Improved User News Feed Customization for an Open Source Search Engine

CS 297 Project Report

Presented to

Dr. Christopher Pollett

San Jose State University

By

Timothy Chow

May, 2019

Abstract

Yioop is an open source search engine project hosted on the site of the same name. It offers several features outside of searching, with one such feature being a newsfeed. The current newsfeed system aggregates articles from a curated list of news sites determined by the owner. However in its current state, the feed is limited in size, being able to utilize around 50 sources. One of the goals for my project will be to increase this amount. I will also be implementing the ability for users to personalize the newsfeed for their own use. In order to accomplish this task, I have spent the last five months working to improve my own familiarity with Yioop, first by adding test cases and then slowly working towards adding features.

Introduction

Within the realm of search engines, one particular subgroup that remains increasingly important and relevant is the idea of vertical searches. Whereas a general search

allows for results to be returned based off a general relevance measure, vertical searches aim to provide more directed results based off of some specific group or interest, with one popular vertical slice that is particularly useful to most people being news results. Yioop is one such search engine which provides this service to its users.

Yioop is an open source search engine built from PHP and made to be extensible for general web, news, image searches, and even a fully functional discussion board and wiki system. For this project, what I am interested in is the current implementation of the news search or newsfeed system. Although it works fine right now, there is actually a hidden limitation to it, where the amount of results returned cannot surpass a certain amount. Normally for a regular search, we would ideally be able to traverse as many results as the system has crawled, but in the case of newsfeed items, we have a hard cap. The reasoning behind this is that regular search items are indexed and archived in such a way that allows consecutive smaller sized chunks to read in, whereas the newsfeed items right now are only stored directly on the database. While this provides some benefit, in the form of easily being able sort from the most recent item to the oldest item, we also cannot store as many. With this in mind, this also prevents us from being able to add too many sources, otherwise the number of items per source would either be small or unbalanced. In order to accomodate for more data in the future, I will be working towards creating a system in which newsfeed items are stored in a similar fashion to regular search items and eliminate this size limitation. Additionally I will have a new system set up to take advantage of this change and allow users to suggest news sources to Yioop, similar to how it already is for general web page crawl sources.

Deliverable 1: Additional Test Cases for IndexShardTest

To ease into this project, I started off with the simple task of writing up new tests for the existing functions in Yioop. To keep it relevant, I wrote test cases for the IndexShard class, which is the data structure used to store a generation’s worth of the

inverted index. IndexShards are then compiled into an IndexArchiveBundle which is the primary way that crawl results are stored and retrieved from for the search engine.

The main goal of this particular deliverable is to gain some knowledge of what sort of methods are used from an instance of IndexShard. This is sort of accomplished by looking at the unit test code to see what main methods are usually called. From the test cases, the major function of IndexShard is to act as a data structure meant to store either documents or links to documents. This is done by using addDocumentWords(), which takes a document id, a posting list, some meta id, and finally an offset. There is also an additional argument for whether we are adding a document or a link. Once documents have been added, we can then perform a search using getPostingSliceById, which will return a result set containing the all documents that contain a certain word in their posting list. Other operations include taking two IndexShards and combining them together to create a bigger one, as well as changing the offset of the documents in the shard.

One of the first things that I noticed while working with this test was that some of the tests were out of date. For example, the flag for adding either documents or links was completely reversed in IndexShard, but the tests do not reflect that change. Additionally, the test to merge multiple shards implies that three shards supposed to be merged, but the actual test only constructs two shards. The documents that were supposed to be added to a third shard were actually being added to the first shard. Miraculously, none of the existing tests failed, but I went ahead and fixed those as well as added a few more just to more it more robust. Modifying this test was also a good opportunity to get into the workflow of pushing new changes to Yioop. This included getting the most recent revision of Yioop at all times, adding my own changes, then creating a patch to upload to the issue tracker for Yioop. While seemingly a slow process, it served as a way to really look over the changes I made in the code and making sure there was no unexpected behavior.

Deliverable 2: Word Tracker to Display Trending Words on Yioop

After some experience in modifying something simple in Yioop, it was time to add something a little more ambitious to the system. The idea was simple, to work with the job which was most closely related to the newsfeed system and put some extra functionality into it. For this deliverable, I added a new tool, tentatively called the word tracker. In order to develop word tracker, it was necessary to understand a few thing: the how Yioop updated its index, how data was temporarily stored, and how to subsequently read that data and display on a web page. By the end of this deliverable, I should have good knowledge on how index construction happens, how to add data to the existing database, and how to add a new page to Yioop.

The first part of the process begins in MediaJob, which in turn calls FeedUpdateJob. FeedUpdateJob is a process that runs every hour to update the index shard. New documents are obtained from a predeterminded source list, and their contents are added into the database. After all the documents have been parsed, the job will rebuild the feed shard looking only at fresh documents which, in our case, are those which at less than a week old. After rebuilding the shard, old items are pruned. To add my functionality for word tracking, I have it so that the word occurrences are saved into a new table within the database. Here, stop words are filtered out before also getting stemmed in order to avoid redundant words. The table keeps track the top hourly, daily, and weekly 25 words. The top hourly words are the simply the top words for this run of FeedUpdateJob, but the top daily and weekly words are calculated using the past 24 hours and the past 7 days worth of data respectively. Once this is calculated and saved, old data is deleted from the table.

Now that we have actual data to work with, we need some place to view it. At its core, Yioop uses a basic model-view-controller architecture, creating a basic loop of

users interacting with a controller which either modifies or grabs data from the model, and that in turn displays a view back to the user. Web pages using a basic view page as the base before rendering individual elements on top of that view. Each element should have the necessary data passed into it using a corresponding model. In my case, I wrote a TrendingModel which pulls the necessary data from the database and passes to a TrackerElement which renders on top of the SearchView. I also reworked the SearchController and main index file such that you can reach the word tracker tool from "yioop.com/trending". Alternatively, one can reach it from the tools page located on the footer.

Deliverable 3: Prototype user Interface for user added news sources

One part of the final product is to expand on the number of sources that can be added to the newsfeed and, subsequently, the number of results. A different part is to allow non administrative users to add news sources that will be crawled. This deliverable deals with planning the latter part of this task.

From the top, we need to deal with two important questions, how should users be able to add these new sources to Yioop, and after they are added, what can we do with them? In my prototype, users will need to be logged into Yioop in order to even suggest a source, as a mostly to preventative measure against potential spam from random anonymous people. Once logged into their account, there will be a new option to suggest a source using a form. This form is mostly derived from the same exact format that admins can use to add sources, and in essence it is a standard form with dynamic fields based on which type of media source is being added. So far, the supported types include RSS, HTML, JSON, and regular expression feeds, as well as podcasts. At the bare minimum, each source needs be given a name to identify by, the URL that Yioop will run FeedUpdateJob on, and the language of the source. With other sources, they may be some additional fields, such as indicators to parse feeds. In order to incorporate

this feature, there will need to be changes made to SocialComponent and SourceModel, plus a new SuggestnewsElement will need to be added.

When a suggestion is made by a user, the item is automatically added into the database. However, it is entirely possible and probable that the source might not work entirely. As a result, these recently added sources will default to being disabled, and the FeedUpdateJob will ignore them. The final decision to use the source is left to users with root privilege and can manually enable or disable sources within their account. Ideally root users should use the built-in testing function within Yioop to check for errors before letting sources be used in order to avoid errors down the road.

Deliverable 4: Prototype framework for NewsFeedBundle

The other part that will need to be fleshed out for this project is to create a system that will allow more news results to be stored. Currently, Yioop runs MediaUpdater which aggregates several different update jobs. The job we are interested in is the FeedUpdateJob, which looks at a list of sources from the MEDIA_SOURCES table in the database. For each source, we parse out the necessary information, add it into the database in FEED_ITEMS. The problem with this existing approach is that storing it exclusively in the database puts some limitations on how many items we can store. In contrast, the main search engine part of Yioop stores items using IndexShards, which are grouped into bundles as each shard is only meant to store up to a certain limit. During a crawl, we just add whatever document or link that we see and then move on. For a news crawl however, it would be prudent to design it in such a way that we access the newest items first before moving backwards in time. Since it is stored on the database right now, it is simple to just sort through by timestamp in descending order, but the goal of this project is to migrate this storage into shards and bundles, hence NewsFeedBundles.

There are two current approaches to going about this, one would be designing a new class that constructs the shards and the dictionaries inside each one in reverse order, and the other would be to keep the existing construction method, but instead we change it so that we traverse it backwards. The plan right now is go with the second method, with the expectation of making as few minimal changes as possible to the existing code. For IndexArchiveBundle, I might need to create a flag setting which would tell subsequent iterators to read the shards and their in reverse as opposed to the usual. Just for convenience, I will refer to bundles with this flag set as reverse bundles. When an iterator reads in a reverse bundle, one way we could accommodate this is to start our numbering at zero and actually move into the negatives the more recent something is. That way, our iterator would still be going from recent to old, while also retaining the old format of going from small to high.

Conclusion

The work that was performed in CS 297 is intended to help or benefit myself greatly when I finally start to work on the actual final deliverable for CS 298. For each of these deliverables, there is a certain focus to it, mostly centered around working and dealing with Yioop, since that is basically what I will continue to work with. The first deliverable served as a simple introduction to the sort of classes that I should be paying attention to, while the second and third deliverables had me actually work and improve on the existing code for Yioop in both the front and back end. Finally, the fourth deliverable is intended to act as a starting point from where I can begin CS 298. Assuming that everything works out by the end of CS 298, there should eventually be a full on revision to the existing method of feed item storage.

References

[1] Jongdeog Lee, Daniel Xu, Md Tanvir Al Amin, Tarek Abdelzaher; iApollo: A Newsfeed Summary Service on NDN; iEEE, 2017.

[2] Nicola Ferro, Yubin Kim, Mark Sanderson; Using Collection Shards to Study Retrieval Performance Effect Sizes; ACM, 2019.

[3] Bo Long, Yi Change; Relevance Ranking for Vertical Search Engines; 2014

[4] Pollett, C. "Open Source Search Engine Software!" Open Source Search Engine Software. https://www.seekquarry.com/