Index Storage Mechanism of MongoDB Based on K-center PAM Optimization Algorithm

Yunke Liang & Anping Xiong*
School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, China

ABSTRACT: In MongoDB database, data objects are queried by means of the index. Under the pressure of mass data, all of its index could not be stored in memory, which is bound to affect the system query performance. For this issue, this paper proposes a layered index storage mechanism based on K-center PAM optimization algorithm, clusters hot and cold index through the index of data objects, and uses a layered index storage structure based on Redis cache to reduce times to access disk during data query and improve data query performance. Through experimental comparison with the original index mechanism of MongoDB, this paper verifies that K-center PAM optimization index mechanism can enhance the original system query performance.

Keywords: MongoDB; mass data; hot and cold index; Redis; layered storage

1 INTRODUCTION

With the development and application of Internet Web2.0 technology, the amount of data generated by the network presents an exponential growth. How to efficiently store and process mass data has become a problem to be solved urgently. Non-relational database based on cloud computing platform, that is NoSQL (Not Only SQL) system has become an efficient solution to storage and processing of mass data, and has obtained widespread concern in industry, such as Google’s BigTable, Amazon’s Dynamo, 10gen’s MongoDB and the like. MongoDB uses BSON document as the format and syntax of internal storage, in order to be more intuitive, easy to understand and master, so it becomes a widely used document database application system, and also becomes one of research hotspots in industry.

MongoDB supports fast data query through establishment of B+ tree index. Logically, MongoDB is “keep all index fit in memory”. However, under the pressure of mass data, all of its index could not be stored in memory. For data access, there is also a need to traverse all indexes in the master (server used for storage of indexes), so that the efficiency of data access is reduced. The data access generally presents fixed 80/20 data distribution feature \(^1\); that is, 20% of the data are frequently accessed and used, which are called as ”hot data”; 80% of the data are rarely accessed, which are called as “cold data”. To further optimize MongoDB index query performance, this paper researches MongoDB and its index storage mechanism, and proposes a layered index storage mechanism based on K-center clustering index.

2 MONGODB STORAGE MECHANISM

Unique logical storage and physical storage structure of MongoDB \(^2\) and memory mapping storage engine and index mechanism of the system are an important guarantee for MongoDB to massive storage of unstructured data and completion of efficient data processing.

2.1 MongoDB storage structure

MongoDB assigns a file set with gradual increase in the size, and each database is comprised by .ns file and several data files, of which .ns file records a number of collection namespace and index space, and data files store all collection data. Internal data structure of MongoDB is shown in Figure 1.
Among them, a collection namespace has multiple data Extents, and each Extent is a block, which saves more Documents with BSON document format. Extent and Document adopt two-way linked list structure. The collection namespace is achieved by Disklocation data structure, which includes fileNo and file offset.

Index Namespace uses B-tree structure, and stores pointers at root nodes pointed to B tree.

2.2 Relevant research

MongoDB index is almost the same with a relational database, and the vast majority of usage and skills of the relational optimization database are equally applicable to MongoDB. The reference [3] establishes a lexicon via stem reduction to achieve MongoDB flashback tabulate based on the establishment of reverse index of MongoDB, which is used to construct reverse index table of full-text retrieval, but is only applicable to full-text query of mass data; the reference [4] researches the optimization scheme of MongoDB fairway database based on the fairway data modeling of MongoDB and conventional index and geographical index, in order to improve the geographical index query efficiency in MongoDB system processing; in reference [5], Xu Bing, et al improve the index management system of MongoDB based on massive heterogeneous historical data querying, and effectively manage and maintain various types of indexes, and also improve query efficiency. However, without considering the situation of mass data, all of indexes could not be stored in memory, thus ignoring the index storage problem; the reference [6] decomposes the property of value based on key-value data model under cloud environment, and changes the way of index query, and proposes a new index structure, and uses global and local index modes to improve the system throughput and data query performance, but fails to solve the problem of index storage.

In MongoDB, the index is very efficient while extracting smaller sub data set. If the result set accounts for a larger proportion in the original set, the velocity of index will be slower, because the index needs to look for once: look for the appropriate documentation according to the index pointer. However, full table scan only needs to look for twice: look for index entry and look for the index entry is the key [7]. In most of MongoDB application system, the users ignore that all of indexes could not be stored in memory. This paper proposes to distinguish the indexes into hot and cold cluster, and always stores hot indexes in memory, and cold indexes on the hard disk, and constantly periodically updates the hot and cold indexes in memory and on the hard disk to optimize storage order of hot and cold indexes. It not only facilitates to find the index entry, but also improves the efficiency of index query, and optimizes the memory storage space of MongoDB application system.

3 INDEX STORAGE MECHANISM BASED ON K-CENTER PAM OPTIMIZATION ALGORITHM

This section applies for K-center PAM optimization algorithm [8] to cluster hot and cold indexes (hereinafter referred to as PAM optimization algorithm), and adopts the layered storage structure to store high heat indexes in memory Redis cache, and low heat indexes in the persistent storage layer, and updates memory indexes according to certain strategies, in order to ensure that common indexes are stored in the memory cache and improve query efficiency.

3.1 Data object index clustering based on k-center PAM optimization algorithm

PAM optimization algorithm picks up actual data objects to represent the cluster, which is also the hot and cold data object index partition, and the algorithm eventually divides all index data into two clusters, namely hot index cluster and cold index cluster. One representative object is used for each cluster, and the rest of each cluster is assigned to the clusters with the most similar representative objects. Thus, the division method is to divide based on the principle of minimizing all objects p and a sum of dissimilarity between its corresponding representative objects. This paper applies for the PAM (Partioning Around Medoid) optimization algorithm to cluster the hot and cold data object indexes.

First, the heat is used to measure the hot and cold degree of the index data objects accessed. When an index is more frequently used in a cycle time, it indicates that the index with a higher heat is a hot index, and vice versa, cold index. This paper defines the computing equation of the heat of a data index p as the equation (1):

\[ \text{heat} = \frac{1}{n} \sum_{i=1}^{n} \text{heat}_i \]

\[ \text{heat}_i = \sum_{j=1}^{n} |p - q_j| \]

\[ \text{heat} = \sum_{i=1}^{n} \text{heat}_i \]

where \( p \) is the data object index, \( q_j \) is the representative object of \( p \), and \( n \) is the number of representative objects.

The equation (1) can be used to compute the heat of all data objects in the system, and then use the equation (2) to cluster them:

\[ \text{cluster} = \arg \min_{c} \sum_{i=1}^{n} \min_{j \in c} |p - q_j| \]

where \( c \) is the cluster of data objects, and \( j \) is the representative object of \( p \). This equation (2) can be used to minimize the sum of the dissimilarity between each data object and its representative object in the same cluster.

After obtaining the heat values of all data objects, the algorithm then uses the equation (3) to cluster them:

\[ \text{cluster} = \arg \min_{c} \sum_{i=1}^{n} \sum_{j \in c} |p - q_j| \]

where \( c \) is the cluster of data objects, and \( j \) is the representative object of each cluster. This equation (3) can be used to minimize the sum of the dissimilarity between each data object and its representative object in the same cluster.

After obtaining the heat values of all data objects, the algorithm then uses the equation (4) to cluster them:

\[ \text{cluster} = \arg \min_{c} \sum_{i=1}^{n} \sum_{j \in c} |p - q_j| \]

where \( c \) is the cluster of data objects, and \( j \) is the representative object of each cluster. This equation (4) can be used to minimize the sum of the dissimilarity between each data object and its representative object in the same cluster.

The algorithm then uses the equation (5) to cluster them:

\[ \text{cluster} = \arg \min_{c} \sum_{i=1}^{n} \sum_{j \in c} |p - q_j| \]

where \( c \) is the cluster of data objects, and \( j \) is the representative object of each cluster. This equation (5) can be used to minimize the sum of the dissimilarity between each data object and its representative object in the same cluster.
\[
HOT_p = a \cdot \frac{n \cdot S}{t} = \frac{S}{s}
\]

Where, \(t\) represents the cycle time used to calculate heat; \(n\) represents the access number of a data object index within the heat cycle time. When \(0 < a < 1\), it indicates the weight of data object index in \(t\) time.

In MongoDB application system, there are unique index, sparse index, multi-value index and many other types of index, and the index data types are also different in different application systems, and the weight of index is different, for example, the weight of data index established for emergency news data is higher, so there is a need to define the size of weight according to the specific scenario. \(S\) represents the size of system memory; \(s\) represents the size of memory of each index entry. Due to limited space size of system memory, a hot index collection changes with state of system operation, and memory is even unable to accommodate all hot index entries, and some index items in the hot index set are unable to be clustered in memory. Considering this situation, when the heat of index is almost the same, the index entries sharing a small memory and the indexes with more index entries are stored in memory as much as possible, while the index entries sharing a large memory and the indexes with less index entries are stored in the persistent storage layer.

For MongoDB index attribute list, we add the attribute “access number (n)”, and add the index record function, in order to facilitate to record access when the index is used, and automatically add 1 for “access number (n)”, and reset “access number (n)” as 0 at an interval of every fixed heat cycle time \((t)\).

Equation (1) calculates and stores the heat value of each index. The process of server will use PAM optimization algorithm at an interval of every heat cycle time \((t)\) to complete hot and cold index clustering according to \(HOT\) value of the index, and store the clustered hot index in Redis cache, and cold index on the hard disk.

This paper defines PAM optimization algorithm based on heat value, and the equation (2) is defined as follows:

\[
E_{O_i} = \sum_{k=1}^{k} \sum_{p \in C_i} \text{dist}(HOT_p, HOT_{O_i})
\]

Here, \(E_{O_i}\) is a sum of absolute error of all indexes \(p\) in the data set \(C_i\) and representative index object \(O_i\) of all index collections; \(k = 2\), indicating that the cluster of this paper has two categories-cold index and hot index. Here, PAM optimization algorithm is to select any other non-representative index objects as the representative index object \(O_i\), and make dynamic cycle to look for the best central point of clustering cluster \(O_i\), based on the principle of minimizing a sum of absolute error, and cluster the index objects to the corresponding cluster. Faced with MongoDB of mass data, at the initial phase of the system, the memory will burden huge pressure, so there is a need to start PAM optimization algorithm to cluster hot and cold data indexes when the system is initialized, in order to improve the efficiency of system user query and user’s read-write operation as much as possible.

3.2 Redis-based storage strategy

3.2.1 Layered storage structure
To further enhance Mongodb data query efficiency, with constant changes of hot and cold indexes in memory, in order to better store and manage hot index data in memory, the mechanism introduces Redis as cache\(^9\), and uses the layered index storage structure to store hot data index objects obtained based on the section 3.1 in memory, and store cold data index objects with a low amount of access in the persistent storage layer, that is, on the hard disk, using the layered storage structure, as shown in Figure 2:

![Figure 2. Logical structure of layered storage of MongoDB mass data.](Image)

On implementation, hot indexes are stored in Redis memory database, and Redis automatically completes the above rapid query process, so it is quite efficient.

3.2.2 Redis layered storage strategy
Multiple hash slots are built in Redis cluster on implementation. When key-value is required to being set in Redis cluster, Redis first calculates a result of key by the use of CRC16 algorithm, and Redis maps hash slots to different nodes based on the principle of an approximately equal number of nodes. The advantage of using hash slots is to easily add or remove nodes, which is applicable for constant change or migration of hot and cold index data in memory.

In Redis, the maximum memory size, server. maxmemory is allowed to be set by the user. Therefore, the memory size will also affect the heat of index, which can also re-cluster hot and cold indexes through the fixed heat cycle time \((t)\). In this case, there is a need to restore index objects according to the latest clustering results. Storage strategies are shown in Figure 3.
3.3 Data query process

After the use of mechanism of this paper, the query process of MongoDB for any data object is shown in Figure 4.

In Figure 4, the user first submits a query request by the client, and queries hot indexes through querying engine and storage layer of hot indexes in memory. If the cache is not hit, then the request is forwarded to the persistent index storage layer based on MongoDB for retrieval. Therefore, the layered index storage strategy based on PAM optimization algorithm makes the vast majority of data query directly hit indexes in memory, instead of querying indexes with a hard disk, thereby reducing the access overhead of hard disk, and improving the overall query performance.

In summary, the mechanism not only considers the time distance of index access, but also considers the frequency of index access, and tries to ensure that the hot index data is in memory, and takes into account the factor of the memory size of index, and tries to store hot index entries in memory as much as possible, in order to optimize the query efficiency of optimized indexes, and ease the pressure on memory, and improve the user’s efficiency in data read-write operation.

4 EXPERIMENT AND ANALYSIS

For the original index query mechanism and improved index query mechanism of MongoDB, this paper carries out experimental comparison of the hit ratio and query response time of index in memory under the condition of mass data query.

4.1 Experimental environment

The experiment builds MongoDB cluster via ten sets of virtual machines, and each virtual machine owns the same hardware configuration: 4G memory and CPU frequency of 24GHz. Each virtual machine uses CentOS system, with the bandwidth of 100Mbps. MongoDB version is 2.0.2.

In this paper, we implement a big data benchmark test experiment of Brown University. Big data benchmark test of Brown University is a benchmark test proposed by Brown University, which aims at comparing with big data query performance of the relational database and Hadoop. We use the benchmark test to continuously query about 100,000 data sets at 1000 times, and the query is in line with Zipf distribution [10], in order to test the cache hit ratio and query response time under different index cache ratio.

4.2 Experimental results and analysis

Figure 5 shows the comparison with the hit ratio of index in memory of PAM (Partitioning Around Medoid) optimization algorithm indexing strategy and the original MongoDB index query. Figure 6 shows the comparison of the query response time.

As can be seen from Figure 5, the size of memory space is increased, and hit rate of index in memory is improved. Especially when the memory size ratio is
0.4, the hit rate of PAM index optimization algorithm is higher than 10% of that of LRU algorithm of MongoDB, because PAM index optimization algorithm can more accurately record the hot and cold degree of the index, and cache hot indexes in memory. For big data application, cache space is restricted by memory, and PAM index optimization algorithm can give full play to its advantages when the cache ratio is not high, and improve query performance of big data.

![Figure 5. Cache hit rate.](image)

As can be seen from Figure 6, with the increase of cache space in memory, and constant enhancement of hit rate, the use of indexes of two kinds of algorithms to query time is decreased. Due to the advantage of PAM index optimization algorithm on the hit rate, the query response time is significantly better than the query performance of replacing strategies by the use of LRU.

![Figure 6. Query response time.](image)

The above experiment shows that, To introduce PAM optimization algorithm indexing strategy in MongoDB and establish hot and cold index elimination mechanism can dynamically guarantee to always distinguish between hot and cold indexes and layered storage, so that the system query performance has a certain improvement; meanwhile, with the limited application memory of big data, to introduce Redis as a memory cache can better manage hot index data in memory, and further optimize the system query response time to some extent [11].

5 CONCLUSION

Based on the analysis of MongoDB and its index storage mechanism, and hot and cold index PAM optimization algorithm indexing strategy, this paper makes cluster partition of the indexes of hot and cold data objects, and proposes a layered index storage structure based on Redis cache to respectively store hot and cold indexes. To make an experimental comparison with the original index query mechanism of MongoDB through establishment of an experimental platform, the results show that, the layered index storage mechanism based on K-center PAM optimization algorithm can improve the efficiency of system data query. Without considering the load factor of the machine, in the next step of work, it can be considered in PAM optimization algorithm to further enhance the system performance.

REFERENCES