SkipBlock: Self-Indexing for Block-Based Inverted List

Stéphane Campinas, Renaud Delbru and Giovanni Tummarello

Digital Enterprise Research Institute,
National University of Ireland, Galway
Galway, Ireland

Abstract. In large web search engines the performance of Information Retrieval systems is a key issue. Block-based compression methods are often used to improve the search performance, but current self-indexing techniques are not adapted to such data structure and provide sub-optimal performance. In this paper, we present SkipBlock, a self-indexing model for block-based inverted lists. Based on a cost model, we show that it is possible to achieve significant improvements on both search performance and structure's space storage.

1 Introduction

The performance of Information Retrieval systems is a key issue in large web search engines. The use of compression techniques and self-indexing inverted files [8] is partially accountable for the current performance achievement of web search engines. On the one hand, compression maximises IO throughput [3] and therefore increases query throughput. On the other hand, self-indexing inverted files [8] enables the intersection of inverted lists in sub-linear time.

Nowadays efficient inverted index compression methods tend to have a block-based approach [6,10,1]. An inverted list is divided into multiple non-overlapping blocks of records. The coding is then done a block at a time and independently from the others. While block-based coding approaches provide incontestable benefits, the self-indexing method [8] achieves only sub-optimal performance on block-based inverted lists. The reason is that the self-indexing technique disregards the block-based structure of the list which however can be used for designing a more efficient self-indexing as we will show in this paper.

In this paper we present an approach for self-indexing of block-based inverted lists. We demonstrate the benefits of our block-based self-indexing technique by comparing it against the original self-indexing approach based on a cost model. In Section 2 we first review the original self-indexing technique based on the Skip Lists data structure, before presenting in Section 3 our approach. Section 4 discuss the problem of searching within Skip Lists intervals. In Section 5 we define a cost model and compare four implementations of our model against the original model. In Section 6 we recall the main finding of the research and the task that remains.

1.1 Related Work

The Skip Lists data structure is introduced by [9] as a probabilistic alternative to balanced trees and is shown in [5] to be as elegant and easier to use than binary search trees. Such a structure is later employed for self-indexing of inverted lists in [8]. Self-indexing inverted list enables a sub-linear complexity in average when intersecting two inverted lists. [2] proposes a way to compress efficiently a Skip Lists directly into an inverted list and shows that it is possible to achieve a substantial performance improvement. In [4], the authors introduce a method to place skips optimally given the knowledge of the query distribution. [7] presents a generalized Skip Lists data structure for concurrent operations. In this paper, we introduce a new model for self-indexing of block-based inverted lists which is based on an adaptation of the Skip Lists data structure. Our work is orthogonal to the previous works, since they might be extended to our model.

2 Background: Self-Indexing for Inverted Lists

An inverted list is an ordered list of compressed records (e.g., documents identifiers). When intersecting two or more inverted lists, we often need to access random records in those lists. The basic approach is to scan linearly the lists to find them. Such an operation is not optimal and can be reduced to sub-linear complexity in average by the use of the self-indexing approach [8]. Self-indexing is based on Skip Lists to build a sparse index over the inverted lists and to provide fast record lookups. In this section, we first present the Skip Lists model and its associated search algorithm. We finally discuss the effect of the probabilistic parameter with respect to the Skip Lists data structure and search complexity.

2.1 The Skip Lists Model

Skip Lists are used to index records in an inverted list at regular interval. These indexing points, called *synchronization points*, are organized into a hierarchy of linked lists, where a linked list at level $i+1$ has a probability p to index a record of the linked list at level i. The probabilistic parameter p is fixed in advance and indicates the *interval* between each synchronization point at each level. For example in Figure 1, a synchronization point is created every $\frac{1}{p^1} = 16$ records at level 1, every $\frac{1}{p^2} = 256$ records at level 2, and so on. In addition to the pointer to the next synchronization point on a same level, a synchronization point at level $i+1$ has a pointer to the same synchronization point at level i. For example in Figure 1, the first synchronization point at level 3 (i.e., for the record 4096) has a pointer to the level 2 which has itself a pointer to the level 1. This hierarchical structure enables to quickly find a given record using a top-down search strategy.

Given the probabilistic parameter p and the size n of an inverted list, we can deduce two characteristics of the resulting Skip Lists data structure: (1) the expected number of levels and (2) the size, i.e., the total number of synchronization points. The number of levels in the Skip Lists is defined by $L(n) = \lfloor \ln_{\frac{1}{p}}(n) \rfloor$, which is the maximum as stated in [9]. The total number of synchronization points is given by $S(n) = \sum_{i=1}^{L(n)} \lfloor n \times p^i \rfloor$, which sums up the number of synchronization points expected at each level.

2.2 Skip Lists Search Algorithm

Searching a Skip Lists enables to retrieve an interval containing the target record and is performed using a top-down strategy. The search within an interval is discussed in Section 4. The search walk starts at the head of the top list, and performs a linear walk on a level as long as the target is greater than a synchronization point. The walk goes down one level if and only if the target is lower than the current synchronisation point. The search finishes when the current synchronization point is (a) equal to the target, or (b) on the bottom level and greater than the target. The reached interval then contains the target. The search complexity is defined by the number of steps necessary to find the interval holding the target. In the worst case, the number of steps at each level is at most $\frac{1}{p}$ in at most $L(n)$ levels. Consequently, the search complexity is $\frac{L(n)}{p}$.

Figure 1 depicts with a solid line the search path in a Skip Lists with $p = \frac{1}{16}$ and $L(n) = 3$ levels to the record 8195. At the top of the Skip Lists, we walk to the record 8192. Then we go down to level 1 and stop because the current synchronization point (i.e., the record 8208) is greater than the target. At this point, we know that the target record is in the next interval on the inverted list.

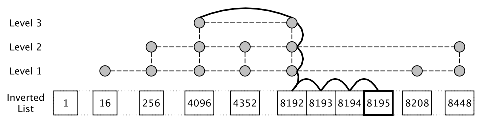

Fig. 1: Skip Lists with $p = \frac{1}{16}$. Dashed lines denote pointers between synchronization points. The solid line shows the search path to the record 8195.

2.3 Impact of the Probabilistic Parameter

In this section, we discuss the consequences of the probabilistic parameter on the Skip Lists data structure. Table 1a reports for low (i.e., $\frac{1}{1024}$) and high (i.e., $\frac{1}{2}$) probabilities (1) the complexity $\frac{L(n)}{p}$ to find the interval containing the target record, and (2) the size $S(n)$ of the Skip Lists structure. There is a trade-off to achieve when selecting p: a high probability provides a low search complexity but at a larger space cost, and a low probability reduces considerably the required space at the cost of higher search complexity. The SkipBlock model presents a way to reduce even more the search complexity in exchange of a larger data structure.

$\lvert I\rvert$	2	16	64	128	1024
$S(n)$	99 999 988	6 666 664	1 587 300	787 400	97 751
C	54	112	320	512	3072

(a) Skip Lists with $\lvert I\rvert = \frac{1}{p}$.

$\lvert I\rvert$	16		64		128		1024	
p:$\lvert B\rvert$	$\frac{1}{4}$:4	$\frac{1}{8}$:2	$\frac{1}{4}$:16	$\frac{1}{8}$:8	$\frac{1}{4}$:32	$\frac{1}{8}$:16	$\frac{1}{4}$:256	$\frac{1}{8}$:128
$S_B(n)$	8 333 328	7 142 853	2 083 328	1 785 710	1 041 660	892 853	130 203	111 603
C	48	64	44	56	40	56	36	48

(b) SkipBlock with $\lvert I\rvert = \frac{\lvert B\rvert}{p}$.

Table 1: Search and size costs of Skip Lists and SkipBlock with $n = 10^8$. $\lvert I\rvert$ stands for an interval length. C reports the search complexity to find an interval (Sections 2.2 and 3.2).

3 SkipBlock: A Block-Based Skip Lists Model

In this section, we introduce the SkipBlock model and present its associated search algorithm. Finally we discuss how the SkipBlock model offers finer control over the Skip Lists data structure in order to trade search cost against storage cost.

3.1 The SkipBlock Model

The SkipBlock model operates on *blocks* of records of a fixed size, in place of the records themselves. Consequently, the probabilistic parameter p is defined with respect to a block unit. A synchronization point is created every $\frac{1}{p^i}$ blocks on a level i, thus every $\frac{|B|}{p^i}$ records where $|B|$ denotes the block size. A synchronization point links to the first record of a block interval. Compared to Figure 1, a SkipBlock structure with $p = \frac{1}{8}$ and $|B| = 2$ also has an interval of $\frac{|B|}{p^1} = 16$ records. However, on level 2, the synchronization points are separated by $\frac{|B|}{p^2} = 128$ instead of 256 records. We note that with $|B| = 1$, the SkipBlock model is equivalent to the original Skip Lists model, and therefore it is a generalization. The number of levels is defined by $L_B(n) = \left\lfloor \ln_{\frac{1}{p}} \left(\frac{n}{|B|} \right) \right\rfloor$ and the size by $S_B(n) = \sum_{i=1}^{L_B(n)} \left\lfloor \frac{n \times p^i}{|B|} \right\rfloor$.

3.2 SkipBlock Search Algorithm

With the block-based Skip Lists model, the search algorithm returns an interval of blocks containing the target record. We discuss the search in that interval in Section 4. The search walk is identical to the one presented in Section 2.2: we walk from the top to the bottom level, and compare at each step the current synchronization point with the target. The walk stops at the same termination criteria as in Skip Lists. The search complexity in the worst case becomes $\frac{L_B(n)}{p}$.

3.3 Impact of the Probability and of the Block's Size

The SkipBlock model provides two parameters to control its Skip Lists structure: the probabilistic parameter p and the block size $|B|$. The block size parameter enables more control over the Skip Lists structure. For example, to build a structure with an interval of length 64, the original Skip Lists model proposes only one configuration given by $p = \frac{1}{64}$. For this same interval length, SkipBlock proposes all the configurations that verify the equation $\frac{|B|}{p} = 64$. Table 1b reports statistics of some SkipBlock configurations for the same interval lengths as in Table 1a. Compared to Skip Lists on a same interval length, SkipBlock shows a lower search complexity in exchange of a larger structure.

4 Searching Records in an Interval

The Skip Lists and SkipBlock techniques enables the retrieval of an interval given a target record. The next step is to find the target record within that interval. A first strategy (S1) is to linearly scan all the records within that interval until the target is found. Its complexity is therefore $O(|I|)$.

SkipBlock takes advantage of the block-based structure of the interval to perform more efficient search strategy. We define here four additional strategies for searching a block-based interval, parameterized to a probability p. The second strategy (S2) performs (a) a linear scan over the blocks of the interval to find the block holding the target and (b) a linear scan of the records of that block to find the target. The search complexity is $\frac{1}{p} + |B|$ with $\frac{1}{p}$ denoting the linear scan over the blocks and $|B|$ the linear scan over the records of one block. Similarly to S2, the third strategy (S3) performs the step (a). Then, it uses a inner-block Skip Lists structure, restricted to one level only, to find the target. The complexity is $\frac{1}{p} + \frac{1}{q} + \lfloor |B| \times q \rfloor$ with q the probability of the inner Skip Lists. In contrast to S3, the fourth strategy (S4) uses a non-restricted inner-block Skip Lists structure. The complexity is $\frac{1}{p} + \frac{L(|B|)+1}{q}$ with q the inner Skip Lists probability. The fifth one (S5) builds a Skip Lists structure on the whole interval instead of on a block. Its complexity is then $\frac{L\left(\frac{|B|}{p}\right)+1}{q}$, with q the inner Skip Lists probability. The strategies S3, S4 and S5 are equivalent to S2 when the block size is too small for creating synchronization points.

5 Cost-Based Comparison

In this section, we define a cost model that we use to compare five SkipBlock implementations and the original Skip Lists implementation.

Cost Model For both the Skip Lists and the SkipBlock, we define a cost model by (a) the cost to search for the target, and (b) the cost of the data structure's size. The search cost consists in the number of synchronization points traversed to reach the interval containing the target, plus the number of records scanned in that interval to find the target. The size cost consists in the total number of synchronization points in the data structure, including the additional ones for S3, S4 and S5 in the intervals.

Implementations We define as the baseline implementation, denoted I_1, the Skip Lists model using the strategy (*S1*). We define five implementations of the SkipBlock model, denoted by I_2, I_3, I_4, I_5 and I_6, based on the five interval search strategies, i.e., *S1*, *S2*, *S3*,*S4* and *S5* respectively. The inner Skip Lists in implementations I_4, I_5 and I_6 is configured with probability $q = \frac{1}{16}$. The inner Skip Lists in I_5 and I_6 have at least 2 levels. The size costs are $S(n)$ for I_1, $S_B(n)$ for I_2, $S_B(n) + \frac{n}{|B|}$ for I_3, $S_B(n) + \lfloor n \times q \rfloor$ for I_4, $S_B(n) + \frac{S(|B|) \times n}{|B|}$ for I_5 and $S_B(n) + \frac{S(p \times |B|) \times n}{p \times |B|}$ for I_6.

Comparison With respect to the SkipBlock model, we tested all the possible configurations for a given interval length. We report that all of them were providing better search cost than the baseline. We only report in Table 2 the configurations providing the best search cost given an interval length with the associated size cost. We observe that I_2 already provides better search cost than the baseline I_1 using the same search strategy *S1*, in exchange of a larger size cost. The other implementations, i.e., I_3, I_4, I_5 and I_6 which use a more efficient interval search strategies further decrease the search cost. In addition, their size cost decreases significantly with the size of the interval. On a large interval (512), I_4 is able to provide a low search cost (64) while sustaining a small size cost ($6.5e6$). The inner Skip Lists of

I_5 and I_6 are built on lists too small to provide benefits. To conclude, I_4 seems to provide a good compromise between search cost and size cost with large intervals.

$\|I\|$	8						16						512						1152					
	I_1	I_2	I_3	I_4	I_5	I_6	I_1	I_2	I_3	I_4	I_5	I_6	I_1	I_2	I_3	I_4	I_5	I_6	I_1	I_2	I_3	I_4	I_5	I_6
SC	72	56	54				112	62	56				1536	548	120	64	86	84	3456	1186	154	74	84	81
ZC$\times e6$	14.3	16.7	50.0				6.7	12.5	25.0				0.2	0.3	1.8	6.5	7.0	6.9	0.09	0.17	1.7	6.4	6.6	6.7

Table 2: Search (i.e., SC) and size (i.e., ZC, in million) costs with $n = 10^8$. SkipBlock implementations report the best search cost with the associated size cost.

6 Conclusion and future work

We presented SkipBlock, a self-indexing model for block-based inverted lists. The SkipBlock model extends the original Skip Lists model and provides a backbone for developing efficient interval search strategies. Compared to the original Skip Lists model, SkipBlock can achieve a lower search complexity and a smaller data structure size. In addition, SkipBlock allows finer control over the Skip Lists data structure and so additional possibilities for trading search costs against storage costs. Future work will focus on real world data benchmarks in order to assess the performance benefits of the SkipBlock model.

References

1. Anh, V.N., Moffat, A.: Index compression using 64-bit words. Softw. Pract. Exper. 40(2), 131–147 (2010)
2. Boldi, P., Vigna, S.: Compressed perfect embedded skip lists for quick inverted-index lookups. In: In Proc. SPIRE 2005, Lecture Notes in Computer Science. pp. 25–28. SpringerVerlag (2005)
3. Büttcher, S., Clarke, C.L.A.: Index compression is good, especially for random access. In: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management - CIKM '07. pp. 761–770. ACM Press, New York, New York, USA (2007)
4. Chierichetti, F., Lattanzi, S., Mari, F., Panconesi, A.: On placing skips optimally in expectation. In: WSDM '08: Proceedings of the international conference on Web search and web data mining. pp. 15–24. ACM, New York, NY, USA (2008)
5. Dean, B.C., Jones, Z.H.: Exploring the duality between skip lists and binary search trees. In: ACM-SE 45: Proceedings of the 45th annual southeast regional conference. pp. 395–399. ACM, New York, NY, USA (2007)
6. Goldstein, J., Ramakrishnan, R., Shaft, U.: Compressing relations and indexes. In: In proceedings of IEEE International Conference on Data Engineering. pp. 370–379 (1998)
7. Messeguer, X.: Skip trees, an alternative data structure to skip lists in a concurrent approach. ITA pp. 251–269 (1997)
8. Moffat, A., Zobel, J.: Self-indexing inverted files for fast text retrieval. ACM Trans. Inf. Syst. 14(4), 349–379 (1996)
9. Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. Commun. ACM 33(6), 668–676 (1990)
10. Zukowski, M., Heman, S., Nes, N., Boncz, P.: Super-scalar ram-cpu cache compression. In: ICDE '06: Proceedings of the 22nd International Conference on Data Engineering. p. 59. IEEE Computer Society, Washington, DC, USA (2006)