An Extensible Search Engine for Efficient Caching and Fast Lists Intersection

ABSTRACT

In modern search engines, caching has been widely used to reduce query latency. Most frequently requested documents and text fragments are cached to improve query throughput. Besides, there are previous works which utilize GPU to speed-up lists intersection during query processing. However, these relavant work has not been directly implemented in current Lucene.

Due to these reasons, we extend the CLucene, which written in C++, for efficiency research purpose. We construct an extensible architecture with application programming interfaces to provide a flexible research and experiment platform for researchers readily implement their algorithms and strategies. We have expanded the original project for taking use of parallelism in GPU to speed-up lists intersection of query processing. For the problem of cache design and implement, we integrate a fully functional and flexible cache framework which involving snippet generation and cache strategies. The whole platform is designed to be extensible so as to accommodate new methods without significant development effort. The source codes is available from https://github.com/NoneOS/ExCLucene.

CCS CONCEPTS

•Information systems → Search engine architectures and scalability;

KEYWORDS

Search engine, open source, index compression, cache strategy

ACM Reference format:
. 2017. An Extensible Search Engine for Efficient Caching and Fast Lists Intersection. In *Proceedings of ACM SIGIR Conference Workshop, Tokyo, Japan, August 2017 (LIARR'2017),* 2 pages.
DOI: 10.475/123_4

1 INTRODUCTION

To support more and more complex features, many available open source projects for full-text search have been designed [6]. Thanks to the highly optimizations and cross-platform feature, many of them were implemented in Java language, e.g. Lucene[1]. Nowadays, Lucene and its expansions have been widely used in industry situations.

But in the situation of efficiency research, it is difficult for Java programs to directly utilize some parallel mechanism in modern computer system, like SIMD (Single Instruction, Multiple Data) and GPU. These two features have been studied to improve query performance in [1, 3, 5, 7, 8, 10]. Although it is possible for Java programs to use JNI (Java Native Interface) to call C/C++ codes, the efficiency loss of transformation is still a serious problem.

As a result, this paper designs an efficient and extensible search engine for full-text searching written in C++. We choose to reconstruct the CLucene[2], which is a C++ migration of Lucene. Our work is an improvement on previous work by [12]. More specifically, we expand their works in index compression, query processing and result summarization. We implement more efficient encoding methods, especially the ones utilizing SIMD instructions. For query processing, we have integrated more common query operations, like WAND [2] and BMW [4]. We also support parallel query processing function, which utilizes GPU to boost list intersections [1]. In the situation of cache framework, we improve their work with more cache types and replacement strategies.

2 SYSTEM ARCHITECTURE

The architecture of our system is shown in Figure 1. The modules we have implemented including four major components: corpus indexer, index compression, query processing and result summarization. All these modules are implemented by following the inherit hierarchy of CLucene to keep the standard modular architecture.

The documents indexing module parse original documents in the corpus, e.g. HTML webpages and PDF files. This module has implemented by [12], which directly support to analyse TREC and WARC file formats. The index compression module will compress invert index files and decompress relevant posting lists during query processing. Query processing module will parse each query and retrieval relevant documents with ranking result. For each result, the result summarization module will check cache hit to determine whether generate snippet from origin text or fetch corresponding result from cache directly.

2.1 Query Processing

In the query processing module, we integrate the support for two typical rank query operations, WAND [2] and BMW [4]. The extra information required by these operations, like upper-bound scores of terms or blocks of postings, are stored outside of CLucene index.

Besides, we also integrate a CPU-GPU cooperative module to support fast lists intesection utilizing GPU. We first transform original CLucene inverted index into a searchable compressed index and upload it to GPU. Each posting list is split into blocks (each chunk has fix number of docIDs) and each block is compressed independently. During query processing, queries are grouped into batches. Each batch are uploaded from CPU to GPU, while the

[1] https://lucene.apache.org/

[2] http://clucene.sourceforge.net/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

LIARR'2017, Tokyo, Japan
© 2017 ACM. 123-4567-24-567/08/06…$15.00
DOI: 10.475/123_4

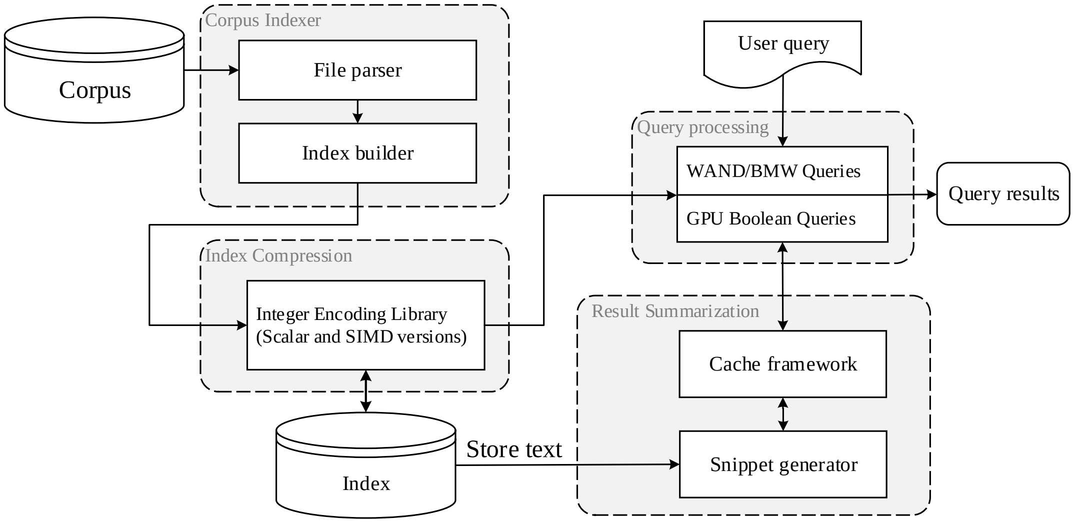

Figure 1: System architecture

results are downloaded from GPU to CPU. The lists intersection in GPU are handled in DAAT(Document-At-A-Time) method. Due to the space limit, the more details can be found in [1, 11].

2.2 Result Summarization

In most search engines, result documents with title and relevant sentences, called snippet, are returned with ranking list. We integrate existing snippet generator, namely Highlighter, with an extensible cache framework. We define the kernel data structure *cache table* which implemented by HashMap. The entry in cache table is *cache node*, which consists of a pair of key-value and can be inserted or deleted independently.

In this module, there are three types of cache, i.e. query result cache, snippet cache and document cache. When the search engine receive a new query, it will first check whether the query can be found in query result cache. If it hits, search engine will directly return result list with snippets. Otherwise query processing module will search relevant posting lists to find result documents. For each result document, search engine will first lookup the snippet cache by docID and query, if it hits, snippet for this document can be directly returned. Otherwise, it will check whether the document cached in document cache to avoid read text from disk. Besides, we also implement several typical cache replacement strategies, e.g. LRU, LFU, QTF(Query-Term-Frequency), QTFDF and etc. The more details about cache replacement strategies can be found in [9].

3 CONCLUSIONS

In this paper, we have described the design and implementation of a search engine based on CLucene project. In order to accommodate new methods expediently, the architecture and core library has been well-designed to be extensible. For this purpose, we defined a series of typical application programming interfaces which involves different stages during full-text search. Our purpose is to provide a flexible research and experiment platform for researchers readily implement their algorithms and strategies on the aspect of efficiency. As future work, we are going to implement an distribute system for full-text search based on this work.

REFERENCES

[1] Naiyong Ao, Fan Zhang, Di Wu, Douglas S Stones, Gang Wang, Xiaoguang Liu, Jing Liu, and Sheng Lin. 2011. Efficient parallel lists intersection and index compression algorithms using graphics processing units. *Proc. VLDB Endowment* 4, 8 (2011), 470–481.

[2] Andrei Z Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proc. 12th ACM CIKM.* 426–434.

[3] Shuai Ding, Jinru He, Hao Yan, and Torsten Suel. 2009. Using graphics processors for high performance IR query processing. In *Proc. 18th International Conference on World Wide Web.* 421–430.

[4] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011.* 993–1002.

[5] Daniel Lemire, Leonid Boytsov, and Nathan Kurz. 2014. SIMD Compression and the Intersection of Sorted Integers. *CoRR* abs/1401.6399 (2014).

[6] Christian Middleton and Ricardo Baeza-Yates. 2007. *A comparison of open source search engines.* Technical Report. Department of Technologies, Universitat Pompeu Fabra.

[7] Benjamin Schlegel, Thomas Willhalm, and Wolfgang Lehner. 2011. Fast Sorted-Set Intersection using SIMD Instructions. In *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures.* 1–8.

[8] Alexander A Stepanov, Anil R Gangolli, Daniel E Rose, Ryan J Ernst, and Paramjit S Oberoi. 2011. SIMD-based decoding of posting lists. In *Proc. 20th ACM International Conference on Information and Knowledge Management.* 317–326.

[9] Jiancong Tong, Gang Wang, and Xiaoguang Liu. 2013. Latency-aware strategy for static list caching in flash-based web search engines. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013.* 1209–1212.

[10] Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, and Jan Schaffner. 2009. SIMD-scan: ultra fast in-memory table scan using on-chip vector processing units. *Proc. VLDB Endowment* 2 (2009), 385–394.

[11] Di Wu, Fan Zhang, Naiyong Ao, Gang Wang, Xiaoguang Liu, and Jing Liu. 2010. Efficient lists intersection by CPU-GPU cooperative computing. In *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, Georgia, USA, 19-23 April 2010 - Workshop Proceedings.* 1–8.

[12] Zhaohua Zhang, Benjun Ye, Jiayi Huang, Rebecca J. Stones, Gang Wang, and Xiaoguang Liu. 2016. NBLucene: Flexible and Efficient Open Source Search Engine. In *Web-Age Information Management - 17th International Conference, WAIM 2016, Nanchang, China, June 3-5, 2016, Proceedings, Part I.* 504–516.