CIS5550 Final Project Report - Penngle

Zhengyi Xiao, Jiayun He, Yuting Zheng, Jinyu Tan

May 3, 2023

Enhanced Features and Challenges

In this project, we have collected data from 1 million web pages and processed approximately 50,000 of them. We started at the local test environment used for this project from a 16GB Linux-based system, while the cloud environment is deployed on a 128GB AWS virtual server (r5.4xlarge).

Crawler

Our web crawler focuses on extracting data from BBC news and wiki pages and generates tables of URLs and plain text web content for use in our search engine. However, we have encountered several challenges during the crawling process, such as broken or unresponsive pages, limitations on available memory, and issues with performance. In order to address these challenges, we have implemented several enhancements to our crawler. And finally, we crawl one million documents in this stage.

- We fine-tuned the find_urls function and improved our normalization techniques to make them more robust.
- We set a timeout to discard low-responsive pages and prevent the crawler from wasting time on unproductive targets.
- To make the crawler restartable and easy to monitor, we saved the frontier at the end of each iteration and constantly monitored its progress.
- We kept a record of the crawled URLs to avoid infinite redirects and improve performance while minimizing additional key-value store read/write operations.
- To accelerate the crawler, we made as few HTTP requests as possible and optimized our network traffic.
- Finally, we manually monitored the crawler and deployed it to Amazon EC2 for better scalability and reliability.

Index

To find the similarity between a given query and documents, we use Equation 1 to calculate the similarity score between a document d_j and a query q:

$$sim(d_j, q) = \cos\theta = \frac{\vec{d_j} \cdot \vec{q}}{\mid \vec{d_j} \mid \times \mid \vec{q} \mid} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}} \tag{1}$$

where $w_{i,j} = \text{tf}(i,j) \times \text{idf}(i)$ and t is the total number of vocabularies.

Our first major challenge was dealing with the sheer size of the document collection: with one million documents, computing the similarity scores on-the-fly would be infeasible. To overcome this, we adopted a strategy of precomputing as many weights as possible and storing them in persistence tables. To optimize the process further, we implemented the following strategies:

- We computed the normalizing constants $\mid \vec{d_j} \mid$ for all documents beforehand and persisted all the middle stages, including $\text{tf}(i,j)$ and $\text{idf}(i)$, so that we can quickly lookup for a search query.

- We left computing $\mid \vec{q} \mid$ and $\sum_{i=1}^{t} w_{i,j} \times w_{i,q}$ in (1) on the fly, but both of them require iterating over the entire set of vocabulary, and it is not necessary. A simple observation is that for word w_i that is not presented in the query, $w_{i,q} = 0$. Therefore, the new normalizing constant is $\mid \vec{q} \mid = \sqrt{\sum_{i=1}^{t'} w_{i,q}^2}$, where t' is the number of distinct words in the query and the new numerator is $\sum_{i=1}^{t'} w_{i,j} \times w_{i,q}$. In our first design, this reduces from $t = 1,980,395$ to the length of the query.

- Without any preprocessing techniques over the corpus, the size of the vocabulary plus one million pages makes this job undoable on the majority of EC2 instances. Our estimation is that this will require at least 8TB of memory in total over different machines. Therefore, we develop the following strategy for preprocessing:

  - We defined a set of regular English words from [1], which contains over 466,550 common English words.
  - We used Porter stemmer [4] steamed the vocabulary and reduced the vocabulary size to 322,470.
  - We then removed all non-English chars and all words that are longer than 17 chars and smaller than 2 chars from the list. Our finalized word list is of size 299,937. And we only count those words in documents and ignore the rest, such as non-English words and numbers. We believe numbers need a more dedicated approach to deal with.

- To let the tables spread out more evenly among KVSs, we initially created a table with the row entry to be the hash of URL and columns to be vocabularies, however even after we reduced the number vocabulary, we found it was still very time-consuming[1] to compute the normalizing constants $\mid \vec{d_j} \mid$ over one million pages. It is also impossible to iterate over one million pages to compute the similarity on the fly. Therefore, we propose the following heuristic greedy procedure

```
function QUERY(q, threshold)
    result ← ∅
    rankedDocuments ← ranked list of documents based on PageRank
    for document in rankedDocuments do
        if sim(document, q) > threshold then
            result.append(document)
        end if
        if len(result) > 100 then
            return result
        end if
    end for
```

[1] We guess it is one-month computation on r5.4xlarge.

end function

The query function is designed to take a query q and a likelihood threshold, and it will greedily search the high PageRank website first till it finds 100 websites with similarity scores higher than the threshold.

Pagerank

This paper focuses on the engineering implementation of the PageRank algorithm while maintaining consistency with the mathematical principles proposed by Larry Page[3].

- In order to address memory overflow issues in the engineering implementation, the primary optimization techniques employed are job partitioning and the use of intermediate files.
  As illustrated in the following diagram, during multiple iterations of the PageRank algorithm, the join and aggregate processes generate a large number of intermediate files. This leads to substantial memory pressure and can potentially cause memory overflow problems. It is necessary to partition the jobs appropriately, save the intermediate results of each computation, and then utilize these intermediate results in the subsequent calculations.

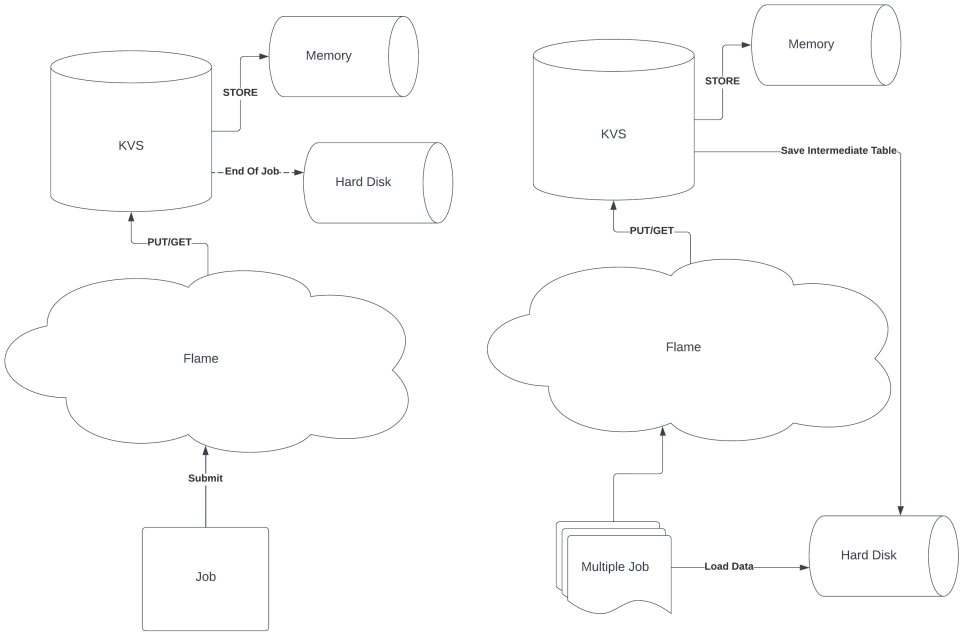

Figure 1: Model Architecture

This approach effectively reduces memory usage and prevents memory overflow issues, thereby improving the overall efficiency and performance of the algorithm implementation. In practical applications, on a computer with 16GB of memory, the implementation using intermediate files can support a data volume 10 times larger than an implementation without intermediate files. Furthermore, the former can continue

running until the PageRank algorithm converges, while the latter cannot guarantee convergence. This demonstrates the effectiveness of employing intermediate files and job partitioning in addressing memory overflow issues and ensuring the stability and accuracy of the PageRank algorithm implementation.

- Another improvement is in the handling of URLs. In this project, to address the issue of uneven distribution of row keys and the inherent data limitations of key-value stores (KVS), we have applied Base64 encoding to process the URL data. By transforming the URLs into Base64 encoded strings, we can ensure a more uniform distribution of row keys and effectively overcome the constraints posed by the KVS data structure. This enhancement contributes to the overall efficiency and performance of the PageRank algorithm implementation.

Challenging Aspects and Possible Improvements

Three Challenging Aspects

Memory To manage the large amount of data involved in web crawling and calculating PageRank, we had to carefully manage memory usage. One major issue we encountered was that the KVS cannot clear unused data. To optimize our memory usage, we had to change the original KVS to support additional features. It is not uncommon to see that after we woke up in the morning, our machine crashed.

KVS Data Discrepancy Issues. Web crawling is performed using a hierarchical traversal method to gather data from the web. However, due to the limited depth of the hierarchical traversal, the overall data distribution tends to form a pyramid-like structure. Many web pages do not contain references to themselves, and there are certain issues with the mutual referencing relationships among different web pages.
As a result, when calculating PageRank, data sinks and URL island phenomena may occur (i.e., the URLs included in a page are not present in the table, and no other URLs reference them). Possible Solutions: 1. Increase the depth of the hierarchical traversal. 2. Limit the width of each traverse.

AWS Another challenge encountered is the unexpectedly slow performance of virtualized cloud services, such as Amazon Web Services (AWS). Compared to a local computer, the execution speed on AWS is only 1/1000th. Upon analyzing the data, it was found that the I/O performance on AWS servers is significantly slower than anticipated.

According to a research paper by Keith R. Jackson, this slowdown is attributed to the network structure of the Virtual Cluster Architecture[2]. The storage disks in this architecture communicate through a shared network bridge, which results in a considerable speed difference compared to direct endpoint communication on a local machine.

Possible Improvements

If we had known what the final project would look like during our earlier homework, we would have implemented additional functions, such as 'putBatch' and 'getBatch', because the major bottleneck in our implementation was the HTTP requests between instances. Furthermore, we would have implemented an appropriate garbage collection system to reduce memory overhead.

Bibliography

[1] Dwyl. Dwyl/english-words: A text file containing 479k english words for all your dictionary/word-based projects e.g: Auto-completion / autosuggestion.

[2] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 159–168, Indianapolis, IN, USA, November 2010. IEEE.

[3] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[4] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.