Compact Snippet Caching for Flash-based Search Engines

Rui Zhang[1†] Pengyu Sun[1,2†] Jiancong Tong[1,2]
Rebecca J. Stones[1] Gang Wang[1‡] Xiaoguang Liu[1‡]

[1]College of Computer and Control Engineering & College of software, Nankai University, China
[2]Baidu, Inc
{zhangruiann, sunpengyu, jctong, becky, wgzwp, liuxg}@nbjl.nankai.edu.cn

ABSTRACT

In response to a user query, search engines return the top-k relevant results, each of which contains a small piece of text, called a snippet, extracted from the corresponding document. Obtaining a snippet is time consuming as it requires both document retrieval (disk access) and string matching (CPU computation), so caching of snippets is used to reduce latency. With the trend of using flash-based solid state drives (SSDs) instead of hard disk drives for search engine storage, the bottleneck of snippet generation shifts from I/O to computation. We propose a simple, but effective method for exploiting this trend, which we call fragment caching: instead of caching the whole snippet, we only cache snippet metadata which describe how to retrieve the snippet from the document. While this approach increases I/O time, the cost is insignificant on SSDs. The major benefit of fragment caching is the ability to cache the same snippets (without loss of quality) while only using a fraction of the memory the traditional method requires. In our experiments, we find around 10 times less memory is required to achieve comparable snippet generation times for dynamic memory, and we consistently achieve a vastly greater hit ratio for static caching.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval-Search process

Keywords

Cache; Snippet; Solid State Drive

1. INTRODUCTION

Caching is an important method for reducing query latency in search engines. A large-scale search engine typically consists of three sets of servers [2, 5]: (a) web servers, which interact with users and coordinate the whole query processing, utilizing a query result cache (QRC), (b) index servers, which match top-k documents that are most related to the query, and (c) document servers, which generate the query result pages, which include the title, URL, and a snippet for each of the top-k documents.

For a given query, a snippet is the portion of a document which matches the query best and the query terms are highlighted. In order to alleviate query latency, frequently requested documents and snippets are cached. These two in-memory caches are called *document cache* (DC) [10] and *snippet cache* (SC) [3], respectively. DC stores document pages to decrease disk accesses when generating snippets, while SC stores snippets for documents to avoid repeated computation and disk access.

Recently, solid state drives (SSDs) have been increasingly used by the industries as a secondary storage medium. Many companies are replacing their hard disk drives (HDDs) by SSDs. SSDs have properties such as fast read (10 to 100 times faster than HDD) and write, lower power consumption, and better shock resistance [4].

Snippet generation is an expensive process in terms of both disk access and CPU computation [10]. In traditional HDD-based search engines, the latency of snippet processing is dominated by disk access operations (i.e., I/O). As SSD has faster read (especially random read), snippet calculation time becomes the bottleneck in snippet processing [11]. In this context, techniques for increasing caching efficiency will have changed.

To address this hardware difference, we adopt a relative location representation, motivated by Hoobin *et al.* [6], which allows more snippets to be cached, and so fewer snippets need to be generated on the fly. Instead of caching the full snippet, we cache the snippet metadata, indicating how to retrieve the snippet, along with the text that needs highlighting, from the document; the idea being to increase the number of queries that can be cached within a limited memory space.

To the best of our knowledge, this is the first study that takes advantage of SSDs to improve the efficiency of snippet caching (although Tong *et al.* [8] used SSDs to improve list caching).

2. RELATED WORK

Snippet generation is one of the most time-consuming processes in document servers. The problem of reducing snippet processing latency has been studied from different

[†]The authors contributed equally to this work.
[‡]Corresponding authors.

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.

DOI: http://dx.doi.org/10.1145/2766462.2767764.

directions. Liu *et al.* [7] used a CPU-GPU hybrid system to accelerate snippet generation. Ceccarelli *et al.* [3] introduced an effective "supersnippet" method, but this method may degrade the accuracy of snippet. Turpin *et al.* [10] used zlib to compress documents and reduced snippet generation latency by 58%; their method can dramatically reduce I/O cost in HDD-based search engine. Tsegay *et al.* [9] stored "surrogates" instead of the whole document to reduce I/O time.

Once SSDs are used to replace HDDs, methods which aim to reduce I/O time may not be of significant benefit. To illustrate, Wang *et al.* [11] found that document caching, which may significantly reduce the I/O of a document server, is not as effective on SSD-based search engines.

3. FRAGMENT CACHE

The use of SSDs dramatically reduces the disk accessing time of snippet processing. As a result, the CPU computation time plays a proportionally larger role in snippet generation. Thus, we will present a snippet caching model with this in mind.

For a given snippet, we define its *fragment* to be the data structure

$$\langle \text{docID}, \text{Frag}_s, \text{Frag}_e, \text{Highlight}_s, \text{Highlight}_e \rangle$$

where docID denotes the ID number of the corresponding document, Frag_s and Frag_e record the start and end points of the snippet to be returned, respectively, and Highlight_s and Highlight_e are two arrays which record the start and end points of the highlighted terms. A fragment records where on the document the relevant snippet data can be found.

Figure 1 depicts what is recorded in a toy example. In it, we have $\text{Frag}_s = \text{offset}$, $\text{Frag}_e = \text{offset}+\text{length}$, $\text{Highlight}_s = [\text{offset}_1, \text{offset}_2]$, and $\text{Highlight}_e = [\text{offset}_1+\text{length}_1, \text{offset}_2+\text{length}_2]$.

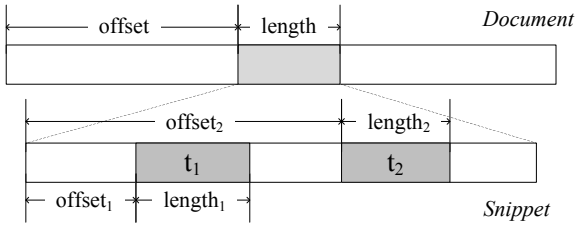

Figure 1: Illustrating the snippet structure. The top image shows the full document stored on the SSD (and possibly also in the document cache). The zoomed in image shows the snippet to be returned, along with which parts of it need highlighting.

This method of generating fragments is independent of the snippet generation algorithm used, so any snippet generation method can be used while utilizing fragment caching. Caching fragments instead of the full snippets will reduce the size of cached items, thereby increasing the number of snippets represented in the cache and increasing the cache hit ratio.

Figure 2 depicts the snippet processing and the fragment processing methods on document servers. The cache stores fragments instead of snippets is called *fragment cache* (FC).

We can model the snippet generation time mathematically. For a random query q for which the snippet for the document d = docID is required, we define the following variables:

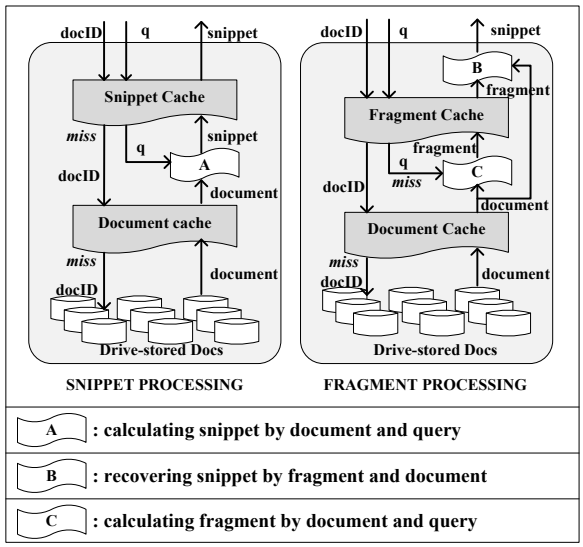

Figure 2: An outline of the workflow in snippet processing and fragment processing. The query q and document ID docID **will be sent from the web server. Both methods return exactly the same snippet.**

- $C^{(\text{sn})}_{(q,d)}$ and $C^{(\text{fr})}_{(q,d)}$ are the total times of processing (q,d) using snippet caching and fragment caching, respectively;
- $\overline{C}_{\text{drive}}$ is the average time it takes to retrieve a document from the drive;
- $p^{(\text{sn-miss})}$ and $p^{(\text{fr-miss})}$ are the probabilities of a snippet cache miss or a fragment cache miss, respectively;
- r is the probability a document is not represented in the document cache;
- $C_{\text{doc-cache}}$ is the time it takes to check the document cache for a document;
- $C_{\text{sn-cache}}$ and $C_{\text{fr-cache}}$ are the respective times it takes to check the snippet cache and fragment cache;
- $\overline{C}_{\text{sn-gen}}$ and $\overline{C}_{\text{fr-gen}}$ are the respective average times it takes to generate the snippet or fragment from (q,d);
- In the case of fragment caching, $\overline{C}_{\text{fr-sn-rec}}$ is the average time it takes to recover the snippet, given the fragment and document.

In snippet processing, we always check the snippet cache. If we hit the snippet cache, we're done, otherwise we check the document cache (which occurs with probability $p^{(\text{sn-miss})}$). If we miss the document cache (which occurs with probability r), we also check the disk, and in either case, we generate the snippet from the document and the query.

In fragment processing, we always check the fragment cache, and if we miss, we will need to generate the fragment from the query and the document. Whether or not we hit the fragment cache, we search for the document in the document cache, and if we miss the document cache (which occurs with probability r), we also check the drive. When the document is found after a fragment cache miss, we generate the fragment, and subsequently the snippet from the fragment.

As a result of the above discussion, the expected runtimes for snippet processing and fragment processing can be modelled as follows (using linearity of expectation):

$$\mathrm{E}(C^{(\mathrm{sn})}_{(q,d)}) = C_{\text{sn-cache}} + p^{(\text{sn-miss})}\,(C_{\text{doc-cache}} + r\,\overline{C}_{\text{drive}} + \overline{C}_{\text{sn-gen}}),$$

$$\mathrm{E}(C^{(\mathrm{fr})}_{(q,d)}) = C_{\text{fr-cache}} + C_{\text{doc-cache}} + r\,\overline{C}_{\text{drive}} + \overline{C}_{\text{fr-sn-rec}} + p^{(\text{fr-miss})}\,\overline{C}_{\text{fr-gen}}.$$

For SSD-based search engines, these equations are dominated by the $p^{(\text{sn-miss})}\overline{C}_{\text{sn-gen}}$ and $p^{(\text{fr-miss})}\overline{C}_{\text{fr-gen}}$ terms, respectively (i.e., computation time). Since $\overline{C}_{\text{sn-gen}} \approx \overline{C}_{\text{fr-gen}}$, when $p^{(\text{fr-miss})} < p^{(\text{sn-miss})}$ (i.e., the hit ratio of fragment caching is higher than the hit ratio of snippet caching), we can expect fragment caching to perform better than snippet caching.

This mathematical model also predicts that on HDD-based systems, where these equations are instead dominated by $p^{(\text{sn-miss})} r\,\overline{C}_{\text{drive}}$ and $r\,\overline{C}_{\text{drive}}$, respectively, fragment caching will perform worse than snippet caching.

4. PERFORMANCE EVALUATION

We evaluate the proposed fragment caching by comparing it vs. traditional snippet caching. We test both static and dynamic cache strategies. For static caching, we use MFU (Most Frequently Used) [1], and for dynamic caching, we use LRU (Least Recently Used). We control the hit ratio of the query result cache (in the web server) as 30% in order to keep the workload of document server realistic during the experiments.

4.1 Experimental setup

In our experiments, we use a collection of 12 million webpages crawled by Sogou Labs[1]. The experiments are performed by replaying real queries from the Sogou search engine. The query log contains 2M queries. The first half is used as the training set to "warm" the cache and the second half is used as the test set. We deploy a Lucene[2] based search engine, in which we implement fragment caching. The server is a Intel Xeon E5645 (2.4GHz) machine, with 12GB of main memory and Windows 7 SP1. We carry out experiments on a 120GB OCZ Vertex-3 SSD.

4.2 Experimental results

In our experiments, we find the average snippet size is 937 bytes while the average fragment size is only 61 bytes, which implies an average increase in the number of items cached by factor of approximately 15.

The total snippet generation time is given by $C_{\text{snip}} + C_{\text{doc}}$, where C_{doc} denotes the overhead of document retrieval, and C_{snip} denotes the remaining time (i.e., the snippet calculation time for snippet caching (A in Figure 2), and the sum of the fragment calculation time and snippet recovering time for fragment caching (B and C in Figure 2)).

Figures 3 and 4 compare the average snippet generation time for snippet caching and fragment caching using both static and dynamic cache strategies (split into C_{snip} and C_{doc}). Tables 1 and 2 list the hit ratios of snippet caching and fragment caching with different cache sizes under MFU and LRU, respectively. In Figures 3 and 4 and Tables 1 and 2, by "Snippet/fragment cache size" of size X, we mean, of the total memory (shared by SC/FC and DC), the portion reserved for snippet caching or fragment caching is approximately equal to the memory required to store $1000X$ snippets.

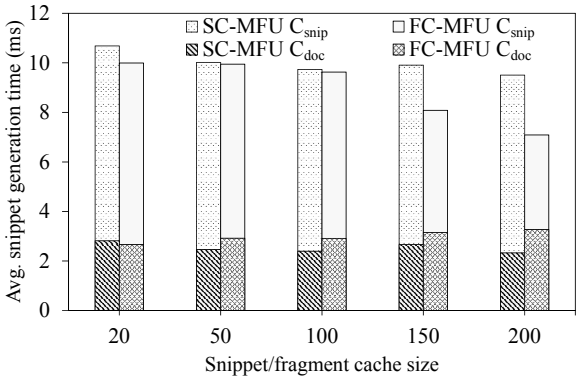

Figure 3: The latency of snippet processing and fragment processing under static caching (MFU).

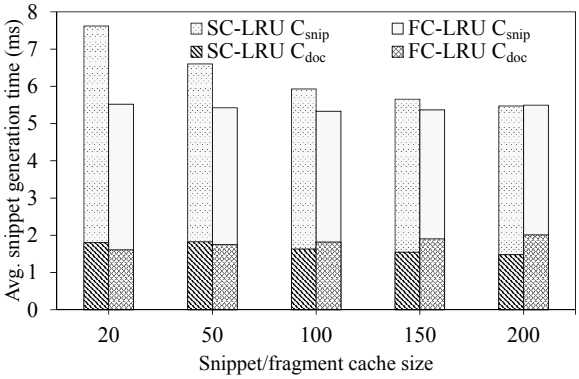

Figure 4: The latency of snippet processing and fragment processing under dynamic caching (LRU).

Table 1: The cache hit ratio for snippet caching and fragment caching under static caching (MFU).

Snippet/fragment cache size	20	50	100	150	200
Snippet (%)	24.8	29.9	33.8	35.9	37.4
Fragment (%)	39.9	45.5	51.3	68.2	77.7

4.3 Discussion

In static caching, we fill the caches in the document server using MFU according to the training set. As snippets can not be cached if the cache space is already filled during the training phase, the constrained cache size restricts the performance of snippet caching. The use of fragment caching decreases the probability of a cache miss substantially (see Table 1), which leads to a reduction of C_{snip}. And due to the fast read of SSD, the increase of C_{doc} is not substantial. As a result, we see better performance by fragment processing than snippet processing, particularly for larger memory sizes. E.g., when we allocate a snippet/fragment cache size of 200, fragment processing is 25% faster than snippet processing (see Figure 3).

We observe that fragment caching can also perform better than snippet caching with dynamic caching (see Figure 4). Using fragment caching, we only need a fragment cache size of 20 to get the similar snippet generation time as snippet

[1] http://www.sogou.com/labs/resources.html?v=1
[2] http://lucene.apache.org

Table 2: The cache hit ratio for snippet caching and fragment caching under dynamic caching (LRU).

Snippet/fragment cache size	20	50	100	150	200	550
Snippet (%)	38.5	52.8	59.8	62.9	64.8	70.7
Fragment (%)	67.5	72.2	75.9	77.5	77.7	77.7

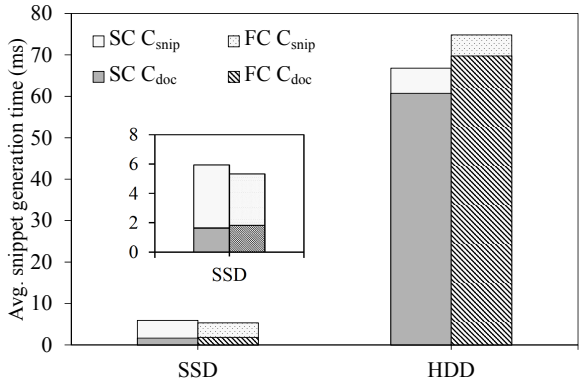

Figure 5: The latency of snippet processing and fragment processing under dynamic caching (LRU) on HDD-based and SSD-based search engine (cache size is 100).

caching with a snippet cache size of 200, i.e., 10 times as large. For snippet caching, snippet generation latency decreases when more memory is used to cache snippets. However the situation changes if fragment caching is used. The cached fragments occupy so much less memory space that even a substantially smaller fragment cache can cache most of the popular fragments. Consequently, allocating more memory to the fragment cache does not decrease C_{snip} significantly, but results in more document cache misses (due to document cache memory being reallocated as fragment cache memory) which results in an increased C_{doc}. Therefore, when the fragment cache size exceeds 100, we see the snippet generation latency increasing slightly as the fragment cache size increases. We conclude that when the query stream contains many distinct snippets that can not be held completely by the snippet/fragment cache, fragment caching will show its performance advantage.

Table 2 lists the cache hit ratios for snippet caching and fragment caching under dynamic caching with different snippet/fragment cache sizes. The hit ratio for snippet caching increases rapidly as more memory space is allocated to the snippet cache, whereas fragment caching achieves a high hit ratio even for substantially smaller cache sizes (a factor of 10 smaller). After the popular fragments have been cached in the fragment cache, increasing the cache size further does not improve its hit ratio substantially. With a fragment cache size of 200, fragment caching achieves the highest possible hit ratio, which indicates all the fragments except the unique ones have been cached. By contrast, snippet caching does not achieve this hit ratio even when the snippet cache size is 550. We conclude that fragment caching can achieve a high hit ratio while using a greatly smaller space than snippet caching.

We also briefly test these caching strategies on a 500GB WDC HDD (5400rpm). As expected, (a) regardless of the caching method used, we found a drastic reduction in latency in SSDs vs. HDDs, and (b) fragment caching results in far more document accessing than snippet caching, and performs worse (see Figure 5).

5. CONCLUSION

Many search engines servers are moving from hard disk drives (HDDs) to solid state drives (SSDs) to store masses of documents, largely due to SSDs massive advantage in I/O speed. In this work, we present a simple, yet effective caching method for compactly caching snippets, called fragment caching, which takes advantage of SSD-based hardware. Instead of caching whole snippets in memory, we instead cache snippet metadata which are used to retrieve the snippets from documents when required. This greatly reduces the memory required for caching, while the increased I/O is not a major concern on SSDs (as it is for HDDs).

6. ACKNOWLEDGEMENTS

This work is partially supported by NSF of China (61373018, 11301288, 11450110409), Program for New Century Excellent Talents in University (NCET130301) and the Fundamental Research Funds for the Central Universities (65141021).

7. REFERENCES

[1] R. A. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proc. SIGIR*, pages 183–190, 2007.

[2] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[3] D. Ceccarelli, C. Lucchese, S. Orlando, R. Perego, and F. Silvestri. Caching query-biased snippets for efficient retrieval. In *Proc. EDBT*, pages 93–104, 2011.

[4] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proc. SIGMETRICS*, pages 181–192, 2009.

[5] J. Dean. Challenges in building large-scale information retrieval systems: invited talk. In *Proc. WSDM*, page 1, 2009.

[6] C. Hoobin, S. J. Puglisi, and J. Zobel. Relative Lempel-Ziv factorization for efficient storage and retrieval of web collections. *Proc. VLDB Endowment*, 5(3):265–273, 2011.

[7] D. Liu, R. Li, X. Gu, K. Wen, H. He, and G. Gao. Fast snippet generation based on CPU-GPU hybrid system. In *Proc. ICPADS*, pages 252–259, 2011.

[8] J. Tong, G. Wang, and X. Liu. Latency-aware strategy for static list caching in flash-based web search engines. In *Proc. CIKM*, pages 1209–1212, 2013.

[9] Y. Tsegay, S. J. Puglisi, A. Turpin, and J. Zobel. Document compaction for efficient query biased snippet generation. In *Advances in Information Retrieval, ECIR*, pages 509–520, 2009.

[10] A. Turpin, Y. Tsegay, D. Hawking, and H. E. Williams. Fast generation of result snippets in web search. In *Proc. SIGIR*, pages 127–134, 2007.

[11] J. Wang, E. Lo, M. L. Yiu, J. Tong, G. Wang, and X. Liu. The impact of solid state drive on search engine cache management. In *Proc. SIGIR*, pages 693–702, 2013.