A Dynamic Management Mechanism Based on H-Index Secondary Indexes

Heng Qin[1, a], Anping Xiong[2,b] and Yuan Tian[3,c]

[1,2,3] College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, ChongQing, China

[a]2903094420@qq.com, [b]1136367521@qq.com

Keywords: Hbase; Hindex; Secondary index; Heat value; Caching strategy

Abstract. Hbase is a kind of non-relational database providing scalable technology methods and platform for managing big data, but it can only query through the primary key. This paper proposes a secondary index mechanism based on the heat value which derives from H-Index, a secondary index scheme of Huawei. It updates data regularly by tagging heat values of data on the secondary index, then it gets heat values together and combines with secondary index read caching strategies to improve query performance. Experimental results show that the proposed mechanism can reduce the average response time for data query.

Introduction

The era of big data comes with the increasing amount of data. NoSQL receives extensive attention because of its scalable property and high-performance when dealing with big data. Currently, there are common NoSQL databases such as Redis, Leveldb, Mongodb, HBase, etc., in which Hbase, the Apache open source project, are most popular.

Hbase is a database based on key-value. The database itself uses only RowKey as the primary index, so users can just search based on RowKey. When users need to retrieve according to the Value, Hbase can’t offer it. Therefore, researchers began to study Hbase based on secondary indexes to meet the needs of retrieval according to theValue. At present, there are various programs of Hbase based on secondary indexes, such as ITHbase[1] based on MapReduce secondary indexes, Coprocessor[2] and Solr+Hbase[3], in which Coprocessor of Huawei is best known. It achieves more practical secondary indexes by cleverly constructing secondary indexes and combining with Coprocessor. However, it becomes a focus of concern about how to further improve the efficiency of the secondary index retrieval.

Related Work

Related Researches. The existing tasks to improve the efficiency of Hbase retrieval are: non-primary key index for select query optimization, non-primary key index for range query optimization and query based on memory cache and so on.

Non-Primary Key Index for Select Query Optimization. NGDATA's Hbase-indexer[3] provides a non-primary key index for Hbase. It periodically pushes data to the SolrCloud service clusters by the index servers, and it inquires by Solr service. Since the indexing mechanism will periodically update indexes asynchronously, it leads to real-time problems.

Non-Primary Key Index for Range Query Optimization. The Interval Index[4] is a kind of query index based on Hbase put forward by the university of Patras. It can be a good support for range queries, but both the space overhead and maintenance costs of the index are large. The CAS(Chinese Academy of Sciences) proposed a multi-dimensional range query index based on non-primary key index named CCIndex[5,6]. This program has high-performance, low space overhead and high availability. The Bloom Filter[7], a kind of binary vector data structure, has a good efficiency of time and space. Its role is to detect whether an element is in a collection.

Memory-based Caching Query. The TBF[8] proposed a cache replacement method based on solid-state drive combining the advantages of Bloom Filter with CLOCK. It reduces space overhead of replacing two metadata of the cache to some extent. One of the metadata is whether data record is cached, another is last visit information. But TBF has defects from CLOCK. It can't quantify frequency of recent accessed data and can't store accessed data outside a CLOCK period.

In the big data environment, due to Hbase retrieval is based on scan of StoreFile[9]. Therefore, retrieval needs to scan large amounts of data at the same time, generating unnecessary overhead. In the actual scenario, part of data would be accessed more frequently named thermal data.

This paper optimizes the existing secondary indexes by combining Thermal data, which concentrates the secondary indexes of heat data relatively, thus improving the retrieval efficiency. And it further improves the hit rate of the thermal data in line with the index cache management mechanism, improving the mechanism of secondary indexes.

Hindex of Hbase. Hbase clusters consist of multiple RegionServers. Every RegionServer manages multiple Regions. And every Region has multiple stores. Hbase will have a table be split into multiple Regions and assign them into different RegionSevers to manage.

Its basic structure is shown in Fig. 1.

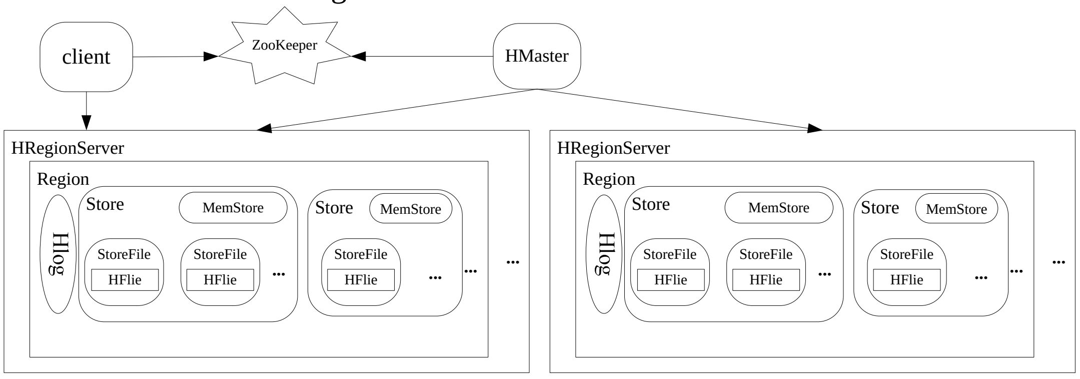


Fig. 1. Hbase structure

HIndex is a kind of secondary index mechanism based on Hbase that Huawei developed. In Hindex, the RowKey structure of secondary indexes is as follows: RegionStartkey-Value-RowKey. Since the data of Hbase is arranged in alphabetic order, this architecture makes the secondary indexes and metadata exist in the same Region certainly, in more detail, in the header of Region certainly, which improves efficiency when retrieving.

As shown in Table 1:

Table 1. Secondary index structure of Hindex

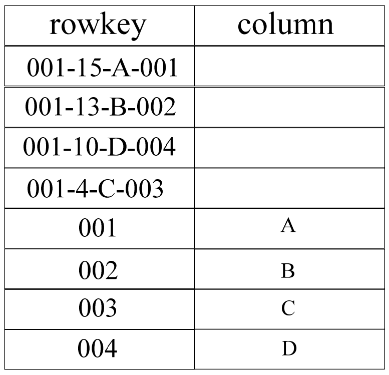
rowkey	column
001-A-001	
001-B-002	
001-C-003	
001-D-004	
001	A
002	B
003	C
004	D

The beginning of secondary indexes is RowKey Region Startkey: 001. According to Hbase collation, they are in a Region together with metadata, and in the header. The end of the RowKey of secondary indexes are the RowKey of metadata: 001,002, 003, 004. Thus, after finding necessary secondary indexes, it can determine the Rowkey of needed metadata directly according to the end information of the secondary index RowKey.

Secondary index mechanism based on Hindex

In order to improve search efficiency of HIndex secondary index in massive data, this paper proposes a kind of dynamic management mechanism of secondary indexes based on the thermal data on the basis of Hindex. The mechanism tags heat value of secondary indexes, and then periodically refreshes the secondary indexes based on the heat value. Finally, it coordinates with index cache strategies to improve the efficiency of data retrieval.

Tag of Heat Value of Secondary Index. Due to the need to tag the heat value of secondary index, the first step is to change RowKey structure of Hindex secondary indexes. The structure of RowKey secondary index is changed as: Region Startkey- heat value - Value-RowKey.
As shown in Fig. 2:



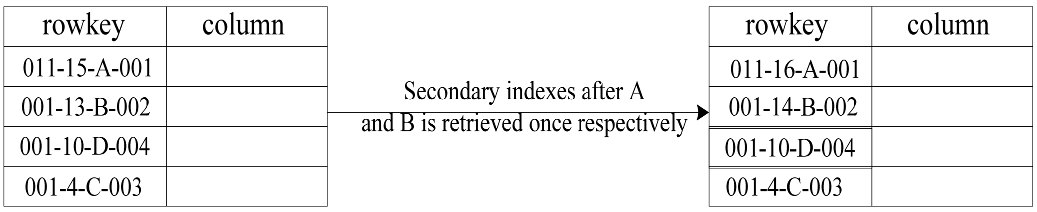
rowkey	column
001-15-A-001	
001-13-B-002	
001-10-D-004	
001-4-C-003	
001	A
002	B
003	C
004	D

Fig. 2. The secondary index RowKey

In case ‘001-13-B-002’, ‘001’ is the RegionStartkey, ‘13’ is the heat value. ‘B’ is the Value of the metadata, and ‘002’ is the RowKey of the metadata.

Hmaster storage file will retain data access records, and the system will scan Hmaster periodically to record heat value. The management of heat value is as follows: when new data is added to create a new secondary index, its heat value is initialized to 0. And when the data is retrieved, the corresponding heat value of secondary index plus one according to the recording. The system periodically clears all heat values of secondary indexes, which has two advantages: The first one is to prevent taking up too much storage space with the accumulation of heat value. The second is to make sure that the thermal data which the system records are always the most recent thermal data.

According to the secondary index structure in Fig. 2, the heat value tags can be obtained after retrieving once which is shown in Fig. 3:



rowkey	column
011-15-A-001	
001-13-B-002	
001-10-D-004	
001-4-C-003	

Secondary indexes after A
and B is retrieved once respectively

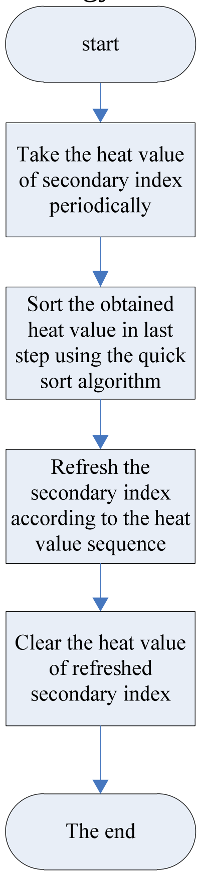
rowkey	column
011-16-A-001	
001-14-B-002	
001-10-D-004	
001-4-C-003	

Fig. 3. The tag of the heat value

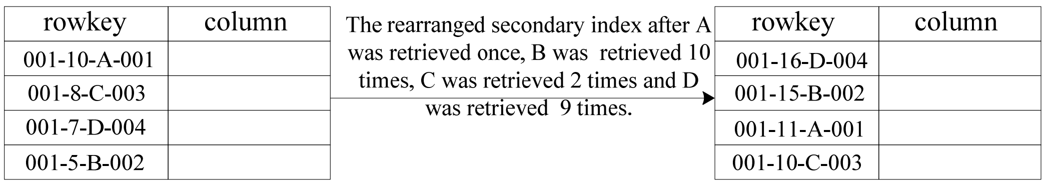
Before retrieval, the heat value of A, B, D, C is respectively 15, 13, 10, 4. After A, B are retrieved once, their heat value plus one respectively. In the end, the heat value of A, B, D, C is severally 16, 14, 10, 4.

Secondary Index Strategy Based on the Heat Data. The system can refresh and sort secondary indexes periodically. And the secondary indexes are sorted in descending order according to heat value, which ensures that the heat dataset are in the front of the secondary indexes. When a retrieval request arrives, Hbase locates the Region where the data reside, and determines which Stores they are in the Region. Then Hbase loads these Stores into Scanners to scan, and extracts the required data. Original secondary indexes of heat data are distributed. And it needs to load a large amount of Stores scanning when retrieving, which is precisely the most expensive link of retrieval. The thermal data

will relatively be concentrated after being sorted based on the heat value. When retrieval arrives, it only needs to load a small amount of Stores scanning which leads to high retrieval efficiency.
The algorithm based on secondary indexes strategy of heat value is shown in Fig. 4:


start
Take the heat value of secondary index periodically
Sort the obtained heat value in last step using the quick sort algorithm
Refresh the secondary index according to the heat value sequence
Clear the heat value of refreshed secondary index
The end
Fig. 4. The secondary index strategy

According to the tag of heat value in Fig. 3, the secondary indexes that are not cleared after refreshing are shown in Fig. 5.



rowkey	column
001-10-A-001	
001-8-C-003	
001-7-D-004	
001-5-B-002	

The rearranged secondary index after A was retrieved once, B was retrieved 10 times, C was retrieved 2 times and D was retrieved 9 times.

rowkey	column
001-16-D-004	
001-15-B-002	
001-11-A-001	
001-10-C-003	

Fig. 5. The secondary indexes based on heat values

Before retrieval, the heat values of A, C, D, B is respectively 10,8,7,5. After A, B, C, D is each retrieved 1, 10, 2, 9 times, the heat value of A, C, D, B is respectively 11,10,16,15. After sorting according to the heat value, the secondary index order is as follows: 001-16-D-004, 001-15-B-002, 001-11-A-001, 001-10-C-003.
Caching Strategy Based on Thermal Index. Cache management algorithm of original secondary index is generally LRU or LRU-K, in which the former one has poor performance on hot data hits, while the latter has large overhead.

The cache management policy of secondary index is based on the original LRU algorithm in this paper. The difference in this paper is that when the secondary index reorders, the system will clear the list that secondary index reads cache, and make the data with the maximum of the heat value in secondary index be the new list in the cache. In fact, this is a similar management method to LRU-K. The difference is that this paper has recorded the number of visits for each data on the secondary index, which is recorded in the cache in LRU-K. This will not only be close to LRU-K[10] hot hits, but also save cache overhead.

The algorithm of secondary index cache management strategy is shown in Fig. 6.

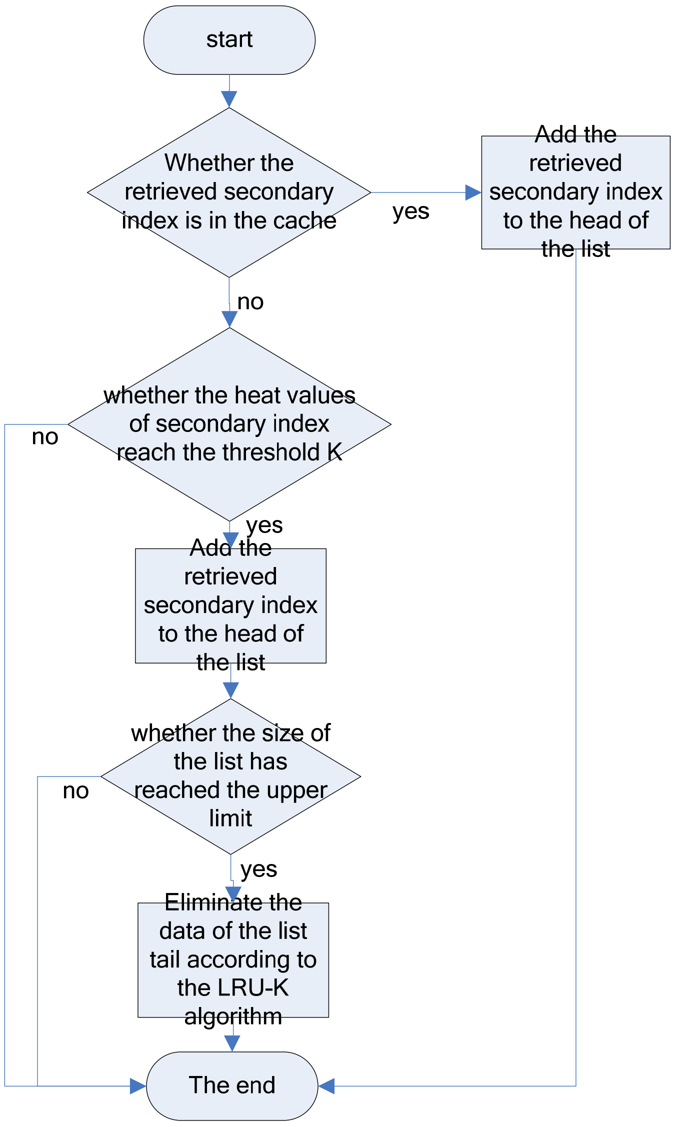
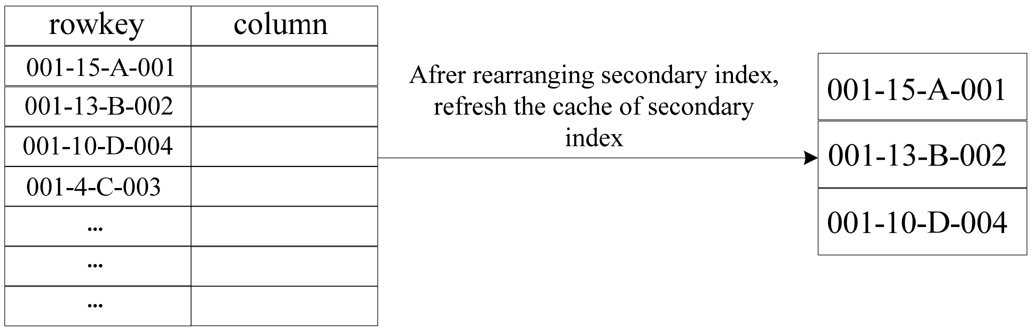

start
Whether the retrieved secondary index is in the cache
yes
Add the retrieved secondary index to the head of the list
no
whether the heat values of secondary index reach the threshold K
no
yes
Add the retrieved secondary index to the head of the list
whether the size of the list has reached the upper limit
no
yes
Eliminate the data of the list tail according to the LRU-K algorithm
The end

Fig. 6. The secondary index cache management policy

The secondary index of heat indexing strategy is shown in Fig. 7:



rowkey	column
001-15-A-001	
001-13-B-002	
001-10-D-004	
001-4-C-003	
...	
...	
...	

Afrer rearranging secondary index, refresh the cache of secondary index
001-15-A-001
001-13-B-002
001-10-D-004

Fig7. After cache refreshing and reordering of secondary index, the top four is A, B, D and C. If the cache list just takes the top three data according to heat value of secondary index, the new cache list is 001-15-A-001,001-13-B-002 and 001-10-D-004.

Experimental Results and Analysis

To verify the validity of the proposed indexing mechanism, this paper builds the experimental environment, and uses the call data of mobile Internet users as the test data, comparing retrieval efficiency and cache hit rate with the secondary index of Huawei.

Experimental Environment. In this paper, the experimental environment is built on distributed clusters of Hadoop2.4.0.The cluster server configuration of Zookeeper and RegionSever is shown in Table 2. The version of Hbase is 0.94.8. And Hindex is built on Hbase.

Table 2. Clusters Configuration

Cluster Components	configuration
Zookeeper	CPU：Intel 2.40GHz 24 cores RAM：24G Hard disk：300G
Master	CPU：Intel 2.40GHz 24 cores RAM：24G Hard disk：300G
RegionSever	CPU：Intel 2.20GHz 24 cores RAM：64G Hard disk：30T

Experimental data is call data of mobile internet users which is around 1.5 billion items. First, take 1 billion data of them to Hbase, and use Hindex and secondary index mechanism in this paper to generate secondary indexes respectively. According to the secondary index order of HIndex, the experiment generates three kinds of data: centralized retrieved data, relative hash retrieved data and complete hash retrieved data. Then this paper has the generating data be retrieved at HIndex and the secondary index mechanism herein, recording the query response time and cache hit rate.

Experimental Results and Analysis. According to the above experiment, the comparison chart according to query response time is shown in Fig. 8 and the cache hit rate between Hindex and the retrieval mechanism proposed in this paper is shown in Fig. 9:

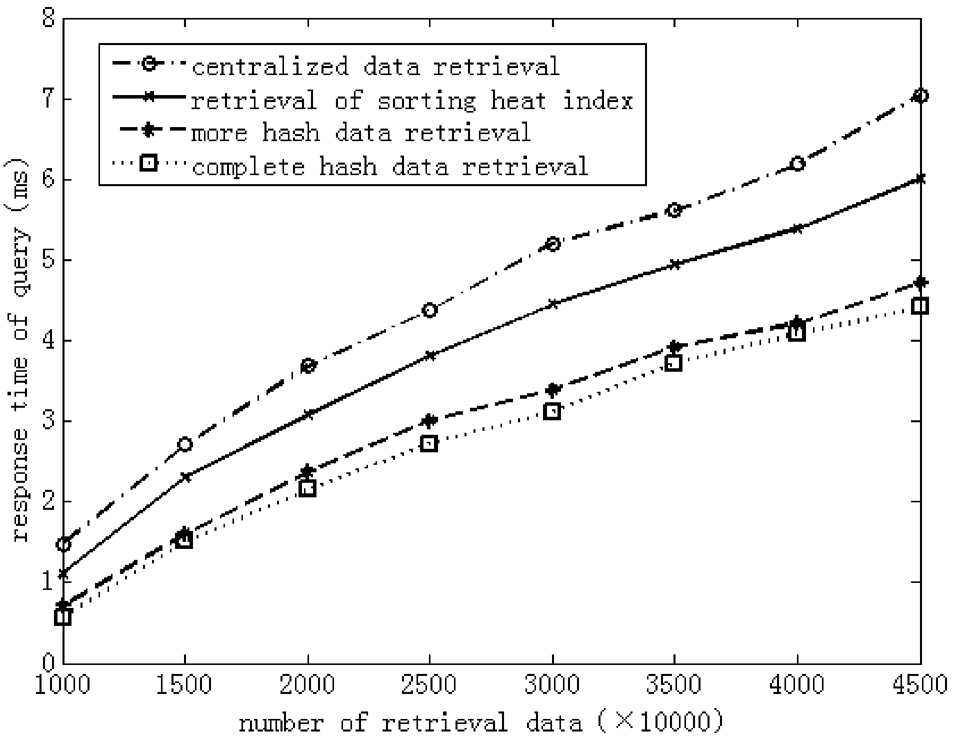


Fig. 8 Comparison chart according to query response time

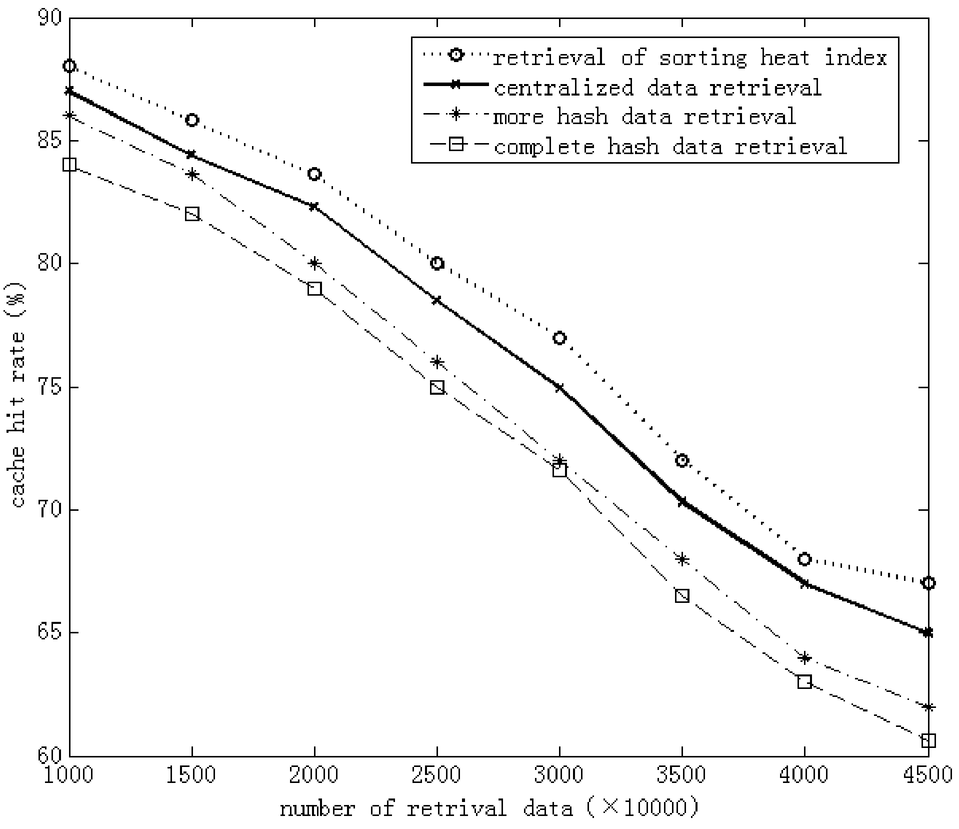


Fig. 9 Comparison chart according to cache hit rate

As can be seen from Fig. 8, the retrieval response time of heat index sorting is slightly better than retrieval response time of Hindex centralized data. And compared with Hindex, the mechanism in this paper improves retrieval efficiency significantly on more relative hash data and complete hash data.

As can be seen from Fig. 9, the cache hit rate of the heat sorting index is slightly better than Hindex centralized data. And compared with Hindex, the mechanism in this paper improves the cache hit rate significantly on more relative hash data and complete hash data.

The above results are in line with expectations. After heat index sorting, original distributed thermal data are put together. Retrieval response time with mechanism in this paper is similar to Hindex centralized data retrieval. The caching mechanism in this paper makes the data with maximum retrieval times recently stored in the cache, improving the cache hit rate significantly.

Conclusions

HIndex provides content-based retrieval by secondary index. However when the heat data is distributed, corresponding query response time is longer. This paper proposes a kind of secondary index mechanism based on analyzing Hbase secondary index and its cache mechanism. It concentrates the thermal data secondary index and improves index cache hit rate by marking the heat data indexes and using management strategies of secondary index based on heat value and cache strategies based on heat index. And it also reduces query response time and improves query efficiency of Hbase based on secondary index.

References

[1] Chang C R, Hsieh M J, Wu J J, et al. HSQL: a highly scalable cloud database for multi-user query processing[C]//Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. Honolulu, HI: IEEE, 2012: 943-944.

[2] George, L. 2011. HBase: the definitive guide. "O'Reilly Media, Inc.".

[3] Grainger, T, Potter, T, Seeley Solr, Y. 2014. in action. Manning.

[4] George, S, Ioannis, P, Nikos, N, Triantafillou, P. 2013. Interval indexing and querying on key-value cloud stores. Proceedings of the 29 th IEEE International Conference on Data Engineering (ICDE). Brisbane, Australia. pp: 805-816.

[5] Zou, Y.Q, Liu, J, Wang, S.C, Li, Z, Xu, Z.W. 2010. CCIndex: A Complemental Clustering Index on Distributed Ordered Tables for Multi-dimensional Range Queries. Proceedings of Network and Parallel Computing (NPC), Zhengzhou, China.pp: 247-261.

[6] Chen, Feng, Zou,Y.Q, Xu, Z.W. 2011. CCIndex for Cassandra: A Novel Scheme for Multi-dimensional Range Queries in Cassandra. Proceedings of 7th International Conference on Semantics, Knowledge and Grid (SKG). pp: 130-136.

[7] Bloom, B.H. 1970. Space/Time Trade-Offs in Hash Coding with Allowable Errors. Communications of the ACM. 13(7):422-426.

[8] Ungureanu, C, Debnath, B, Rago, S, Aranya, A. 2013. TBF: A memory-efficient replacement policy for flash-based caches. Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE). Brisbane, Australia. pp:1117-1128.

[9] Feng, X.P. 2014. The research and application of HBase storage . Beijing University of Posts and Telecommunications(In Chinese).

[10] O'Neil, E.J, O'Neil, P.E, Weikum, G. 1996. The LRU-K Page Replacement Algorithm For Database Disk Buffering. ACM Sigmod Record. 22(2):297-306.