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 9 (3): 474-480, 2010
ISSN 1812-5638

An Incremental Block-Level Snapshot Indexing Algorithm using Multi-Version Bitmaps

[1]Guangjun Wu, [1,2]Binxing Fang, [2]Xiaochun Yun, [2]Shupeng Wang and [1]Xiangzhan Yu
[1]Research Center of Computer Network and Information Security Technology,
Harbin Institute of Technology, Harbin, China
[2]Institute of Computing Technology Chinese Academy of Sciences, Beijing, China

Abstract: In this study, we proposed a hierarchical indexing structure with merging window to reduce redundant entries in the snapshot query path. The window size is controlled by Indexing Redundancy (IR) computed by bitmap vectors. Performance analysis of the novel indexing method indicates that the new approach provides efficient access to long-lived snapshot at low cost and it could potentially be suitable for write-intensive replicated system. The experiment results exposed that there was 70% reduction in the merging windows size and just increased 8% indexing metadata compared to current indexing method.

Key words: Snapshot, indexing, version management, block level, indexing redundancy

INTRODUCTION

In recent years, data protection has become an important issue in organizations and individuals (McKnight *et al*., 2006). The emergence of the large volume storage network provides an opportunity to store long-lived multi-version snapshots. Block-level snapshot is an instant data image which supports fine-granularity data recovery and can provide near CDP data protection. The common snapshot methods are COW (Copy-On-Write) (Santry *et al*., 1999; Charles and Grunwald, 2003; Peterson and Burns, 2005) and ROW (Redirect-On-Write) (Patterson *et al*., 2002) which only duplicate the incremental updating data to storage medium. The interesting version of snapshot has scattered in different incremental versions. The challenge is how to query the interesting version snapshot in the long-lived multi-version context. Some approaches have been proposed to access data historical state in the former literatures.

Salzberg and Tsotars (1999) gave a survey of partial-persistent indexing techniques, such as multi-version and overlapping structures. But these techniques operated at the logical database level, indexing records and targeting specific queries. For example, Snapshot Index (Tsotras and Kangelaris, 1995) found all records in the target version; Multi-version B Tree (Becker *et al*., 1996) and Time Split B tree (Lomet and Salzberg, 1989) targeted key-range queries. The block-level snapshot operates at the block level and it is independent from the upper layer applications: file systems or databases. Content Based Image Retrieval (CBIR) has emerged during the last several years (Smeulders *et al*., 2000). The CBIR indexed images based on objects and relations between them. The main idea is to represent each image as a feature vector and to measure the similarity between images with distance between their feature vectors (Sudhamani and Venugopal, 2008). The CBIR can retrieve images efficiently those are visually similar to the retrieved one. Some content-based indexing techniques have also been proposed in the massive logging and information management system with the help of data mining and knowledge discovery techniques. Block-level snapshot constitutes binary blocks which are generated by different applications, even running compression and encryption operations, so the content-based indexing methods can not be used directly.

Currently, some positive policies have been proposed in the split-storage system. SNAP (Shrira and Xu, 2005) used VMOB to cache old snapshot at the cost of extra memory. Thresher (Shrira and Xu, 2006) discriminated the lifetimes of snapshots and copied them into different archive areas. These methods were limited helpful to the long-lived context. Skippy used multiple layer Maplogs to skip over hot data which frequently appeared in the indexing structure (Shaull *et al*., 2008). Skippy could increase snapshot query efficiency dramatically, but it used fixed-size merging window to filter out the redundant indexing entries. The larger window could merge more redundant entries to the upper layer Maplog, but more memory and computing resources are needed in the merging process, which had impact on the performance in the write-intensive replicated system. On the other hand, the smaller window could potentially generate more

Corresponding Author: Guangjun Wu,Research Center of Computer Network and Information Security Technology,
Harbin Institute of Technology, Harbin, China

redundant indexing entries into the upper layer Maplog and increased the volume of indexing metadata. The fixed-size window method is awkward in the write-skew situations and Skippy (Shaull *et al.*, 2008) can just enlarge window size to merge more redundant entries. It is important to set different window size according to skew situations in the hierarchical indexing structures. We are further in this trend and present an indexing method which uses bitmap vectors to compute the indexing redundancy caused by write-skew. The detailed contributions of this study are listed as follows:

- Describes the block-level snapshot merging process and presents a multi-version bitmap vectors snapshot merging algorithm, which can control the window size according to the writing skew situations
- Analyzes the hierarchical indexing structure theoretical performance and evaluates the algorithm in practice. Experiment exposed that the indexing structure could provide a practical solution for the long-lived block-level snapshot query

DEFINITIONS

Block-level snapshot has temporal and spatial attributes. The temporal attribute is used to discriminate between snapshot versions. In CDP and COW system, the version sequence is kept by a version number or the writing sequence. Because of access locality, there is large number of snapshot with the same address. It leads to scan redundant indexing entries repeatedly in a snapshot query process. A common method is to set a window and merge the redundant entries to the upper layer. The upper layer structure only keeps the distinct entries from the low-layer's merging window. The snapshot query is to travel the merged indexing and the bottom window.

Figure 1 shows the merging process in the snapshot indexing structure. The K_1 entry emerges in version V_1 and V_3 and the K_2 emerges in version V_2 and V_5. The merged results only preserve the last version of the entries in the low-layer window and the redundant ones are filtered out. It is called Snapshot Merging Process and the lower layer's merged structure is called Merging Window. A bitmap is often used to depict the spatial distribution for the block-level snapshot. We use the multiple bitmaps in the snapshot merging process to control the windows size.

Definition 1. Bitmap vector: The replicated blocks constitute data set S. A bitmap R can be used to represent the logical address distribution with 01 strings. It can use a function f to represent this mapping process, f(S) = R. R is called bitmap vector for set S. Similarly, a block x can be represented as: f(x) = r and r is called x's bitmap vector. We can predict whether a block x is included in blocks set S through their bitmap vectors computing. That is f(x) = r, f(S) = R, if for all the 1 bit in r, the corresponding bits in R are also 1, then x is included in S and it can be written as $r \in R$. Otherwise, we can predict that x is not included in S and it is written as $r \notin R$.

Definition 2. Vector merging: For block x and blocks set S, f(x) = r, f(S) = R, if $r \notin R$, when we add x to S, the new set's bitmap vector equals r and R OR binary operation. This process is called vector merging.

Vector merging operation can generate union of set's bitmap vector directly. If $f(S_1) = R_1$, $f(S_2) = R_2$ and $S=S_1 \cup S_2$, then S's bitmap vector R equals R_1 OR R_2. We can compute the degree of indexing entry redundancy in the snapshot merging process through their bitmap vectors.

Definition 3. Indexing redundancy: In snapshot replicated process, we name the ratio of the total number of indexing entries (N_{sum}) and the amount of distinct logical address entries (N_{disc}) Indexing Redundancy (IR). IR can be depicted as Eq. 1.

$$\beta = N_{sum} / N_{disc} \quad (1)$$

Block-level indexing entry is often used to index binary block (for example, 4 kb block in linux), so we use the total number of replicated blocks to represent N_{sum} and the number of distinct logical address blocks for N_{disc}. According to Eq. 1, the value of IR (β) is not smaller than 1. We used multiple bitmap vectors to compute indexing structure's IR. When the IR is bigger than some fixed value, that is $\beta > \beta_0$, we generate the upper layer indexing structure through merging process.

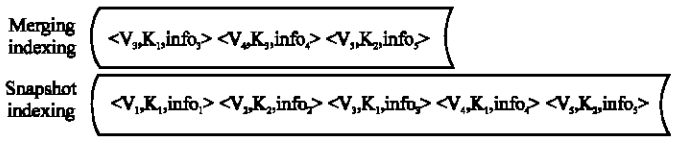

Fig. 1: Snapshot merging process in the indexing structure

ALGORITHMS

We introduce 5 parameters to control the merging window size for each layer indexing structure. They are {r, $N_{sum}(i)$, $N_{disc}(i)$, R(i), β_0}: r is the bitmap vector for the new incoming block; i is the layer number, $i \geq 0$; $N_{sum}(i)$ is the ith layer's window size and $N_{disc}(i)$ is the ith layer's distinct logical address block in $N_{sum}(i)$; R(i) is the current window blocks set bitmap vector; β_0 is a predefined IR value. If $N_{sum}(i)/N_{disc}(i) \geq \beta_0$, the merging process happens. Without loss of generality, we set each layer's predefined IR the same value. The concrete parameters setting rules and algorithm are listed as follows.

Rule 1. Parameters setting rule:

$N_{sum}(i) = N_{sum}(i) + 1$;
if ($r \notin R$)
{$R(i) = R(i)$ OR r; $N_{disc}(i) = N_{disc}(i)+1$;}

Rule 2. Parameters resetting rule:

$N_{sum}(i) = 0$; $N_{disc}(i) = 0$; $r(i) = \{0\}$

Algorithm 1: BVSM algorithm

1. Compute the new incoming block x bitmap vector, f(x) = r;
2. while($r \notin R$){
3. Set i layer's parameters as Rule 1;
4. $N_{sum}(i) / N_{Disc}(i) \rightarrow \beta'$;
5. if($\beta' \geq \beta_0$) {
6. Generate upper layer indexing structure through snapshot merging process;
7. Reset ith layer's parameter as Rule 2;}//end if
8. $i+1 \rightarrow i$;}//end while

When a new entry is incoming in ith layer merging window, the $N_{sum}(i)$ increases 1, if the entry is not an redundant one, $N_{disc}(i)$ increases 1. When the current IR reaches the fixed value, $N_{sum}(i)/N_{disc}(i) \geq \beta_0$, the new layer structure is created. After ith layer's window merged to (i+1)th layer, the parameters are reset and prepare for the new counting. The concrete algorithm is shown in algorithm 1. We name it Bitmap Vector Snapshot Merging (BVSM) algorithm.

The BVSM algorithm includes two parts: IR computing and snapshot merging. After the new incoming block's bitmap vector r is computed, it can be written to storage medium directly. The following IR computing and snapshot merging process could be done by r. The IR is computed in each layer's merging window, until the indexing redundant happens as step 2. When a window's IR reaches β_0, the merging process begins by scanning the merging window and the merged result is copied to the upper layer's current window. This process is done continuously and the hierarchical indexing structure is built as Fig. 2.

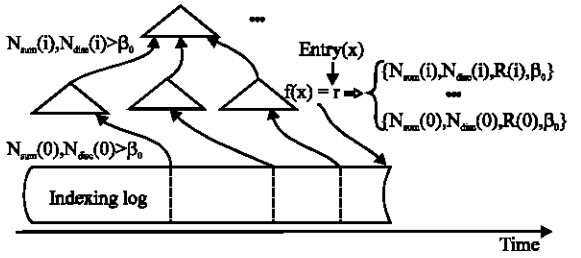

Fig. 2: Hierarchical indexing structure built by snapshot merging algorithm using bitmap vectors

Algorithm 2: BVSM algorithm with window thresholds

1. Compute the new incoming block x bitmap vector, f(x) = r;
2. while($r \notin R$){
3. Set i layer's parameters as Rule 1;
4. if($N_{sum}(i) < MIN\_NODE\_SIZE$)
5. $i+1 \rightarrow i$; continue;
6. $N_{sum}(i)/N_{Disc}(i) \rightarrow \beta'$;
7. if($\beta' \geq \beta_0$ or $N_{sum}(i) > MAX\_NODE\_SIZE$ and $\beta' > 1.0$) {
8. Generate upper layer indexing structure through snapshot merging process;
9. Reset ith layer's parameter as Rule 2;}//end if
10. $i+1 \rightarrow i$;}//end while

The algorithm 1 gives the basic ideas of the dynamic merging process. In practice, the IR might reach the predefined value within just a few entries and it will lead to frequent merging process. We introduce MIN_NODE_SIZE and MAX_NODE_SIZE for threshold values to avoid the extreme cases. The MIN_NODE_SIZE is a lower-bound for the merging window size, until $N_{sum}(i)$ is bigger than MIN_NODE_SIZE, the merging process can be done by BVSM algorithm. The MAX_NODE_SIZE is an upper-bound threshold, even if current window's IR is smaller than β_0, the merging process also begins to avoid the infinite enlarging window size. When BVSM algorithm introduces threshold values, it can be implemented in algorithm 2.

PERFORMANCE ANALYSIS

In traditional temporal and spatial indexing structure, the data lifetime information is recorded in the entry (Salzberg *et al.*, 2004) and the snapshot query process is to retrieve all the entries still alive in the lifetime. The lifetime setting could burden the indexing structure maintenance in the long-live snapshot replicated system. We use the same idea as skippy (Shaull *et al.*, 2008) and the snapshot query is implemented based on merging process. A version tree is used to describe the query process clearly. Each version of the merged snapshot by BVSM is denoted as a node $T_{[i,j]}$, i represents the layer number and j represents the version number. The source node is denoted by double-rings which records snapshot

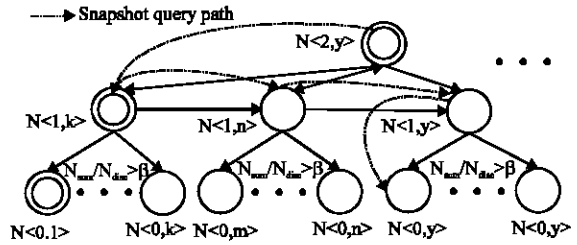

Fig. 3: Block-level snapshot query process in a version tree

from the initial time. The root node's version number equals the newest version number of its children nodes. Each node keeps two pointers for its children nodes (if i>0) and one pointer to its right-cousin node. The snapshot query process can be done from top to down iteratively, as shown in Fig. 3 and the traveled nodes have to be merged to target version of snapshot.

According to algorithm 1 and 2, the IR of each node in the version tree is controlled under some fixed value and it can avoid scanning the redundant entries in the snapshot query path. Some detailed performance analysis can be deduced from following lemmas.

Lemma 1: In BVSM algorithm, each merging window's IR is not bigger than β_0+1 ($\beta_0>1$), β_0 is a predefined IR value.

Proof: For merging consistency, the indexing merging process happens from bottom to up. It leads to IR increasing in the upper layer indexing structure. Suppose the merging process happens between the (i-1)th and ith layers (i>0). When the first merging process happens, it just copies the distinct block entries in (i-1)th layer's window to the upper layer. $N_{sum}(i)/N_{disc}(i) = 1<\beta_0$ and the lemma 1 is satisfied.

Let after k times merging process, the ith layer IR satisfied $N_{sum}(i)`/N_{disc}(i)`<\beta_0$. In the (k+1)th merging process, the ith layer's parameters can be denoted by:

$$N_{sum}(i) = N_{sum}(i)` + N_{disc}(i-1) \quad (2)$$

$$N_{disc}(i) = \text{Distinct}\{N_{disc}(i)`, N_{disc}(i-1)\} \quad (3)$$

$$\beta = N_{sum}(i)/ N_{disc}(i) \quad (4)$$

$N_{sum}(i)`$ and $N_{disc}(i)`$ denote i layer's parameters before merging process. β is current IR after the (k+1)th merging process. The β could reach its maximum value when Distinct$\{N_{disc}(i)`, N_{disc}(i-1)\}$ reaches its minimum value. At that time, one set is just included in another set. We suppose, $N_{disc}(i)`$ is included in $N_{disc}(i-1)$ and then $N_{disc}(i)` \le N_{disc}(i-1)$. According to Eq. 2-4:

$$\beta \le (N_{sum}(i)` + N_{disc}(i-1))/N_{disc}(i-1) < \beta_0 + 1 \quad (5)$$

So, the lemma 1 is satisfied. If current merging window's IR reaches β_0, the merging process happens and the ith layer's parameters are cleared by rule 2.

Lemma 2. The hierarchical indexing structure created by BVSM uses O(n) space.

Poof: Let the space efficiency is denoted by S and

$$S = \sum_{i=0}^{k} N_{sum}(i) \quad (6)$$

According to lemma 1, the space efficiency can be written as:

$$S = (n+n/\beta+n/\beta^2+ \ldots+n/\beta^k) = O(\beta/(\beta-1)n) = O(n) \quad (7)$$

In Eq. 7, n is the number of updating data and $\beta=\beta_0 + 1$. The indexing structure generated by BVSM is space efficient using O(n) space and the coefficient is $(\beta_0+1)/\beta_0$ (β_0 is the predefine IR value).

RESULTS AND DISCUSSION

We use cello99 trace to simulate block-level workloads. Cello99 is a disk-level I/O trace collected under Unix at HP lab. We replay the former 20 h trace of the cello99-03-03.

IR with write-skew: We first give IR variation with the write-skew in the simulated process. IR is different between disks. For example in disk0 the IR can reach 10 times at the end of 20 h and disk1 just only reaches 4 (Fig. 4). At the same time, IR is not monotonically increasing with the merging window size in the same disk. As shown by Fig. 5, the IR even decreases in disk0 when the merging window size enlarges.

The write-skew exists between disks and even in the same disk at different time. The fixed-size windows algorithm can not control the skew situation and it leads to IR difference in each layer's merging window, it is clearly shown by Fig. 6 and it increases not only indexing metadata and but the IR in the snapshot query path.

BVSM algorithm performance evaluations: We further carried out the detailed evaluations between the fixed-size window method and our BVSM algorithm. We select disk2 trace from 00:00 to 18:00 h in cello99-03-03.

In our experiment, we used algorithm 2 to implement BVSM. The windows size thresholds were 8 and 64 k and

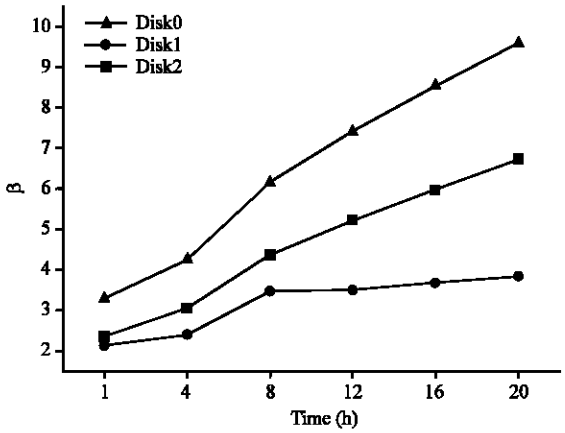

Fig. 4: IR variation between disks

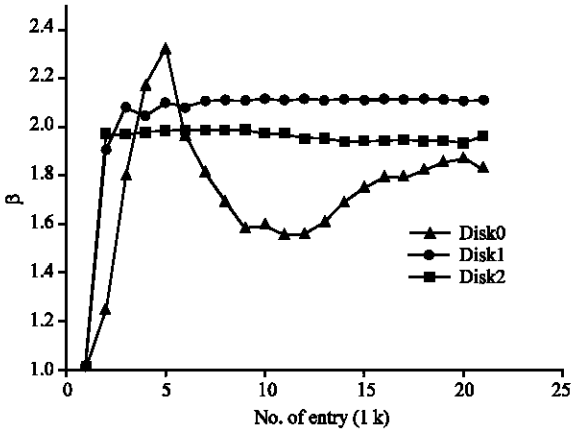

Fig. 5: IR variation with window size

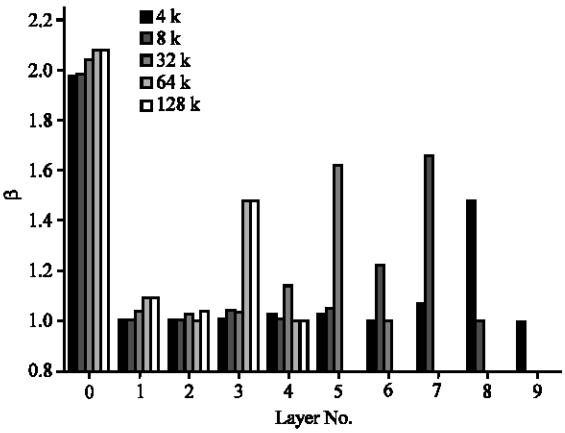

Fig. 6: IR variation in each layer indexing structure using fixed-size window algorithm

the predefined IR was 1.8. At the same time, we also gave the fixed window size algorithm under 8 and 64 k, respectively. We compared the IR and average windows

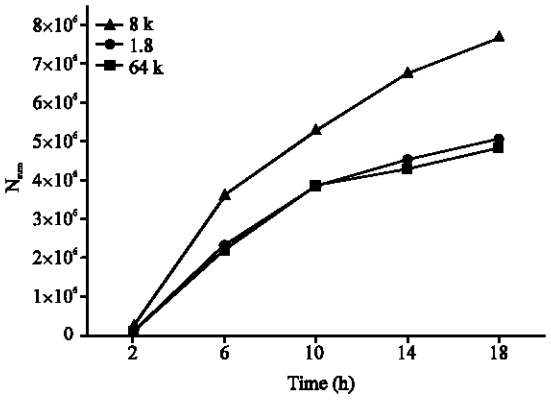

Fig. 7: N_{sum} comparisons

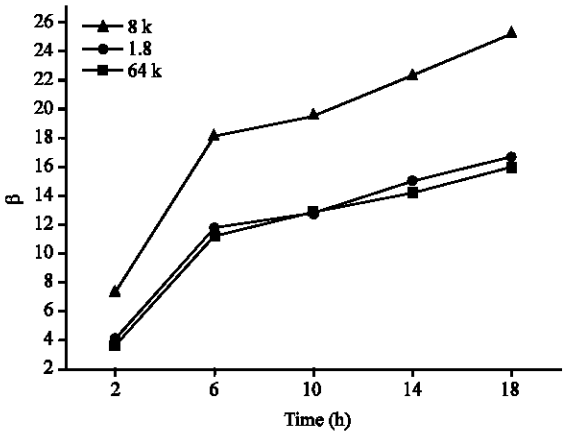

Fig. 8: IR comparisons

size in different algorithms. The larger windows size could achieve better merging performance. The total number of indexing metadata (N_{sum}) was smallest in 64 k. The BVSM algorithm was closer to its upper-bound 64 k and it just increased 8% metadata at the end of the simulation. Figure 7 shows the N_{sum} comparison at each time point and Fig. 8 further gives the IR comparison in this course.

We use wait-1 method in the fixed windows size structure. When the windows size reaches to the fixed window size, it waits for the first redundant entry encounter before merging current windows. In Skippy (Shaull *et al*., 2008) each merged window is called node and the length of node is called node size. We use the same name as Skippy. Figure 9 shows the total number of nodes comparison between wait-1 mechanism and BVSM algorithm. The number of node reflects the snapshot merging times in the snapshot replicated process. The BVSM generated the mean the node number between the two extremes and its average window size was closer to 8K threshold. As shown in Fig. 10, the BVSM average windows size reduction was 70% compared to 64 k

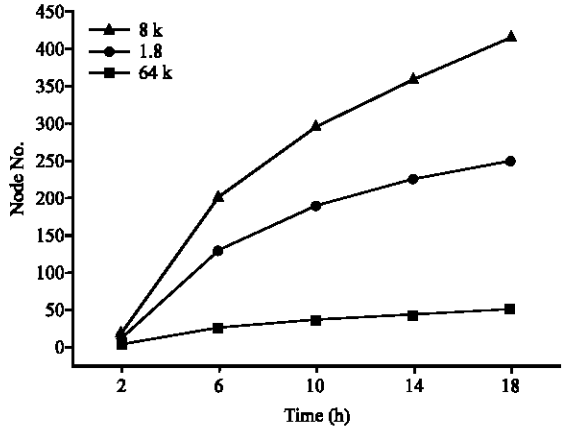

Fig. 9: Node no. comparisons

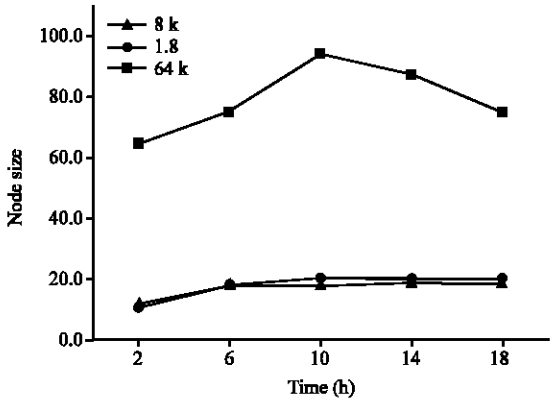

Fig. 10: Node size comparisons

Table 1: The indexing layer number comparison

Algorithm	1	6	10	14	18
	----------(h)----------				
64 k	1	4	5	5	5
8 k	3	7	7	8	8
1.8	1	4	5	5	5

threshold at the end of 18 h. Table 1 lists the indexing layers number. The BVSM creates the same indexing layer as 64 k window size.

The BVSM algorithm can reduce merging window size in the long-lived frequent block-level snapshot and generates less indexing metadata. The most important thing is that it ensures the redundant indexing entries ratio under some fixed value in a snapshot query process.

CONCLUSION

The Indexing Redundancy (IR) caused by disk access locality has impact on the long-lived block-level snapshot query performance. We present a novel indexing method which uses the multi-version bitmap vectors to control the merging window size and generates hierarchical indexing structure. The new algorithm can generate less indexing metadata and accelerates merging-based snapshot query by eliminating the redundant entries on the query path. The algorithm could potentially be adapted to the long-lived block-level snapshot indexing management in nowadays block to block storage network.

ACKNOWLEDGMENTS

This study was supported by the National High-Tech Research and Development Plan of China under Grant No.2009AA01A403, 2007AA01Z406, 2009AA01Z437 and the 973 National Basic Research Program of China under Grant No. 2007CB3111003.

REFERENCES

Becker, B., S. Gschwind, T. Ohler, B. Seeger and P. Widmayer, 1996. An asymptotically optimal multiversion B-tree. Int. J. Very Large Data Bases, 5: 264-275.

Charles, B.M. and D. Grunwald, 2003. Peabody: The time traveling disk. Proceedings of 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, April 7-10, IEEE Computer Society, pp: 241-253.

Lomet, D. and B. Salzberg, 1989. Access methods for multiversion data. Proceedings of the 1989 ACM SIGMOD International Conference on Management of data, May 31-June 2, Portland, Oregon, United States, pp: 315-324.

McKnight, J., T. Asaro and B. Babineau, 2006. Digital archiving: End-user survey and market forecast 2006-2010. The Enterprise Strategy Group.

Patterson, H., S. Manley, M. Federwisch, D. Hitz, S. Kleiman and S. Owara, 2002. Snapmirror: File system based asynchronous mirroring for disaster recovery. Proceedings of the Conference on File and Storage Technologies, Jan. 28-30, Usenix Association, pp: 117-129.

Peterson, Z. and R. Burns, 2005. Ext3cow: A time-shifting file system for regulatory compliance. ACM Trans. Storage, 1: 190-212.

Salzberg, B. and V.J. Tsotars, 1999. Comparison of access methods for time-evolving data. Comput. Surveys, 31: 158-221.

Salzberg, B., L. Jiang, D. Lomet, M. Barrena, J. Shan and K. Evangelos, 2004. A framework for access methods for versioned data. Lecture Notes Comput. Sci., 2992: 730-747.

Santry, D.S., M.J. Feeley, N.C. Hutchinson and A.C. Veitch, 1999. Deciding when to forget in the Elephant file system. Operat. Syst. Rev., 33: 110-123.

Shaull, R, L. Shrira and H. Xu, 2008. Skippy: A new snapshot indexing method for time travel in the storage manager. Proceedings of the 2008 ACM Sigmod International Conference on Management of Data, June 09-12, Vancouver, Canada, pp: 637-648.

Shrira, L. and H. Xu, 2005. SNAP: Efficient snapshots for back-in-time execution. Proceedings of the 21st International Conference on Data Engineering, April 5-8, IEEE Computer Society, pp: 434-445.

Shrira, L. and H. Xu, 2006. Thresher: An efficient storage manager for copy-on-write snapshots. Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference, May 30-June 03, USENIX Association, pp: 57-70.

Smeulders, A.W.M., M. Worring, S. Santini, A. Gupta and R. Jain, 2000. Content-based image retrieval at the end of the early years. IEEE Trans. Patt. Anal. Mach. Intell., 22: 1349-1380.

Sudhamani, M.V., C.R. Venugopal, 2008. Multidimensional indexing structure for content-based image retrieval: A survey Int. J. Innovative Comput. Inform. Control, 4: 867-881.

Tsotras, V.J. and N. Kangelaris, 1995. The snapshot index-an I/O-optimal access method for timeslice queries. Inform. Syst. J., 20: 237-260.