node-indri: moving the Indri toolkit to the modern Web stack

Felipe Moraes and Claudia Hauff

Delft University of Technology
{f.moraes,c.hauff}@tudelft.nl

Abstract. We introduce node-indri, a Node.js module that acts as a wrapper around the Indri toolkit, and thus makes an established IR toolkit accessible to the modern web stack. node-indri exposes many of Indri's functionalities and provides direct access to document content and retrieval scores for web development (in contrast to, for instance, the Pyndri wrapper). This setup reduces the amount of *glue code* that has to be developed and maintained when researching search interfaces, which today tend to be developed with specific JavaScript libraries such as React.js, Angular.js or Vue.js. The node-indri repository is open-sourced at https://github.com/felipemoraes/node-indri.

1 Introduction

The information retrieval (IR) field is aided by numerous efficient search engine implementations, aimed at research and industry, such as Indri [1], Lucene[1], Terrier [2] and Anserini [3]. In the data science field, some of these efforts have evolved into frameworks such as Elasticsearch[2] and Terrier-Spark [4]. Recently, in order to enable data scientists to make use of Indri as part of their workflow, Van Gysel et. [5] have made Indri accessible to the Python ecosystem (via Pyndri).

In this paper, we make Indri accessible to the modern web stack. Many modern web applications and frameworks make use of Node.js.[3] A significant advantage of this framework is the single programming language on the client and server-side (JavaScript), which simplifies development; in addition, Node.js is highly scalable [6]. In order to design and evaluate web search interfaces, a backend, implemented in Node.js, requires access to a search system. One option is to call Indri via system calls. However, the disadvantage of system calls via shell commands is the extra layer of communication with the operating system.

Here, we present an alternative, node-indri, a Node.js module implemented with an easy-to-use API. It provides access to basic Indri functionalities such as search with relevance feedback and document scoring. Importantly, node-indri is implemented in a non-blocking manner. We here discuss node-indri's module

[1] http://lucene.apache.org/
[2] https://www.elastic.co/products/elasticsearch
[3] https://nodejs.org/

features and we compare its efficiency with `Indri` and `Pyndri` on the Aquaint and ClueWeb12 corpora with a load of 10K queries. We find that `node-indri` can be efficiently used in modern web backend development with comparable efficiency to `Indri` and `Pyndri`.

2 The `node-indri` module

`node-indri`'s development started with the need to make `Indri`'s state-of-the-art relevance feedback models accessible to students, that (i) tend to have little experience with C++, but are familiar with modern web programming paradigms and (ii) are not IR experts and thus struggle to make sense of `Indri`'s internals.

2.1 Functionalities

Figure 1 shows the three layers of abstraction of `node-indri`. Our module exposes `Indri` features through the `Searcher`, `Reader`, and `Scorer` classes. These classes are implemented in C++ with the help of Native Abstractions for `Node.js`[4], a series of macros that abstract away the differences between the `V8` and `libuv` API versions (which together form the core of the `Node.js` framework and are written in C++).

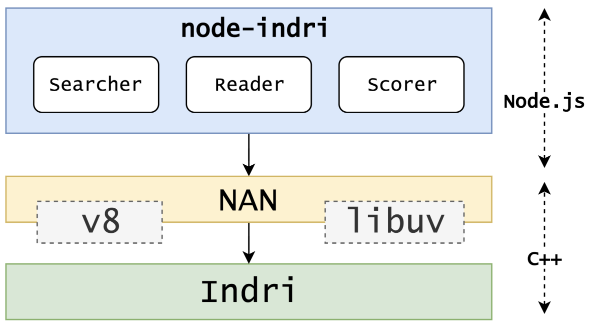

Fig. 1. Overview of `node-indri` layers of abstractions and their implementation languages and platforms.

In Table 1, we list the arguments of `node-indri`'s three classes. Each class has at most two methods with arguments that depend on the functionalities exposed from `Indri`. The last argument is a callback function implementing the error-first pattern. In this manner, `node-indri` is an asynchronous module, with most of these functions assessing lower-level system functionalities through `libuvl`. This in turn means that the methods are executed in `Node.js`' thread pool, making `node-indri` naturally parallel.

[4] https://github.com/nodejs/nan

Table 1. Overview of the arguments necessary for `node-indri`'s method calls. Our API is simple and includes only one method per class. The last argument is always a callback that is executed when the data has been retrieved. Underlined are the required parameters.

`Searcher.search`	`Reader.getDocument`	`Scorer.scoreDocuments`	`Scorer.retrieveTopKScores`
query, page, results-per-page, feedback-docs, callback	docid, callback	query, docs, callback	query, number-of-results, callback

The models' hyperparameter settings (e.g. μ in the case of language modeling with Dirichlet smoothing) are manually set via a configuration file. We now discuss the goal of each of the three classes `node-indri` makes available to its users in turn:

`Searcher` This class exposes the functionalities of `Indri`'s `QueryEnvironment` and `RMExpander` classes through the method `search` which returns a list of search results in a paginated manner. When a `Searcher` object is instantiated, it takes a configuration object as argument (these settings include the retrieval models' hyperparameters and flags of the type of data to return). When a call to `search()` is made and no feedback documents are provided as argument, the standard query likelihood model is employed, otherwise RM3 is [7]. Depending on the configuration settings, the returned result list may contain document snippets (as provided by `Indri`'s `SnippetBuilder`), document scores, document text and other metadata.

`Reader` This class exposes the functionalities of an `Indri` index through the method `getDocument` in order to return a document's meta- and text data.

`Scorer` This class provides access to the retrieval scores of a list of documents via the method `scoreDocuments`. In addition, it provides `retrieveTopKScores` to retrieve the scores and document ids of the top ranked documents for a query.

2.2 Use Cases

We have used `node-indri` as search results' provider in the backend of a large-scale collaborative search system, `SearchX` [8]. `SearchX`'s backend supports the inclusion of many IR backends such as Elasticsearch and Bing API calls. In order to include `node-indri` as one of the supported backends, we implemented `Searcher` to provide search results (with or without snippets) in a pagination manner leveraging feedback documents, `Reader` to enable the rendering of a document's content when a user has clicked on it, and `Scorer` to enable our backend to have direct access to documents' scores for reranking purposes.

3 Efficiency Analysis

We now present an efficiency analysis of `node-indri`, comparing it to `Indri` and `Pyndri`. We indexed two standard test corpora—Aquaint and ClueWeb12B—

with `Indri` and measured the execution time for 10k queries of the TREC 2007 Million Query track[5] across the three toolkits. As retrieval model we fixed language modeling with Dirichlet smoothing; up to 1000 documents were returned per query. We use `Indri`'s `IndriRunQuery` application for this purpose; for `Pyndri` and `node-indri` we implemented scripts to achieve the same behaviour. Specifically, `Pyndri`'s results were obtained with a `Python` script implemented with `multiprocessing`. In `node-indri`, we make use of `Promises.all`. We limit the execution to 20 threads for all three toolkits[6]. Table 2 presents the overall query execution time of the three toolkits.

Our results show that `node-indri` has execution times comparable to `Indri` and `Pyndri`. We can thus use `node-indri` efficiently in modern web backend development. We find for a small collection such as Aquaint (1 million documents), all three toolkits to obtain very similar execution times (between 25 and 29 seconds). In contrast, for a larger collection such as ClueWeb12B (50 million documents), the execution times differ to some extent: `Indri` takes on average 27 minutes to process 10K queries, while `node-indri` and `Pyndri` take 34 and 37 minutes respectively. This is expected, as both `node-indri` and `Pyndri` have additional overhead due to the frameworks they are built upon.

Table 2. Overview of retrieval efficiency (in seconds) across two corpora. 10K queries from the TREC 2017 Million Queries track were executed 20 times. Reported are the average (standard deviation) execution time of the batches.

	Aquaint	**ClueWeb12B**
`Indri`	29s (0.30s)	1645s (20s)
`Pyndri`	25s (1.22s)	2262s (340s)
`node-indri`	25s (0.58s)	2058s (338s)

4 Conclusions

We have introduced `node-indri`, a `Node.js` module to enable users with a good web development background (but minimal C++ knowledge) to efficiently implement search applications. We have described how `node-indri` exposes `Indri`'s functionalities and how it is currently being used in the backend of a large-scale collaborative search system, that has been successfully tested with several hundred users. Furthermore, we compared `node-indri`'s efficiency in the batch setting with `Indri`'s and `Pyndri`'s.

Acknowledgements: This work was funded by NWO projects LACrOSSE (612.001.605) and SearchX (639.022.722). The authors would like to thank Harry Scells for his input.

[5] https://trec.nist.gov/data/million.query07.html
[6] More details and benchmark code are included in the GitHub repository.

References

1. Strohman, T., Metzler, D., Turtle, H., Croft, W.B.: Indri: A language model-based search engine for complex queries. In: ICIA. (2005)
2. Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., Lioma, C.: Terrier: A high performance and scalable information retrieval platform. In: OSIR
3. Yang, P., Fang, H., Lin, J.: Anserini: Enabling the use of lucene for information retrieval research. In: SIGIR. (2017)
4. Macdonald, C.: Combining terrier with apache spark to create agile experimental information retrieval pipelines. In: SIGIR. (2018)
5. Van Gysel, C., Kanoulas, E., de Rijke, M.: Pyndri: a python interface to the indri search engine. In: ECIR. (2017)
6. Tilkov, S., Vinoski, S.: Node. js: Using javascript to build high-performance network programs. IEEE Internet Computing (2010)
7. Lavrenko, V., Croft, W.B.: Relevance based language models. In: SIGIR. (2001)
8. Putra, S.R., Moraes, F., Hauff, C.: Searchx: Empowering collaborative search research. In: SIGIR. (2018)