International Journal of Grid Distribution Computing
Vol.8, No.2 (2015), pp.25-36
http://dx.doi.org/10.14257/ijgdc.2015.8.2.04

The String Similarity Query Processing in Cloud Computing System

LiaoYuanLai

Heyuan Polytechnic
HeYuan 517000, China
zsblyl@163.com

Abstract

The paper target at string similarity search in cloud systems. Existing works focus on query processing within a single server, and it incurs main memory overflow and external memory overflow while dealing with big data. For the above problems, the paper proposes a distributed index to support string similarity search in cloud environments. To provide efficient searching in a single node, an external memory index is designed, which adopts multiple filtering techniques and optimizing strategies. The external memory resident index supports length filter, positional filter in disks. This paper proposes the index construction method. During query processing, asymmetric q-gram is used to reduce the number of inverted lists read from disks. An adaptive algorithm is given to choose inverted lists, and seek the tradeoff between two aspects of query cost. The global index partitions the entire string dataset according the content of strings, and a char vector space partition method is proposed. In char vector space partition method, similar strings are partitioned into the same computing nodes, thus the number of computing nodes involved in a single query is reduced. The partition method is also adopted to determine necessary computing node set for a query to access. Simulation results validate the efficiency and effectiveness of our proposed index.

Keywords: *Cloud computing, String similarity, Query processing, Aggregation*

1. Introduction

String query similarity is the basis of operation for many applications, such as data cleaning, spell check, bioinformatics and information integration. With the rapid development of the explosive growth in data and information storage technology, large-scale data set is increasingly common string. The N-gram data Google has set contains 1T tuple; in the biological information, data size of GeneBank is to 416G, and contains more than one million records in the string, in huge collection efficiently handle string similarity query is the basis of many applications.

At present, the research work on string similarity query are based on q-gram memory inverted index, query process is divided into two stages filtering and verification. These methods convert a string to q-gram set, and a string containing the same q-gram Id organization to an inverted list. All of the q-gram corresponding to the inverted list is consisting of inverted index, used to support the string similarity query. Memory existing methods need to use the memory capacity is often more than the raw data size in large string collections. If the string set size is much larger than the memory capacity, the system cannot provide sufficient memory for building inverted index. Behm [1] realizes the inverted index of disk (hereinafter referred to as Behm-Index), is used to support data string attribute similarity query. The method of using the memory stored in the disk address string, and the inverted index in external storage. Behm-Index uses the prefix length filters and filter to reduce the number of disk reads the inverted list, and design the

ISSN: 2005-4262 IJGDC

adaptive inverted table selection method, in order to obtain the inverted table read cost and candidate test results at the expense of compromise, so as to improve the overall performance of the query [2-3].

Similarity search techniques are individual computing nodes existing string, in dealing with large-scale string data collection will cause memory overflow and external memory overflow two problems. Firstly, the existing methods are based on q-gram and inverted index establishment index using more space than the original string data set size. Therefore, in the large-scale string data to establishment inverted index of q-gram will occupy a lot of space in the disk, if the string data occupies more than half of the disk storage space; the remaining space cannot accommodate the inverted index q-gram. Even if the index disk puts on a separate disk, because the query processing need to read more inverted lists, memory consumes huge in the process of spatial query, if the query string is too long, the query using inverted lists a number too much, memory may not accommodate the inverted table. On the other hand, if the string data set is too large, it cannot fit in one disk; existing methods for a single compute node will fail [4-5].

This paper presents a distributed index DLPA-Index; cloud computing system supports the string similarity query. In order to obtain better local query efficiency, this paper will query optimization techniques of the existing string used in the external environment, including the length of filter, filter, filter and the position of the prefix asymmetric Gram mode [6]. This paper gives the LPA-Index disk index structures support length filters and position of filter, method and its construction and implementation details. In the LPA-Index query process, used asymmetric gram model, it is adaptive from a query string to select part of q-gram element, in order to obtain the query using the inverted list. Combined with the prefix filter, LPA-Index used to be selected inverted list processing string similarity query. Compared with the existing work Behm-Index, LPA-Index added to position filter in the index structure, so as to reduce the number of query candidate in the process, reduced the query in the verification phase cost; on the other hand, LPA-Index uses asymmetric gram model during a query, search by gram selection algorithm can effectively reduce the inverted list read number, reduces the cost of filtering query process. Therefore, LPA-Index can achieve better query performance. In order to reduce the query in terms of the number of nodes in the system, this paper designs the data partition method based on the character vector is used to divide the data set, the string. This method can be a string into a similar to the same computing node, and query needs to calculate the node access sets during query processing [7-8].

2. System Framework

This paper describes the structure and working flow of DLPA-Index in the cloud computing system. Figure 1 describes the DLPA-Index framework of a control node and four nodes in the cloud computing system.

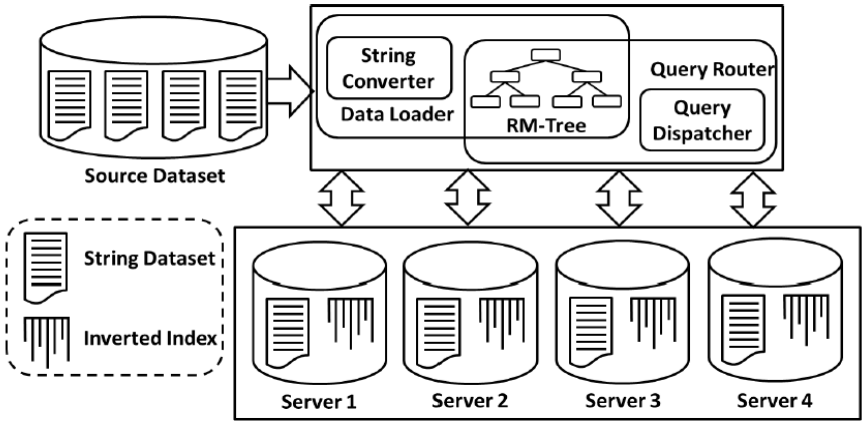

Figure 1. System Architecture of DLPA-Index

It is shown in Figure1, the original string data set through the control node is assigned to each computing node, in the distribution process of each computing nodes string data is roughly the same size. The control node consists of two modules, namely, data loading module and query routing module. Where, data loading module includes a data converter and a built in memory of the RM tree. Data converter will be responsible for the string data in a collection of data into a format that can be inserted into RM tree tuple. By the RM tree, the system can be mapped to a given string of nodes in the system.

The string data set of various import system computing node, control node used the RM tree to find the storage destination for each string. Query routing module and data loading module shared RM tree, and RM tree is to determine the query needs to be sent to those computing nodes. Subsequently, the query outing module used distributor module to send queries to corresponding nodes. Each computing node in the file memory saves string data distribution by the control non- node, and established LPA-Index in the file memory for indexing the data set, and answer similarity query string.

3. Local Query Processing

3.1. LPA-index

The structure of LPA-Index included memory Gram Trie disk and the inverted index, it is shown in Figure 2. Memory Gram Trie used a Trie structure to save all $q\text{-}gram$ in the set of the string. Each leaf node is corresponding to a $q\text{-}gram$, and contains the $q\text{-}gram$ corresponding to the inverted list memory address in the memory. LPA-Index used the length of filter and position filter to reduce the candidate result number, so the calculation of the string s'$q\text{-}gram$ set need to join the s length and the position of the $q\text{-}gram$ in each element. Given $GS(s) = \{g_0, ..., g_{|s|+q-2}\}$, where, g_i is the i-th $q\text{-}gram$ of s.Then LPA-Index calculated s set of $q\text{-}gram$ is $\{(g_0, |s|, 0), ..., (g_{|s|+q-2}, |s|, |s|+q-2)\}$.Denoted as $GS(s)^{LPA}$

The $(g_i, |s|, i)$ inserted into the Gram Trie, $|s|$ and i were regarded as characters. For example, when q = 2, $GS(Tim) = \{(\#T, 3, 0), (Ti, 3, 1), (im, 3, 2), (m\$, 3, 3)\}$, when inserted these elements in the Gram Trie, inserted into the four character string length is 4.

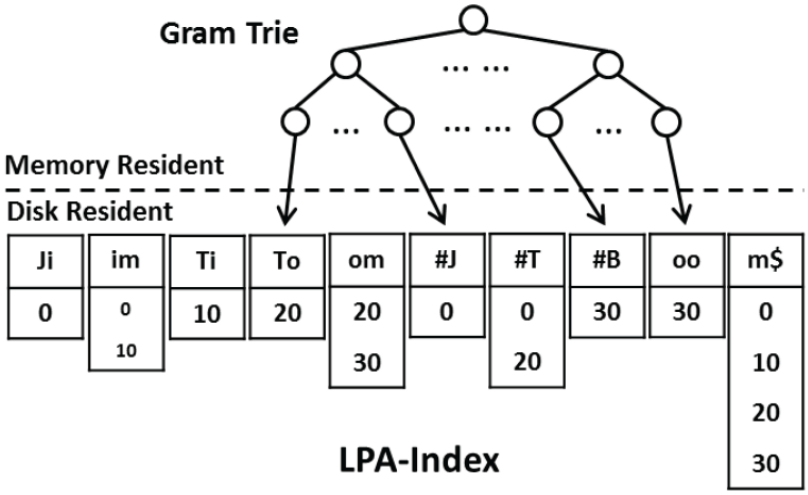

Figure 2. Structure of LPA-Index

3.2 Local Query Processing

Algorithm 1 describes the basic found process method. Basic Query Processing method with the existing work method is different from the reading process of inverted list (2-9). Firstly, algorithm 1 is calculated Gram set $GS(s)$ of the query s, secondly, the element modified the string length and position of $gram$ segment of the $q\text{-}gram$, and the LPA-Index reads query the inverted Table according to store sequentially. Then, with inverted list of the same character content g is all in the same list $L(g)$, and the elements are sorting according to the string memory address. Completed all the inverted list, algorithm 1 used MergeSkip methods to find a Cand set, finally, verify the elements in the Cand and return the final query result.

Algorithm 1. BASIC QUERY PROCESSING

Input: query $Q(s,\theta)$

Output: string set $\{t \mid t \in S, ED(s,t) \le \theta\}$

1 Computing $GS(s)$

2 For i=1to $|GS(s)|$ do

3 Initialization list $L_i = \varnothing$

4 For j= $(|s|-\theta)/l_{len}$ *to* $(|s|+\theta)/l_{len}$ do

5 For j= $(|i|-\theta)/l_{pos}$ *to* $(|i|+\theta)/l_{pos}$ do

6 Read the inverted list of $(GS(s)[i], j, k)$ and added to the L_i

7 The elements in L_i is in accordance with the increasing order by sorting

8 Cand = MergeSkip $(L_i \mid 1 \le i \le |GS(s)|, |GS(s)| - \theta \times q)$

9 return $\{t \mid t \in Cand, ED(s,t) \le \theta\}$

4. Global Query Processing

The query process of DLPA-Index includes three parts. Firstly, the control node area character vector space node is responsible for determining the query $Q(s,\theta)$ to need to access the computing nodes set $CN^{Q(s,\theta)}$ .Secondly, Query $Q(s,\theta)$ is sent to the all

computing nodes of $CN^{Q(s,\theta)}$ in the query, received computing nodes of query used local LPA-Index to process query, and returns the result to the control node. Finally, the control node receives the returned query results of all computing nodes.

Algorithm 2 describes the DLPA-Index string similarity query processing process. Firstly, the control node computing query requires to access the set target (1-10) of computing nodes, a process is running on the control node.

The control node computed the query string and the local string edit distance lower bounds by calculating each character vector of regional node, if the lower is smaller than query edit distance threshold, you need to execute a query in the local computing nodes. Secondly, the query $Q(s,\theta)$ is sent to the CN target of all nodes, and it is used LPA-Index queries in local, and merge the local query results Result^{CN} to the entire query result set Result (11-15).Execution time of the algorithm2 is the latest to return query results computing nodes in the local query execution time, therefore the efficient local query index is crucial for the whole system.

Algorithm 2. DLPA QUERY PROCESSING

Input: query $Q(s,\theta)$

Output: string set $\{t \mid t \in S, ED(s,t) \le \theta\}$

1 Result = $\varnothing$
2 S will query $Q(s,\theta)$ sent to the control node
3 Target is the set of all computing nodes
4 For each computing node CN do
5 Computing MinDist
6 For $CN \in target$ do
7 The query is sent to the CN
8 CN in the local $Q(s,\theta)$ gets Result_{CN}
9 Result = Result $\cup$ Result_{CN}
10 return Result

4 Experiment Design and Discussion

4.1. Local Query Performance

4.1.1. The Experimental Setup. In this paper, by real data and synthetic data test is to use efficiency of LPA-Index string similarity query processing. Experiments used a PC, configuration is Intel Core i5-2400 CPU, frequency is 3.10GHz; memory is 4GB; hard disk is 500GB.This paper used Java to realize the LPA-Index and Behm-Index of disk storage and query processing algorithms, JDK version is 1.6. Operating system is Ubuntu12.04LTS

Real data of experiments were Title and Author, where, contains published the title and author in the DBLP database. Contains 2078174 string in Title data set, the average length is 68; while the Author data set contains 1122253 string, the average length is 15. In addition, this paper uses Title and Author data sets to Synthesize Mix data sets. Title and Author were randomly selected a string s_1, s_2, and s_1 and s_2 merged into a new string s and deposited in Mix, the operation is repeated until the Mix reaches the specified size. In this paper, the number of strings contained in Mix respectively were 2×10^6, 4×10^6, 6×10^6, 8×10^6, 10×10^6.

In this paper, each data set randomly selected 100 string as the query performance test target. For the Title data set and Mix data set, the distance threshold edit tests are1, 2, 4, 8, 16; so string length of the average is smaller in Author data, the edit distance threshold of the Author data set is 1, 2, 3, 4.

In order to test the query running time in the external environment, after the completion of each query, by Java called the environment of Ubuntu commands to clear disk cache. Experiment used the disk page size is 8KB

4.1.2. The Performance Analysis

Firstly, the paper test LPA-Index construction costs, including the construction of time and storage memory overhead. Secondly test the LPA-Index query performance in different data sets. In this paper, the test LPA-Index query performance in Title, Author and Mix of the three data sets, and compared with Behm-Index.

In order to illustrate the effectiveness of LPA-Index optimization strategy, join the LPA-Index asymmetric gram model and query method in Behm-Index, denoted by Behm-A; used gram model of symmetry in the LPA-Index, denoted by LPA-NA. The following experiments compared the four methods to show the validity of LPA-Index and its optimization strategy.

Firstly, tested LPA-Index construction cost. Figure 3 and Figure 4 give the concentration time and space overhead construction of LPA-Index in Mix data. The construct cost of the LPA-Index is better than Behm-Index in the two aspects of time and space. In Figure 4, B shows Behm-Index, L shows LPA-Index, where, Phase1 (a) is a string and build in memory of inverted index, Phase1 (b) is a memory inverted index disk writes in time; Phase 2 is the integration time of temporary inverted index. LPA-Index and Behm-Index are basically the same in the consumption of time in Phase1 (a). LPA-Index will be the memory of the inverted list to writ to disk, as well as integrated a plurality of temporary file on disk, and consumed more time. The experiment results showed that LPA-Index basic construction cost is proportional to the string number in Mix data set.

Secondly, Figure 5 compares the Title data query time. The Behm-Index used the longest time, while LPA-Index used the shortest time. The performance of Behm-A and LPA-NA are in the middle. Compared with Behm-Index, LPA-Index in the edit distance is small, the query time is Behm-Index 48% to 55%; in the edit distance is large, the use of time is only 20.5% of Behm-Index. Figure6 compared four methods the number of inverted lists of 100 queries. Where, used non-symmetric gram mode of LPA-Index and Behm-A to read the inverted lists the number is less than LPA-NA and Behm-Index, indicating that the query used the methods of LPA-Index can effectively reduce the inverted list read number.

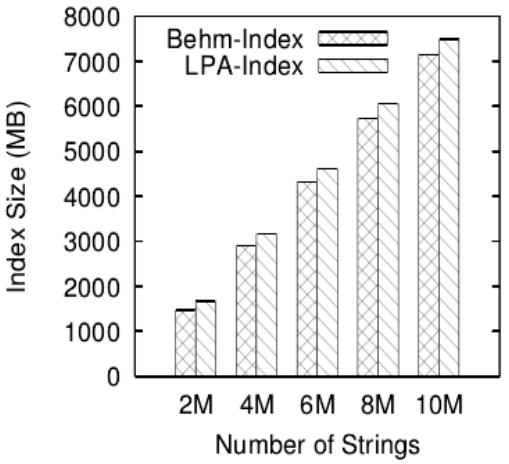

Figure 3. Index Size

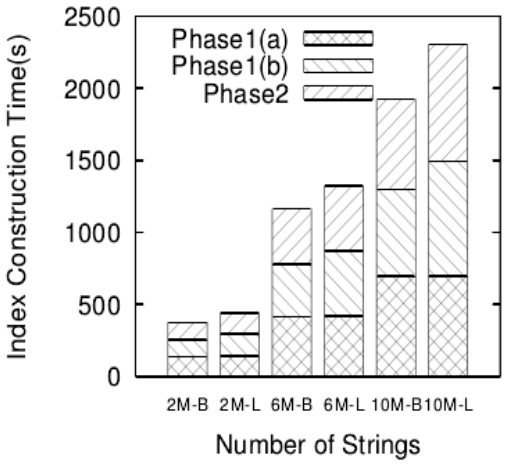

Figure 4. Construction Time

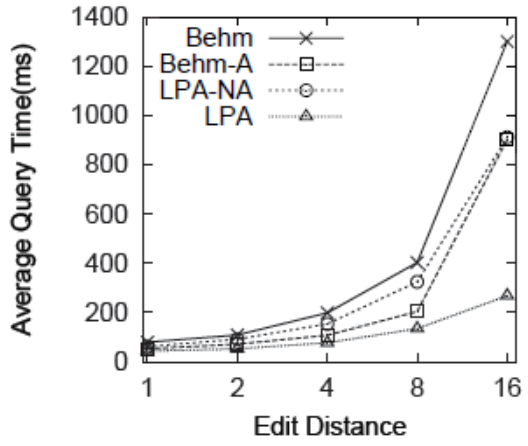

Figure 5. Average Query Latency

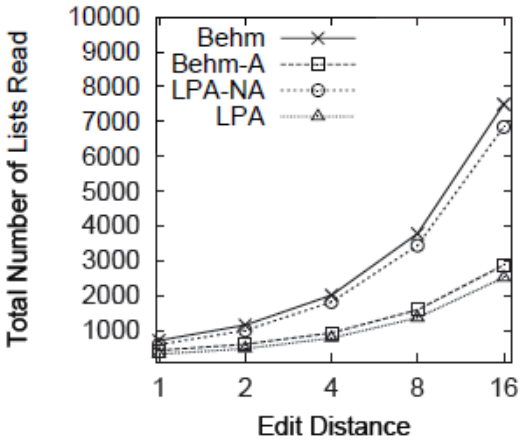

Figure 6. Number of Lists used

Figure 7 compared four methods the number elements of inverted lists of 100 queries. Where, LPA-Index and LPA-NA read number less than Behm-Index and Behm-A. This is because the LPA-Index and LPA-NA support position of filter, Therefore, the number of elements is much smaller than Behm-Index and Behm-A.

Figure8 compared four methods for 100 times the number of candidate query results. LPA-Index generates candidate results at least; asymmetric mode LPA-NA also produced less candidate results. This shows that location filters in the query process was effective in reducing the number of candidate results, reduce the cost of query verification stage.

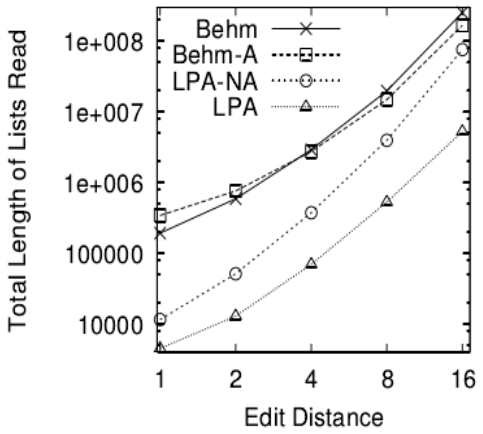

Figure 7. Average Query Latency

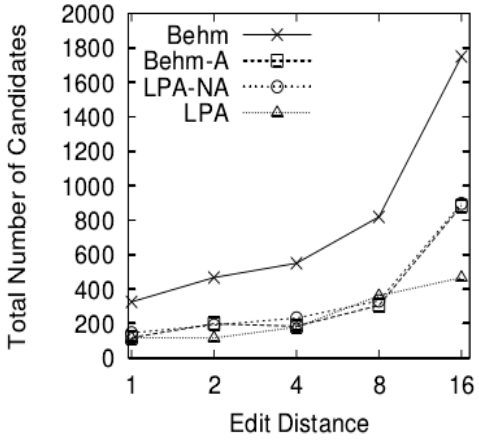

Figure 8. Number of Lists used

4.2 Global Query Performance

4.2.1 The Experimental Setup. Because the calculation of communications do not include DLPA-Index processing distributed queries, used the simulation can accurately calculate the query in the real system time overhead, so this paper tested the DLPA-Index in distributed environment string handling performance of similarity queries by the simulation experiment, the simulation environment is a PC machine, configuration and 4.1 section PC machine in the same. The paper used the setting method of computing system simulation environment and the existing cloud in the work [9-10] similar. The regional character vector space computational nodes used various simulation experiments in the query needs to calculate the node set, then each model of the experimental computing nodes in a PC are to create LPA-Index, and used these computing nodes in the LPA-Index treatment to receive query, and record each computing node local query execution time.

Experiment used author title data set data set of enlarged, and the synthetic mix data set, where, expanding method of the title and author random modified each string of single characters, produced 15 string modified, and the modified string added into the data set, the title set and author data set are expanded 16 times, while the mix data set is consists of 16×10^6 strings, generating method with the same 4.1 session.

The experiment used the query string was random extracted 200 strings from corresponding data set, the distance threshold editing the query in the title and mix data set value is 1, 2, 4, 8 and 16. Value of the author data set is 1, 2, 3 and 4. The simulation experiment in each local query processing calls the Linux system command clear disk cache, in order to obtain the accurate query time.

In this paper, compare objects is MapReduce computing framework, where, each query is sent to all the nodes, and the nodes with all query results are as a query result. Comparison object used single index is existing optimal Behm-Index, so the comparison method in the below called MRB.

4.1.2 The Performance Analysis. Figure 9 and figure 10 describe author data average time overhead in the single query and batch query of 200 times overhead. It is shown in Figure 9, the DLPA-Index processed single query time overhead is 20% to 50% of MRB method. Processed single query time overhead depends on the longest running query in the system processes, so used LPA-Index method can achieve better single query efficiency.

In the batch query performance, because the MRB method, it is shown in Figure 10, it is only 13% to 16% of MRB method time overhead. Performance improvement of batch

queries is higher than a single query, this is because the DLPA-Index effectively reduces the number of computing nodes in the query. The local index has better efficiency, so the batch query performance of DLPA-Index is greatly higher than the MRB method.

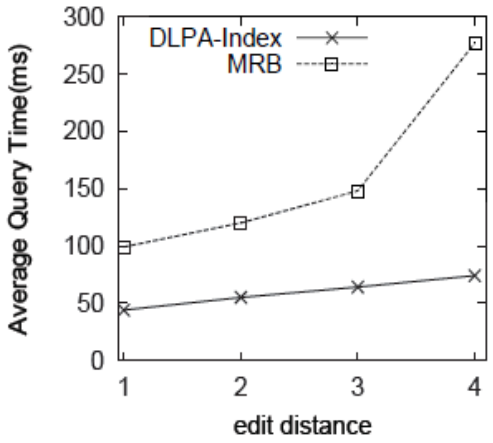

Figure 9. Average Query Time

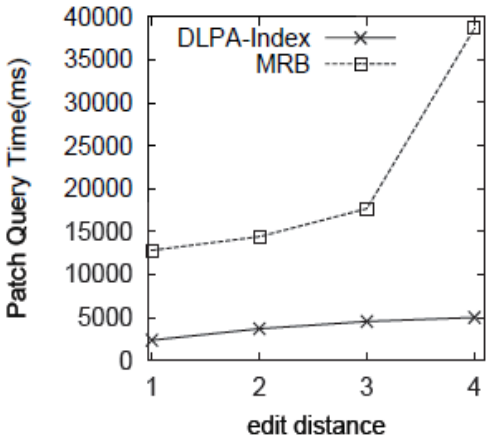

Figure 10. Patch Query Time

Figure 11 and Figure 12 give the title data set in a single query and batch query (query 200) time overhead. In Figure 11, in 1 to 16 query of the edit distance, DLPA-Index query time is always less than the MRB method. It is shown in Figure 12, in the edit distance increased from 1 to 8, batch query time of DLPA-Index method is 14% to 30% of MRB method, because the local index DLPA-Index use efficiency better, and it can effectively pruning in between computing nodes. When the edit distance is greater than 8, the local index query time growth of MRB is small, so the single query and batch query time overhead are small growth.

So the edit distance was 16, DLPA-Index in between computing nodes pruning ability is to drop, but individual computing nodes in the query time also increased, therefore DLPA-Index is superior to the query performance of MRB method to reduce.

Figure 13 and Figure 14 are described single query and 200 time cost of the query batch processing in the mix data set. Where, single query time and batch query time of DLPA-Index and MRB method increase with edit distance. And title data set is different, in growth process of the edit distance from 8 to 16, the MRB query time greatly increased. Because the string of mix data set is longer, so the edit distance increased from 8 to 16, the time cost of Behm-Index cannot reach the steady state. From Figure 13 and Figure 14 is shown, DLPA-Index compared with MRB method, the performance advantages of batch queries is more obvious. Where, a single query time of DLPA-Index Behm-Index is 35% to 60%, while the batch query time overhead is 16% of the MRB method. Because the DLPA-Index can effectively reduce the computing nodes number of queries, so the advantage of improving query time overhead is efficiency higher than local query.

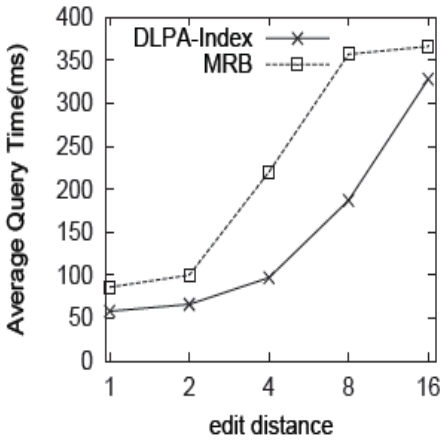

Figure 11. Average Query Time

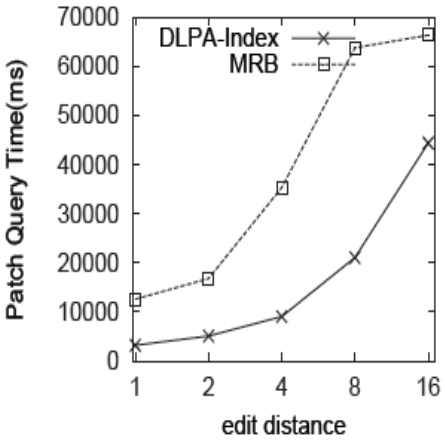

Figure 12. Patch Query Time

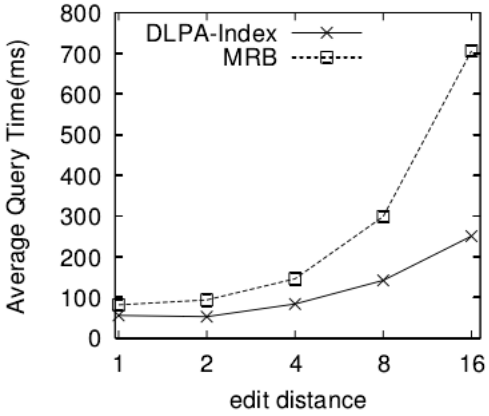

Figure 13. Average Query Time

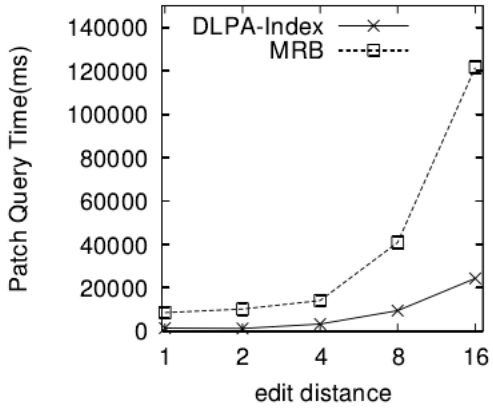

Figure 14. Patch Query Time

5 Conclusion

This paper presents the DLPA-Index query string similarity index structure to efficiently support query processing in the cloud computing system. In order to improve the local query efficiency, this paper proposes a disk index LPA-Index in the single computing nodes to efficiently process queries. LPA-Index used existing filtering techniques to reduce the number of the read and the candidate results inverted index in query process, in order to reduce the time cost of the query. In order to reduce the query in terms of the number of nodes in the system, DLPA-Index design method based on dividing the character vector is used to divide the string data set. The partitioning method can be assigned to a similar string of computing nodes, which reduces the number of the local query process. The experiment results show that LPA-Index used the local index is efficiency greatly superior to the existing Behm-Index index. In a single query and batch query mode, DLPA-Index can effectively improve the query efficiency, and reduce the query response time.

References

[1] C. Olston, B. Reed, U. Srivastava, *et al.*, "Pig Latin: A Not-So-Foreign Language for Data Processing", Proceedings of the 2008 ACM SIGMOD international conference on Management of Data. New York, NY, USA: ACM, **(2008)**, pp. 1099–1110.

[2] M. Isard, M. Budiu, Y. Yu, *et al.*, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks", Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, New York, NY, USA: ACM, **(2007)**, pp. 59–72.

[3] R. Chaiken, B. Jenkins, P. A. Larson, *et al.*, "SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets", Proc. VLDB Endow, vol. 1, no. 2, **(2008)**, pp. 1265–1276.
[4] D. J. DeWitt, E. Paulson, E. Robinson, *et al.*, "Clustera: An Integrated Computation and Data Management System", Proc. VLDB Endow, vol. 1, no. 1, **(2008)**, pp. 28–41.
[5] S. A. Weil, S. A. Brandt, E. L. Miller, *et al.*, "Ceph: A Scalable, High-Performance Distributed File System", Proceedings of the 7th symposium on Operating systems design and implementation. Berkeley, CA, USA: USENIX Association, **(2006)**, pp. 307–320.
[6] H. C. Yang, A. Dasdan, R. L. Hsiao, *et al.*, "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters", Proceedings of the 2007 ACM SIGMOD international conference on Management of Data, New York, NY, USA: ACM, **(2007)**, pp. 1029–1040.
[7] A. F. Gates, O. Natkovich, S. Chopra, *et al.*, "Building a High-Level Dataflow System on Top of Map-Reduce: The Pig Experience", Proc. VLDB Endow, vol. 2, no. 2, **(2009)**, pp. 1414–1425.
[8] B. Panda, J. S. Herbach, S. Basu, *et al.*, "PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce", Proc. VLDB Endow, vol. 2, no. 2, **(2009)**, pp. 1426–1437.
[9] J. Chou, J. Kim and D. Rotem, "Energy-Aware Scheduling in Disk Storage Systems", Distributed Computing Systems (ICDCS). Washington DC: IEEE, **(2011)**, pp. 423–433.
[10] B. Guenter, N. Jain and C. Williams, "Managing Cost, Performance, and Reliability Tradeoffs for Energy-Aware Server Provisioning", INFOCOM. Washington D-C: IEEE, **(2011)**, pp. 1332–1340.

Authors

Liao YuanLai, he received his master's degree of engineering in Guangdong University of Technology. He is a lecturer in Heyuan Polytechnic. He is in the research of Software engineering; Image processing, Image quality assessment and Data mining

