Robust Cache System for Web Search Engine Yioop

Presented by -
Rushikesh Padia
Department of Computer Science

Committee:
Dr. Chris Pollett (Advisor)
Dr. Ben Reed
Dr. Robert Chun
Every millisecond counts!
Agenda

- Introduction
- Preliminary Work
- Implementation Details
- Results
- Conclusion
- Future work
Introduction

- Yioop is an open source web search engine
- It uses result cache to improve response time
- Current implementation uses dynamic cache based on Marker Algorithm
- A dynamic cache based on Marker or LRU algorithm captures short-term trends
- The goal of the project is to explore different caching strategies and implement them in Yioop
Yioop Search Engine Architecture

- Yioop search engine consists of three main components: Crawler, Indexer, and Query Processor.

  - **Crawler**
    - Responsible for discovering and gathering information from web pages.
    - **QueueServer** process queues URLs to be fetched.
    - **Fetcher** process fetches webpages from the internet.

  - **Indexer**
    - Processes fetched pages to extract textual content.
    - Builds an inverted index, which aids in processing queries.
    - Generates textual summaries of the extracted information.

Figure 1: Yioop Architecture
Yioop Search Engine Architecture

- **Query Processor**
  - Evaluates user queries for search results.
  - Cleaning and preprocessing the query through techniques like case folding, stemming, and stopword elimination.
  - Utilizes a result cache to check if the results for a query are already computed.
  - If results are available in the cache, they are returned without further processing, saving time and resources.
  - If not available, the processor retrieves posting lists for each term in the query from the inverted index.
  - Utilizes a ranking algorithm to determine the most relevant documents.
To implement new cache system in Yioop, following algorithms were evaluated

- Static-Dynamic Cache
- Machine Learning Static-Dynamic Cache
- Static-Semistatic-Dynamic Cache
- Static-Topic-Dynamic Cache

Experiments were performed to evaluate each of these algorithms

Following slides give more information about each of these algorithms
**Static-Dynamic Cache**

- Dynamic cache adapts well to the short-term trend in queries
- It does not adapt well in presence of both short-term and long-term trends in queries
- Static-Dynamic cache divides cache into two segments
  - Static segment adapts to the long-term trends in the queries
  - Dynamic segment adapts to the short-term trends in the queries
- Static Cache - Modeled as offline cache allocation problem
- Dynamic Cache - Modeled as online admission-eviction problem
- Query is first checked in static cache and if not present, it is checked against the dynamic cache

Figure 2: Static-Dynamic Cache
Dynamic cache implemented using classical machine learning models.

- Goal of machine learning models is to accurately predict the next appearance of the query i.e. “IAT_NEXT”.
- Features extracted from the query.
- Models cache admission problem as a classification problem to classify whether query should be admitted or not.
- Models cache eviction problem as a regression problem to predict the “IAT_NEXT” value. It removes value having highest “IAT_NEXT”.
Dataset

- AOL Query logs contains 3 months of query logs generated in year 2006 on the AOL search engine.
- It contains total 36 million queries
- Dataset contains raw queries, anonymized user ids, timestamp, url clicked by the user, and the rank of the item.

<table>
<thead>
<tr>
<th>AnonID</th>
<th>Query</th>
<th>QueryTime</th>
<th>ItemRank</th>
<th>ClickURL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mapquest</td>
<td>2006-03-01 15:18:21</td>
<td>1.0</td>
<td><a href="http://www.mapquest.com">http://www.mapquest.com</a></td>
</tr>
<tr>
<td>1</td>
<td>cajun candle</td>
<td>2006-03-01 13:20:18</td>
<td>1.0</td>
<td><a href="http://www.cajuncandles.com">http://www.cajuncandles.com</a></td>
</tr>
<tr>
<td>3</td>
<td>muic.com</td>
<td>2006-03-01 22:42:05</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>i need a company name</td>
<td>2006-03-02 12:32:59</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Table 1: AOL Query log Dataset
Query Features

Based on the available data, a subset of features from original paper were selected.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_HOUR</td>
<td>Hour of the day, the query was fired</td>
</tr>
<tr>
<td>LAST_MIN_FREQ</td>
<td>Frequency of query in last minute</td>
</tr>
<tr>
<td>LAST_HOUR_FREQ</td>
<td>Frequency of query in last hour</td>
</tr>
<tr>
<td>LAST_DAY_FREQ</td>
<td>Frequency of query in last day</td>
</tr>
<tr>
<td>PAST_FREQ</td>
<td>Total frequency of the query</td>
</tr>
<tr>
<td>IS_TIME_COMPAT</td>
<td>Whether query is time compatible</td>
</tr>
<tr>
<td>QUERY_LENGTH</td>
<td>Number of words in the query</td>
</tr>
</tbody>
</table>

Table 2: Query features
Experiment and results

- Regression model was fitted on the dataset of 1M queries
- Training data was highly skewed, over 80% data had IAT_NEXT value less 1% of total data
- Log binning was applied to remove the skewness of the data
- The model achieved the hit rate of 28.33% for cache size of 100 frames and 4K total number of queries
- LRU and Optimal offline algorithm achieved 52.6% and 57.27% for the same data.
- Will requires lot of tuning and feature engineering to achieve acceptable results
Static-Semistatic-Dynamic Cache

- Adds Semistatic layer to Static-Dynamic cache framework
- Based on observations, day time popular queries are different than night time popular queries.
- Categorizes queries into day-time popular, night-time popular, and all-time popular queries
- Static segment contains all-time popular queries
- Semi-static segment toggles between day-time and night-time popular queries
- Dynamic cache implemented with LRU caching algorithm
- Query is checked in each cache in order - static cache, semi-static cache, and dynamic cache

Figure 5: Static-Semistatic-Dynamic cache
Experiment and Results

- Query categorization was done using AOL query logs.
- Queries which appeared more than 80% of the time in day-time were categorized as day-time and similarly for night-time queries.
- If night day-time or night-time, it was termed all-time query.
- Training was performed using 1.5M queries.
- Results were evaluated on other set of 1.5M queries with cache size of 300 frames.
Experiment and Results

- The algorithm has acceptable performance. Only 1.2% lower than Static-Dynamic cache.
- There was no scope for improvement in this approach.
- Requires large cache space and as day-time night-time both requires persistence, it was not selected.

Table 3: Hit rate in percentages (%) for 1.5M queries and cache size of 300 entries

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Configuration</th>
<th>Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>SDC(80-20)</td>
<td>56.65%</td>
</tr>
<tr>
<td>300</td>
<td>SSDC(10-70-20)</td>
<td>55.76%</td>
</tr>
<tr>
<td>300</td>
<td>SSDC(20-60-20)</td>
<td>55.95%</td>
</tr>
<tr>
<td>300</td>
<td>SSDC(40-40-20)</td>
<td>56.21%</td>
</tr>
<tr>
<td>300</td>
<td>SSDC(60-20-20)</td>
<td>55.94%</td>
</tr>
<tr>
<td>300</td>
<td>SSDC(70-10-20)</td>
<td>56.56%</td>
</tr>
<tr>
<td>300</td>
<td>SSDC(0-80-20)</td>
<td>55.46%</td>
</tr>
</tbody>
</table>
Different topics for e.g. weather, tv shows are accessed more frequently during different time of days and have different access patterns.

Static-Dynamic cache does not adapt well to these type of access patterns.

Static-Topic-Dynamic cache adds Topical layer over Static-Dynamic cache to capture such trends.

Topic is assigned to queries using some topic model.

Each topic has own instance of cache managed by a certain policy.

Query is checked in each cache in order - static cache, topical cache, and dynamic cache.
Topic Modeling

- Topic Modeling is a popular technique in NLP to extract topics from text
- Extracts latent topics without unsupervised learning algorithms
- Popular algorithms
  - Latent Semantic Analysis (LSA)
  - Latent Dirichlet Allocation (LDA)
  - k-means Algorithm
Experiments and Results

- Experiment was performed on 10K queries and cache size of 100 and 200 frames
- For topic modeling LDA model was used
- LDA model was trained using 1.2M news headline dataset
- Each topical cache instance was governed using LRU cache
- Dynamic cache was implemented using LRU cache
Experiment and Results

- Static-Dynamic cache performance was close to Static-Dynamic cache
- There was scope for improvement by training the model using actual search engine text data
- Queries can be enriched using user’s clicked URL data

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Configuration</th>
<th>Hit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>LRU</td>
<td>42.33%</td>
</tr>
<tr>
<td>100</td>
<td>Belady</td>
<td>49.08%</td>
</tr>
<tr>
<td>100</td>
<td>SDC(30-70)</td>
<td>43.35%</td>
</tr>
<tr>
<td>100</td>
<td>STDC(30-50-20)</td>
<td>43.16%</td>
</tr>
<tr>
<td>100</td>
<td>STDC-V(30-50-20)</td>
<td>43.10%</td>
</tr>
<tr>
<td>200</td>
<td>LRU</td>
<td>44.49%</td>
</tr>
<tr>
<td>200</td>
<td>Belady</td>
<td>49.81%</td>
</tr>
<tr>
<td>200</td>
<td>SDC(15-85)</td>
<td>45.16%</td>
</tr>
<tr>
<td>200</td>
<td>SDTC(15-50-35)</td>
<td>45.08%</td>
</tr>
<tr>
<td>200</td>
<td>STDC-V(15-50-35)</td>
<td>45.01%</td>
</tr>
</tbody>
</table>

Table 4: Hit rate in percentages (%) for 10K queries and cache size of 100 and 200 entries
Implementation of New Cache in Yioop

- Choice of caching algorithm depends on the use case
- Yioop is used for a variety of purposes e.g. general purpose crawling, crawling set of web pages or crawling user’s website
- Old caching system in Yioop was tightly coupled with Marker Algorithm
- The new system gives Yioop flexibility to switch between different cache types
- Following cache type are added in Yioop [Demo 1]
  - Least Recently Used
  - Static-Dynamic Cache
  - Static-Topic-Dynamic Cache
Cache System Design

- Object Oriented Design of new cache system
- Flexible design to switch internal implementation of Static-Dynamic and Static-Dynamic cache

Figure 7: Class diagram of Cache System in Yioop
Implementation Static-Dynamic Cache

- “StaticDynamicCache” class is implemented in Yioop
- Static cache is populated using most frequent queries in search engine logs
- Dynamic Cache segment uses instance of LRU cache
Implementation of Topical Cache

- “TopicalCache” class is implemented in Yioop
- It uses k-means clustering topic model to extract topic from cache
- k-means algorithm is an unsupervised machine learning model used for clustering
- k-means algorithm can also be used as topic model where centroids of k-means acts as latent topics
- k-means algorithm can be trained to classify text into k number of topics
- Each of these topic have corresponding cache governed by “LRUCache”
Dataset

- To train k-means algorithm a text data was created using Yioop’s indexer.
- As queries are usually 2-3 words long, it needs to be enriched with additional contextual information.
- Contextual information was added using user’s clicked URL and webpages extracted from Yioop’s crawl data.
- Yioop summarizer’s text was added to the query to enrich the queries for training.
- As Yioop does not get lot of traffic, instead of Yioop’s query logs, AOL query logs were used.
- Total of 10K clicked URL queries were used for dataset creation.
- Thus a dataset was created using combination of Yioop’s crawl data and AOL query logs.
Word Embeddings

- Machine learning algorithms require text to be represented as vectors.
- In Yioop, CountVectorizer is implemented to convert terms into vectors.
- CountVectorizer first creates vocabulary from text corpus.
- Assigns each word a unique index in a vector.
- Increments count of index of each term in the text.
- To reduce the cost of memory and CPU, all vectors are implemented as sparse vectors in Yioop.
Training k-means algorithm

- To train k-means algorithm, each document in the training dataset was converted into the vector
- Each vector was appended to form a document-term matrix
- k=10 centroids were selected to train the algorithm
- To avoid training and creating vocabulary, serialization and deserialization capability is added to both KMeansClustering model as well as CountVectorizer
Results of k-means algorithm

Figure 8: Restaurants

Figure 9: Holidays

Figure 10: Technology
Implementation of Static-Topic-Dynamic Cache

- “StaticTopicDynamicCache” is implemented in Yioop
- Uses “TopicalCache” for topical segment of its cache
- Static and dynamic segments are implemented with “StaticCache” and “LRUCache”
- “StaticTopicDynamicCache” first checks whether cache is present in static cache.
- If it is not present it checks in topical Cache.
- If it is not present it checks with dynamic cache.
- If it is found in any segment, result hit is returned
To evaluate the performance of cache, “CacheMetricWrapper” class is implemented.

It delegates get and put calls to the internal cache implementation and tracks performance based on the output.

Currently it tracks numbers of hits and misses and hit rate.

The performance was evaluated on 5k, 10k, and 20k queries.

Cache sizes were varied between 200-500 entries.
<table>
<thead>
<tr>
<th>Cache Size</th>
<th>LRU</th>
<th>SDC (20-80)</th>
<th>SDC (30-70)</th>
<th>SDC (40-60)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>69.8</td>
<td>70.6</td>
<td>71.04</td>
<td>71.4</td>
</tr>
<tr>
<td>300</td>
<td>69.96</td>
<td>71.12</td>
<td>71.72</td>
<td>72.34</td>
</tr>
<tr>
<td>400</td>
<td>69.98</td>
<td>71.58</td>
<td>72.4</td>
<td>73.2</td>
</tr>
<tr>
<td>500</td>
<td>70</td>
<td>72.02</td>
<td>73.02</td>
<td>74.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>STDC (20-40-40)</th>
<th>STDC (20-50-30)</th>
<th>STDC (20-60-20)</th>
<th>STDC (20-70-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>70.22</td>
<td>69.96</td>
<td>69.68</td>
<td>69.56</td>
</tr>
<tr>
<td>300</td>
<td>70.98</td>
<td>70.74</td>
<td>70.54</td>
<td>70.36</td>
</tr>
<tr>
<td>400</td>
<td>71.46</td>
<td>71.38</td>
<td>71.12</td>
<td>70.9</td>
</tr>
<tr>
<td>500</td>
<td>71.92</td>
<td>71.88</td>
<td>71.72</td>
<td>71.56</td>
</tr>
</tbody>
</table>

Table 5: Hit rate in percentages (%) for 5K queries and cache size between 200-500 entries
## Results

Table 6: Hit rate in percentages (%) for 10K queries and cache size between 200-500 entries

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>LRU</th>
<th>SDC (20-80)</th>
<th>SDC (30-70)</th>
<th>SDC (40-60)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>61.38</td>
<td>61.96</td>
<td>62.21</td>
<td>62.31</td>
</tr>
<tr>
<td>300</td>
<td>61.55</td>
<td>62.3</td>
<td>62.6</td>
<td>62.91</td>
</tr>
<tr>
<td>400</td>
<td>61.6</td>
<td>62.54</td>
<td>62.98</td>
<td>63.41</td>
</tr>
<tr>
<td>500</td>
<td>61.65</td>
<td>62.79</td>
<td>63.36</td>
<td>63.87</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>STDC (20-40-40)</th>
<th>STDC (20-50-30)</th>
<th>STDC (20-60-20)</th>
<th>STDC (20-70-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>61.54</td>
<td>61.26</td>
<td>60.95</td>
<td>60.73</td>
</tr>
<tr>
<td>300</td>
<td>62.1</td>
<td>61.89</td>
<td>61.53</td>
<td>61.33</td>
</tr>
<tr>
<td>400</td>
<td>62.42</td>
<td>62.29</td>
<td>61.99</td>
<td>61.69</td>
</tr>
<tr>
<td>500</td>
<td>62.7</td>
<td>62.66</td>
<td>62.44</td>
<td>62.2</td>
</tr>
</tbody>
</table>
### Results

Table 7: Hit rate in percentages (%) for 20K queries and cache size between 200-500 entries

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>LRU</th>
<th>SDC (20-80)</th>
<th>SDC (30-70)</th>
<th>SDC (40-60)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>60.19</td>
<td>60.62</td>
<td>60.66</td>
<td>60.75</td>
</tr>
<tr>
<td>300</td>
<td>60.37</td>
<td>60.87</td>
<td>61.11</td>
<td>61.25</td>
</tr>
<tr>
<td>400</td>
<td>60.52</td>
<td>61.14</td>
<td>61.38</td>
<td>61.6</td>
</tr>
<tr>
<td>500</td>
<td>60.97</td>
<td>62.15</td>
<td>62.75</td>
<td>63.36</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>STDC (20-40-40)</th>
<th>STDC (20-50-30)</th>
<th>STDC (20-60-20)</th>
<th>STDC (20-70-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>60.11</td>
<td>59.8</td>
<td>59.41</td>
<td>59.04</td>
</tr>
<tr>
<td>300</td>
<td>60.54</td>
<td>60.28</td>
<td>59.92</td>
<td>59.63</td>
</tr>
<tr>
<td>400</td>
<td>60.91</td>
<td>60.73</td>
<td>60.42</td>
<td>60.13</td>
</tr>
<tr>
<td>500</td>
<td>61.95</td>
<td>61.86</td>
<td>61.76</td>
<td>61.66</td>
</tr>
</tbody>
</table>
Figure 12: Hit rate in percentages (%) for 5K queries and cache size between 200-500 entries
Figure 13: Hit rate in percentages (%) for 10K queries and cache size between 200-500 entries
Results

Figure 14: Hit rate in percentages (%) for 20K queries and cache size between 200-500 entries
Conclusion

- Static-Topic-Dynamic Cache performs consistently better than Marker cache and LRU cache
- Performance of Static-Dynamic cache is slightly better than Static-Topic-Dynamic cache
- Static-Dynamic cache shows improvement of 2.3% over Marker cache
- Static-Topic-Dynamic shows improvement of 1.17% over Marker cache
- Users have ability to switch between the implementations based on the requirement
Future Work

- Currently topic model is trained on AOL query logs, it can be changed to use Yioop’s impression data
- Instead of k-means algorithm, other topic models like LDA can be also be implemented
- Topic models also exist which are particularly designed to extract topic from small texts
- Other vectorization method like tf-idf can be used instead of count vectorizer
- Customization feature can also be added to Yioop to customize individual instances of Static-Dynamic and Static-Topic-Dynamic cache
References


Thank you!

Questions?