http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (20): 5415-5423, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.5415.5423

Efficiently Indexing Sparse Wide Tablesin Cloud Computing

[1,2]Huang Bin and [3]Peng Yuxing
[1]Department of Computer, Huaihua University, 418008, Huaihua, China
[2]School of Computer Science, Wuhan University, 430072, Wuhan, China
[3]National Laboratory of Parallel and Distributed Processing,
National University of Defense Technology, 410073, Changsha, China

Abstract: For its perception of unlimited resources and infinite scalability, Cloud Computing has emerged as a pervasive paradigm for hosting? data-centric applications in large computing infrastructures. The data produced by these applications are essentially sparse and wide and may change schema frequently, traditional relational data model is inappropriate for their data management. A new data model, called Sparse Wide Table, was introduced for this task. Unfortunately, we have to face many challenges in building the secondary index for Sparse Wide Table in cloud, as the distributed and column-oriented storage which eliminates a number of NULLs. In this study, we present a three-level index scheme for efficient data processing in the Cloud. Our approach can be summarized as follows. First, we build an index for each column by which the records can be rebuilt easily. Second, we build a bitmap index for each storage node which only indexes the data residing on the node. Third, we organize the storage nodes as a structured overlay and each node maintains a portion of the global index for the all different data. The global index is a bitmap index to indicate the node each data resides in. Finally, based on the three-level index scheme, some query algorithms are implemented. We conduct extensive experiments on a LAN and the results demonstrate that our indexing scheme is dynamic, efficient and scalable.

Key words: Cloud computing, bitmap, Index, wide table, column-oriented storage

INTRODUCTION

Now, ┌ data-centric web applications have become very popular. They provide the wide variety of Internet services based on massive data. For its perception of unlimited resources and infinite scalability, Cloud computing has emerged as a pervasive paradigm for hosting ┌ data-centric applications in large computing infrastructures. Currently, the Internet scale applications in cloud computing mainly include the following four broad categories: (1) New generation of e-commerce system, e.g., Amazon, (2) Data publishing and annotating services based on Web 2.0, such as Delicious, Flickr, Google Co-op (3) Search Engine, such as Google and (4) Some novel web applications such as Google Base.

Data: These applications bring a large amount of new type of data whic have following characteristics:

- **Massive:** The dataset is terabyte-scale even petabyte-scale and increases constantly. Cloud computing can meet the requirements, as it provides scalable cluster storage by partitioning the dataset into a number of small pieces called data shards. As a distribution unit, each data shard is stored on a unique storage node
- **Large No. of columns:** For example, (Agrawal *et al*., 2001) points out that in an e-commerce marketplace for the electronics industry the catalog may contain 2000 categories, with 5000 attributes per category. The current database system always has a limit on the numbers of columns in a table. For instance, the maximal number of the column in DB2 is 1012, Oracle too
- **Sparsity:** Though the dataset has a great many of attributes, most objects in the dataset have non-null values for only a small number of these attributes. So, the data is highly sparsely populated, in other words, a lot of attributes are NULLs
- **Constant schema evolution:** The increasing of heterogeneous data brings the table into frequently changing which is expensive for conventional relation database

These new characteristics above make conventional relational database not support for representing, storing

Corresponding Author: Huang Bin, Department of Computer, Huaihua University, 418008, Huaihua, China

and managing these datasets efficiently, because the prevalent n-ary horizontal representation introduces a large number of NULLs which waste storage space and the structured query performance is deteriorated seriously as the data records are wide and the query just focuses on a few columns. The current practice to solve this problem is to adopt the column-oriented (Yang *et al.,* 2008; Stonebraker *et al*., 2005; Abu Sayed and Hoque, 2002; Boncz *et al*., 2005; Chang *et al*., 2006) sparse Wide Table (Yu *et al*., 2008; Abadi, 2007). Based on the standard relational data model, the wide table logically stores a sparse dataset in a single table (i.e., each column represents a distinct attribute and each row represents an object), however, it employs column-oriented implementation mechamisms to represent and store data. The mechamisms eliminate NULLs and reduce the waste of storage space; they also improve query performance by only reading these data of some attributes relevant with the query than all attributes (Aguilera *et al*., 2008).

As mentioned in the 2.1, three data models are employed for representing and storing SWT and they are respectively: 2-ary, 3-ary and the hybrid of the former two (Yang *et al*., 2008). Though they eliminate NULLs, they result in some challenges in building secondary index.

Index: When the entire sparse data set is stored in a single table, it is crucial that we minimize the need to scan the whole table. A common approach to avoid table scans is indexing.

Currently, only the primary index is built by either employing a pure key-value data model, where both key and value are arbitrary byte strings (e.g., Dynamo (DeCandia *et al*., 2007), or its variant, where the key is an arbitrary byte string and the value is a structured record consisting of a number of named columns (e.g., Big Table (Chang *et al*., 2006) which supports efficient retrieval of values via a given key or key range). However, many applications also need the secondary index for improving query efficiency. In real world, users tend to query data with more than one key. For example, in an online video system, such as Youtube, each video could be stored in a key-value store with a unique video ID as the key and video information, including title, upload time and number of views as the value. Although the video can be efficiently retrieved via video ID, a common scenario is that the end user wants to find videos with given titles or within a date range. Therefore, the secondary index is necessary for the data in cloud.

Currently, only inverted indices are achieved by running a MapReduce (Dean and Ghemawat, 2004) job that scans the whole wide table and produces the necessary second indices in an offline batch manner. Although, inverted indexes avoid table scans for keyword queries, they cannot be used for range queries. Furthermore, the inverted index built by this approach is not up-to-date and newly inserted tuples cannot be queried until they are indexed. For instance, when a new item is inserted into Google Base, the item could be delayed for one day to be seen by users.

So, B-tree indices or their equivalent must be used. Unfortunately, we have to face the following challenges in building the secondary index for Sparse Wide Table in cloud:

- The distribution of data and scalability of resources require a distributed and scale index for the dataset
- After a wide table with N columns is stored based on column-oriented storage technology, N attribute values in same row is laid in N column file with different location, for example, for the same record, the location of the value of attribute I in column file I is X, however, it is Y for the value of attribute J in column file J. This results in some problems with building a secondary index

Contributions of this study: This study presents the CSG-index, a secondary index scheme for SWT in cloud storage systems. It is tailored for online queries and maintained in an incremental way. The CSG-index supports usual dictionary operations (insert, delete and lookup), a s well as range search with a given key range. It shares many implementation strategies with shared-nothing databases (DeWitt and Gray, 1992), peer-to-peer computing (Crainiceanu *et al*., 2007; Jagadish *et al*., 2005), column-oriented storage (Yang *et al*., 2008; Stonebraker *et al*., 2005; Abu Sayed and Hoque, 2002; Boncz *et al*., 2005; Chang *et al*., 2006) and existing cloud storage systems (DeCandia *et al*., 2007; Ghemawat *et al*., 2003).

CSG-index consists of three components: CF-index, DS-index and global-index.For the variety of the location of all attribute values in the same row, the CSG-index builds a CF-indexfor each column in each data shard. The index is an ordered list of the surrogate-handle pair according to the surrogate order. The surrogate is an identifier of a row data object and may be either a primarykey or an OID. The handle is the disk block number of the node.

Instead of building an index for the whole dataset, the CSG-index builds a local bitmap index for each data shard named as DS-index. The index is a distribution unit of the CSG-index which is stored and maintained on a unique

index server. CSG-index relies on this index distribution technique for desired scalability. Queries are served by searching all qualified index shards.

As the data set is partitioned into multiple data shards and each index server is responsible for query over its local DS-index, the CSG-index builds a Global-index to be aware of and optimized for the form of dataset partitioning. To route queries among the servers, all index servers are organized as a structured peer-to-peer network, BATON (Jagadish *et al*., 2005). All different values in some attribute among all data shard are split into N equal-size portions. Each index server stores one portion and builds a global bitmap index for its portion. Each index entry in the global bitmap index is a s_k-handle pair, where s_k is the secondary key that will be indexed and handle is a bitmap which could be used to fetch the corresponding nodes which the s_k resides in. A query routing algorithm traverses the network with neighbor links and returns all s_k-handle pairs.

Compared with recent studies on SWTs and indices technologies in cloud computing, our contributions of this study are summarized as follows:

- To the best of our knowledge, the CSG-index is the first secondary indexing mechanism designed to support structured range queries and point queries over column-oriented SWTs prevalent in cloud computing. The recent studies on SWTs, mainly focus on optimizing the storage scheme of datasets (Chu *et al*., 2007; Beckmann *et al*., 2006) and inverted indices (Yu *et al*., 2007). CG-index (Wu *et al*., 2010) proposes the secondary indexing mechanism only for row-oriented structure data in cloud computing
- The global search in the CSG-index is performed in a accurate data shard set in parallel, different from CG-index (Sai *et al*., 2010) which involves redundant data shard
- We present a novel query processing strategy for structure queries over the whole SWT which is suitable for any range query and point query

The rest of the study is organized as follows: Section 2 reviews related work. Section 3 introduces our system architecture. Section 4 presents the proposed index scheme in detail. Section 5 presents query algorithms. Section 6 empirically validates the effectiveness and efficiency of our proposed indexing scheme. We conclude in section 7.

RELATED WORK

Column-oriented wide table: Currently, three data models are employed for representing and storing the wide table and they are: 2-ary, 3-ary and the hybrid of the former two.

- **2-ary model:** The Decomposed Storage Model (DSM) (Copeland and Khoshafian, 1985; Khoshafian *et al*., 1987) decomposeshorizontal table into as many 2-ary tables as thenumber of columns. Each 2-ary table contains a surrogateand one attribute. A surrogate is always the object identifier.It is represented as:

$$R_b(\text{surrogate, attribute value})$$

Figure 2 shows the 2-ary vertical representation of thetable in Fig. 1.

- **3-ary model:** A single row in a horizontaltable is split into as many rows as the number of non-NULL attributes. Each 3-ary table contains object identifier,attribute name and attribute value. The schema is:

$$R_v(\text{Oid, name, value})$$

Figure 3 shows the 3-ary vertical representation of thetable in Fig. 1.

- **Hybrid model:** The hybrid representation first employs binary representation for each column

Oid	A	B	C	D	E
1	b	⊥	⊥	c	⊥
2	⊥	g	⊥	⊥	f
3	⊥	⊥	d	e	⊥
4	⊥	a	⊥	⊥	⊥

Fig. 1: A horizontaltable

(a)

sur	Value
1	b

(b)

sur	Value
2	b
4	a

sur	Value
2	b
4	g

(c)

sur	Value
3	d

(d)

sur	Value
1	c
3	e

(e)

sur	Value
2	f

sur	sur
	1
	2
	3
	4

Fig. 2(a-e): 2-ary binary representation

Oid	Name	Value
1	A	b
1	D	c
2	B	g
2	E	f
3	C	d
3	D	e
4	B	a

Fig. 3: 3-ary vertical representation

(a)

Row	Qualifier	Value
1	A	b

(b)

Row	Qualifier	Value
2	B	g
3	C	d
4	B	a

(c)

Row	Qualifier	Value
1	D	c
2	E	f
3	D	e

Fig. 4(a-c): hybrid representation

families and usesthe vertical representation in each column family table. Suppose there are three columnfamilies of the table in Fig. 1, family 1 just has attribute A,family 2 has B and C, family 3 has D and E. The tableusing hybrid representation is described in Fig. 4

BATON: BATON is a balanced tree structure overlayon a peer-to-peer network capable of supporting both exact queries and range queries efficiently. Usinganin-order traversal, it attains a linear ordering ofthe nodes in the tree. With this, each peer in thenetwork stores a link to its parent, a link to its leftchild, a link to its right child, a link to its left adjacentnode, a link to its right adjacent node, a left routing table to selected nodes on its left hand side at thesame level and a right routing table to selected nodeson its right hand side at the same level. While the tree structure is binary, it hasscalability and robustness similar to that of the B-tree.An immediate benefit of a tree structured overlay network is to conveniently support for range queries whichcannot be supported by conventional distributed hash tables. In spite of the tree structure causing distinctions to be made between nodes atdifferent levels in the tree, theload at each node is approximately equal.

It assigns to each node, both leaf and internal, arange of values. It records for each link the rangeof values managed by the node at the target of thelink. The range of values directly managed by a node isrequired to be to the right of the range managed byits left subtree and less than the range managed by itsright subtree. With this, the BATON overlay structure immediately behaves like an index tree.

Indices for large-scale data: Khoshafian *et al.* (1987), a distributed B+-tree algorithm was proposed for indexing the large-scale dataset in the cluster. The B+-tree is distributed among the available nodes by randomly disseminating each B+- tree node to a storage node (also called server node in). This strategy has two weaknesses. First, although it uses a B+-tree based index, the index is mainly designed for simple lookup queries and is therefore not capable of handling range queries efficiently. To process a range query [l, u], it must first locate the leaf node responsible for l. Then, if u is not contained by the leaf node, it needs to retrieve the next leaf node from some storage server based on the sibling pointer. Such form of retrieval continues until the whole range has been searched. Second, it incurs high maintenance cost for the server nodes and huge memory overhead in the client machines, as the client node (user's own PC) lazily replicates all the corresponding internal nodes.

Sai *et al.* (2010), a scalable B+-tree based indexing scheme was proposed for efficient data processing in the Cloud. It builds a local B+-tree index for each storage node which only indexes data residing on the node. Then it organizes the storage nodes as a structured overlay-BATON and publishes a portion of the local B+-tree nodes to the overlay for efficient query processing. This strategy has three weaknesses. First, in local-index, it suffers too much same value in leaf nodes, as a result,it slows query down. Second, as the query range may be narrower than the range of nodes published to CG-index, the result from searching in CG-index is inaccurate and it results in some unwanted nodes searching in local-index. Third, the CG-index ignoresthat some rang values which are resided in multiple nodes may be overlap with each other.

Both the CG-index and the distributed B+-tree are used to build indices for row-oriented dataset, whereas the CSG-index does for the column-oriented SWT. What's more, the CSG-index has some differences with them. Compared with the CG-index, it can attain an accurate node set from the global-index of the CSG-index. Compared with the distributed B+-tree, each index node maintains a portion of all index entries and it can efficiently support point queries and range queries.

SYSTEM ARCHITECTURE

Figure 5 shows the system architecture of our cluster system. A set of low-cost workstations join the cluster as storage (or processing) nodes. This is a shared nothing and stable system where each node has its own memory and hard disk. To facilitate search, nodes are connected based on the BATON protocol. Namely, if two nodes are routing neighbors in BATON, we will keep a TCP/IP connection between them. Note that BATON was proposed for a dynamic Peer-to-Peer network. It is designed for handling dynamic and frequent node departure and joining. Cloud computing is different in that nodes are organized by the service provider to enhance performance. In this study, the overlay protocols are only used for routing purposes. Amazon's Dynamo adopts the same idea by applying consistent hashing for routing in clusters. BATON is used as the basis to demonstrate our ideas due to its tree topology. Other overlays supporting range queries, such as P-Ring and P-Grid (Crainiceanu *et al*., 2007), can be easily adapted as well.

In our system, data is randomly distributed to storage nodes based on their primary keys; as a result, data are partitioned into N data shards if there are N nodes in our cluster. Employing the column-oriented storage technique, we store each data shard into the same many column-files as the attribute of a row data object. We build an index for each column-file. The index is located in the lowest level in the CSG-index. It is an ordered list of the surrogate-handle pair according to the surrogate order. The surrogate is an identifier of a row data object and may be either a primary-key or an OID. The handle is the disk block number of the node and could be used to fetch the corresponding value in the cloud storage system. By the surrogate, we can rebuild the row corresponding to the surrogate.

To facilitate search in the attribute, each storage node index its local data by building a B+-tree for different values, its leaf nodes are key-bitmap pairs. A bitmap for every different value indicates which records the value resides in. In this way, given a value, we can efficiently receive a bitmap by which a surrogate set can be built easily.

Every node in the BATON manages a continuous range value, so this index information of these values in the range is stored in the node which is a key-bitmap pair. This bitmap indicates the value resides in which nodes. These indexes compose the Global-index in our system. To process a query, we first look up the Global-index for the corresponding storage nodes based on the overlay routing protocols. And then following the bitmap of the Global-index, we search the local index in parallel.

CSG-INDEX

In our system, data is randomly distributed to storage nodes based on their primary keys; as a result, data is

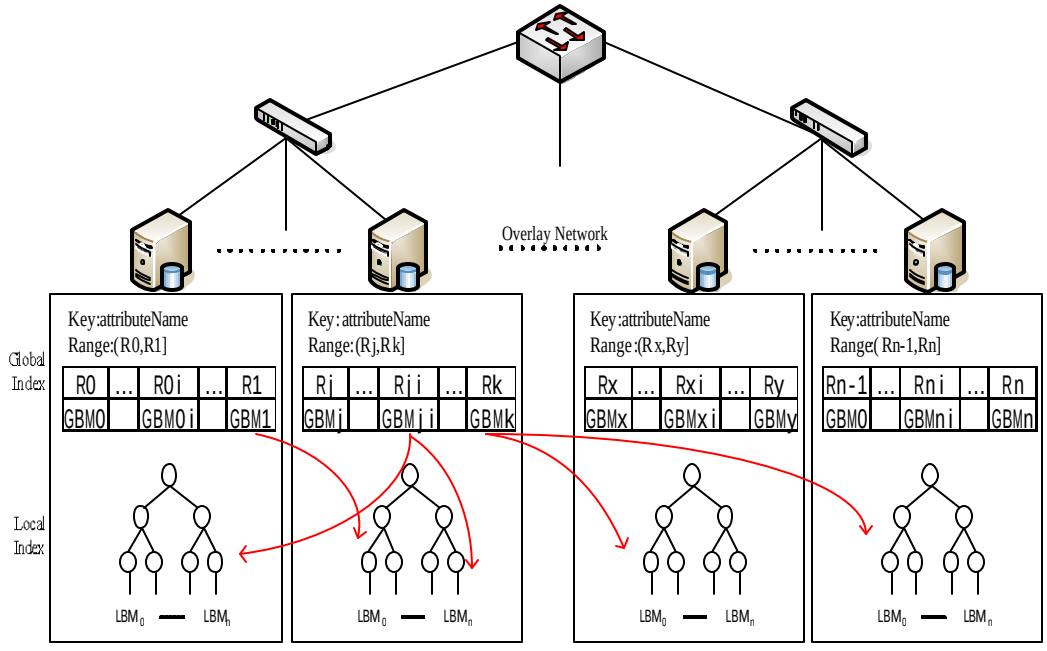

Fig. 5: System architecture

partitioned into data shards and these records whose keys are continuous may be distributed to different nodes. For a non-key attribute, it has a lot of same values. To process queries in the cluster, a traditional scheme will broadcast the queries to all the nodes, where local search is performed in parallel. This strategy, though simple, is not cost efficient and is not scalable too. Another approach is to maintain data partitioning information in a centralized server. The query processor needs to look up the partitioning information for every query. The central server risks being the bottle-neck. To facilitate search in a non-key attribute, we build the CSG-index: the global bitmap index and the local bitmap index. The former is used to seek nodes the required value resides in and the latter is used to search the record address the required value in the local data of each storage node.

CF-index (column file-index): As mentioned in the 2.1, a SWT may be represented into one of three data model (2-ary,3-ary,hybrid). This section introduces their index methods, respectively.

- **Indexing 2-ary:** The index is an ordered list on the surrogate order of each row record. Each index entry is a triple(surrogate,blk,length), where the blk is the disk block number of the node and the length is the byte number of the attribute value. It is represented as:

$$I_b(\text{surrogate,blk,length})$$

The index is suited for fixed or variable length attributes as the length indicates the data length read. As the values in each attribute are stored in a data file individually, the index only finds values in an attribute and cannot find the values in others attributes at the same time. If the indexed attributes have their index files and others attributes do not, the structure query involving some attributes which are not indexed will scan these data files in sequence. As a result, it is an inefficient search. Thus, for handling the structure query with arbitrary number of attributes, we build a CF-index for each attribute in the column level. Supposed AP stand for the processed attribute set, the algorithm of query and rebuild a record is:

Algorithm 1: b_query (AP, surrogate)

1:foreach (A_i in AP)
2:Achieve the blk and length corresponding to surrogate the from the I_b of A_i
3:Achieve the value from the data file of A_i
4:data set←value

- **Index 3-ary:** Like the index of 2-ary, the index of 3-ary is also an ordered list and each index entry is a surrogate-blk pair. It is represented as:

$$I_v(\text{surrogate, blk})$$

As the values of all attributes are stored in a data file, only an index file is needed to maintain. However, it is needed to rebuild a record by analyzing the read data bulk and choosing the values of wanted attributes when a structure query with multiple attributes is processed. The algorithm of query is:

Algorithm 2 v_query(AP, surrogate)

1:Achieve the blk corresponding to the surrogate from the I_v
2: repeat
3: read a record R
4:if the attribute A of R in AP
5: data set?the value of R
6: until(the surrogate of R ?surrogate)

- **Index hybrid:** Like the index of 3-ary, its index is represented as:

$$I_h(\text{surrogate,blk})$$

However, different from the index of 3-ary, the number of I_h is equal to the number of the attribute clusters. So, the algorithm of query is:

Algorithm 3 H_query(AP, surrogate)

1: foreach (A_i in AP)
2: do v_query over the I_h and data file of A_i

DS-index(data shard-index): In cloud computing, due to a lot of values are same for a non-key attribute. The same value may reside in multiple nodes, one of while may has many duplicate values. If indexing these values by B+-tree, the index entry is too much. It results in pages of the index data file very much and the B+-tree is very high, as a result, it slows query down. For a data set in which much data has many duplicate values, the bitmap index is a good index method; especially in cloud computing, query is more than modification. Thus we build a bitmap index as the data shard-index. The bitmap indicates the OIDs of the wanted value in the local data shard. To facilitate find the bitmap of the required value, we build a B+-tree for all bitmaps which leafs are key-bitmap pairs.

We use run-length encoding (Garcia-Molina *et al*., 2000) to compress all bitmaps which can be used to make the number of bits closer to n, independent of the number of different values (DeWitt and Gray, 1992). One advantage of run-length encoding is to reduce storage

requirement. Another advantage is that the last insert value only needs to modify its bitmap while others do not.

Global-index: To route queries among theservers, all index servers are organized as a structured peer-to-peernetwork, BATON. All data are randomly distributed to storage nodesbased on their primary keys; as a result, the same valueisstored in multiple nodes for their different primary keys. To facilitate search, we build a distributed global-index for the attribute.

As previously mentioned, BATON is a balanced tree structure in which each node, both leaf and internal, is assigned arange of values andusinganin-order traversal, it attains a linear ordering of the range of values managed by each node in the tree. With this advantage, we organize index entries of some attribute based on BATON. Supposed the range of the attribute value is [l,h) and the BATON has N nodes, we split the range into N equal-size continuous portions (P_1, P_2,... P_i,..., P_n). There are:

$$P_i = [l + (i-1) \times \frac{h-1}{N}, l + i \times \frac{h-1}{N})$$

and:

$$\bigcup_{i=1}^{i=N} P_i = [l, h)$$

Each portion is in order assigned to a node on the linear ordering of the nodes in the BATON which is attained by using an in-order traversal. Thus the range managed by ith node is P_i. Therefore the indexing scheme is distributed and scalable.

In each node, a B+-tree is built for these values managed by it and belonging to the global index. Each index entry is a s_k-handle pair, where s_k is the secondary key that will be indexed and handle is an bitmap which could be used to fetch the corresponding node which the sk resides in.

When a new value is inserted, we obtain a node which range contains the value based on BATON's routing protocols and publish an index entry to the global index managed by the node.

QUERY PROCESSING

Our index scheme is capable of supporting both point queries and range queries based on the structure and protocols of the BATON. For a dimension index, the point query is the extreme of the range query (where l = h), so this section only introduce the range query with a dimension index.

Given a range query Q(AP, $l = A_i = u$), we first search the global index to locate the B+-tree nodes whose ranges overlap with Q, then find in DS-index and get a surrogate set, finally find and rebuild the record relevant to AP. Algorithm 4 shows the range search algorithm. Starting from the lower bound of Q, we follow the right adjacent links to search sibling nodes until reaching the upper bound of Q.

Algorithm 4 query(AP, $l=A_i=u$)

```
1:Ni = lookup(l)
2:while(Ni = Ni.right and value<u)
3: bmp = bmp op value.bitmap
4: Ni send the query to these nodes of bmp[i] = 1 and they do concurrently:
5: search in DS-index, get a surrogate set
6: search in CF-index for each surrogate and rebuild the record
7: return the record set to Ni
8: aggregate the records from these nodes of bmp[i]=1,and return
```

EXPERIMENT EVALUATION

Experiment environment: The experiment environment is located in a LAN and contains ten computers with a 2.8 GHz Pentium dual-core processor, 2GB memory and 160 GB storage. All computers are connected into a subnet by a two-layer switch which ports' speed is 1000 Mbps. Our system is implemented in Java 1.6.0. In our system, each node hosts 500k tuples. The tuple format is (key, string). The key is an integer key with the value in the range of [0, 10^9] and the string is a randomly generated string with 50 bytes.

In experiment, each node is both a client which issues query or update command and an index server which serves query requests. We generate exact queries and range queries for the keys in uniform distribution. The major metrics in the experiment are query throughput.

Performance of query: Figure 6 shows query throughput under different search ranges. The best performance is achieved for the exact search query (s = 0).When the

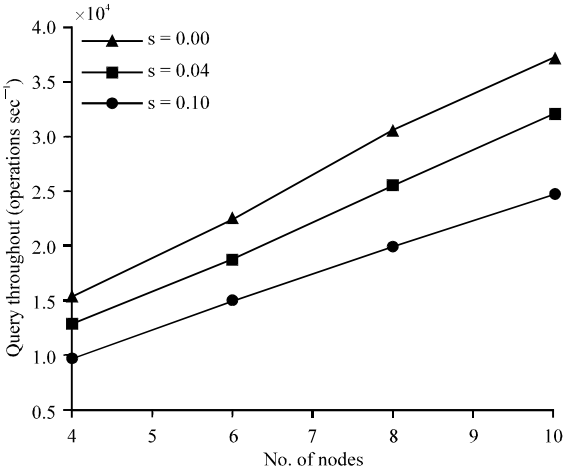

Fig. 6: Throughout of query

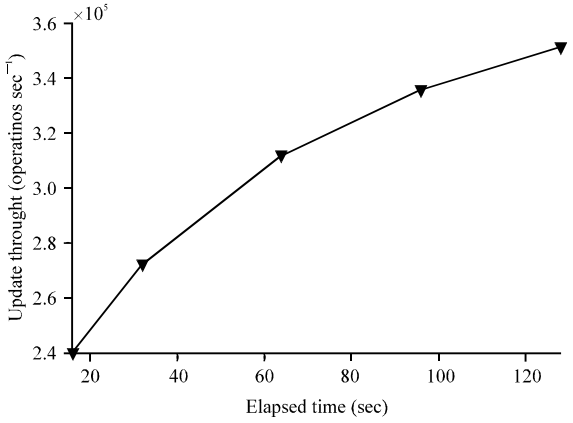

Fig. 7: Update throughput

search range is enlarged, throughput degrades as more nodes are involved. Scalability increases when we increase the number of processing nodes. Figure 7 shows the update throughput of the system (in logarithmic scale) where the node generates uniform insertions. Their scalability increases when we increase the number of processing.

CONCLUSION

We have presented the design and implementation of a scalableand high-throughput indexing scheme for SWTs in Cloud. We first build a index foreach column by which the records can rebuild easily, then assume a local bitmap index is built for thedataset stored in each storage node. And to enhance the throughputof the system, we organize storage nodes as a structured overlayand build a global bitmap index. Each computing node stores a portion of all index entries. Based on the overlay's routing protocol,the CSG-index can support point queries, range queries. Our scheme has been implementedand evaluated in a LAN.The experimental results show that our approach isefficientand scalable.

ACKNOWLEDGMENT

The authors would like to thank for the support by National Basic Research Program of China (973 Program) under Grant No.2011CB302601, National High Technology Research and Development program of China (863 Program) under Grant No. 2011AA01A202, Science and technology program of Hunan Province under Grant2013FJ4335 and 2013FJ4295 and the constructing program of the key discipline in Huaihua University.

REFERENCES

Abadi, D.J., 2007. Column-stores for wide and sparse data. Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research, January 7-10, 2007, Asilomar, CA, USA.

Abu Sayed, M. and L. Hoque, 2002. Storage and querying of high dimensional sparsely populated data in compressed representation. Proceedings of the 1st EurAsian Conference on Information and Communication Technology, October 29-31, 2002, Iran, pp: 418-425.

Agrawal, R., A. Somani and Y. Xu, 2001. Storage and querying of e-commerce data. Proceedings of the 27th International Conference on Very Large Data Bases, September 11-14, 2001, Rome, Italy, pp: 149-158.

Aguilera, M.K., W. Golab and M.A. Shah, 2008. A practical scalable distributed B-tree. Very Proceedings of the Large Data Base Endowment, August 24-30, 2008, Auckland, New Zealand, pp: 598-609.

Beckmann, J.L., A. Halverson, R. Krishnamurthy and J.F. Naughton, 2006. Extending RDBMSs to support sparse datasets using an interpreted attribute storage format. Proceedings of the 22nd International Conference on Data Engineering, April 3-7, 2006, Atlanta, Georgia, pp: 58-58.

Boncz, P., M. Zukowski and N. Nes, 2005. MonetDB/X100: Hyper-pipelining query execution. Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research, January 4-7, 2005, Asilomar, California.

Chang, F., J. Dean and S. Ghemawat, 2006. Bigtable: A distributed storage system for structured data. Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, November 06-08, 2006, Seattle, Washington, DC., pp: 205-218.

Chu, E., J. Beckmann and J. Naughton, 2007. The case for a wide-table approach to manage sparse relational data sets. Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, June 11-14, 2007, Beijing, China, pp: 821-832.

Copeland, G.P. and S.N. Khoshafian, 1985. A decomposition storage model. Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, May 28-31, 1985, Austin, Texas, pp: 268-279.

Crainiceanu, A., P. Linga, A. Machanavajjhala, J. Gehrke and J. Shanmugasundaram, 2007. P-ring: An efficient and robust p2p range index structure. Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, June 11-14, 2007, Beijing, China, pp: 223-234.

DeCandia, G., D. Hastorun, M. Jampani, G. Kakulapati and A. Lakshman *et al*., 2007. Dynamo: Amazon's highly available key-value store. Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles, October 14-17, 2007, Stevenson, WA, USA, pp: 205-220.

DeWitt, D. and J. Gray, 1992. Parallel database systems: The future of high performance database systems. Commun. ACM, 35: 85-98.

Dean, J. and S. Ghemawat, 2004. MapReduce: Simplified data processing on large clusters. Proceedings of the 6th Conference on Symposium on Opearting Systems Design and Implementation, Vol. 6, December 06-08, 2004, San Francisco, CA., pp: 10-10.

Garcia-Molina, H., J.D. Ullman and J. Widom, 2000. Database System Implementation. Prentice Hall, Upper Saddle River, New Jersey, pp: 225-231.

Ghemawat, S., H. Gobioff and S.T. Leung, 2003. The google file system. Proceedign of the 19th ACM Symposium on Operating Systems Principles, October 19-22, 2003, ACM, Lake George, NY., pp: 29-43.

Jagadish, H.V., B.C. Ooi and Q.H. Vu, 2005. BATON: A balanced tree structure for peer-to-peer networks. Proceedings of the 31st International Conference on Very Large Data Bases, October 04-06, 2005, Italy, pp: 661-672.

Khoshafian, S., G.P. Copeland, T. Jagodis, H. Boral and P. Valduriez, 1987. A query processing strategy for the decomposed storage model. Proceedings of the 3rd International Conference on Data Engineering, February 3-5, 1987, Washington, DC, USA, pp: 636-643.

Stonebraker, M., D.J. Abadi, A. Batkin, X. Chen and M. Cherniack *et al*., 2005. C-store: A column-oriented DBMS. Proceedings of the 31st International Conference on Very Large Data Bases, October 04-06, 2005, Italy, pp: 553-564.

Wu, S., D.W. Jiang, B.C. Ooi and K.L. Wu, 2010. Ef?cient b-tree based indexing for cloud data processing. Proceedings of the VLDB Endowment, Vol. 3, September 2010, Singapore, pp: 1207-1218.

Yang, B., W.N. Qian and A.Y. Zhou, 2008. Using wide table to manage web data: A survey. Front. Comput. Sci. China, 2: 211-223.

Yu, B., G.L. Li, B.C. Ooi and L. Zhou, 2007. One table stores all: Enabling painless free and easy data publishing and sharing. Classless Inter-Domain Routing, pp: 142-153. http://www.bibsonomy.org/bibtex/163ff0c4f514993844aa14a9bbfcba1f8

Yu, B., G.L. Li, B.C. Ooi and L.Z. Zhou, 2008. One table stores all: Enabling painless free-and-easy data publishing and sharing. Classless Inter-Domain Routing. http://www.comp.nus.edu.sg/~utab/paper/cidr07p16.pdf