Robust Cache System for Web Search Engine Yioop

A Project Report

Presented to Dr. Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfilment

of the Requirements for the Class

Fall 2022: CS 297

By

Rushikesh Padia

December 2022

ABSTRACT

Caches are the most effective mechanism utilized by web search engines to optimize the performance of search queries. Search engines employ caching at multiple levels to improve its performance, for example, caching posting list and/or caching result set. Caching query results reduces overhead of processing frequent queries and thus saves a lot of time and computing power. Employing a good caching mechanism can significantly improve the performance of a web search engine. Yioop is an open-source web search engine which utilizes result cache to optimize searches. The current implementation utilizes single dynamic cache based on Marker's algorithm. Our goal is to improve the performance of cache in Yioop. In this report, we focus on understanding Yioop Media Jobs and different caching mechanisms.

Keywords - Yioop, Web Search Engine, Result Caching, Result cache performance

TABLE OF CONTENTS

I. INTRODUCTION

The most critical objective of a web search engine is to produce search results within a fraction of a second. With the billions of pages currently present on the internet and the number still exponential growing, it is crucial to employ efficient caching mechanisms to search queries. The goal of caching is to reduce the response time by storing results of previously executed operations. Most of the modern search engines caches data at multiple levels. Caching posting lists, posting intersections, result sets are the most common approaches used for caching in web search engine. Result caching is the highest-level caching which caches query and the corresponding result. It reduces the overhead of processing the query. Thus, it is the most efficient in reducing the response time of a search query.

The main problem of result caching is to detect which query to be cached while which one to remove. These problems are called query admission problem and query eviction problem. Several approaches are available to determine admission and eviction of the query. For e.g., LRU admits every query while it evicts query which appeared least recently in the cache. Although LRU is one the most widely used caching algorithms, it does not capture global long-term trends. To capture these trends, static cache is often employed along with the dynamic cache [1]. Many recent caching algorithms modify this framework to achieve better results.

Yioop, an open-source web search engine uses result cache to improve the performance of search queries. It uses simple Marker's algorithm based dynamic cache for caching query results. The goal for this semester is to get familiarized with Yioop MediaJobs and evaluate three different caching algorithms.

We are now going to discuss the organization of the rest of the report. In Deliverable 1, a MediaJob was created to periodically refresh the cache. Deliverable 2 was the implementation of the Machine Learning based dynamic caching algorithm. The algorithm used features extracted from query log to take eviction decision. Deliverable 3 and 4 the caching algorithms that exploit temporal locality of the query topic and the query respectively to populate the cache.

II. DELIVERABLE 1 – CACHE REFRESH MEDIA JOB

The goal of this deliverable was to get familiar with Yioop code base, analyze important classes and methods related to Media Job. Along with this, implement a new Media Job to refresh cache with popular queries. Currently Yioop has only a dynamic cache which is populated with popular queries offline during the restart. To refresh the cache, a script has to be run manually. The top most popular queries have been empirically decided and are based on past data. The dynamic contents are updated in cache with execution of every search query.

Static-Dynamic Cache [1] is one of the most popular approaches and is widely used for benchmarking the performance of cache implementations. Almost every state-of-the-art caching mechanism [2, 3] employs some form of static and dynamic cache for storage. Static parts of these caches are refreshed offline periodically while dynamic parts are updated online. To periodically refresh the cache, a Media Job has been created. Media jobs are the processes that are scheduled to run periodically by Media Updater of Yioop. Analysis of classes and methods of Media Job and Media Updater is shown in Table 1 and Table 2. Having cache refreshed automatically and periodically ensured that the cache is up to date with popular queries.

CacheRefreshJob is a Media Job that is created to refresh the cache. It reads all the files present in the folder "/cache/cache_queries" and one by one executes each query on the search engine. Search engine caches these queries in the result cache according to the policy currently configured.

Thus, we analyzed the important classes and methods related to Media Jobs. Also, we have implemented a Media Job to refresh the cache which will be helpful in the next semester to implement any algorithm which require static cache to be populated periodically.

Classes	Description
MediaJob	It is a parent class for the Media Job to be executed. All jobs need to extend this class and override callback methods to execute their intended functionality.
MediaUpdater	It is a class reponsible for executing and managing lifecycle of Media Jobs

Table 1: Classes related to Media Job

Methods	Description
init()	Callback method called after constructor to initialize the job.
checkPrerequisites()	Method that should returns true if media job has to be executed, false otherwise.
nondistributedTasks()	Method is called when Media Job is running in non-distributed mode. Non-distributed version on task can be added in this method
prepareTasks()	Method is responsible for preparing data for the tasks executed on client machines in distributed mode. The method is executed on the NameServer only.
getTasks()	Client machines call this method on NameServer to get their data. This method uses the output of prepareTask() method and send client its indiavidual share of the data.
doTasks()	The method to process the data fetched from getTasks() method. This is called by MediaUpdater on each client after they complete getTasks().
putTasks()	Each client calls this method on the NameServer to store their processed results on the NameServer
finishTasks()	The method is called on the NameServer to complete the final computation after all tasks are completed

Table 2: Description of methods of class MediaJob

III. Deliverable 2 – Implement Machine Learning Dynamic Caching Algorithm

The goal of this deliverable is to implement a cache eviction policy based on the framework described in the paper [4]. The paper proposes a machine learning based cache admission and eviction algorithm. In this framework, a set of features are derived from query logs. The admission and eviction decision is based on a machine learning model trained from these features. The paper proposes machine learning models for both static and dynamic cache. Populating static cache has been modeled as an offline cache allocation problem while the latter has been modeled as an online eviction problem.

To derive features from the query log, AOL [5] query logs were used. Table 3 shows the extracted features. This subset of features was selected based on the availability of query logs and Yioop indexing data structures. 'IAT_NEXT' is the ground truth value that represents the next time the same query appears i.e., the number of queries between the next appearance of the query. The goal of the machine learning model was to predict the IAT_NEXT value as close to the actual IAT_NEXT value. The query having the highest IAT_NEXT value was evicted if the cache was full.

To create the features, queries were first cleaned. The query was converted to lowercase, stop words were removed, and words were stemmed. Features were then extracted from the clean queries. The key observation from the data was that the data was highly skewed. More than 80% of the data had an IAT_NEXT value of less than 1% of the total data (Figure 1). To mitigate this issue, IAT_NEXT values were binned using logarithmic binning. This reduced the effect of skewness in the data (Figure 2). A regression model was fitted and evaluated on this data. The model achieved a hit rate of 28.33% for cache size of 100 and 4200 queries. While its counterpart LRU achieved the hit rate of 52.60% and optimal offline algorithm - Belady's algorithm achieved a 57.27% hit rate.

As the hit rate is significantly low, we can conclude that the performance of this algorithm is not suitable for our implementation. Although the performance can be improved by adding more features but will require lot of effort in fine tuning and feature engineering. Hence, this algorithm will not be considered for the further development.

Feature	Description
QUERY_HOUR	Hour of the day, query was fired
LAST_MIN_FREQ	Frequency of query in last minute
LAST_HOUR_FREQ	Frequency of query in last hour
LAST_DAY_FREQ	Frequency of query in last day
PAST_FREQ	Total frequency of the query
IS_TIME_COMPAT	Whether query is time compatible
QUERY_LENGTH	Number of words in the query

Table 3: Features extracted from query logs

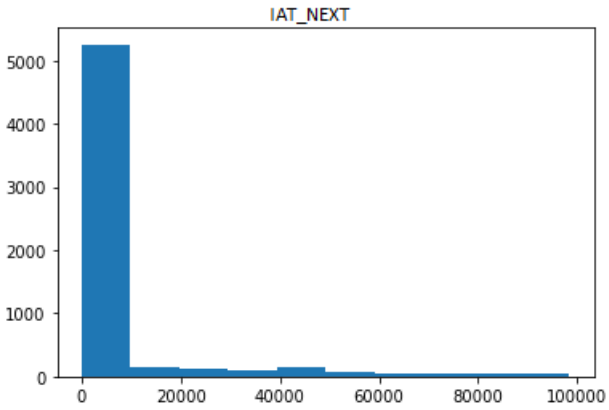

Figure 1: Distribution of IAT_NEXT values

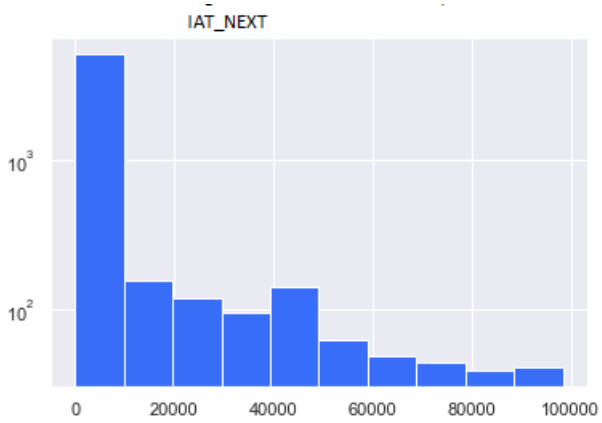

Figure 2: Distribution of IAT_NEXT values after log binning

IV. Deliverable 3 – Implement Static-Topic-Dynamic Cache Algorithm

The goal of this deliverable is to implement Static-Topic-Dynamic Cache (STDC) [2]. The given approach refines the traditional Static-Dynamic Cache (SDC) to capture the temporal locality of the topic in query streams. In SDC, static cache is populated periodically with the popular queries with assumption being popular queries in past remains popular in future. The dynamic part of the cache using LRU strategy captures the burst of short-term popular queries. The drawback of this approach is that it fails to capture queries with long-term popularity. There are certain queries which are neither too popular to be cached in static cache nor too frequent in short-term to be captured by dynamic cache. For instance, queries for certain topics like weather can be popular in the morning and late in the evening but are not popular in other parts of the day. To capture this temporal locality of the topic, topic cache is introduced along with the static and the dynamic part.

Assigning topic to a query is a challenging part as queries are generally very short. To classify queries correctly, queries are enriched with the search results. For this deliverable, queries are not enriched with search results, instead a news dataset is used to train the Latent Dirichlet Allocation (LDA) model. LDA model is an unsupervised statistical method to discover the hidden topic in the text. Given, k different topic, it assigns individual scores between [0, 1] to each topic. The topic is assigned to the query if the topic has maximum score and is greater than minimum threshold.

The topical cache has individual instances of cache of each topic. These cache instances can be LRU or SDC. The size of each instance can be configured to be fixed or variable depending upon the popularity of the topic. For this deliverable, we have evaluated STDC with both fixed sized (STDC) and variable size

(STDC) topic cache and compared its performance with traditional SDC, LRU and optimal offline algorithm for 10K queries. The results are shown in Table 4.

The performance of this algorithm is significant improvement over the LRU baseline algorithm. The algorithm improved the hit rate by 2% resulting in gap reduction of 12% with optimal Belady. Also, compared to SDC, the performance is not significantly lower and can improved the better data set and fine tuning the number of topic. Thus, we will further explore this algorithm in the next semester.

Cache Size	Configuration	Hit Rate
100	LRU	42.33
100	Belady	49.08
100	SDC(30-70)	43.35
100	STDC(30-50-20)	43.16
100	STDC-V(30-50-20)	43.10
200	LRU	44.49
200	Belady	49.81
200	SDC(15-85)	45.16
200	SDTC(15-50-35)	45.08
200	STDC-V(15-50-35)	45.01

Table 4: Result of STDC algorithm

V. Deliverable 4 – Implement Static-Semi-Static-Dynamic Cache

The goal of this deliverable is to implement Static-Semi-Static-Dynamic Cache (SSDC) [3]. The previous paper STDC used dynamic cache to capture the temporal locality in cache while this approach uses semi-static cache to capture the temporal locality. The key observation the paper relied on was that daytime queries have higher submission rate during day while nighttime queries have higher submission rate during night.

To capture the temporal query frequency variations, new attributes called normalized temporal query frequency (NT-Freq) attributes were added to the queries. NT-Freq values are hourly submission rate of the query i.e., total frequency of the query during a period divided by number of hours in the period. NT-Freq attribute for daytime hours is calculated as total frequency of query between 07:00 to 19:00 divided by 12. Similarly, nighttime NT-Freq is calculated as frequency during other time of the day divided by 12. All time NT-Freq is calculated as total frequency in period of 24 hours divided by 24.

SSDC utilizes semi-static cache along with traditional SD cache. The semi-static cache contains daytime popular and nighttime popular queries corresponding to the time of the day. Daytime and nighttime popular queries are calculated using training data. For this deliverable, training was performed using 1.5 million queries and is tested on other 1.5 million queries with cache size of 300. The cache size has been increased as SSDC approach is designed for large caches only. The result of this model is shown in Table 5.

The hit rate of the algorithm is 1.2% lower than the SDC algorithm. It less than the performance of STDC and has additional constraint of having large static cache. The algorithm is easy implement and has a considerable performance but has a very little scope for fine tuning as most of the parameters are fixed. Hence, it might not be considered in the next semester.

Cache Size	Configuration	Hit Rate
300	SDC(80-20)	56.65
300	SSDC(10-70-20)	55.76
300	SSDC(20-60-20)	55.95
300	SSDC(40-40-20)	56.21
300	SSDC(60-20-20)	55.94
300	SSDC(70-10-20)	56.56
300	SSDC(0-80-20)	55.46

Table 5: Results of SSDC algorithm

VI. CONCLUSION

In this semester, we got familiarized with the current caching mechanism and the code base of Yioop. Additional components like MediaUpdater and MediaJobs were also analyzed and were used to implement important functionality for future use in SD cache. Along with this, three different caching algorithms were implemented and evaluated against real life AOL query log data. SDC, STDC and SSDC did show promising results and were much more efficient than the traditional LRU algorithm. Although the efficiency of the STDC was limited by lack of dataset having enriched queries from search results, it still showed promising results. Fine tuning STDC may increase its efficiency. Hence, we will explore it further in the next semester.

In the next semester, we will first fine tune the STDC model to improve its efficiency. Integrate newly created cache implementation is Yioop. As current implementation is done in Python, it has to be migrated to PHP as Yioop is written in PHP. Currently, Yioop does not have module to track performance of cache. A new module to evaluate the performance of cache will be added. The performance of cache will finally be tested using a newly integrated module in Yioop.

REFERENCES

[1] T. Fagni, R. Perego, F. Silvestri, and S. Orlando, “Boosting the Performance of Web Search Engines: Caching and Prefetching Query Results by Exploiting Historical Usage Data,” *ACM Trans. Inf. Syst.*, vol. 24, no. 1, pp. 51–78, Jan. 2006, doi: 10.1145/1125857.1125859.

[2] I. Mele, N. Tonellotto, O. Frieder, and R. Perego, "Topical result caching in web search engines", *Information Processing & Management*, vol. 57, no. 3, p. 102193, 2020, doi: https://doi.org/10.1016/j.ipm.2019.102193.

[3] T. Kucukyilmaz, “Exploiting temporal changes in query submission behavior for improving the search engine result cache performance,” *Information Processing & Management*, vol. 58, no. 3, p. 102533, 2021, doi: https://doi.org/10.1016/j.ipm.2021.102533.

[4] T. Kucukyilmaz, B. B. Cambazoglu, C. Aykanat, and R. Baeza-Yates, "A machine learning approach for result caching in web search engines," *Information Processing & Management*, vol. 53, no. 4, pp. 834-850, 2017, doi: https://doi.org/10.1016/j.ipm.2017.02.006.

[5] G. Pass, A. Chowdhury, and C. Torgeson, “A Picture of Search,” in *Proceedings of the 1st International Conference on Scalable Information Systems*, 2006, pp. 1-es. doi: 10.1145/1146847.1146848.