SkipBlock: Self-Indexing for Block-Based Inverted List

Stéphane Campinas[1 2], Renaud Delbru[1] and Giovanni Tummarello[1]

[1] Digital Enterprise Research Institute,
National University of Ireland, Galway
Galway, Ireland
[2] École Pour l'Informatique et les Techniques Avancées,
Le Kremlin-Bicêtre, France
{stephane.campinas,renaud.delbru,giovanni.tummarello}@deri.org

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 independently. Despite block-based coding approaches providing 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 inverted list that can be used for designing a more efficient self-indexing structure as we will show in this paper.

We present in this paper 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 List data structure, before presenting in Section 3 our approach. Section 4 discusses the problem of searching within Skip List intervals. In Section 5 we define a cost model and compare four implementations of the SkipBlock model against the original Skip List model. In Section 6 we recall the main finding of the research and the remaining task.

1.1 Related Work

The Skip List data structure is introduced in [9] as a probabilistic alternative to balanced trees and it 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 of inverted lists enables a sub-linear complexity in average when intersecting two inverted lists. [2] proposes a way to compress efficiently a Skip List 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 based on a query distribution. In [7], the authors present a generalized Skip List data structure for concurrent operations. In this paper, we introduce a new model for self-indexing of block-based inverted lists based on an extension of the Skip List data structure. Our work is orthogonal to the previous works, since each of them could be adapted 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. A naive 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 technique [8]. Self-indexing relies on a Skip List data structure to build a sparse index over the inverted lists and to provide fast record lookups. In this section, we first present the Skip List model and its associated search algorithm. We finally discuss the effect of the probabilistic parameter with respect to the Skip List data structure and search complexity.

2.1 The Skip List Model

Skip List 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 itself has 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 List 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 List 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 List Search Algorithm

Searching an element in a Skip List is performed with a top-down strategy. The search starts at the head of the top list and performs a linear walk over the list as long as the target is greater than a synchronization point. The search goes down one level if and only if the target is lower than the current synchronisation point, and resumes the linear walk. The search stops when the current synchronization point is (a) equal to the target, or (b) on the bottom level and the upper bound of the target. At this stage, it means we have found the interval of records containing our target element.

Figure 1 depicts with a solid line the search path in a Skip List with $p = \frac{1}{16}$ and $L(n) = 3$ levels to the record 8195. At the top of the Skip List, 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.

The search complexity is defined by the number of steps necessary to find the record interval containing the target element. 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 $\frac{L(n)}{p}$.

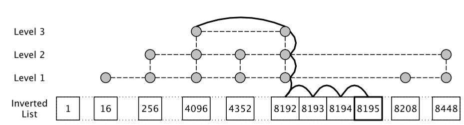

Fig. 1: Skip List with $p = \frac{1}{16}$. Dashed lines denote pointers between synchronization points. The solid line shows the search path to the record 8195.

2.3 Impact of the Probabilistic Parameter

In this section, we discuss the consequences of the probabilistic parameter on the Skip List 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 List 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 provides 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 List with $|I| = \frac{1}{p}$.

$\lvert I \rvert$	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 $|I| = \frac{|B|}{p}$.

Table 1: Search and size costs of Skip List and SkipBlock with $n = 10^8$. $|I|$ 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 List 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 List data structure in order to trade search against storage costs.

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 List model. Therefore this model is a generalization of the original Skip List model. 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

Within the SkipBlock model, the search algorithm returns an interval of blocks containing the target record. In Section 4, we discuss for searching a record within that interval. The search strategy 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 search strategy applies the same termination criteria as in the Skip List search algorithm. 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 List data structure: the probabilistic parameter p and the block size $|B|$. Compared to the original Skip List model, the block size parameter enables a finer control over the Skip List structure. For example, to build a structure with an interval of length 64, the original Skip List 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 List 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 List and SkipBlock techniques enable the retrieval of a record interval given a target record. The next step consists in finding 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 $O(|I|)$ with $|I|$ the length of an interval.

SkipBlock takes advantage of the block-based structure of the interval to perform more efficient search strategies. We define here four additional strategies for searching a block-based interval. 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 an inner-block Skip List structure to find the target, restricted to one level only. The complexity is $\frac{1}{p} + \frac{1}{q} + \lfloor |B| \times q \rfloor$ with q the probability of the inner Skip List. In contrast to S3, the fourth strategy (S4) uses a non-restricted inner-block Skip List structure. The complexity is $\frac{1}{p} + \frac{L(|B|)+1}{q}$ with q the inner Skip List probability. The fifth one (S5) builds a Skip List 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 List 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 is used to compare five SkipBlock implementations and the original Skip List implementation.

Cost Model For both the Skip List 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 of 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 in the intervals for S3, S4 and S5.

Implementations We define as the baseline implementation, denoted I_1, the Skip List 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 List in implementations I_4, I_5 and I_6 is configured with probability $q = \frac{1}{16}$. The inner Skip List 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 report in Table 2 the search and size cost of the configurations that are providing the best search cost given an interval length. 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 (1152), I_5 and I_6 allow yet smaller search cost (69) than I_4 with a similar size cost. Compared to the Skip List with a smaller interval (16), they achieve a smaller search cost with a similar size. To conclude, I_4, I_5 and I_6 seem to provide a good compromise between search and size costs with large intervals; I_5 and I_6 offering slightly better search cost in exchange of a slightly greater size cost.

$\lvert I \rvert$	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	70		3456	1186	154	74	69	
ZC$\times e6$	14.3	16.7	50.0				6.7	12.5	25.0				0.2	0.3	1.8	6.5	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 List model and provides a backbone for developing efficient interval search strategies. Compared to the original Skip List model, SkipBlock achieves with a structure of similar size a lower search cost. In addition, SkipBlock allows finer control over the Skip List 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)