Performance Evaluation of SSD-Index Maintenance Schemes in IR Applications

Du-Seok Jin, Hoe-Kyung Jung, *Member, KIMICS*

***Abstract*— With the advent of flash memory based new storage device (SSD), there is considerable interest within the computer industry in using flash memory based storage devices for many different types of application. The dynamic index structure of large text collections has been a primary issue in the Information Retrieval Applications among them. Previous studies have proven the three approaches to be effective: In-Place, merge-based index structure and a combination of both. The above-mentioned strategies have been researched with the traditional storage device (HDD) which has a constraint on how keep the contiguity of dynamic data. However, in case of the new storage device, we don't have any constraint contiguity problems due to its low access latency time. But, although the new storage device has superiority such as low access latency and improved I/O throughput speeds, it is still not well suited for traditional dynamic index structures because of the poor random write throughput in practical systems. Therefore, using the experimental performance evaluation of various index maintenance schemes on the new storage device, we propose an efficient index structure for new storage device that improves significantly the index maintenance speed without degradation of query performance.**

***Index Terms*— SSD(Solid State Drive), Information Retrieval, Performance evaluation, Index Maintenance.**

I. INTRODUCTION

Flash-based storage devices are now considered to have great potential as a new storage device that can be substitute for magnetic disk in a diverse set of applications, such as enterprise database server, mobile personal computers and IR storage systems. It also might seem natural to achieve much higher performance for all the above applications but it is not, because new storage devices have much different properties from traditional magnetic disks. The key properties of SSD device that directly influence a performance are related to the method in which the device can be read or written. In the SSD, a page can only be written after erasing the entire block to which the page belongs. Page write cost is typically higher than read, and the block erase requirement makes writes even more expensive. Furthermore, when a small sized writes are requested in a random order we are more significantly confronted with the problem that the flash memory based device performs quite poorly.

In particular, since the previous strategies of dynamic index maintenance has been researched with the traditional magnetic storage device which has a constraint on how keep the contiguity of dynamic data. The existing schemes of index maintenance, which has proven to be effective for the large text collections in the IR applications, are not well suited for the flash memory based new storage device. Thus, to obtain the best attainable performance on SSD, elaborate SSD-aware index structures and algorithms of large text data collections are needed in practical systems. In this paper, we evaluate the effect of properties which influence on maintain index structure and propose an efficient index structure for new storage device.

Manuscript received July 1, 2010; accepted July 1, 2010.

Du-Seok Jin is with the Department of Information Technology Research, Korea Institute of Science and Technology Information, Daejeon, 305-806, Korea (Email: dsjin@kisti.re.kr)

Hoe-Kyung Jung(Corresponding author) is with the Department of Computer Engineering, Paichai University, Daejeon, 302-735, Korea (Email: hkjung@pcu.ac.kr)

II. RELATED WORKS

In this section we review the related works on the techniques which were used extensively in the index maintenance and the optimizing the random writes on flash memory based storage.

A. Index Schemes in IR Applications

Inverted index structures [1],[2] have become the most efficient data structure for high performance indexing of large text collections, especially online index maintenance, In-place and merge-based index structures are the two main competing strategies for index construction in dynamic search environments [3]. In the strategies, a contiguity of posting information is the mainstay of design for online index maintenance and query time. During the merging events, the In-place and merge-based strategies can either maintain each inverted list in a single contiguous disk location, or to allow it occupy multiple, discontiguous locations [4],[5].

The In-place update is an intuitive and very simple index update strategy. Whenever a document is arrived to the system, the posting list for each term is appended to the main index. In general this requires relocating existing on-disk postings list for each term in a new document. Here, since a new document will usually have hundreds of unique terms, this kind of in-place update require hundreds of time-consuming relocations of postings list in the main index. The frequent relocations of postings list in the main index will degrade the update performance seriously. Overallocation of postings list – which leaves some amount of free space after every postings list – in the main index, is proposed to relieve these costly operations. However, overallocation size is difficult to predict in real databases and overallocation can be degenerated into garbage if there is few update of addition of documents, or falls into same situation of no-overallocation after a huge amount of documents are inserted hence filling over-allocated free spaces in the posting list. Therefore we do not regard the overallocation strategy in this experiment.

The Merge-based strategies were introduced for incoming documents to reduce the relocation problem. Whenever a new document is arrived, the indexing results are appended to the auxiliary (in-memory) index rather than to the main (on-disk) index. At initial state, the auxiliary index is empty or small, hence new postings lists from the incoming documents will be short and can be easily inserted to the auxiliary index. However as new documents are accumulated, postings lists in the auxiliary index grow in size resulting in deteriorated performance due to frequent relocations of grown postings lists. Therefore, to be small enough auxiliary index, every auxiliary postings list are merged into the main index or the posting list which longer than a given threshold is merged into the main index. This way, there is no postings list longer than the given threshold and also the total size of auxiliary index do not exceed the defined limits. As a result, updating of postings lists for every unique term in a document can be very small compared with the direct In-place update index strategy.

B. Optimizing the random writes on SSD Storage

Flash-based storage devices are now becoming available with price and performance characteristics and these are considered to have tremendous potential as a new storage medium that can replace magnetic disk for enterprise database server. The new storage devices have much different properties from traditional magnetic disks. The key characteristics of SSD device that directly influence the performance are related to the method in which the device can be read or written. All read and write operations are performed at page granularity and a page can only be written after erasing the entire block to which the page belongs. Page write cost is typically higher than read, and the block erase requirement makes writes even more expensive. Especially small random writes, on the device are inherently much slower than reads because of the erase-before-write mechanism [6]. To avoid performance degradation caused by this erase-before-write limitation, The In-Page Logging (IPL) approach that allows the changes made to a page are written (or logged) to the database on the per-page basis, instead of writing the page in its entirety was proposed [7] and The BFTL was proposed to improve the inferior random write performance [8]. However, BFTL entails a high search cost since it accesses multiple disk pages to search a single tree node. Furthermore, the memory consumption is still high for large trees. Flash DB was proposed to implement a self-tuning scheme between standard B+-tree and BFTL under various workloads on different flash SSDs [9].

III. DESIGN PROPERTIES OF INDEX STRUCTURE ON SSD

There are several considerations in designing a structure of index maintenance for retrieval system. The appropriate indexing scheme and its configuration parameters must be selected based on the new storage device properties. In this section we discuss the design properties that affect an index maintenance performance on SSD.

A. Segmentation (Partitioning)

As inverted lists grow, if insufficient free space is available to append new information at the end of the list, the combined list is copied to a new location and the original list is deleted. Most of the strategies based on In-place updates have assumed that each posting lists have to be in a contiguous part of the on-disk index. Keeping posting lists in a contiguous part of the inverted file maximizes query performance, but requires frequent relocations of most lists in the index. Relocating inverted lists means destroying the original ordering of the lists on disk and also the disordered lists make update costs even more expensive. To avoid this problem, researchers have proposed a variety of different strategies based on In-place scheme to updates index which is composed of a set of index partitions. However, query performance degrades with the number of index partitions because of processing overhead associated with each partition. For these reasons, many index maintenance studies indicating that the merge-based approach is usually more efficient than in-place update.

In contrast, if the index partitions were stored on an SSD, the query performance would still maintain its advantage on In-pace updates because the cost of a seek time would drop to nearly zero and also keep its superior performance of index maintenance. Consequently, the principle property for the SSD to be available at the server storage is that the index is divided into a number of partitions. In other words, the presence of multiple partitions for on-line index construction is faster than methods based on a single partitioning, while the query performance is almost similar to that of single partition approaches. Accordingly, we will examine the effectiveness of multi-partitioning property in inverted lists. Here, we refer to this property as the '*segmentation*'.

B.Merging Extent/Period

Merge-based update strategies share the common idea that disk read/write operations are most efficient when they are carried out in a sequential manner, minimizing disk head movement. In merge-based strategies, recently updated documents are typically gathered in-memory area which is a subset of the indexed documents. Whenever main memory is exhausted, the in-memory postings are merged with the existing on-disk index. However, the main problem of merge-based strategies is that whenever main memory is exhausted, the entire on-disk index has to be re-processed even though they have not been changed. To overcome the shortcoming, Büttcher and Clarke and Lester et al. have proposed [10],[11],[12] merge-based update strategies that indexing efficiency is greatly increased, while query processing performance remains almost unchanged by allowing a controlled number of on-disk indices to exist in parallel.

However, when the inverted index were stored on the SSD, the advantage of merge-based strategy was an insignificant matter because the cost of a seek time on SSD would be nearly zero. Therefore, in this work, the merging extent (what is value for a merging extent selected?) and period (when is time for a merging period selected?) are more important property than how to optimize the merging methods. Accordingly, we evaluate the performance of index maintenance using two kinds of merging extents: the one is to be merged each posting list between the same terms like a common merge method, the other is to be merged as a single physical unit which combined all posting lists in a merge index. In this paper, we refer to the former as '*Sot-merging*' and the latter as '*Sod-merging*'.

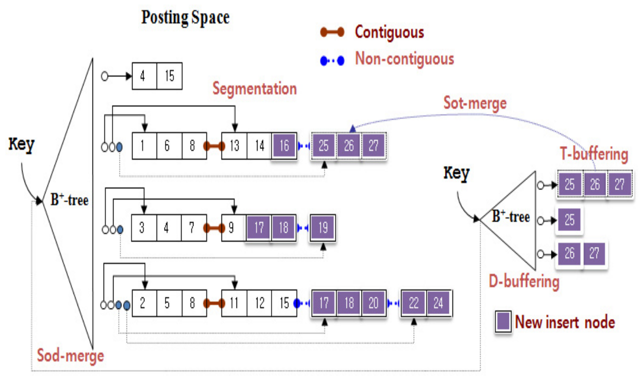

Fig. 1. The inverted index design properties for index maintenance on SSD.

Another important design property of the merge-based approach is a buffering scheme. In this paper, three kinds of merging period's schemes are explored: The first, when the size of posting list of certain term is above the predefined threshold, the posting list of term has to be appended to the end of the term's posting list of on-disk index immediately. We refer to the approach as '*T-buffering*'. The second is similar with general merge-based approach but it has been composed of multi segmentation which is referred as '*D-buffering*'. The last is combined of '*T-buffering*' and '*D-buffering*'. We refer to this approach as '*TD-buffering*'. Those approaches decrease the number of merging events.

C. Transactional Index Maintenance

Since most of the index maintenance strategies lack the concept of integrity constraint, they have opportunities to regard a block of new documents – for efficiency – as a unit of index maintenance. However, to meet the field requirements from database managers in text service area, index maintenance should be supported in the on-disk storage level, not in-memory. This means the logical unit of index maintenance process should consist of each document (not multiple documents) and its accompanying index information. Furthermore, to support ACID property of database systems, an IR system must support logged processing of index maintenance. We have tested the index maintenance strategies for per-document basis regarding each document and its index as a transaction unit.

For all experiments in this work, every change in index data is logged to disk in per-document basis transaction unit. If the transaction –i.e., insertion of a new document- is successful, the log is discarded. On the other hand, if the transaction fails for some reason, the log is used to roll-back to the state of just before the transaction.

Figure 1 shows the inverted index structure with segmented posting lists and an on-disk auxiliary index. In addition, it depicts segment size (DF=3), merge extent and merge period of posting lists. The black node means new posting information which contains term frequency, location and identifier of a document. As shown, initial nodes were stored in the contiguous region but appended nodes would be stored in the separated regions even same term.

IV. PERFORMANCE EVALUATION

In this section we evaluate how well the index schemes adapt with above mentioned properties on SSD. To evaluate this, we experimented with the USA patent data, and measured index maintenance times and querying times for each properties within transactional environment. Experiments were performed on a dual Pentium Xeon 3GHz machine with 8GB of memory and RAID-5 SCSI storage and Intel-X25 80G SSD storage device.

Figure 2 shows an example patent item which consist of Number(Patent number), Title, Pubdate(publish date), Inventors, Assignee, Abstract and Claims.

```
<us-patent>
   <number> US-6981282 </pat-number>
   <title> Systems and methods for transformable suits
   </title>
   <pub-date> 20060103 </pub-date>
   <inventors>
      <name> Marty,Justin Douglas </name>
      <name> Pace,Joshua Craig </name>
   </inventors>
   <assignee> Z Gear, Inc. </assignee>
   <abstract> Methods and systems for transforming a
   volume of material into pant legs by means of a
   transforming fastener. The transforming fastener has
   multiple tracks, each track having a pair of matable
   rows. ⋯
   </abstract>
   <claims> 1. A system comprising: a volume of
   material having a first formation wherein the volume
   is substantially undivided by the material, ⋯
    2. The system of claim 1 wherein said second
   formation comprises pant legs, said pant legs further
   comprise inseams, and said tracks are coupled to the
   inseams. ⋯
   </claims>
</us-patent>
```

Fig. 2. A sample data in USA patent collection.

Table 1 shows average term counts in every section in the USA patent data collection. Remind that to keep the lexicographical order of the terms in an inverted file, usually a vocabulary data structure is separated from the posting lists which have document identifiers with term frequencies and exact locations in the documents. Now refer to table 1 where a document contains more than one sixteen hundreds terms, which means that addition of a document should access the on-disk index more than two times of the term frequency since the inverted index should access two disk data structures – the vocabulary list and posting list – for each term from the input document.

TABLE I
BRIEF DATABASE SCHEMA AND STATISTICS OF INDEX TERMS PER-DOCUMENT OF PATENT DATA

Field	Data Type	Index Type	Total Terms/doc
Number	char[20]	index as is	1.0
Title	string	token	14.2
PubDate	char[8]	Index as is	1.0
Inventors	string	token	10.8
Assignee	String	token	3.57
Abstract	string	token	67.9
:	:	:	:
Claims	string	token	626.9
SUM (90) fields			1601

For the segmentation strategies, we compared insert times of pouring 35K documents with the single in-place strategy and the segmentation in-place strategy. Each point in Figure 3 was obtained by averaging insertion times of every 1K insertions, to reduce the effects of the biases in document lengths. In Figure 3, the upper line indicates that an in-place scheme with a single contiguous region for a long posting list and the bottom line writes a long posting list into multiple segmentations. For the experiment, document frequency of 1000 was chosen for threshold of segmentation, since it balances the trade-off between query processing and index maintenance. As mentioned *Section 3.A*, Figure 3 shows non-segmentation index strategy is very poor compared with segmentation approach. This mean a single contiguous posting list is not feasible for even in very small databases. On the other hand, segmentation index strategy can improve the update efficiency. Furthermore this strategy shows nearly stable performance though many inserts are done.

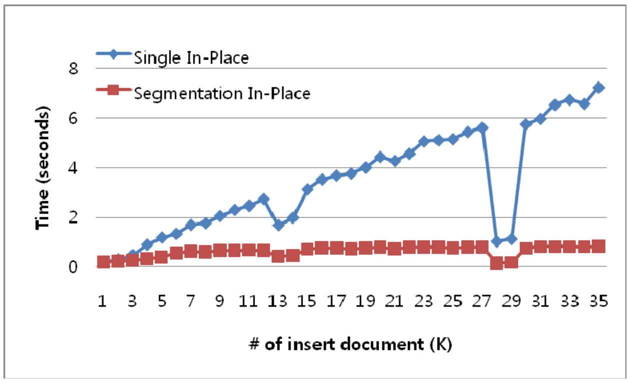

Fig. 3. Insert times for 35,000 input documents on SSD.

For the merge extent and merge period, our experiments were done by '*Sot-merging*' and '*Sod-merging*' approach, respectively. The '*Sot-merging*' was to be merged each posting list between the same terms and the '*Sod-merging*' was to be merged as a single physical unit which combined all posting lists in a merge index. Figure 4 shows the result that '*Sod-merging*' is faster than '*Sot-merging*' to merge posting lists. But the query performance of '*Sod-merging*' is slightly slower than '*Sot-merging*' approach, since the pointers of each term' posting list must be recomputed each time.

As mentioned *Section 3.B*, another important design property of the merge-based approach is how open executed the merging events. To evaluate the characteristic, we experiment an impact of varying buffering scheme with a buffering threshold (≈segment size) on index maintenance times and querying times.

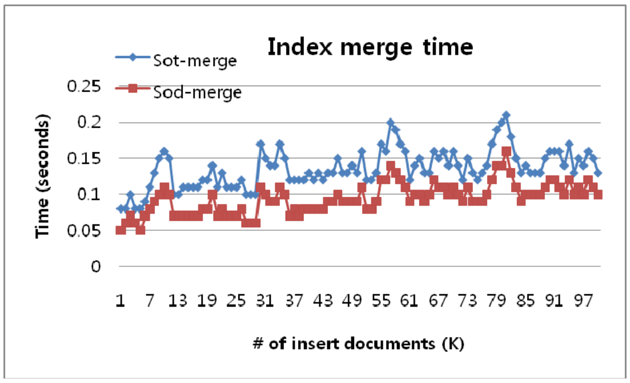

Fig. 4. Merge times for 100,000 input documents on SSD.

Figure 5 shows that *D-buffering* significantly reduces the time of index maintenance compared to *T-buffering* approach. This is due to the decreasing number of merging events. Figure 5 also shows that *TD-buffering* is about two times faster than the *T-buffering* but slightly slower than the *D-buffering* approach. However the query times of *TD-buffering* and *T-buffering* are faster than *D-buffering* approach, because *T/TD-buffering* approaches have an advantage, like as distributed processing.

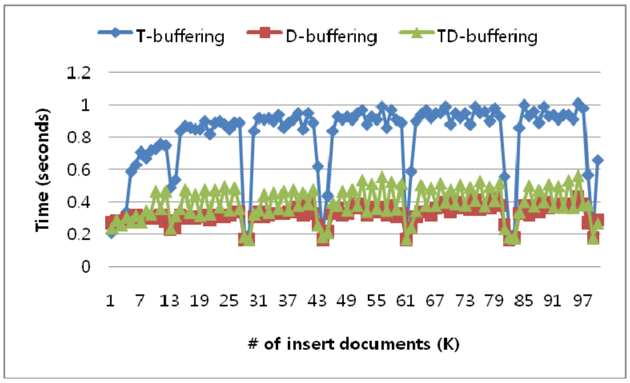

Fig. 5. Insert times for three buffering schemes on SSD.

Finally, we evaluate the query performance of four different index maintenance schemes with two kinds of devices. Several examples of the complex queries are shown below.

- yellow* /N8 (polyurethane* OR urethane*)
- W silicon AND (optic* /N8 signal*) AND module*
- W food* /N3 (wastewater* OR (waste /W1 water*)) AND treat*
- W ceramic* AND (bulletproof* OR (bullet /W1 proof*) OR (bullet /W1 resist*) OR (bullet* /N2 (protect* OR resist*)))
- wood* /N5 (substitut* OR replacement*)
- W (catalyst* OR catalyzer*) /N5 (regenerat* OR ((precious OR valu* OR noble*) /N2 metal* /N5 recover*))

Figure 6 show that the query performance maintained a constant speed for all methods when we used SSD. But, the query performance with the HDD shows quite different results according to the variation of index methods, since various access latency times for each method.

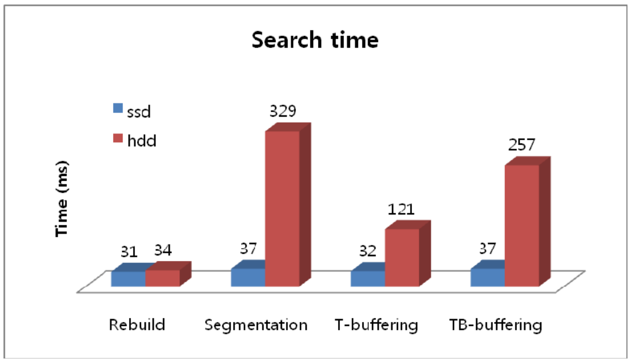

Fig.6. Search times for the input documents on SSD and HDD

IV. CONCLUSIONS

The objective of this work is to evaluate flash memory based storage device (SSD) as stable storage for Information Retrieval Applications. In particular, for the dynamic index maintenance, we identified what is the property that strong effect on the performance and how the property affects the dynamic indexing schemes. To maximize the benefit from the new technology, we also proposed an efficient index structure for improved performance based on SSD-index maintenance scheme.

REFERENCES

[1] R. Baeza-Yates, B. Ribeiro-Neto, *Modern Information Retrieval.* Addison-Wesley, Reading, MA, 1999.
[2] H. Witten, A. Moffat, C. Bell, *Managing gigabytes: compressing and indexing documents and images.* Morgan Kaufmann, Los Altos, CA 94022, USA, second edition, 1999.
[3] N. Lester, J. Zobel, and H. Williams, "In-place verse re-build verse re-merge: Index maintenance strategies for text retrieval systems," *Proc. CRPIT 27th Australasian Computer Science Conference* pp. 15-23, 2004.
[4] E. Brown, J. Callan, and W. Croft, "Fast incremental indexing for full-text information retrieval," *Proc. VLDB,* pp. 192-202, Sep. 1994.
[5] H. Tomasic, Garcia-Molina, and K. Shoens, "Incremental updates of inverted lists for text documents retrieval," *Proc. ACM SIGMOD ,* pp. 289-300, May. 1994.
[6] Andrew Birrell, Michael Isard, Chuck Thacker, and Ted Wobbe, "A Design for High-Performance Flash Disks," *Technical Report* MSR-TR-2005-176, Microsoft Research, Dec. 2005.
[7] Sang-Won Lee, and Bongki Moon, "Design of Flash-Based DBMS: An In-Page Logging Approach," *Proc. ACM SIGMOD,* pp. 55-66, Jun. 2007.

[8] C. H. Wu, L. P. Chang, and T. W. Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," *Proc. RTCSA,* pp. 409-430, 2004.

[9] S Nath, and A Kansal, "FlashDB: Dynamic Self-tuning Database for NAND Flash," *Proc. IPSN* 6*th Information Processing in Sensor Networks Conference*, Apr. 2007.

[10] N. Lester, A. Moffat, and J. Zobel, "Fast On-Line Index Construction by Geometric Partitioning," *Proc. ACM CIKM,* pp. 776-783, 2005.

[11] S. Buttcher, C.L.A. Clarke, and B. Lushman, "Hybrid index maintenance for growing text collections," *Proc. ACM SIGMOD,* pp. 1-4, 2004.

[12] N. Lester, A. Moffat, and J. Zobel, "Efficient Online Index Construction for Text Database," " ACM *Trans. Database Systems,* vol. 33, no. 3, article 19, Aug. 2008.

Du-Seok Jin received the B. S. and M. S. degrees in computer engineering from Chonbuk National University, Korea in 1999 and 2001, respectively. Since 2001, he has been worked as a researcher in the Korea Institute of Science and Technology Information. He is currently interested in Information Retrieval System, Dynamic Index Structure and Data Management

Hoe-Kyung Jung received B.S degree in 1987, and Ph. D. degree in 1993, in the Department of Computer Engineering from Kwangwoon University, Korea. During 1994- 2005, he worked for ETRI as Researcher. Since 1994, he has worked in the department of Computer Engineering at Paichai University where he now works as a Professor. His current research interests Multimedia Document Architecture Modeling, Information Processing, Web Services, Semantic Web, USN and MPEG-21.