Improved User News Feed Customization for an Open Source Search Engine

Timothy Chow

Agenda

- Introduction
- Background of Yioop
- Yioop Indexing
- Index Storage
- Reverse Iteration
- Testing
- Conclusion

Introduction

- In the past, one of the big problems was distribution of stories
    - Newspapers were local, region locked
- Now the Internet allows for stories online
- This allows for two benefits
    - Distribution is no longer dependent on area or supplier
    - Cost to user is generally free
- 61% of Americans get their news online from the Internet on a typical day.
- New problem rises:
    - Now that users can freely choose stories from anywhere online, how to pick which ones

Content Aggregation

- Content is posted on several different pages
- Instead of human visiting all sites, have machine or system
  - System will have to crawl and save all the items
- Collected results are presented at the end to the user
  - Results still need to be ranked or sorted in some meaningful way
- One of earliest examples is Yahoo! News in 1996
- Web syndication

Aggregation Methods

- Typically, website content stored in HTML format
- Data stored using tags and attributes
    - Good for layout and design, not so much for sharing
- Web feed formats created to solve this
    - XML, YAML, JSON, RSS
- Aggregation based on pull strategy
    - Feed document contains text and metadata
    - List of feeds provided to aggregator
    - Aggregator pulls from each feed and stores it

News Ranking

- After items are stored, they need to presented to user in the best way
- Search engines use a scoring system based on relevancy on query terms
  - Calculated using frequency of search terms matching inside a document
- News feeds ranking prioritizes age of document, or freshness
  - Other major factors could include clustered weight and source authority
- More intricate systems will determine temporal freshness
  - More obscure features such as story coverage or query frequency within a given time slot

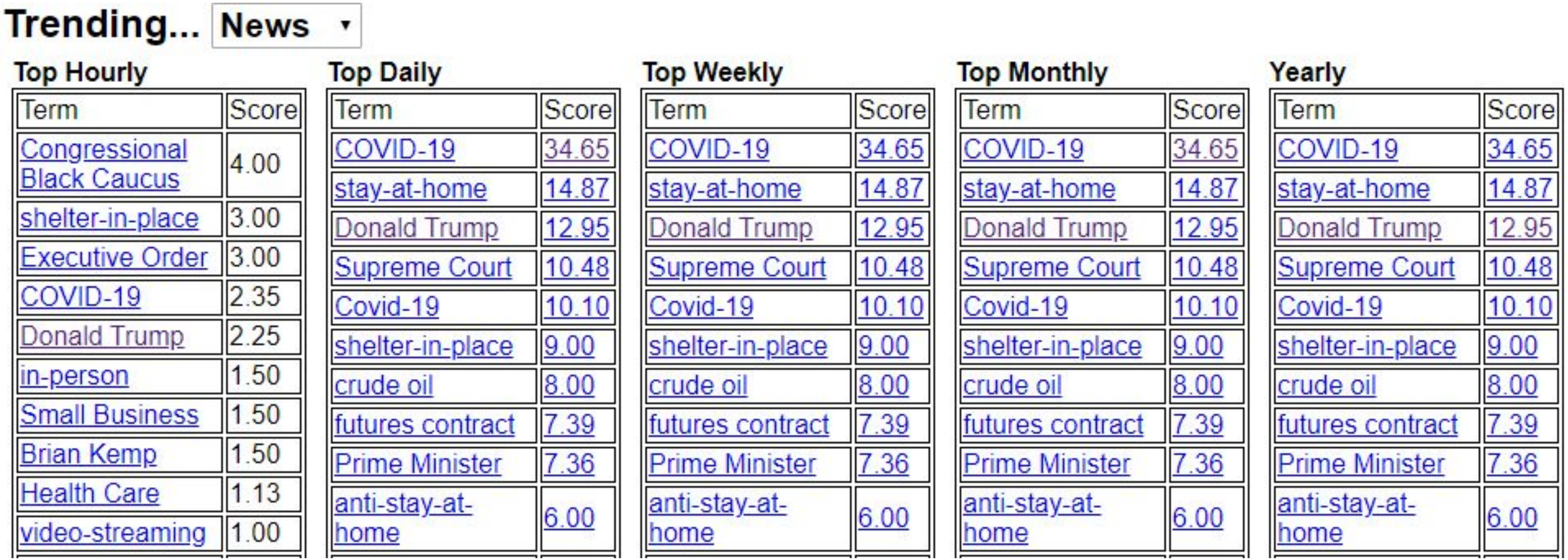
Existing News Aggregators

- Google News
    - Stories are ranked in order of perceived interest
    - Similar stories based on subject are clustered
    - Specified to each user
- Facebook News
    - Stories focused on groups or friends on Facebook
    - Four steps: inventory, signals, predictions, and scoring
    - Also user specific
- RSS feed aggregators
    - Mixes different feeds provided by user, but nothing more
    - Similar to Yioop

Trending Words

- Feature in Yioop used to keep track of the top “trending words”
- Word and their occurrences are saved during a news feed update
- Word count is used to calculate some statistics
- Could be used for clustering or search engine optimization(SEO)

Trending Words



Trending... News ▾

Top Hourly

Term	Score
Congressional Black Caucus	4.00
shelter-in-place	3.00
Executive Order	3.00
COVID-19	2.35
Donald Trump	2.25
in-person	1.50
Small Business	1.50
Brian Kemp	1.50
Health Care	1.13
video-streaming	1.00

Top Daily

Term	Score
COVID-19	34.65
stay-at-home	14.87
Donald Trump	12.95
Supreme Court	10.48
Covid-19	10.10
shelter-in-place	9.00
crude oil	8.00
futures contract	7.39
Prime Minister	7.36
anti-stay-at-home	6.00

Top Weekly

Term	Score
COVID-19	34.65
stay-at-home	14.87
Donald Trump	12.95
Supreme Court	10.48
Covid-19	10.10
shelter-in-place	9.00
crude oil	8.00
futures contract	7.39
Prime Minister	7.36
anti-stay-at-home	6.00

Top Monthly

Term	Score
COVID-19	34.65
stay-at-home	14.87
Donald Trump	12.95
Supreme Court	10.48
Covid-19	10.10
shelter-in-place	9.00
crude oil	8.00
futures contract	7.39
Prime Minister	7.36
anti-stay-at-home	6.00

Yearly

Term	Score
COVID-19	34.65
stay-at-home	14.87
Donald Trump	12.95
Supreme Court	10.48
Covid-19	10.10
shelter-in-place	9.00
crude oil	8.00
futures contract	7.39
Prime Minister	7.36
anti-stay-at-home	6.00

Yioop

- Open source search engine written in PHP
- Designed for crawling the web, archiving, and letting users search
- Index is created using visited sites
- Can be manually set up on personal PC
- Unlike Google, crawl sites can be specified by user, as well as the depth of crawls

Yioop Indexing

- Distributed setup consisting of name servers and queue servers
- Name servers act as nodes, help coordinate crawls
- Each node can have several queue server processes, either to schedule jobs or to index
- Additional fetcher processes that help with downloading and processing pages from crawl
- News feed update job is separate from regular crawling, but similar methodology
    -

Crawling

- Initially set up the list of sites to crawl
- Fetcher processes create a schedule that holds data to be processed later, as well as type of processing required
- Queue server is periodically pinged for list of pages to download before creating a summary
- The summary is a shortened description of the page along with different metadata for indexing
- Unique hash id is assigned to each page and index construction started

Indexing

- In books: an alphabetical list of names, subjects, etc., with references to the places where they occur
- In databases: a copy of a subset of columns which are used to speed up access times
- Overall, two major benefits
  - Index will be smaller in file size than document
  - Lookup on index is faster
- In Yioop, scores for page ranking are also calculated during indexing before POSTing to queue server
- Queue server merges everything into a final inverted index structure

Inverted Index

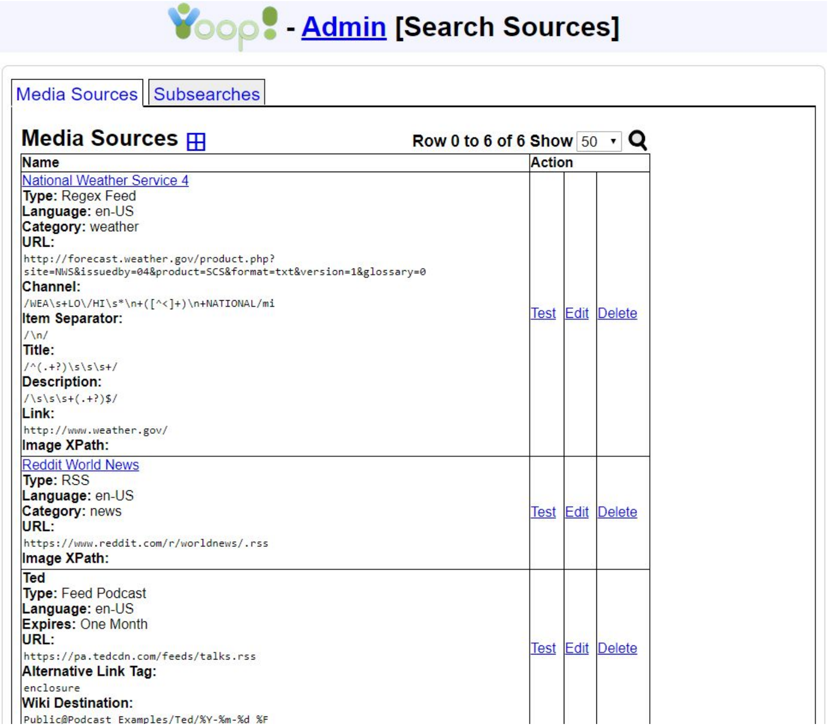
- Consider a collection of documents
- What if I want to return every document that contains a certain term
- Create an index from document->term, known as forward index
    - e.g. doc1 contains term1, term2, term3, term4
      doc2 contains term3, term6
      doc3 contains term1, term9, term10
- Using forward index, create a new index which goes from term->document
- This is the inverted index
    - e.g. term1 is in doc1, doc3
      term2 is in doc1
      term3 is in doc1, doc2

Newsfeed Indexing

- MediaUpdater process handles media jobs
  - Mail server, recommendations, trending, feed update
- News feeds are done by FeedsUpdateJob
- MediaUpdater only runs once per hour, whereas standard crawling is nonstop
- Usual queue server is also designed to crawl with depth in mind, but media jobs only work with a source, e.g. depth of 1

Newsfeed setup

- Media sources can be one of four types
    - RSS, JSON, HTML Regex, or podcast
- Each feed needs correct parameters to function properly
- Assumes sources will be updated with new items over time



Yoop! - Admin [Search Sources]

Media Sources

Row 0 to 6 of 6 Show 50

Name	Action		

National Weather Service 4
Type: Regex Feed
Language: en-US
Category: weather
URL:
`http://forecast.weather.gov/product.php?site=NWS&issuedby=04&product=SCS&format=txt&version=1&glossary=0`
Channel:
`/WEA\s+LO\/HI\s*\n+([^<]+)\n+NATIONAL/mi`
Item Separator:
`/\n/`
Title:
`/^(.+?)\s\s\s+/`
Description:
`/\s\s\s+(.+?)$/`
Link:
`http://www.weather.gov/`
Image XPath:
	Test	Edit	Delete

Reddit World News **Type:** RSS **Language:** en-US **Category:** news **URL:** `https://www.reddit.com/r/worldnews/.rss` **Image XPath:**	Test	Edit	Delete

Ted
Type: Feed Podcast
Language: en-US
Expires: One Month
URL:
`https://pa.tedcdn.com/feeds/talks.rss`
Alternative Link Tag:
`enclosure`
Wiki Destination:
`Public@Podcast Examples/Ted/%Y-%m-%d %F`
	Test	Edit	Delete

Current Bottleneck

- Prior to this project, crawled news items are stored in intermediary database
- Items are then added to a singular IndexShard
- Entire IndexShard needs to be rebuilt for each update
- Database storage performance is influenced by amount of RAM that system has
- Items that are too old have to be removed
- We will explore how index storage works in Yioop and how to change this current implementation

IndexShards

- Lowest level data structure for a index
- Two access modes, read-only and loaded-in-memory
- While in memory, data can also be packed or unpacked
  - New data can only be added while unpacked
  - Only packed data can be serialized to disk
- Each shard has three major components
  - *doc_infos*
  - *word_docs*
  - *words*

IndexShard components

- *doc_infos* - document ids, summary offset, and the total number of words that were found in that document
  - Each record starts with 4 byte offset, followed by 3 bytes to hold doc length, 1 byte to hold number_doc key strings, and the key strings themselves
  - Each key string is 8 bytes containing hash of URL plus a hashed summary
- *word_docs* - string of sequence of postings
  - One posting is a positional offset into a document for where it appears
  - Also contains occurrences of word for that document
  - Only set while IndexShard is loaded and packed

IndexShard components (cont.)

- words - array of word entries stored in shard
    - Exists in two different forms depending on packed or unpacked state
    - In packed state, each word entry is made up of:
        - Term id
        - Generation number
        - Offset into *word_docs* where posting list is stored
        - Length of posting list
    - In unpacked state, each entry is only a string representation of term plus its postings
- When serialized to disk, a shard produces a header with doc statistics and index into *words* component

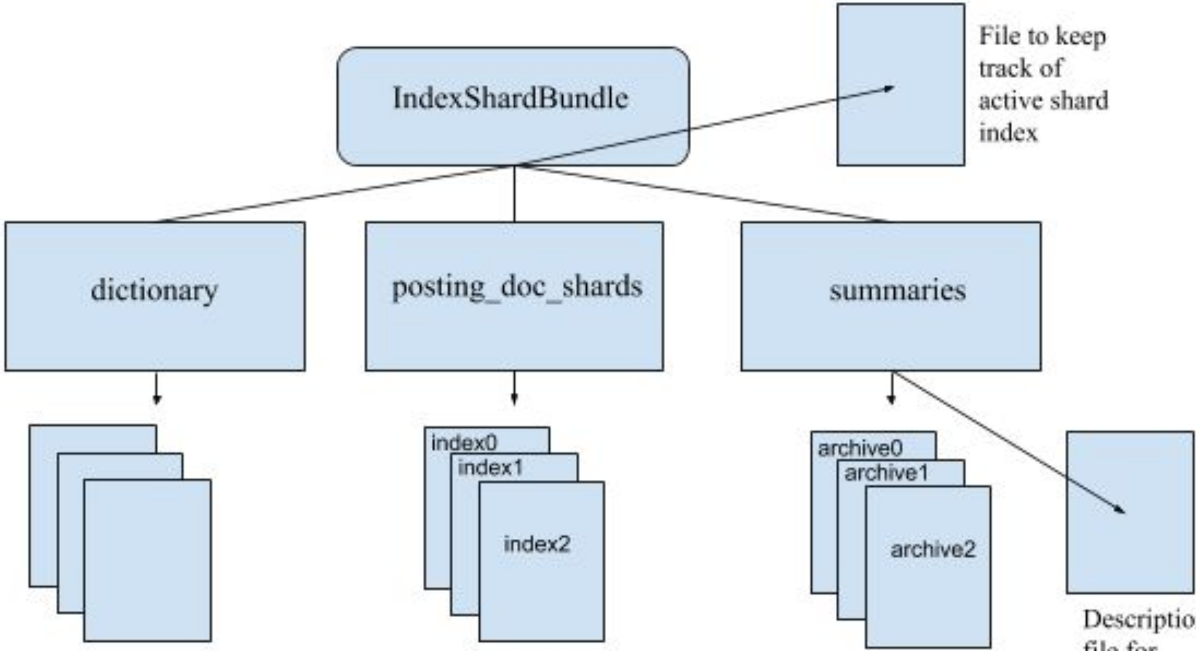
Adding to a shard

- Indexing mostly uses the *addDocumentWords()* method
  - Run after processing a singular page
  - Takes in the document keys and word lists as arguments
  - Keys can include hashed id and host url of a link
  - Word lists is associative array of terms to positions with a document
- Terms are hashed and positions are converted to a concatenated string before being added to *words* component
- Additional parameters such as meta words, description scores, and user rank is added

IndexArchiveBundle

- IndexShards technically have no size limit, but reading a shard into memory is difficult if too big
- Size of IndexShard is determined by how much memory the system has
- To get around this, have multiple generations of IndexShard
- When one shard is full, save to disk and start new generation
- IndexArchiveBundle is a the data structure that holds this together

IndexArchiveBundle structure



IndexShardBundle

File to keep track of active shard index

dictionary

posting_doc_shards

summaries

index0

index1

index2

archive0

archive1

archive2

256 folders corresponding to first letter of hash. Each folder contains a dictionary file with document postings

Collection of IndexShard files saved to disk

Collection of gzipped archives with document summaries

Description file for bundle

Index storage process

- After crawling some pages, we have generated an IndexShard
- First, check if the most recent shard in bundle has enough space to store the new shard
  - If there is, then merge shards
  - If not, then save active shard and start new generation
- At this point, summaries have already been stored in web archive, so summary offsets are added into the IndexShard
- Once everything has been added, IndexShard is successfully added to bundle
- Current news feed storage does not use IndexArchiveBundle

Reverse Iteration

- Because news items added at the end of a shard, we want to be able to move backwards through shards and bundle
- Could have also done backwards construction where items are added at front of shard
- We need a few new things to make this work:
    - New methods to facilitate reverse traversal
    - Some way to designate a bundle’s direction
    - Modification of existing news feed update job to support IndexArchiveBundles

One Slice at a Time

- Information retrieval methods:
    - *first(t)* returns the first position at which the term t occurs in the collection.
    - *last(t)* returns the last position at which the term t occurs in the collection.
    - *next(t, current)* returns the position of the first occurence of t after the current position in the collection.
    - *prev(t, current)* returns the position of the first occurence of t before the current position in the collection.
- Items in IndexShards are retrieved one slice at a time
- A slice is an array of postings and positional information
    - Any location is going to be stored as byte offsets
- We need methods to get move through slices in reverse, and also inside the slice backwards too

Dealing with Offsets

- Retrieve start and end offset of posting list and begin at the end
- *getPostingsSlice()* - given a current offset value, get the offset of previous slice with this term
    - Postings are always 4 bytes long so we know how many postings exist in current slice
- *getPostingAtOffset()* - given an offset, returns a substring from *word_docs* where there is a posting
    - Loop through postings until we reach the start of the posting list
    - When our offset goes below the start offset, we know we have seen all postings for this slice

Dealing with offsets(cont.)

- *nextPostingOffsetDocOffset()* - takes both a current offset and doc offset. Retrieves first posting offset in a slice where the document is also equal or lesser
    - If equal, then next offset in same document, else we want last offset for next document
- Uses exponential search to speed up process
    - Two step search that reduces search range before doing binary search inside that range
- Since working with offsets is finicky, don't let shard access direction be changed

Putting it together

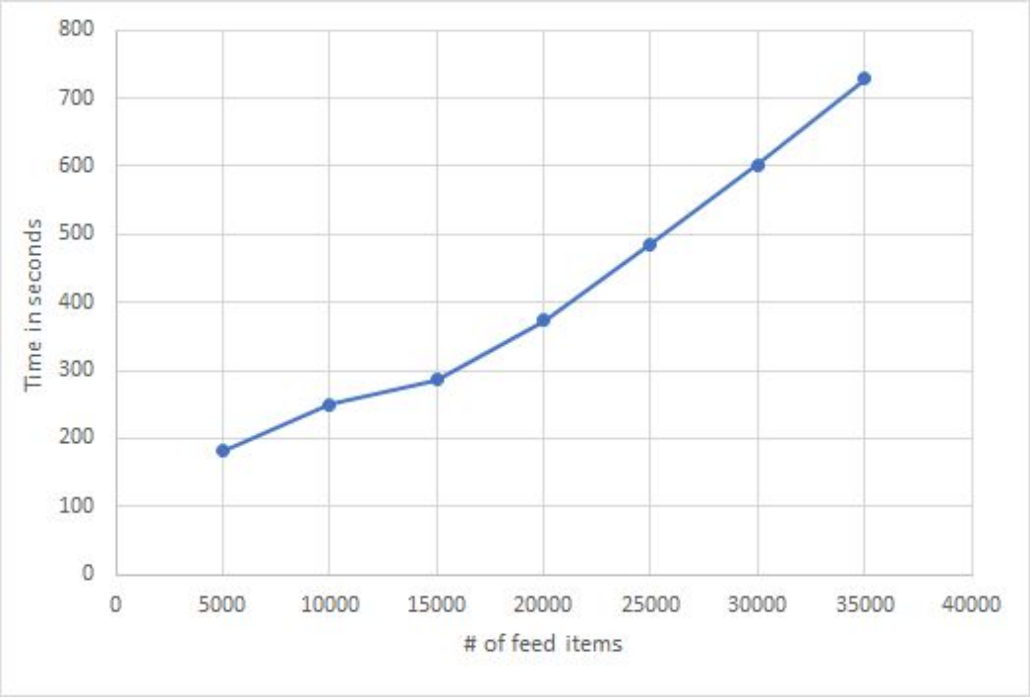
- Instead of having methods in the archive bundle that read shards, we use iterator classes
    - Multiple iterator classes could be used, and we can combine results of multiple iterators
- Iterator looks to IndexDictionary to find shard generations that contain that term
- *advance()* - read in block of shard to memory using start and last offset
    - Only in chunks of up to 800 bytes
- Slight tweaks to news feed update job to create IndexArchiveBundle

Testing

- Performance testing done by setting up fake local RSS feeds
- Feed is populated with miscellaneous data and amount of items is user specified
- Yioop will only pull from these feeds
- Check for speed and scalability
- Finally check to see if each item is retrieved properly after being added

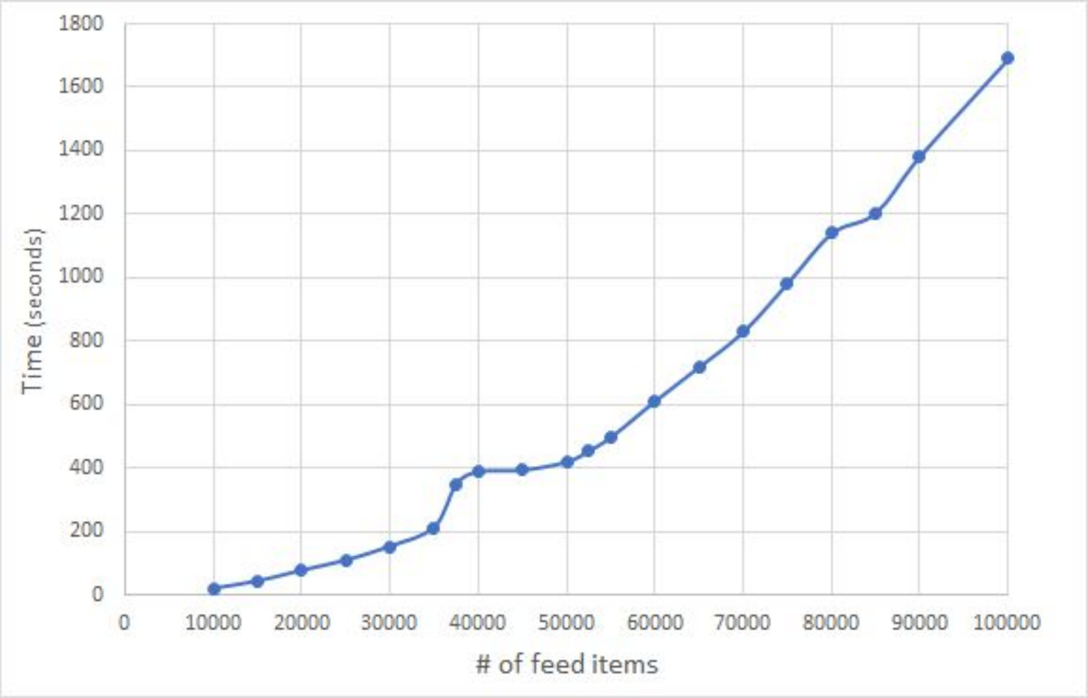
Performance for old Yioop

- Old system is slow when trying to add many items
- LIkely due to database step
- IndexShard only seems to hold approximately 37,500 items
- Old system does not work when adding more than this cap



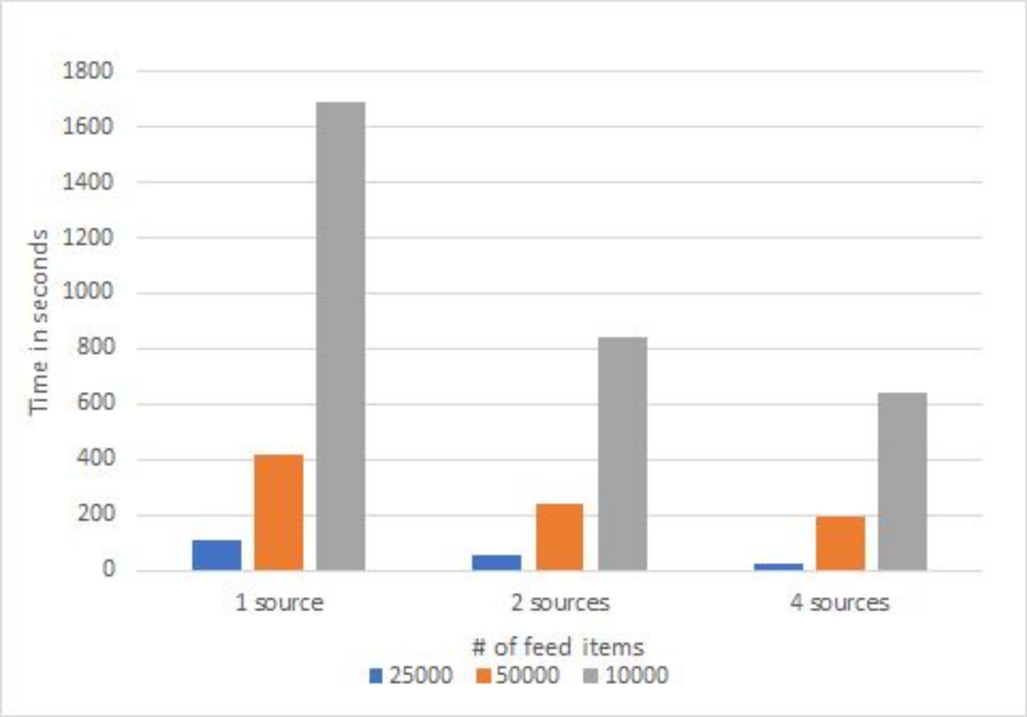
Performance of new system

- Speed is increased greatly over old Yioop
- Not limited in size anymore
- Speed bumps observed whenever new IndexShard is introduced
- Adding a lot of items still slow, but unlikely scenario



Pulling from multiple sources

- Previous testing only use one source
- Multiple sources alleviate the long insertion time
- Closer to real life usage, since most feeds limit to 50-100 items



Conclusion

- New storage solution for news feed allows for better scalability and performance
- Adding the same amount of items is faster, and it overcomes the limitation of holding only one IndexShard worth of data
- Ability to record all seen items instead of removing the oldest ones
- System is already live on Yioop and shown to handle shards correctly

Future work

- Adding to index still gets slow, just not as fast
- Size and format of IndexShards could be optimized further
- Could explore other ways of handling data other than serialized strings
- News feed system currently uses a basic weighting system based on time. Could be changed to be more user specific
- Using trending words, cluster feed items based on topic