Level Caches, IEEE Transactions on Computers 63 (7) (2014) 1656–1667.

[23] P. Missier, C. Goble, Workflows to open provenance graphs, round-trip, Future Generation Computer Systems 27 (6) (2011) 812–819.

[24] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, et al., The open provenance model core specification (v1. 1), Future Generation Computer Systems 27 (6) (2011) 743–756.

[25] K.-K. Muniswamy-Reddy, P. Macko, M. I. Seltzer, Provenance for the cloud, in: Proceedings of the 8th USENIX Conference on File and Storage Technologies,FAST'10, USENIX, San Jose,CA,USA, 2010, pp. 197–210.

[26] S. Stumpf, J. Herlocker, Tasktracer: Enhancing personal information management through machine learning, in: Proceedings of the 29th SIGIR Workshop on Personal Information Management,PIM'06, ACM, Seattle,WA,USA, 2006, pp. 10–11.

[27] J. Frew, D. Metzger, P. Slaughter, Automatic capture and reconstruction of computational provenance, Journal of Concurrency and Computation: Practice and Experience 20 (5) (2008) 485–496.

[28] U. Braun, S. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, M. I. Seltzer, Issues in automatic provenance collection, in: Proceedings of the international conference on Provenance and Annotation of Data,IPAW'06, Springer, Chicago,IL,USA, 2006, pp. 171–183.

[29] E. Riedel, M. Kallahalla, R. Swaminathan, A framework for evaluating storage system security, in: Proceedings of the USENIX Conference on File and Storage Technologies,FAST'02, USENIX, Monterey,CA,USA, 2002, pp. 15–30.

[30] Y.-W. Cheah, B. Plale, J. Kendall-Morwick, D. Leake, L. Ramakrishnan, A noisy 10gb provenance database, in: Proceedings of the 9th International Conference on Business Process Management,BPM'11, Springer, Clermont-Ferrand,France, 2011, pp. 370–381.

[31] Y. L. Simmhan, B. Plale, D. Gannon, S. Marru, Performance evaluation of the karma provenance framework for scientific workflows, in: Proceedings of the international conference on Provenance and Annotation of Data,IPAW'06, Springer, Chicago,IL,USA, 2006, pp. 222–236.

[32] Y. Zhang, L. Liu, Distance-aware bloom filters: Enabling collaborative search for efficient resource discovery, Future Generation Computer Systems 29 (6) (2013) 1621–1630.