Anserini: Enabling the Use of Lucene for Information Retrieval Research

Peilin Yang, Hui Fang
Department of Electrical and Computer Engineering
University of Delaware
{franklyn,hfang}@udel.edu

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo
jimmylin@uwaterloo.ca

ABSTRACT

Software toolkits play an essential role in information retrieval research. Most open-source toolkits developed by academics are designed to facilitate the evaluation of retrieval models over standard test collections. Efforts are generally directed toward better ranking and less attention is usually given to scalability and other operational considerations. On the other hand, Lucene has become the *de facto* platform in industry for building search applications (outside a small number of companies that deploy custom infrastructure). Compared to academic IR toolkits, Lucene can handle heterogeneous web collections at scale, but lacks systematic support for evaluation over standard test collections. This paper introduces Anserini, a new information retrieval toolkit that aims to provide the best of both worlds, to better align information retrieval practice and research. Anserini provides wrappers and extensions on top of core Lucene libraries that allow researchers to use more intuitive APIs to accomplish common research tasks. Our initial efforts have focused on three functionalities: scalable, multi-threaded inverted indexing to handle modern web-scale collections, streamlined IR evaluation for *ad hoc* retrieval on standard test collections, and an extensible architecture for multi-stage ranking. Anserini ships with support for many TREC test collections, providing a convenient way to replicate competitive baselines right out of the box. Experiments verify that our system is both efficient and effective, providing a solid foundation to support future research.

1 INTRODUCTION

Information retrieval researchers have a long history of developing, sharing, and using software toolkits to support their work. Over the past several decades, various IR toolkits have been built to aid in the development of new retrieval models, to test hypotheses about information seeking, and to validate new evaluation methodologies. As the field moves forward, IR toolkits are expected to keep up with emerging requirements such as the ability to handle large web collections and new data formats. The growing complexity of modern software ecosystems and the resource constraints most academic research groups operate under make maintaining open-source toolkits a constant struggle.

SIGIR'17, August 7–11, 2017, Shinjuku, Tokyo, Japan

978-1-4503-5022-8/17/08…$15.00
DOI: http://dx.doi.org/10.1145/3077136.3080721

Most IR toolkits developed by academics, such as Indri,[1] Galago,[2] and Terrier[3] were primarily designed to facilitate evaluation over standard test collections from evaluation forums such as TREC, CLEF, NTCIR, etc. In many cases, scalability took a back seat to efforts around improving retrieval models, and thus these systems often struggle to scale to modern web collection. As an example, the ClueWeb12 collection[4] contains 733 million web pages, totaling 5.54 TB compressed (or 27.3 TB uncompressed). The standard practice for working with this collection, as exemplified by the infrastructure built for the TREC 2014 Session Track [4], is to separately index partitions of the collection and then build a distributed broker architecture that integrates results from each partition. In general, working with web-scale collections using existing academic IR toolkits is time- and resource-intensive, even for basic tasks.

With the exception of a small number of companies (e.g., commercial web search engines), the open-source Lucene system[5] and its derivatives such as Solr and Elasticsearch (for convenience, we simply refer to as "Lucene" collectively in this paper) have become the *de facto* platform for deploying search applications in industry. Examples include LinkedIn, Twitter, Bloomberg, as well as a number of online retailers and many large companies in the financial services space. Despite its undeniable operational success, a large user base, and a vibrant community of contributors, Lucene is not well suited to information retrieval research. For many reasons, including poor documentation of system internals and a number of unintuitive abstractions, Lucene is not as widely used for research as academic toolkits such as Indri or Terrier.

In this paper, we describe our efforts in developing a new open-source information retrieval toolkit called Anserini that builds on Lucene.[6] We aim to bridge the gap described above that separates information retrieval research from the practice of building real-world search applications. Anserini provides wrappers and extensions on top of core Lucene libraries that allow researchers to use more intuitive APIs to accomplish common research tasks. Our initial efforts have focused on three functionalities: scalable, multi-threaded inverted indexing to handle modern web collections, streamlined IR evaluation for *ad hoc* retrieval on standard test collections, and an extensible architecture for multi-stage ranking. Anserini ships with support for standard TREC test collections, providing a convenient way to replicate competitive baselines "right out of the box", supporting the community's aspirations toward reproducible research [1, 7, 8, 10, 16, 18].

[1] http://www.lemurproject.org/indri/
[2] http://www.lemurproject.org/galago.php
[3] http://terrier.org/
[4] http://www.lemurproject.org/clueweb12/
[5] https://lucene.apache.org/
[6] http://anserini.io/

We experimentally evaluate the efficiency and effectiveness of Anserini on a number of standard test collections. In terms of indexing performance, it is able to handle the largest research web collection available today with ease on a single modern server. We observe better indexing performance compared to Indri, a popular choice among researchers today. In terms of retrieval, we also find that Anserini is not only faster than Indri, but returns rankings that are comparable in quality. In other words, Anserini is faster and just as good. We present the case that Anserini should be adopted as the toolkit of choice for information retrieval researchers.

2 ANSERINI OVERVIEW

2.1 Motivation

Despite its popularity in industry and broad adoption for operational search deployments, Lucene remains under-utilized in information retrieval research. We begin with some high-level discussions of why we believe this might be so to motivate our efforts in building Anserini.

From the very beginning, Lucene was written for "real world" search applications, not with researchers in mind. For the most part, its developers targeted an audience that mostly used search engines as black boxes, as opposed to researchers that required access to ranking internals such as scoring models, mechanisms for postings traversal, etc. Because of the target user population, documentation for Lucene internals has always been quite poor, especially in keeping up with the relatively rapid pace at which the developer community has been releasing improved versions of the software. Access to these internals is exactly what information retrieval researchers need for their studies, and therefore poor documentation has been a barrier to entry.

To further compound this issue, the internal APIs in Lucene are not organized in a way that would be intuitive to most IR researchers, with class names that are not indicative of functionality and many levels of indirection. This is not an issue for "black box" users of Lucene, but presents a hurdle for information retrieval researchers who desire access to system internals. As an example, the code to open up a Lucene index and to traverse postings programmatically (without invoking the scoring function) is unnecessarily complex and involves dispatching to several intermediate classes along the way. Some researchers have the impression that Lucene is difficult to use, and indeed there is some truth to this, especially with respect to low-level abstractions.

Another side effect of Lucene's focus on "black box" search is that it has severely lagged behind in the implementation of modern ranking functions. For the longest time, the default scoring model was an *ad hoc* variant of tf-idf. Okapi BM25 was not added to Lucene until 2011,[7] more than a decade after it gained widespread adoption in the research community as being more effective than tf-idf variants. This lag in adopting "research best practices" has contributed to the perception that Lucene is not effective and ill-suited for information retrieval research. However, this perception is no longer accurate today. Lucene comes with implementations of modern baseline retrieval models, and we show that the effectiveness of Lucene's implementations is at least as good as those offered by academic IR toolkits (see Section 3).

[7] https://issues.apache.org/jira/browse/LUCENE-2959

Finally, because Lucene is written in Java, there is sometimes the perception that it is slow and inefficient, particularly when scaling up to modern web collections. Developers often point to the managed memory environment of the Java Virtual Machine (JVM) as not being conducive to efficient low-level implementations of search engine internals. We experimentally show that this is definitely not true (see Section 3). The open-source community has devoted substantial effort to optimizing the performance of Lucene and taking advantage of today's multi-core processors. It is capable of handling large web collections on a single server with ease.

The goal of Anserini is to align the practice of building search applications with research in information retrieval. Colloquially speaking, our toolkit aims to smooth the "rough edges" around Lucene for the purposes of information retrieval research. It is not our goal to replace or to reimplement Lucene, but rather to facilitate its use for research by presenting as gentle a learning curve as possible to newcomers.

2.2 Main Components

Anserini components fall into two categories: *wrappers* and *extensions*. Wrappers provide APIs that leverage core Lucene library components to accomplish specific tasks. They are tightly integrated with "core" Lucene and in some cases, represent custom implementations of existing Lucene APIs. Extensions, on the other hand, are components that are distinct from Lucene and more loosely coupled: these may represent our own implementations or connectors to third-party libraries.

Multi-threaded indexing (*wrapper*). Inverted indexing is one of the most fundamental tasks in information retrieval and the starting point of many research studies. In working with large web collections, it is imperative that indexing operations are efficient and scalable. While academic researchers have attempted to address this issue via MapReduce and related frameworks [5, 11], these solutions impose the burden of requiring clusters and additional software infrastructure.

Lucene supports multi-threaded indexing, and as we experimentally show (Section 3), it is able to scale up to large web collections on a single commodity server. The biggest issue, however, is that Lucene itself only provides access to a collection of indexing components that researchers need to assemble together to build an end-to-end indexer. For example, the developer would need to write from scratch custom document processing pipelines, code for managing individual indexing threads, and implementations of load balancing and synchronization procedures.

We address these issues in Anserini by providing abstractions for document collections that an IR researcher would be comfortable with, as well as the implementation of an efficient, high-throughput, multi-threaded indexer that takes advantage of these abstractions. Anserini models collections as comprised of individual segments (for example, the ClueWeb12 collection is comprised of a number of compressed WARC files) and provides implementations for common document formats—for parsing TREC-style XML documents, web pages stored in WARCs, tweets in JSON format, etc. In fact, Anserini ships with the ability to index many TREC collections "right out of the box". This greatly reduces the learning curve for researchers to get started with Lucene.

Table 1: Indexing performance of Anserini and Indri on smaller collections using 16 threads on a modest commodity server.

			Anserini (count)		Anserini (pos)		Anserini (doc)		Indri	
Collection	docs	terms	time	size	time	size	time	size	time	size
Disk12	742k	219m	00:01:24	199MB	00:01:44	512MB	00:03:09	2.5GB	00:12:28	2.5GB
Disk45	528k	175m	00:01:13	166MB	00:01:33	423MB	00:02:51	2.1GB	00:06:55	1.9GB
AQUAINT	1.03m	318m	00:01:53	305MB	00:02:10	734MB	00:04:32	3.8GB	00:17:36	3.9GB
WT2G	246k	182m	00:02:21	143MB	00:02:55	437MB	00:04:24	2.3GB	00:07:25	2.2GB
WT10G	1.69m	752m	00:04:55	708MB	00:05:05	2.9GB	00:09:51	12GB	00:42:51	9.6GB
Gov2	25.2m	17.3b	01:16:32	11GB	02:32:43	38GB	06:52:35	331GB	14:51:12	215GB

Table 2: Indexing performance of Anserini on web collections using 88 threads on a high-end server.

			Anserini (count)		Anserini (pos)	
Collection	docs	terms	time	size	time	size
CW09b	50m	31b	00:42	28GB	01:13	75GB
CW09	504m	268b	07:32	254GB	12:18	649GB
CW12b13	52m	31b	00:57	29GB	01:25	76GB
CW12	733m	429b	17:01	376GB	22:21	1.1TB

Streamlined IR evaluation (*extension*). Test collections play an important role in information retrieval research, and a substantial amount of research activity in improving ranking models is focused around *ad hoc* retrieval runs. A research toolkit should make this "inner loop" of IR research as easy as possible. Since Lucene was not originally designed for researchers, support for running experiments on standard test collections is largely missing. Anserini fills this gap by implementing missing features: parsers for different query formats, a unified driver program for *ad hoc* experiments that outputs standard `trec_eval` format, etc. For convenience, existing TREC topics and qrels are included directly in our code repository—once again, reducing the learning curve for researchers to get started with Lucene.

There are two main uses for this feature in Anserini: First, our toolkit provides an easy way for researchers to replicate baselines of standard retrieval models such as BM25 and query likelihood. Armstrong et al. [2] previously identified the prevalent problem of weak baselines in experimental IR papers. Lin et al. [10] further observed that authors are often vague about the baseline parameter settings and the implementations they use. For example, Mühleisen et al. [13] reported large differences in effectiveness across four systems that all purport to implement BM25. Trotman et al. [15] pointed out that BM25 and query likelihood with Dirichlet priors can actually refer to at least half a dozen variants, and in some cases, differences in effectiveness are statistically significant. There is substantial community interest in engaging with reproducibility-related issues [1, 8], and Anserini contributes to this discussion. Our proposed solution is to have widely-available baselines that are both competitive in effectiveness and easy to replicate. It is our hope that Anserini can fill this role.

Second, an easy-to-use baseline retrieval component in Anserini provides the starting point for additional ranking extensions. In particular, we advocate a multi-stage ranking architecture [3, 6, 14, 17] so that researchers will not need to directly work with native Lucene scoring APIs. That is, researchers should take advantage of Anserini APIs that generate an initial document ranking and hooks for feature extraction to build subsequent reranking stages. This, in fact, is the common architecture used in commercial web search engines today to support learning to rank [14].

Relevance feedback (*extension*). Relevance feedback techniques provide robust solutions to the vocabulary mismatch problem between expressions of user information needs and relevant documents. Anserini provides a reference implementation of the RM3 variant of relevance models [9], built as a reranking module in the multi-stage architecture described above. Thus, our implementation is useful not only as a baseline for comparing query expansion techniques, but provides an example of how reranking extensions can be implemented in Anserini.

3 EVALUATION

We describe experiments to support three claims about Anserini and the use of Lucene for information retrieval research. First, that Anserini is highly scalable and able to efficiently index large web collections. Second, that Anserini is similarly efficient in searching these collections and ranking documents using standard baseline models. Finally, Anserini is able to achieve scalable indexing and efficient retrieval without compromising ranking effectiveness.

The indexing performance of Anserini on a number of smaller and older collections is shown in Table 1. These experiments were conducted on a server with dual AMD Opteron 6128 processors (2.0GHz, 8 cores) with 40GB RAM running CentOS 6.8. This machine can be characterized as an old, modest commodity server. All experiments were run on an otherwise idle machine. With Anserini, we used 16 threads for indexing and we report results from three different index configurations: count indexes where only term frequency information is stored (count), positional indexes that also store term positions (pos), and positional indexes that also store the raw documents and parsed document vectors (doc). For each condition, we report the indexing time in HH:MM:SS (averaged over two trials) as well as the index size. The size of each collection is also shown for reference. As a comparison condition, we indexed the same collections using Indri 5.9 on the same machine.

In Table 2, we report indexing performance for larger web collections on a server with dual Intel Xeon E5-2699 v4 processors (2.2GHz, 22 cores) and 1 TB RAM running Ubuntu 16.04. The table rows indicate different collections: CW09b refers to the ClueWeb09 (category B) web crawl, CW09 refers to all English pages in the ClueWeb09 web crawl, CW12b13 refers to the smaller ClueWeb12-B13 web crawl, and CW12 refers to the complete ClueWeb12 web crawl. Due to the size of the collections, we only report the count and positional index configurations. For these experiments, we used

Table 3: Retrieval efficiency for Terabyte 06 efficiency queries on Gov2, using a single thread.

	Latency (ms)	Throughput (qps)
Indri	2403	0.42
Anserini	382	2.61

Table 4: Effectiveness comparisons between Anserini and Indri on standard TREC test collections.

Collection	Disk12	Disk45	WT2G	WT10G	Gov2
Queries	51-200	301-450 601-700	401-450	451-550	701-850
BM25 (I)	0.2040	0.2478	0.3152	0.1955	0.2970
BM25 (A)	0.2267	0.2500	0.3015	0.1981	0.3030
LM (I)	0.2269	0.2516	0.3116	0.1915	0.2995
LM (A)	0.2232	0.2465	0.2922	0.2015	0.2951

88 threads on an otherwise idle machine; indexing time is reported in HH:MM (averaged over two trials). On this server, we are able to index *all* of ClueWeb12, one of the largest collections available to researchers today, in less than a day! As seen from Table 1, even on an older server, the indexing performance of Lucene is impressive. Compared to academic toolkits, Lucene does not appear to have any trouble scaling to large modern web collections.

Our next set of experiments were conducted on the Gov2 collection with Terabyte 06 efficiency queries. We issued all 100,000 queries sequentially against both the Anserini and Indri indexes on the slower AMD Opteron server. Results are shown in Table 3, which reports latency (ms) and throughput (queries per second, or qps). In this experiment, we used only a single query thread, and therefore do not take advantage of Lucene's ability to execute queries in parallel on multiple threads (so in our case, throughput is simply the inverse of latency). We see from these experiments that Lucene is roughly six times faster than Indri.

Finally, we compared the retrieval effectiveness of Anserini and Indri. For Indri we refer to the RISE work of Yang and Fang [18], as they fine-tuned model parameters to achieve optimal effectiveness. We considered two baseline ranking models: Okapi BM25 (BM25) and query likelihood with Dirichlet priors (LM). For Anserini, we removed stopwords (the default) and tuned parameters as follows: for BM25, $k = 0.9$ and $b \in [0, 1]$ in increments of 0.1; for LM, $\mu \in [0, 5000]$ in increments of 500. Results on standard TREC collections and queries are shown in Table 4, where (I) refers to Indri and (A) refers to Anserini. We see that effectiveness results are comparable between the two systems.

In summary, our experiments show that Anserini is at least as good as Indri in terms of effectiveness, and much faster in both indexing and retrieval. These results are consistent with findings from the recent Open-Source IR Reproducibility Challenge [10]. Together, empirical evidence presents a compelling case for adopting Lucene for information retrieval research.

4 CONCLUSIONS AND FUTURE WORK

Our message to the information retrieval community is that Lucene is efficient and scalable without compromising effectiveness. Furthermore, Lucene has the benefit of a large user community and broad adoption in industry. Anserini smooths over the "rough edges" of using Lucene for information retrieval research by providing wrappers and extensions that simplify common tasks such as indexing large research web collections and performing standard *ad hoc* retrieval runs. We hope that our toolkit will help to better align the research and practice of information retrieval.

Broadly characterized, Anserini provides the foundation for an IR research toolkit, but currently lacks features that one would associate with cutting-edge research. Ongoing work is focused on addressing this issue, as we are actively exploring retrieval models based on deep learning [12]. Efforts include attempts to replicate existing neural retrieval models within our framework. Given the existence of many deep learning toolkits (Torch, TensorFlow, etc.), it does not make sense to reinvent the wheel. In this spirit, we have been building connectors between Lucene and the PyTorch deep learning toolkit. Moving forward, we anticipate substantial continued interest at the intersection of deep learning and information retrieval, and the multi-stage ranking architecture of Anserini provides a natural integration point for future explorations.

REFERENCES

[1] Jaime Arguello, Matt Crane, Fernando Diaz, Jimmy Lin, and Andrew Trotman. 2015. Report on the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR). *SIGIR Forum* 49, 2 (2015), 107–116.

[2] Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. 2009. Improvements That Don't Add Up: Ad-Hoc Retrieval Results Since 1998. In *CIKM*. 601–610.

[3] Nima Asadi and Jimmy Lin. 2013. Effectiveness/Efficiency Tradeoffs for Candidate Generation in Multi-Stage Retrieval Architectures. In *SIGIR*. 997–1000.

[4] Ben Carterette, Evangelos Kanoulas, Mark Hall, and Paul Clough. 2014. Overview of the TREC 2014 Session Track. In *TREC*.

[5] Marc-Allen Cartright, Samuel Huston, and Henry Feild. 2012. Galago: A Modular Distributed Processing and Retrieval System. In *SIGIR 2012 Workshop on Open Source Information Retrieval*. 25–31.

[6] Charles L. A. Clarke, J. Shane Culpepper, and Alistair Moffat. 2016. Assessing Efficiency—Effectiveness Tradeoffs in Multi-stage Retrieval Systems Without Using Relevance Judgments. *IRJ* 19, 4 (2016), 351–377.

[7] Hui Fang, Hao Wu, Peilin Yang, and ChengXiang Zhai. 2014. VIRLab: A Web-based Virtual Lab for Learning and Studying Information Retrieval Models. In *SIGIR*. 1249–1250.

[8] Nicola Ferro, Norbert Fuhr, Kalervo Järvelin, Noriko Kando, Matthias Lippold, and Justin Zobel. 2016. Increasing Reproducibility in IR: Findings from the Dagstuhl Seminar on "Reproducibility of Data-Oriented Experiments in e-Science". *SIGIR Forum* 50, 1 (2016), 68–82.

[9] Victor Lavrenko and W. Bruce Croft. 2001. Relevance Based Language Models. In *SIGIR*. 120–127.

[10] Jimmy Lin, Matt Crane, Andrew Trotman, Jamie Callan, Ishan Chattopadhyaya, John Foley, Grant Ingersoll, Craig Macdonald, and Sebastiano Vigna. 2016. Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge. In *ECIR*. 408–420.

[11] Jimmy Lin, Donald Metzler, Tamer Elsayed, and Lidan Wang. 2009. Of Ivory and Smurfs: Loxodontan MapReduce Experiments for Web Search. In *TREC*.

[12] Bhaskar Mitra and Nick Craswell. 2017. Neural Models for Information Retrieval. *arXiv:1705.01509*.

[13] Hannes Mühleisen, Thaer Samar, Jimmy Lin, and Arjen de Vries. 2014. Old Dogs Are Great at New Tricks: Column Stores for IR Prototyping. In *SIGIR*. 863–866.

[14] Jan Pedersen. 2010. Query Understanding at Bing. *Invited Talk at SIGIR*.

[15] Andrew Trotman, Antti Puurula, and Blake Burgess. 2014. Improvements to BM25 and Language Models Examined. In *ADCS*. 58–65.

[16] Ellen M. Voorhees, Shahzad Rajput, and Ian Soboroff. 2016. Promoting Repeatability Through Open Runs. In *EVIA*. 17–20.

[17] Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A Cascade Ranking Model for Efficient Ranked Retrieval. In *SIGIR*. 105–114.

[18] Peilin Yang and Hui Fang. 2016. A Reproducibility Study of Information Retrieval Models. In *ICITR*. 77–86.

Acknowledgments. This research was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the U.S. National Science Foundation under IIS-1423002 and CNS-1405688.