Project Group 10

Cy Heffley, Connor Cole, Eduardo Rodriguez, Zach McGee

Objective

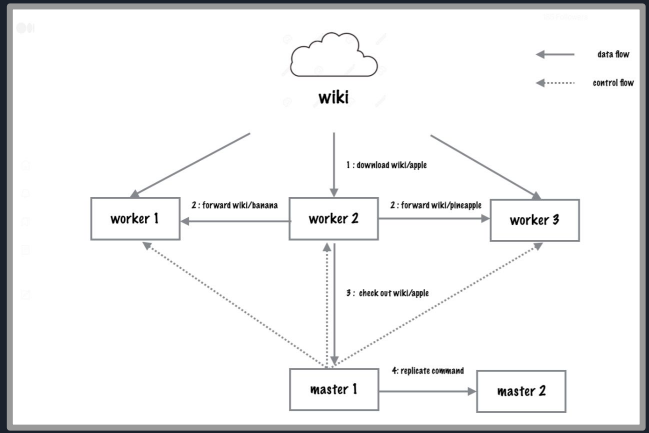
Use modern tools to improve on current internet search technologies, applying a distributed system architecture for scraping websites.

If it is possible to more efficiently crawl the Internet, then a secondary goal is to create a dynamic recommendation system, tailoring results based on the assumption that users with similar interests would wish to receive similar results.

Related Work

In his paper, “Design and Implementation of Distributed Crawler System Based on Scrapy,” Yuhao Fan researched a similar system with quicker persistent memory storage, due to using Redis in conjunction with Scrapy

Issue: False positives in Redis DB increase as time goes on due to Bloom filter used to reduce memory consumption



“Design a Distributed Web Crawler”

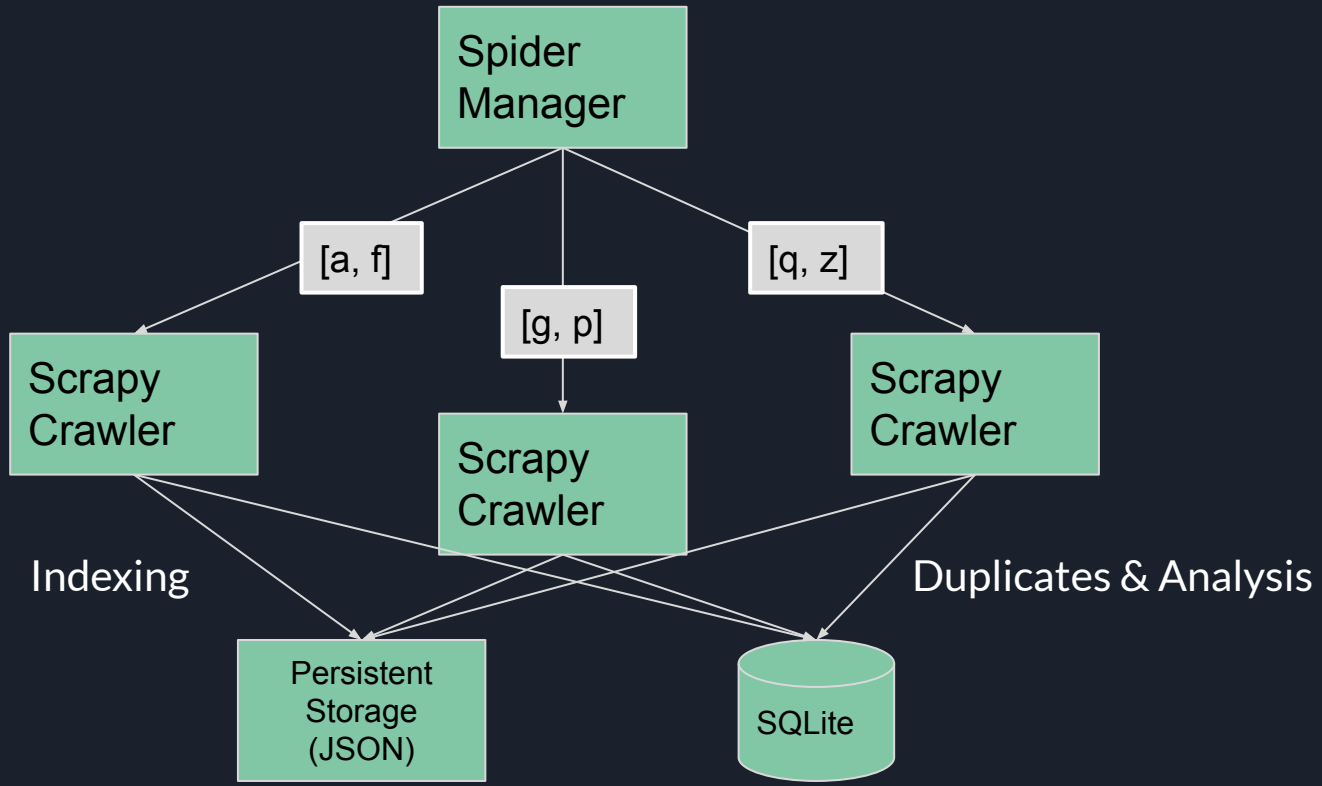
Important definitions

Initial crawling depth - the starting depth of web pages crawled whose links are automatically collected

Filter depth - links are collected past the initial depth, up to the filter depth, only if they have not been previously collected

Maximum depth - links are collected past the filter depth, up to the maximum depth, only if they have not been previously collected and the URL matches the current suffix pattern being crawled

System Architecture



System Architecture and Scrapy

Details:

- Spider manager (multiple_spiders.py) is responsible for running multiple individual scrapy processes
  - Scrapy process: the Scrapy framework makes requests/ handles some data scraping
  - We built custom rules/ functionality/ stats monitoring for our designed framework
    - Specifically, we made rules to help deal w/ different levels of depths and to deal w/ distributed systems vs. just the single system

System Architecture and Scrapy

Details:

- Scrapy crawler (post_spyder) is an individual subprocess
  - Run is managed by multiple_spiders, then acts as an independent asynch process
  - Responsible for that processes stack of URLs, filtering/ decisions made internally on if URL should be added to stack
  - Rules for filtering provided by multiple_spiders

High level code execution

Keywords, depths, seed URL defined before start

Stacks = [start_url]

While there are URLS left in the stack,

- See if URL in database, if we are before initial depth pass, else don't add info/ new links from this URL
- If not in DB, add info/ keywords to DB
- If URL is being considered, add all new links (matching suffix criteria, if applicable) to stack

Periodically writes stats ~every 500 new links a subprocess adds

Thought process behind design

Individual processes use DFS when searching URLs

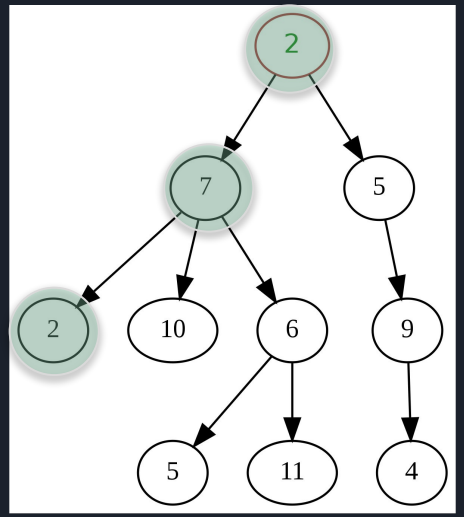
- Similar to traversing down a tree
- Would build up initial stack of URLs to traverse, then new processes would find point in tree to assume responsibility- basically hoping to divide up tree across nodes
- Max depth acts as bottleneck

Tree example- frozen time

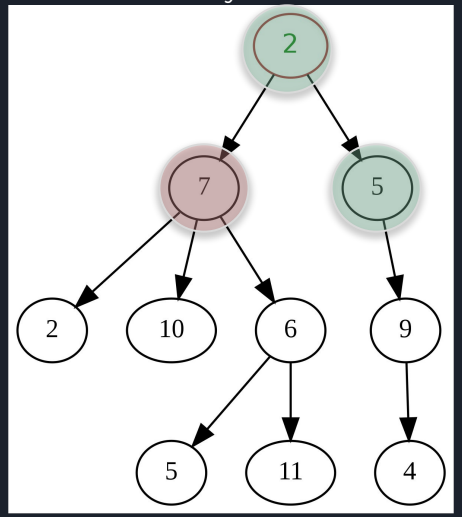
Imagine 2 is a child URL from root URL right at filter depth

- 2 added to booths queues, subprocess 1 takes left side, subprocess 2 sees 7 already in DB so takes right side of tree

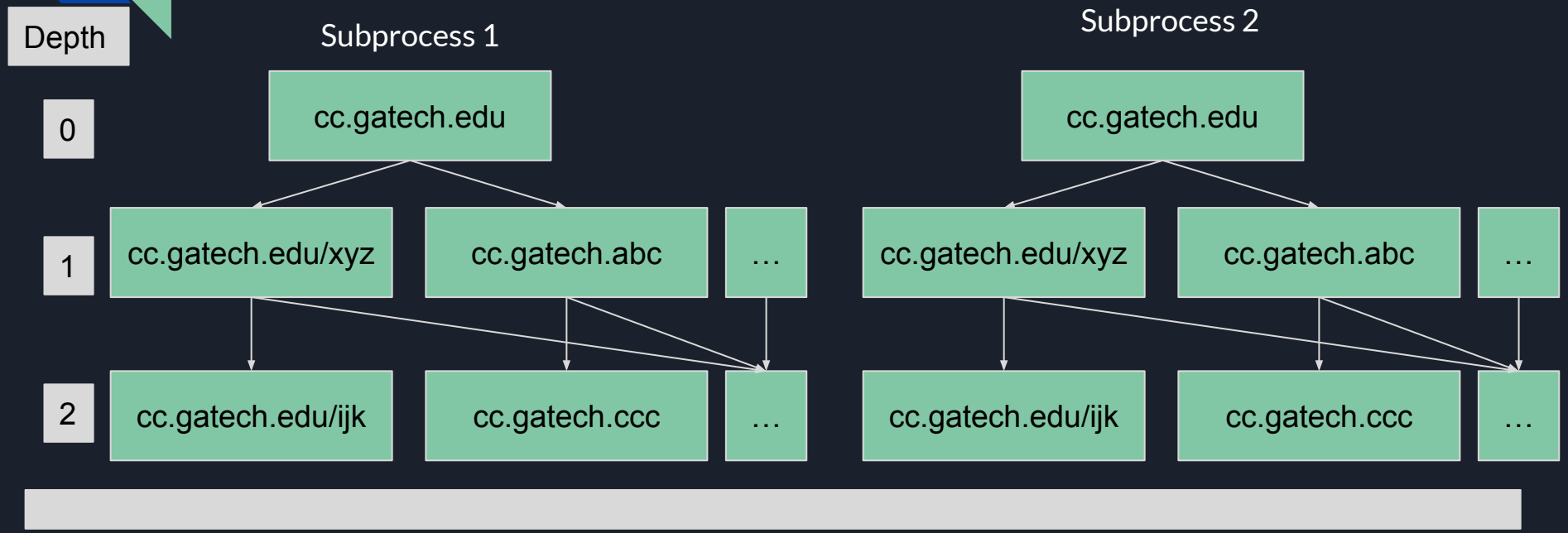
Subprocess 1
Working on URL 2



Subprocess 2
Working on URL 5 as 7 is already in DB



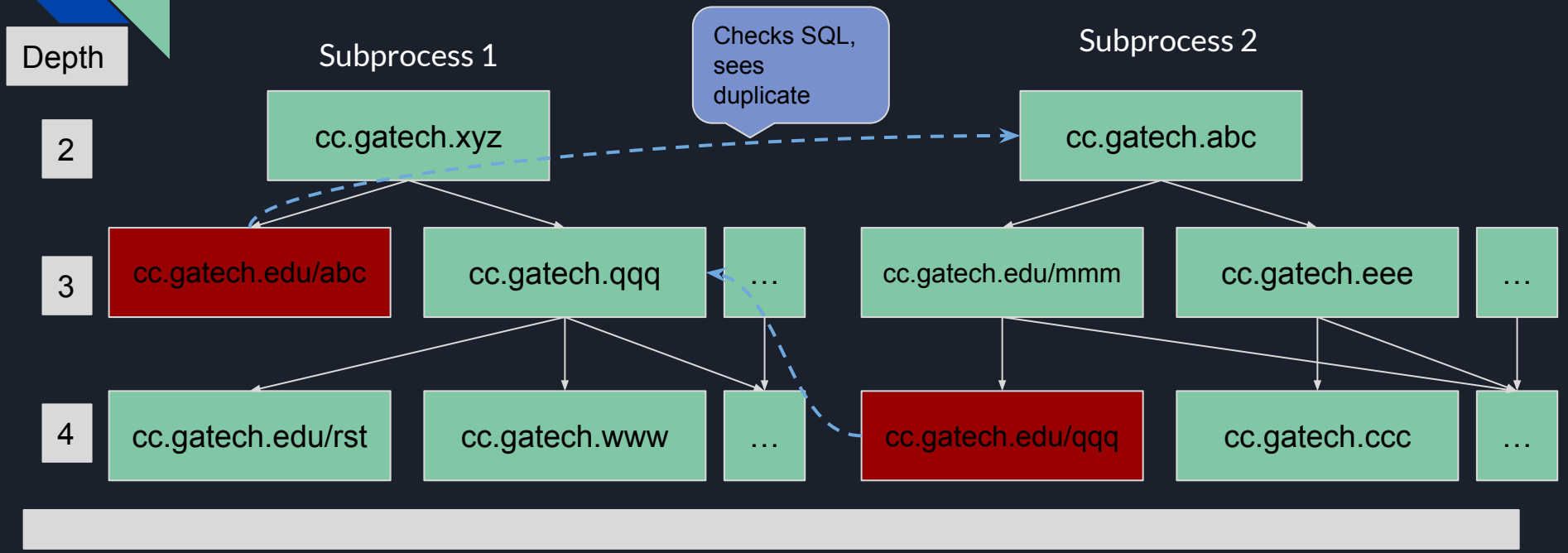
URL Scraping: Initial Depth = 2



Adds all links up to filter depth threshold on each scraper to consider- builds up stack of links to work with

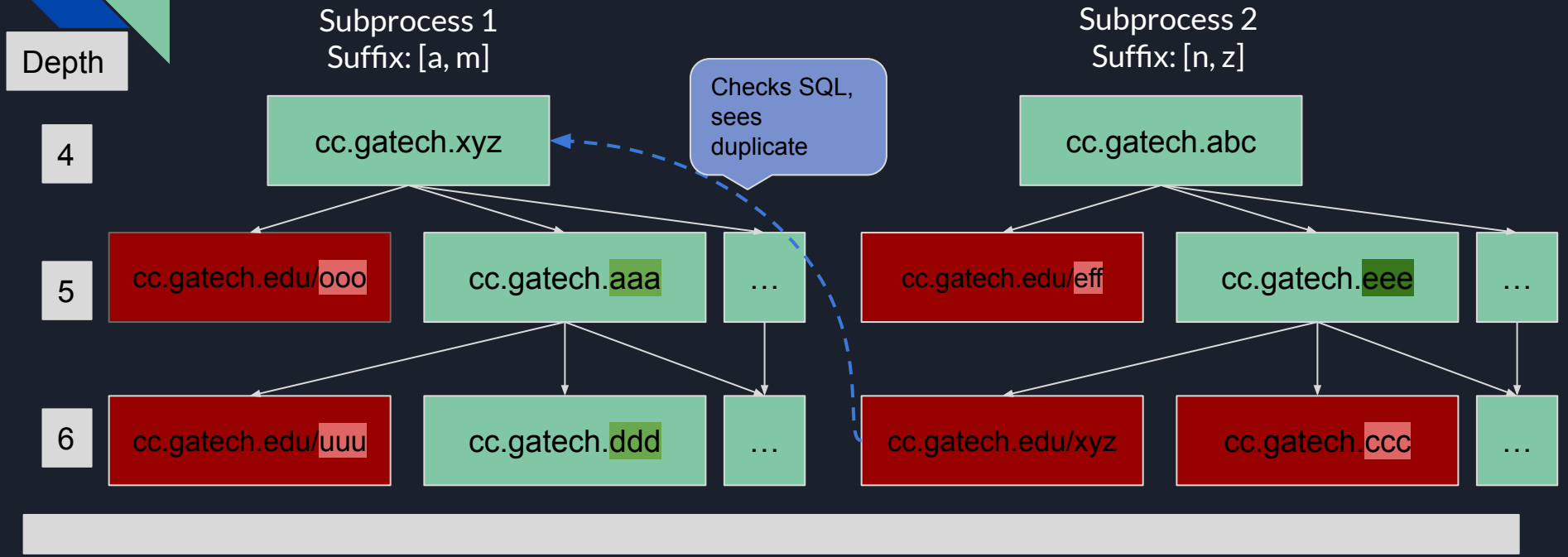
Should be small/ quick to get initial stack built up

URL Scraping: Filter Depth = 4



Queries DB when URL is found to see if exists (if these were perfectly synchronized, cc.gatech.edu/abc has already been added by subprocess 2, not considered in subprocess 1

URL Scraping: Max Depth = 6



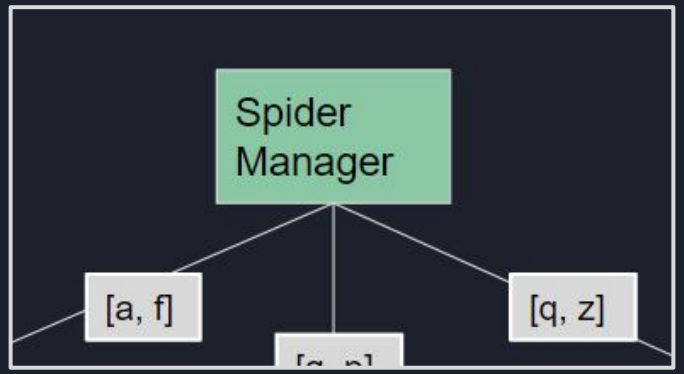
Checks suffix condition, ALSO queries database IF condition is met

Past 6: no links added even if suffix condition met

Spider Manager

Purpose: Divide up different suffixes across multiple scrapers

In true distributed system: manager would be responsible for communication with neighbors, rebalancing suffix rules/ conditions



Technologies

Prototype

Python, Scrapy

SQLite

Production

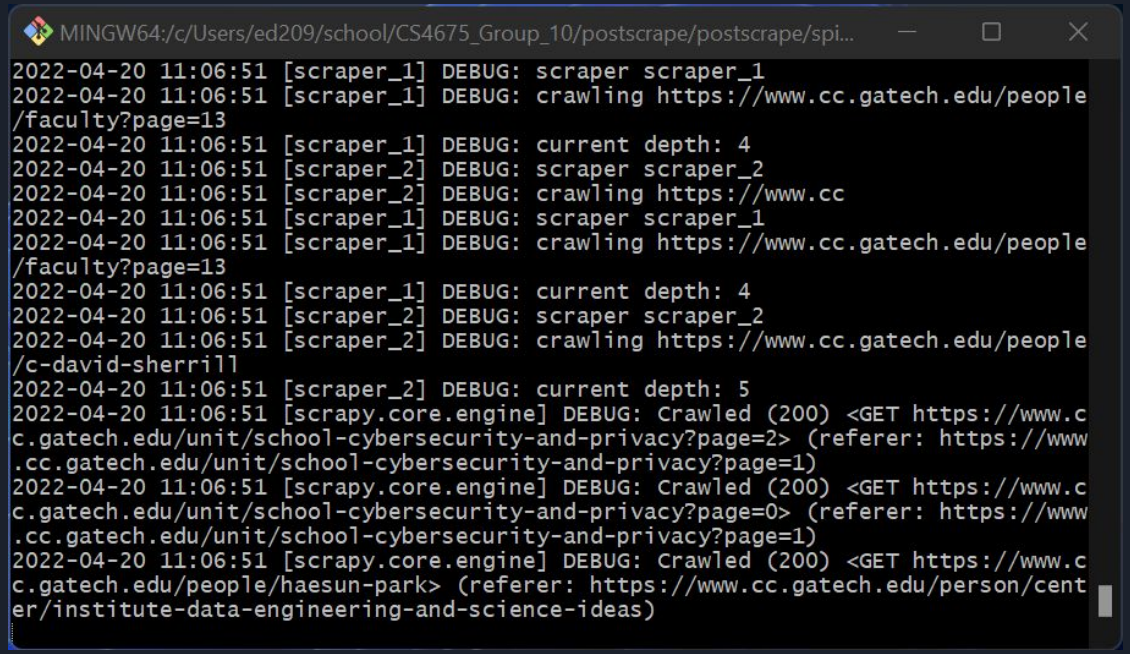
MySQL, PostgreSQL, Apache Hadoop, Redis

C++, Rust, or Apache Nutch

Search service: React + Node, Django, or Spring

Crawling Demonstration

- https://youtu.be/tVVaFffQCNk



Spider Configuration Inputs

Number of spiders in a session: [1, 2, 4]

Initial crawling depth: [2, 3, 5, 10]

Filter depth: [5, 10, 25, 35]

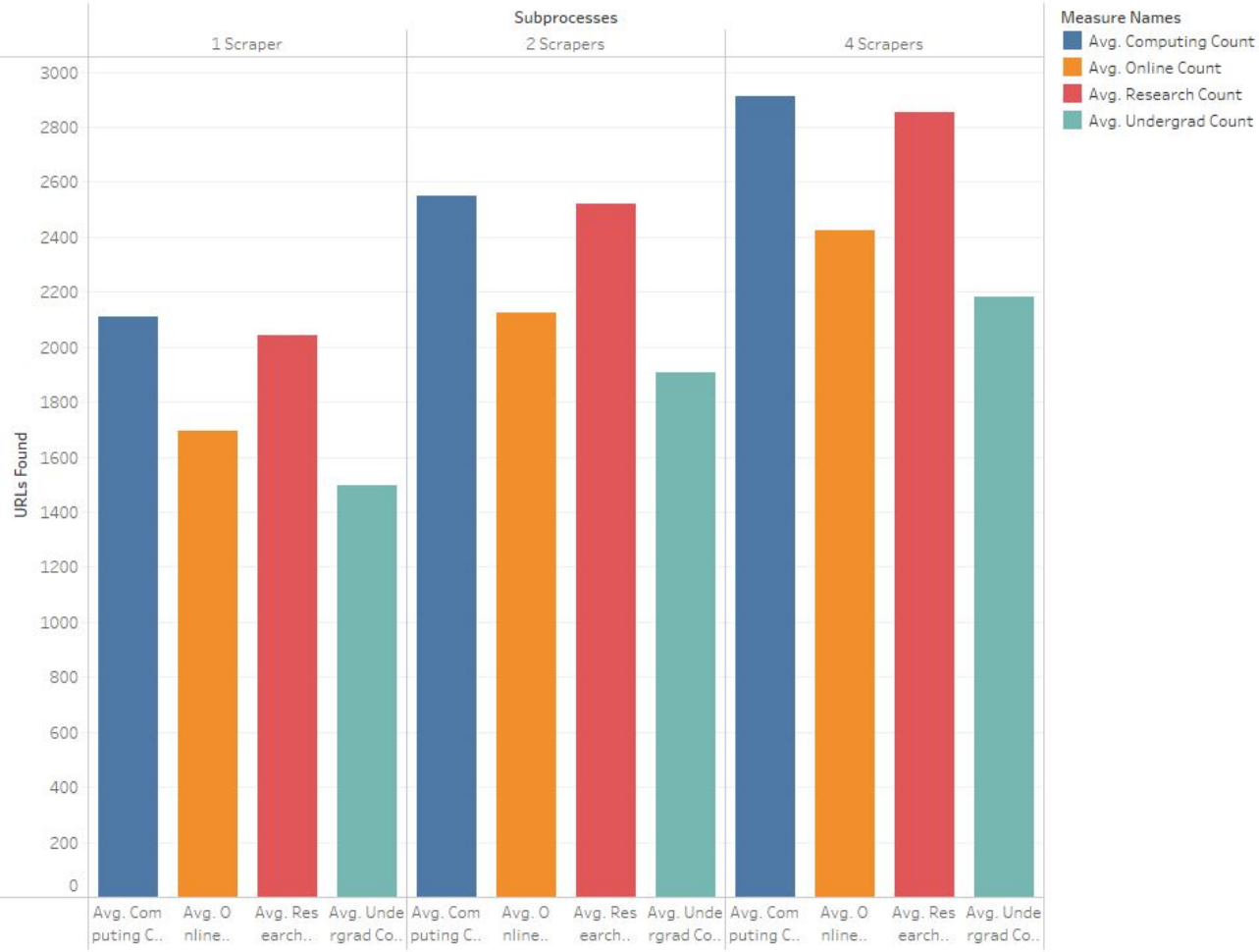
Maximum depth: [10, 45]

Crawling Results of Our Spider Configurations

Evaluation methods/metrics/datasets and results

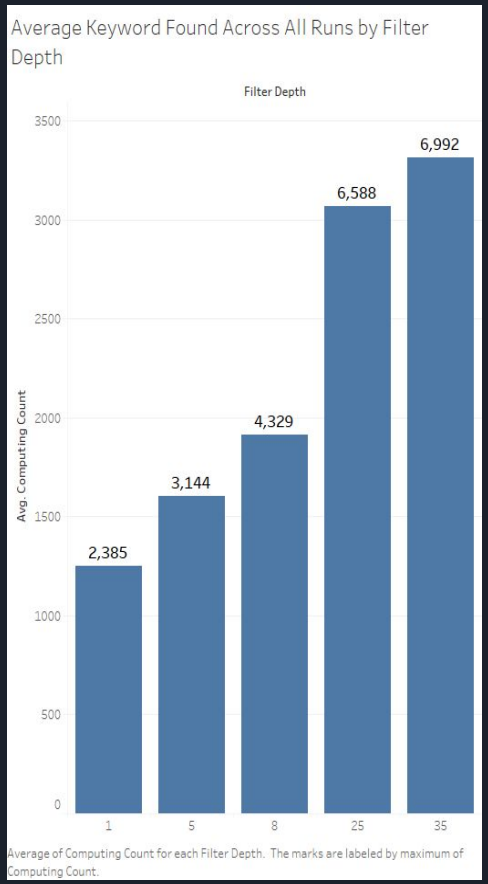
- Compare subprocesses, initial, filter, max depths on:
  - Keywords found
  - Duplicate URLs filtered over time
  - Offsite URLs filtered over time
  - GET Requests made over time

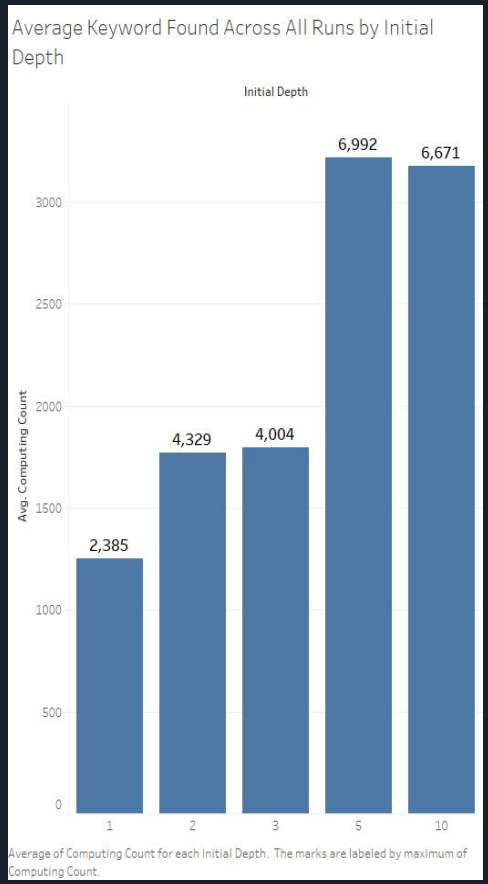
Average Keywords Found By Subprocesses

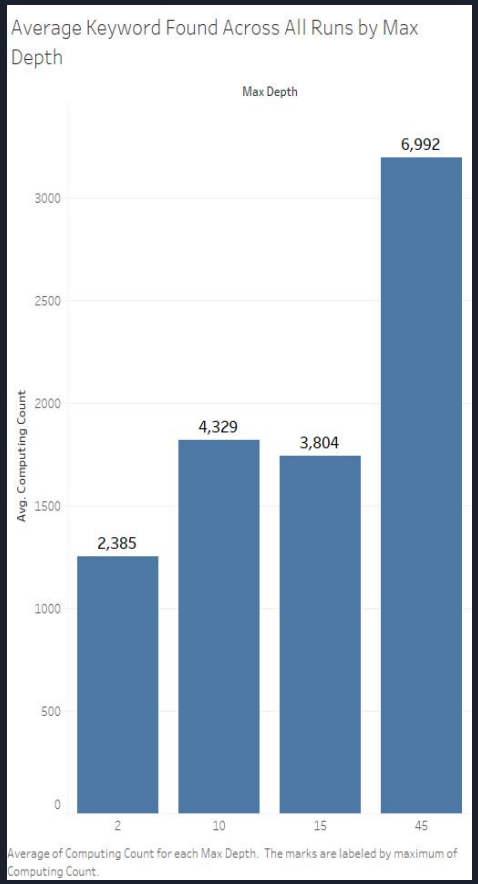


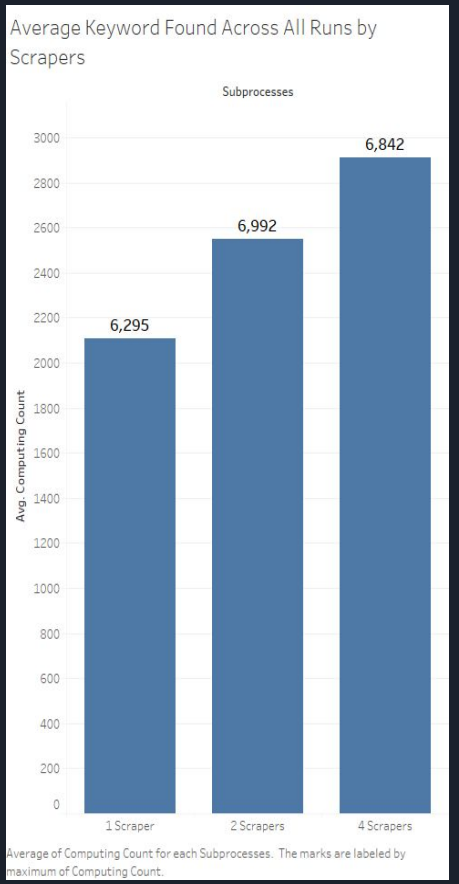
Avg. Computing Count, Avg. Online Count, Avg. Research Count and Avg. Undergrad Count for each Subprocesses. Color shows details about Avg. Computing Count, Avg. Online Count, Avg. Research Count and Avg. Undergrad Count.

We see that generally across all run configurations, adding more scrapers gives better results for each keyword.



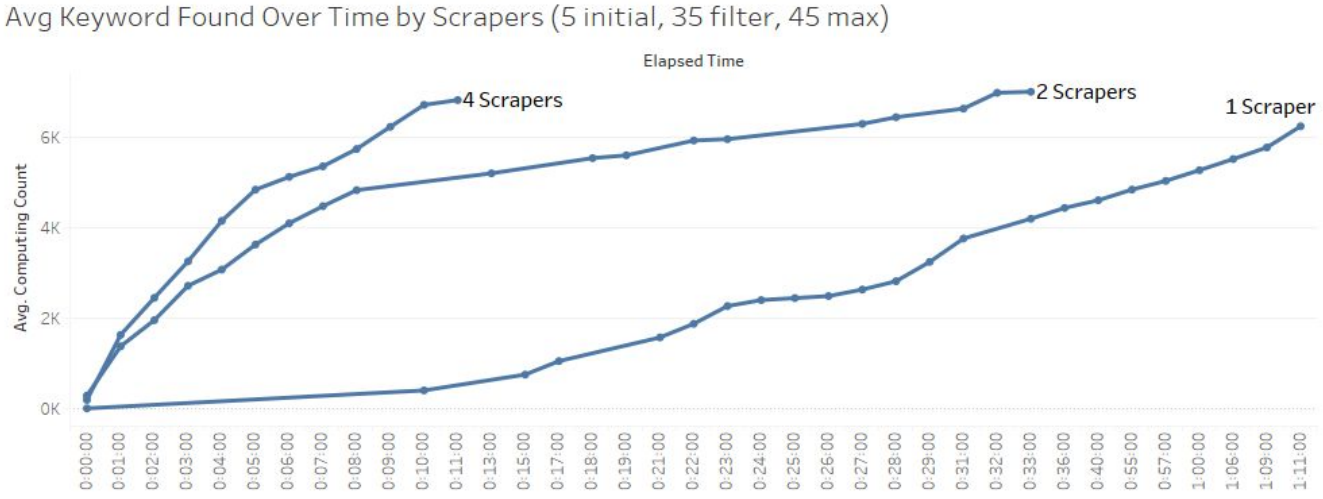


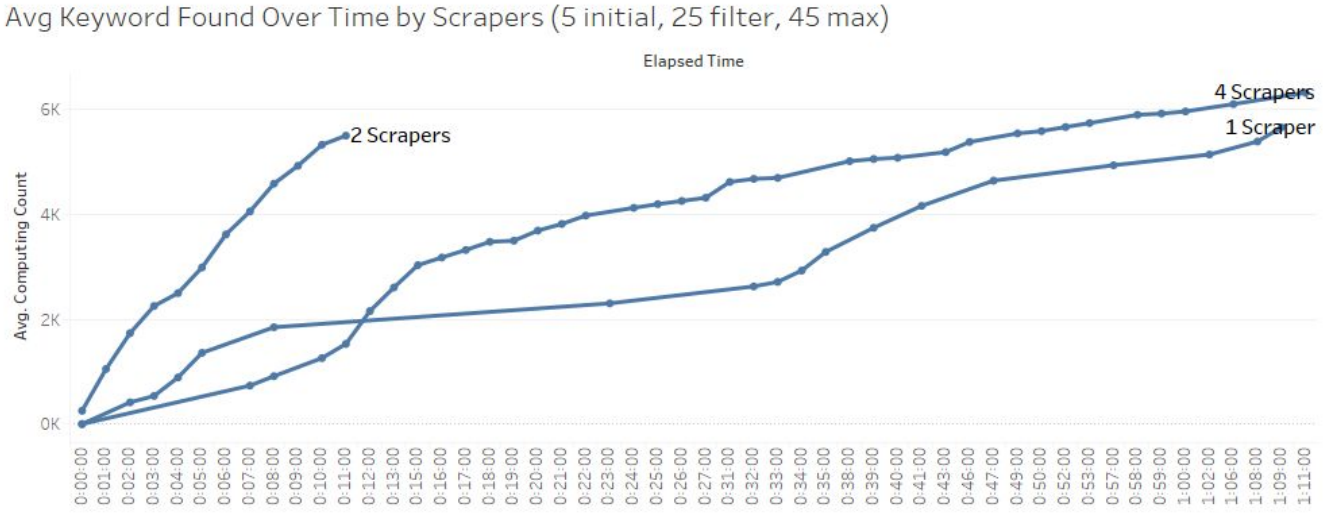




We see a positive correlation between higher depth settings/subprocesses and the number of webpages found containing a given keyword.

Note: This is only the results for the Computing keyword, but all showed the same trends.

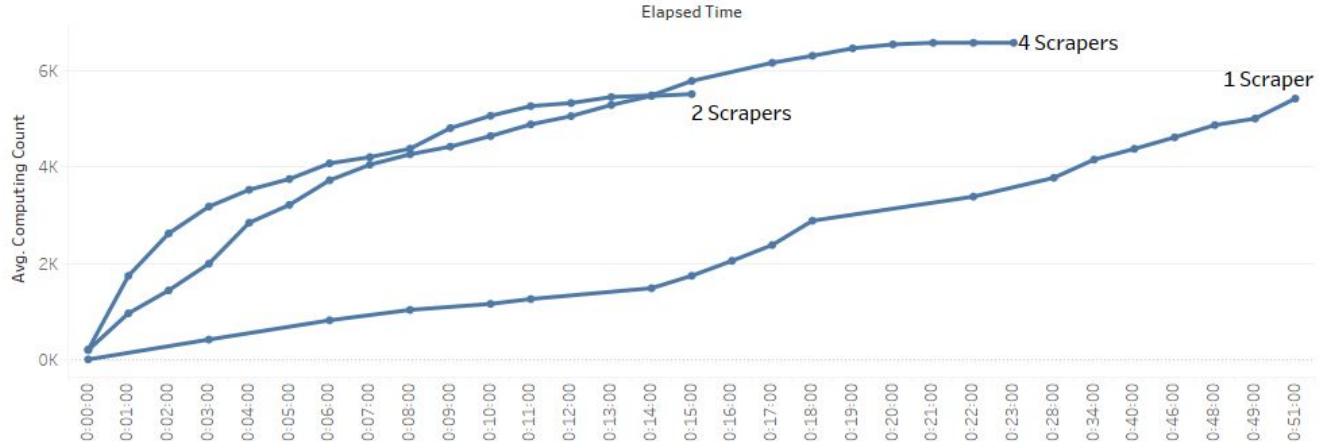




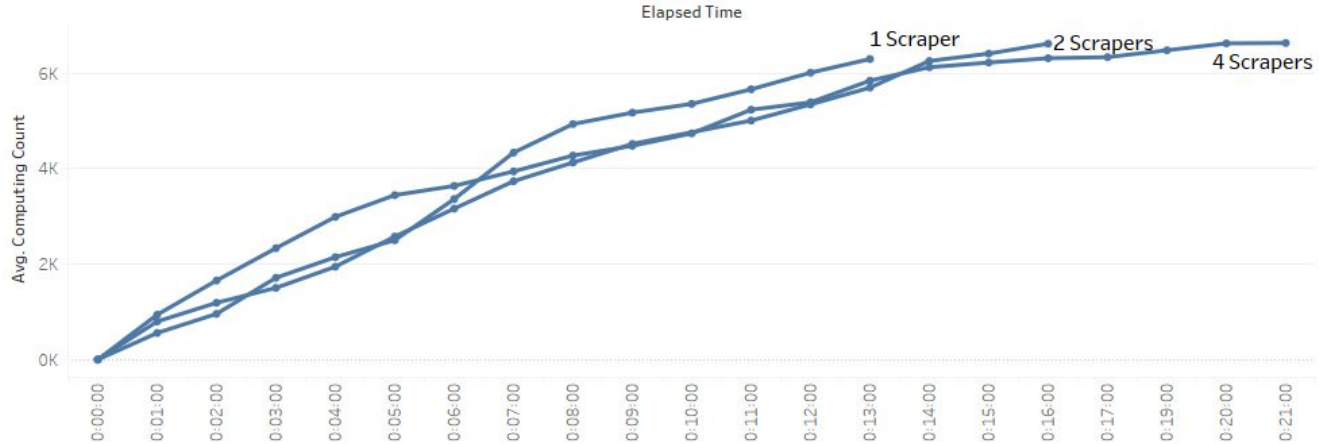
Observations:

- Very different results between filter depths which could indicate an outside factor affecting these quick runs
- The single scraper does consistently worst with a lower initial depth

Avg Keyword Found Over Time by Scrapers (10 initial, 25 filter, 45 max)



Avg Keyword Found Over Time by Scrapers (10 initial, 35 filter, 45 max)



Observations:

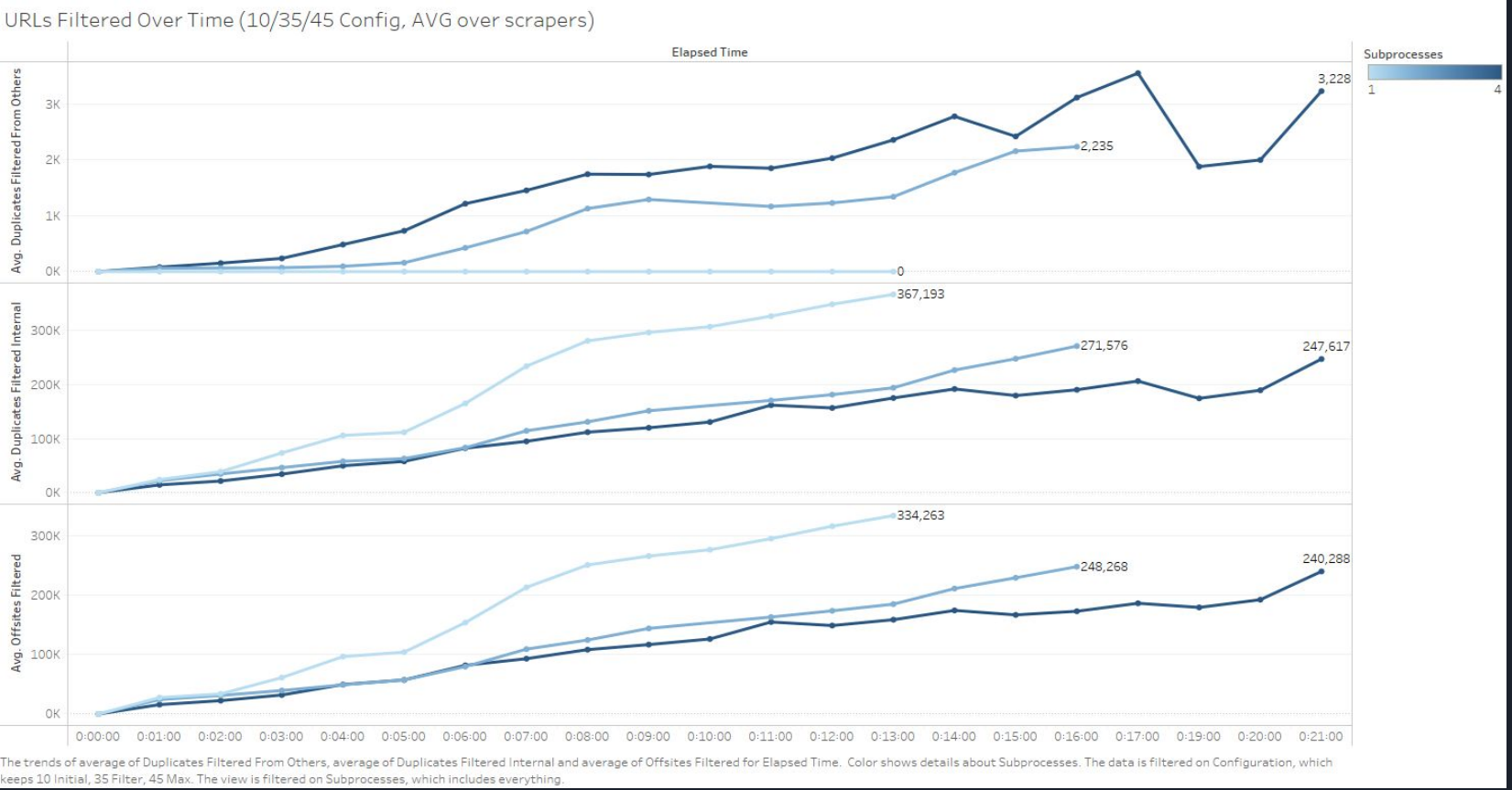
- The single scraper does worse with the lower filter depth
- The 10/35/45 configuration was consistently fastest for all configurations of scrapers

Crawling Results of Our Spider Configurations

To clarify the following graphs

Filtering behavior of spiders:

- **Filtered from others** = URLs filtered because another scraper has already processed it
- **Filtered internal** = URLs that the scraper itself has already encountered
- **Offsites filtered** = URLs that do not match the cc.gatech.edu/ criteria

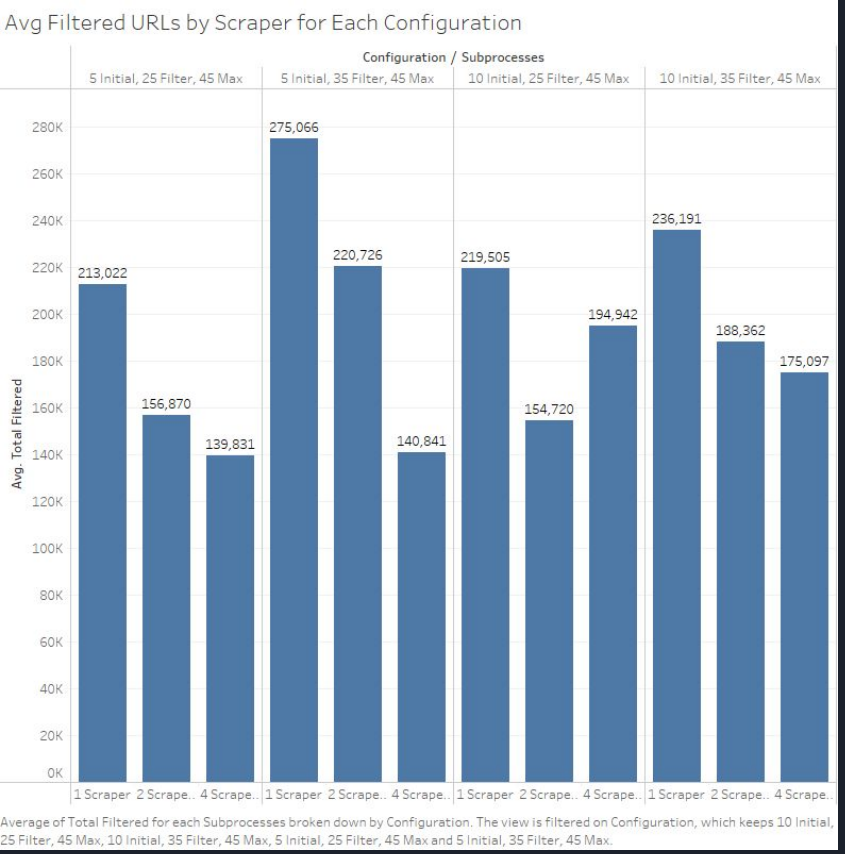


Trends:

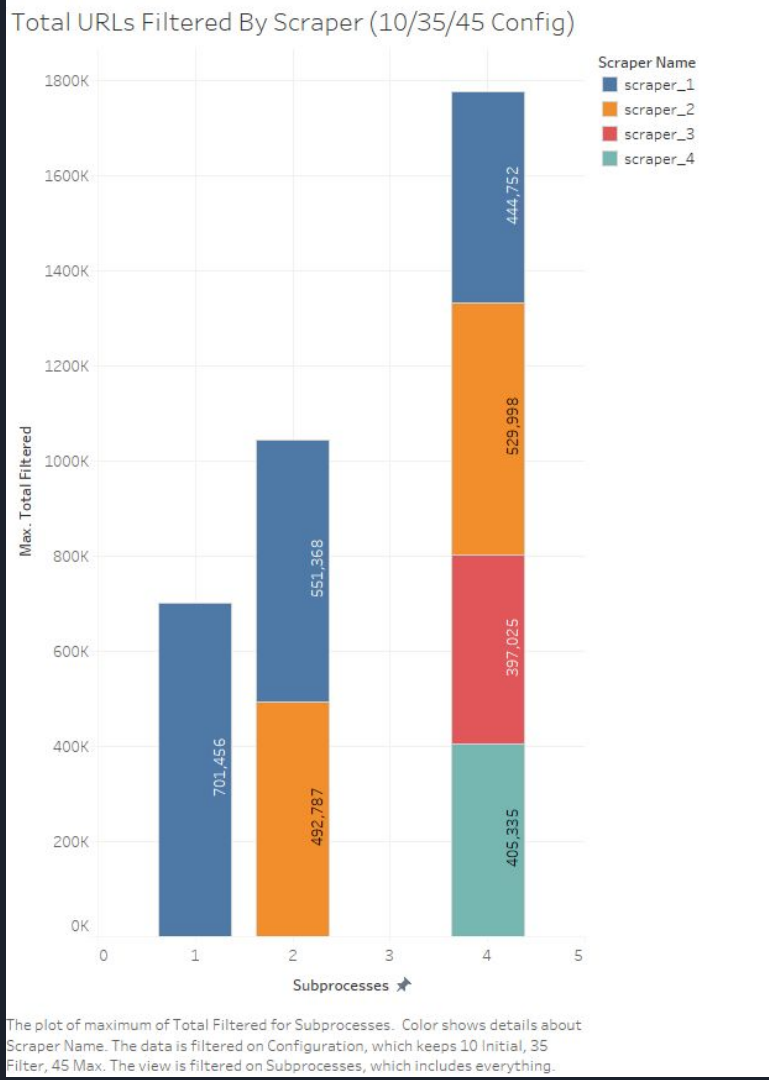
- One Scraper's number of filtered results quickly grows, showing a high number of URLs it filters
- Adding more scrapers results in filters less requests but takes longer

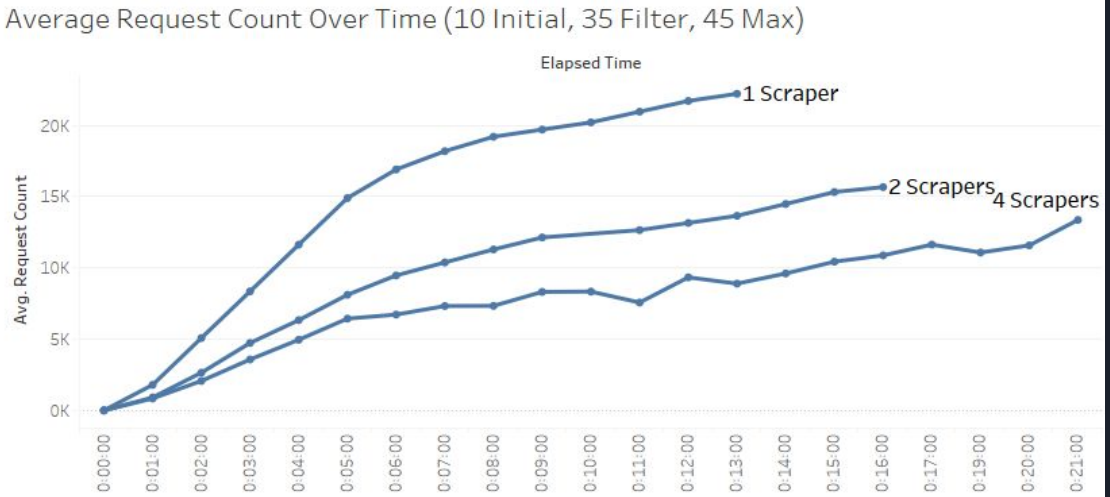
Possible Explanations:

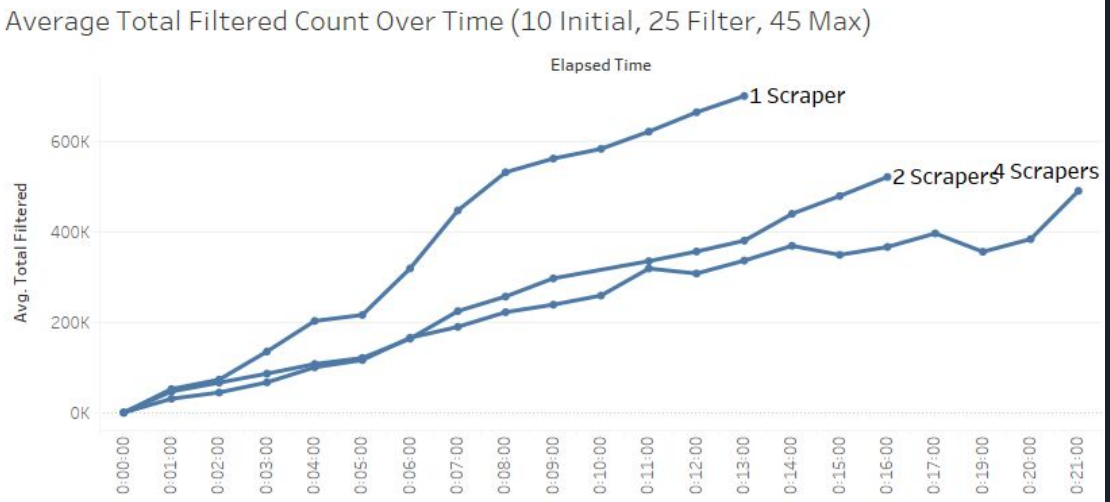
- Queries to check conflicts from other scrapers could be slowing down the scraper.



When we add more scrapers, each individual scraper filters less duplicates total, but adding all scrapers together shows that 4 scrapers is likely filtering the same URLs across multiple scrapers.

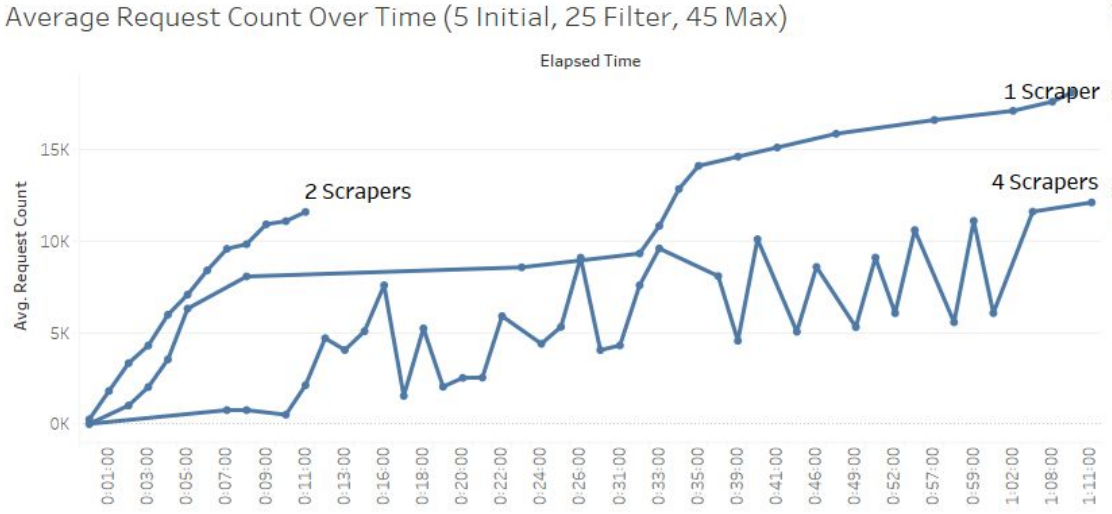


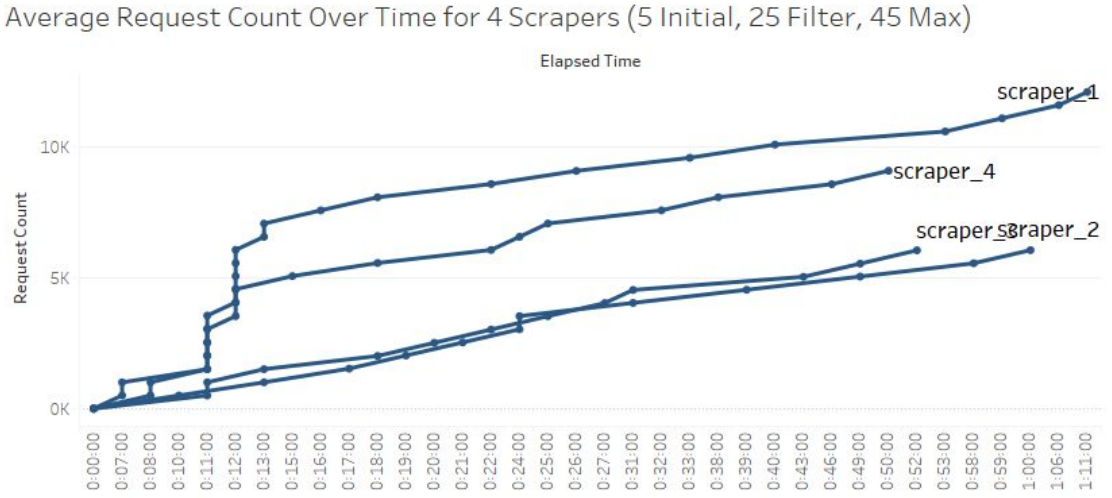




We see a mirroring of average requests made and average total filtered URLs indicicating:

- Filtering more URLs doesn't slow down requests.





Some of our data came out a bit sporadic, closer examination shows that some individual data points are missing, resulting in bad data on certain runs.

These runs and missing data points need to be investigated and conducted again.

Continued work & Aspirations

Setting up the crawling algorithm and getting the results into a useful and presentable form proved to be more difficult than originally anticipated.

Getting the crawlers to work in a truly distributed manner is not complete.

The list of keywords crawled thus far is rather short, so it is not yet useful as part of a search engine.

No real work has been done on the front end, as the data is not stored in a way that can be easily searched.

The dynamic recommendation system has so far eluded us.

Summary and Lessons Learned

Conclusions from the data analysis portion

- Efficiency of multiple crawlers
- Issues dealing with duplicate URLs

Lessons learned about the technologies

- Distributed systems
- Database configurations