ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

KỸ THUẬT DỮ LIỆU - HK231

Similarity search with MapReduce

Giảng viên hướng dẫn: TS.Phan Trọng Nhân
Nhóm: 1
Sinh viên thực hiện: Nguyễn Đức An – 2010102
Nguyễn Trường An – 2370388
Trần Quốc Anh – 1852247

Tp. Hồ Chí Minh, Tháng 11/2023

Mục lục

1 Giới thiệu

1.1 Tổng quan

Tìm kiếm tương tự (Similarity search) là một phương pháp phổ biến, liên quan đến việc tìm các đối tượng có tính chất tương tự với một truy vấn nhất định. Tìm kiếm là một hoạt động cơ bản được sử dụng để lấy thông tin từ dữ liệu số. Tìm kiếm được sử dụng rộng rãi giúp cải thiện trải nghiệm người dùng bằng cách đề xuất hoặc hiển thị các mục tương tự hoặc liên quan. Ví dụ, trong tìm kiếm web và mua sắm trực tuyến, nó giúp người dùng tìm thấy sản phẩm hoặc thông tin mà họ có thể quan tâm.. Với lượng dữ liệu kỹ thuật số ngày càng tăng, tìm kiếm trở thành một công cụ cần thiết và hữu ích.

Tìm kiếm tương tự khác với tìm kiếm khớp chính xác, vì dựa vào sự tương đồng hơn là sự trùng khớp hoàn toàn. Các công cụ tìm kiếm như Google, Yahoo!, và Bing sử dụng phương pháp này để tìm kiếm thông tin trên World Wide Web dựa trên từ khóa của người dùng. Tìm kiếm tương tự còn hỗ trợ nhiều quy trình khác như phân cụm, phân loại, khai thác dữ liệu, và ra quyết định.

Ứng dụng của tìm kiếm tương tự thường liên quan đến việc tìm kiếm dựa trên tính chất tương tự, trong khi tìm kiếm khớp chính xác thường được sử dụng trong các tác vụ yêu cầu sự chính xác tuyệt đối. Tìm kiếm tương tự quan trọng vì nó cho phép chúng ta tìm kiếm thông tin và tri thức một cách hiệu quả dựa trên sự tương đồng, thay vì dựa hoàn toàn vào các từ khóa hoặc cấu trúc dữ liệu cụ thể. Tìm kiếm khớp chính xác có thể bị hạn chế, đặc biệt đối với các đối tượng phức tạp như đa phương tiện và kết quả mong muốn của tác vụ tìm kiếm có thể thay đổi do ngữ cảnh. Ngược lại, tìm kiếm tương tự trả về kết quả có liên quan thay vì kết quả trống khi không có kết quả khớp chính xác. Do đó, tìm kiếm tương tự đã trở thành một hoạt động được sử dụng rộng rãi để phục vụ nhu cầu của rất nhiều người dùng.

Khi sử dụng tìm kiếm tương tự cũng đối mặt với nhiều thách thức, đặc biệt là xử lý dữ liệu lớn. Mô hình MapReduce được áp dụng để giải quyết vấn đề này, sử dụng cách tiếp cận song song để xử lý dữ liệu trên nhiều node, giúp giảm thời gian xử lý và đảm bảo tính nhất quán. Mặc dù vậy việc triển khai mô hình này đòi hỏi kỹ năng lập trình chuyên sâu và sử dụng công cụ phù hợp, yêu cầu phải tập trung vào việc tối ưu hóa và cải tiến mô hình để đáp ứng nhu cầu tìm kiếm tương tự trong tương lai.

1.2 Một số thách thức

Trong quá trình thực hiện thường sẽ gặp phải những khó khăn cụ thể. Khó khăn đầu tiên là quy mô của dữ liệu. Nếu tập dữ liệu cần tìm kiếm ngày càng lớn, như trong trường hợp của một công cụ tìm kiếm, thì tìm kiếm tương tự trở nên phức tạp hơn nếu thuật toán tìm kiếm chỉ chạy trên một máy và phải luôn chạy trên dữ liệu gốc ban đầu. Vì vậy, ta cần phải tiền xử lý dữ liệu một cách thích hợp để có thể giảm thiểu công sức lặp đi lặp lại của việc tính toán đặc trưng cần thiết của mỗi mẫu dữ liệu để so sánh, đồng thời phát triển các thuật toán để tìm kiếm có thể được thực hiện trên nhiều máy - hay tính toán song song - để giải quyết vấn đề có tính co dãn cao trên các tập dữ liệu lớn.

Khó khăn thứ hai là số chiều của không gian tìm kiếm. Thông thường, một văn bản có thể chứa từ vài trăm đến vài nghìn từ, và trong ngôn ngữ tiếng Anh chỉ có khoảng 170.000 từ được sử dụng phổ biến. Nếu chuyển dữ liệu này thành tập hợp hoặc vector để thực hiện tìm kiếm, ta sẽ gặp vấn đề về tính đồng nhất - khi số chiều của không gian tìm kiếm quá lớn, độ đo khoảng cách và độ tương tự giữa hai văn bản sẽ không còn có ý nghĩa nữa vì tất cả khoảng cách đều quá lớn và có thể coi như đồng nhất.

1.3 Một số phương pháp tìm kiếm tương tự

Tìm kiếm tương tự là bài toán tìm 1-lân cận gần nhất, là tập con của bài toán k-lân cận gần nhất trong lĩnh vực khai phá dữ liệu, do đó các phương pháp có thể sử dụng cũng sẽ có nhiều nét tương đồng tuy nhiên sẽ đơn giản hơn. Tùy vào lĩnh vực và yêu cầu ứng dụng mà phương pháp được sử dụng và độ đo để tính toán độ tương tự giữa hai đối tượng sẽ khác nhau.

- **Phương pháp sử dụng vector:** Đối tượng truy vấn và các đối tượng khác trong cơ sở dữ liệu được biểu diễn dưới dạng các vector nhiều chiều và độ đo thường được sử dụng để tính toán độ tương tự giữa hai vector là độ tương tự cos, khoảng cách Euclid, hay độ đo Jaccard. Hai độ đo cos và Jaccard sẽ được giới thiệu ở phần sau.
- **Phương pháp sử dụng đặc trưng:** Các đặc trưng của đối tượng và truy vấn được trích xuất và sau đó sử dụng để đo độ tương tự. Phương pháp này cũng tương tự như ở phương pháp trước, tuy nhiên phương pháp này sẽ dựa nhiều vào cách mà dữ liệu được biểu diễn (ảnh, văn bản,...) để trích xuất ra các đặc trưng tương ứng.
- **Phương pháp sử dụng đồ thị:** Mô hình hóa các đối tượng và quan hệ giữa chúng một cách tự nhiên dưới dạng đồ thị với các nốt biểu diễn các đối tượng và các cạnh biểu diễn độ tương tự giữa hai đối tượng nào đó. Độ đo giữa các đối tượng được tính toán bằng những độ đo đặc thù của cấu trúc đồ thị, như là khoảng cách biên tập đồ thị (graph edit distance), bước đi ngẫu nhiên (random walk-based), hoặc nhân đồ thị (graph kernel).
- **Phương pháp sử dụng hàm băm**: Băm các đối tượng thành những mã nhị phân (binary code) sử dụng một hàm băm và sau đó sử dụng khoảng cách Hamming hoặc những độ đo khác của kiểu dữ liệu mã nhị phân để đo độ tương tự. Phương pháp băm luôn tốt trong những ứng dụng cần sự hiệu quả về tốc độ truy vấn và tính co dãn, vì vậy nên phương pháp này thường được dùng trong những ứng dụng tìm kiếm quy mô lớn.
- **Phương pháp sử dụng học máy:** Với sự phát triển của những mạng neuron học sâu và phương pháp tự động mã hóa (auto-encoder), mô hình học máy có thể học cách tổng hợp thông tin đầu vào thành một vector biểu diễn trung gian (mô hình sequence2vector), sao cho những đối tượng tương tự với nhau sẽ được biểu diễn thành những vector gần nhau. Phương pháp này mạnh mẽ ở chỗ những đặc trưng được trích xuất sẽ do mô hình tự học, do đó với lượng dữ liệu đủ lớn phương pháp này có thể áp dụng cho nhiều dạng dữ liệu khác nhau như hình ảnh, văn bản, âm thanh.

2 Cơ sở lý thuyết

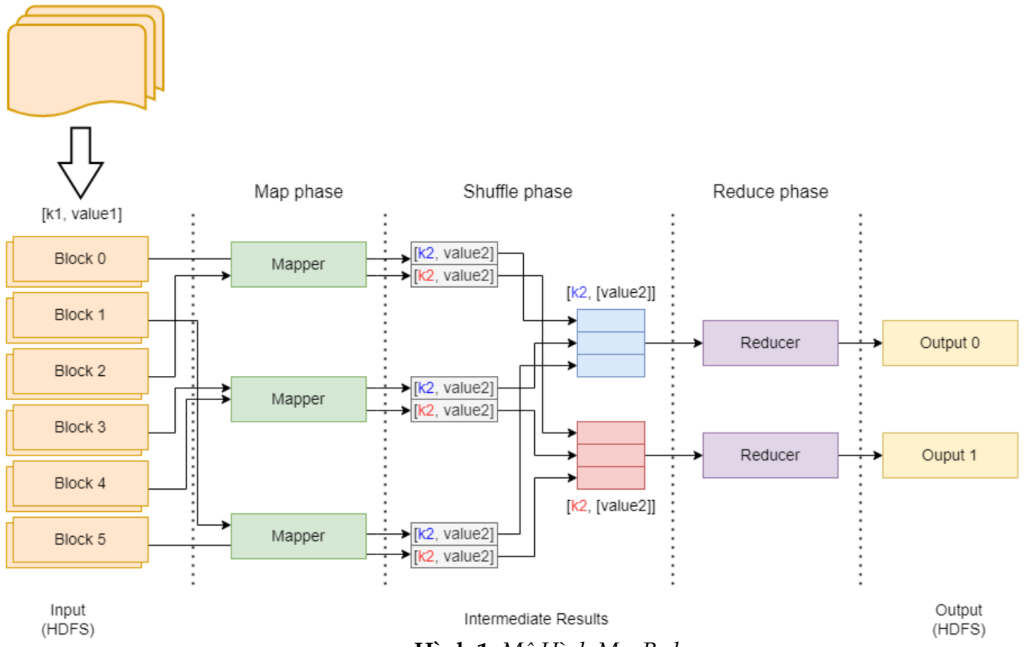
2.1 Độ đo tương tự Jaccard

Giả sử có 2 document Di và Dj, mỗi document bao gồm các term, và được kí hiệu là $D = \{term_1, term_2, ..., term_t\}$. Trong đó, mỗi term có thể xuất hiện ở một hay nhiều document khác nhau.

Độ tương tự Jaccard xác định kích thước phần giao (intersection) chia cho kích thước liên kết (union) của hai tập hợp. Hệ số tương tự Jaccard là một giá trị nằm trong khoảng từ 0 đến 1, trong đó 0 có nghĩa là các tập hợp hoàn toàn không giống nhau và 1 có nghĩa là các tập hợp giống hệt nhau và được tính bởi công thức: $SIM(D_i, D_j) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|}$

2.2 Mô hình MapReduce

MapReduce là một mô hình lập trình cho phép xử lý các tập dữ liệu lớn trên môi trường tính toán phân tán. Mô hình này được thiết kế để tách các tập dữ liệu lớn thành các khối (chunk) nhỏ hơn và xử lý chúng song song trên nhiều nút tính toán. Để sử dụng mô hình MapReduce, người dùng cần định nghĩa các tác vụ tính toán thông qua hàm MAP và hàm REDUCE. Trong mô hình, hàm MAP xử lý các cặp key-value để tạo ra một tập key-value trung gian và hàm REDUCE kết hợp tất cả các giá trị trung gian liên kết với cùng một key trung gian Khi được triển khai trên một cụm tính toán, một máy được chọn làm máy master và các máy còn lại đóng vai trò là các workers. Máy master sẽ phân chia các các MAP task và REDUCE task cho các worker.



Hình 1: *Mô Hình MapReduce*

Quy trình làm việc chung của một tác vụ MapReduce có thể được chia thành các bước sau:

- **Bước 1. Partition:** Đọc dữ liệu đầu vào từ hệ thống lưu trữ phân tán (distributed file system), chẳng hạn như Hệ thống tệp phân tán Hadoop (HDFS). Sau đó, dữ liệu này được chia thành các khối dựa trên kích thước khối (block size).
- **Bước 2. Map:** Mỗi mapper sẽ đọc khối dữ liệu tương ứng và phân giải (parse) thành các cặp [key1, value1]. Sau đó, mapper thực thi hàm MAP được chỉ định và tạo ra các cặp key-value trung gian [key2, value2]. Các cặp key-value này sẽ được đệm vào bộ nhớ, phân vùng thành r region và được ghi định kỳ vào đĩa cục bộ của mapper
- **Bước 3. Shuffle và Sort:** Khi nhận tín hiệu từ master, reducer pulls các các cặp key-value trung gian từ đĩa cục bộ của mapper liên quan đến phân vùng của nó, sau đó nhóm bởi key, tạo ra các cặp key-value mới có dạng [key2, [value2]]
- **Bước 4. Reduce:** Mỗi tác vụ 'REDUCE' nhận vào một khóa trung gian và một tập các giá trị trung gian ứng với khóa đó. Sau đó reducer sẽ merge các giá trị này để tạo thành một tập giá trị nhỏ hơn (sum, min, max ...). Các giá trị trung gian được cung cấp cho hàm REDUCE thông qua một iterator, cho phép thực thi các danh sách các giá trị trung gian phù hợp với bộ nhớ.

- **Bước 5. Output:** Các cặp khóa-giá trị đầu ra được tạo bởi các tác vụ 'Reduce' sau đó được ghi vào hệ thống lưu trữ, chẳng hạn như HDFS hoặc vào cơ sở dữ liệu, tùy thuộc vào yêu cầu của ứng dụng.

3 Hiện thực

3.1 Giải thuật chạy tuần tự

Trong cả giải thuật tuần tự và song song, nhóm đều sử dụng Python để hiện thực giải thuật. Python thì là một ngôn ngữ thông dịch được ưa chuộng vì sự đơn giản nhưng mạnh mẽ trong khả năng mở rộng và tích hợp những thư viện bên ngoài. Trong số đó những ứng dụng phổ biến nhất của Python, xử lý ngôn ngữ tự nhiên là một trong những tác vụ được ưa chuộng. Sau đây là phần hiện thực độ thuật toán tuần tự. Đầu tiên, là phần hiện thực hàm tính độ đo Jaccard.

Algorithm 1 Jaccard Similarity Calculation

Input: $set1$, $set2$
Output: sim: Similarity scores

```
intersection ← size(intersect(set1, set2))
union ← size(union(set1, set2))
similarity ← intersection/union
return similarity=
```

Tiếp theo, là phần hiện thực giải thuật tìm kiếm tương tự bằng cách so sánh độ tương đồng Jaccard theo hướng tuần tự.

Algorithm 2 Linear file search with Jaccard

Input: $rootPath$, $searchTerm$
Output: $filePath$, sim

```
foundTimes ← {}
searchSet ← set(wordTokenize(searchTerm.lower()))
while True do
    subDir, dirs, files ← os.walk(rootPath)
    for each file in given path do
        if file ends with '.txt' then
            filePath ← os.path.join(subDir, file)
            fileSet ← set()
            for each line in file do
                lineSet ← set(wordTokenize(line.lower()))
                fileSet.update(lineSet)
            intersection, union, similarity ← jaccard(searchSet, fileSet)
            if similarity > 0 then
                yield filepath, similarity
```

Nếu hệ số tương tự Jaccard lớn hơn 0, tức là tệp văn bản chứa từ tìm kiếm, hàm sẽ in ra thông tin về tệp văn bản đó, bao gồm đường dẫn, số phần tử chung, tổng số phần tử và hệ số tương tự Jaccard. Hàm cũng tính thời gian tìm kiếm và in ra thời gian tổng cộng.

Tóm lại, thuật toán tìm kiếm tương tự sử dụng phương pháp so sánh Jaccard với cách tiếp cận tuần

tự này sẽ duyệt qua từng tệp văn bản trong thư mục cần tìm kiếm và tính hệ số tương tự Jaccard giữa từng tệp và thuật ngữ tìm kiếm. Điều này có thể trở nên khá chậm nếu thư mục cần tìm kiếm chứa nhiều tệp văn bản hoặc các tệp văn bản rất lớn. Để giải quyết vấn đề này, ta có thể sử dụng phương pháp tìm kiếm tương tự bằng MapReduce để xử lý dữ liệu phân tán và tăng tốc độ tìm kiếm.

3.2 Môi trường chạy MapReduce

Để chạy được giải thuật MapReduce cần một cụm các máy tính được kết nối với nhau trong một mạng và tất cả đều cài phần mềm để điều phối việc chạy MapReduce. Ở đây nhóm chọn thử nghiệm sử dụng Hadoop và giả lập, kết nối các worker trong một mạng các Docker container.

Để thuận tiện trong lập trình, nhóm sử dụng Hadoop Streaming API. Hadoop Streaming API là một chương trình được cung cấp bởi Hadoop, cho phép thực thi các giải thuật thông qua standard input và standard output. Trong một giai đoạn MapReduce, các worker MAP và REDUCE sẽ nhận input từ stdin và sau đó in kết quả ra stdout, và Hadoop Streaming sẽ làm những công việc còn lại. Điều này cho phép người lập trình được tự do chọn lựa ngôn ngữ phù hợp và giao tiếp với nhau thông qua một giao thức định sẵn. Ngoài sự tự do đó, đối với những tác vụ cần hiệu năng tính toán tốt hơn ngôn ngữ Java mặc định, người dùng còn có thể biên dịch chương trình của mình thành một file thực thi và sau đó thực thi các file này sử dụng Hadoop Streaming.

3.3 Mã giả và hiện thực giải thuật song song

3.3.1 Giai đoạn 1

Ở đây nhóm chọn hiện thực giải thuật tìm kiếm tương tự theo độ đô Jaccard. Giải thuật sẽ được chia làm 2 giai đoạn: xây dựng chỉ mục ngược (inverted index) và tìm kiếm tương tự. Mỗi giai đoạn bao gồm một tác vụ MAP và REDUCE. [1] [2] Ký hiệu:

- D_i**:** văn bản thứ i
- $term_k$**:** từ thứ k
- URL_i**:** đường dẫn xác định văn bản D_i
- W_i**:** số lượng từ riêng biệt có trong D_i
- @**:** ký tự đặc biệt để phân tách các giá trị

Ở bước MAP của giai đoạn 1, các từ được tách ra từ đoạn văn bản ban đầu, cùng với đường dẫn xác định duy nhất văn bản đó (URL). Một tập hợp các từ xuất hiện trong văn bản được xây dựng và sau đó số lượng từ xuất hiện trong văn bản đó sẽ được trích xuất ra. Với những từ trùng nhau ở câu truy vấn q và tập hợp từ, chương trình sẽ phát ra - trong trường hợp của Hadoop Streaming thì là in ra stdout - những cặp khóa-trị như mã giả dưới đây.

Sau khi lọc được những văn bản có chứa những từ trong câu truy vấn, bước7 REDUCE sẽ gom những văn bản chứa một từ khóa nhất định (GroupByTerm) vào chung với nhau để hoàn thiện bảng chỉ mục ngược.

Algorithm 3 MAP-1

Input:

- A set of documents [D_i]
- A query object q

Output:

- A part of customized inverted indices $[term_k, URL_i@W_i]$

```
for each line in D_i do
    [term_k] ←ExtractTerms(line)
length ← GetLength([term_k])
URL_i ←GetURL(D_i)
[term] ← [term_k]∩ExtractTerms(q)
for each [term_k] in[term] do
    Emit([term_k], URL_i ∩ length)
```

Algorithm 4 REDUCE-1

Input:

- A part of inverted indices $[term_k.URL_i@W_i]$
- Number of total documents n

Output:

- A customized inverted index $[term_k, [URL_i@W_i]_{ord}]$

```
data ←Read(Input)
for each term, group in GroupByTerm(data) do
    if GetLength(group) <> n and GetLength <> 1 then
        Emit(term, sorted(group))
```

3.3.2 Giai đoạn 2

Kết quả chỉ mục đảo ngược ở giai đoạn 1 sẽ được sử dụng để tính toán độ tương tự giữa các tài liệu. Trong giai đoạn này, các Mapper sau khi nhận được chỉ mục, sẽ tiến hành cắt tỉa những từ không xuất hiện trong câu query (Query Parameter Filter) và kết quả sau khi được tính ở Reducer sẽ được lọc ra theo mong muốn dựa trên giá trị ngưỡng (Range Query Filtering) hoặc lựa chọn k kết quả tốt nhất (k-NN Query Filtering).

Cụ thể, giải thuật MAP-2 nhận đầu vào là một chỉ mục đảo ngược (customized inverted index) có dạng $[term_k, [URLi@Wi]_{ord}]$ và đầu ra là danh sách các cặp ứng viên. Giải thuật đầu tiên đọc dữ liệu đầu vào, với mỗi term, thông tin về câu truy vấn được trích xuất sử dụng hàm GetQueryObject. Sau đó, trích xuất URL_Q và W_Q từ câu truy vấn sử dụng hàm ExtractURLQ và ExtractWQ tương ứng. Tiếp theo, giải thuật lặp qua từng tài liệu trong danh sách các tài liệu chứa term. Với mỗi tài liệu không phải query, giải thuật trích xuất URL và W và tạo cặp ứng viên giữa query với tài liệu đó ở dạng $[URL_{ij}@W_i@W_j, 1]$, trong đó URL_{ij} là kết hợp của URL_i và URL_q, số 1 là cờ biểu diễn D_i và D_j nhận term hiện tại làm phần giao.

Algorithm 5 MAP-2

Input:

- A customized inverted index $[term_k, [URL_i@W_i]_{ord}]$

Output:

- A list of candidate pairs $[URL_{ij}@W_i@W_j, 1]$

```
1  data ←Read(Input)
2  for each term, elements in data do
3      q ←GetQueryObject(elements)
4      urlq ←ExtractURLQ(q)
5      wq ←ExtractWQ(q)
6      for each element in elements do
7          if element <> q then
8              url ←ExtractURLQ(element)
9              w ←ExtractW(element)
10             Emit(url ∩ urlq ∩ w ∩ wq, 1)
```

Algorithm 6 REDUCE-2

Input:

- A list of candidate pairs $[URL_{ij}@W_i@W_j, 1]$

Output:

- Final similarity scores $[URL_{ij}, SIM(D_i, D_j)]$

```
1  data ←Read(Input)
2  for each pair, group in GroupBy(data) do
3      for each pair, value in group do
4          val ← val + value
5      wi, wj ←ExtractW(pair)
6      sim ← val/ + (wi + wj − val)
7      Emit(pair, sim)
```

Giải thuật REDUCE-2 nhận danh sách các cặp ứng viên cùng với các trọng số và tính toán điểm tương tự cho mỗi cặp. Giải thuật đọc dữ liệu đầu vào và nhóm chúng theo URL_s. Với mỗi nhóm, tính tổng giá trị cờ phần giao val sau đó trích xuất giá trị w_i và w_j của ứng viên và tính toán độ tương tự bằng công thức: $sim = val/(w_i + w_j - val)$. Cuối cùng, kết quả được xuất ra stdout.

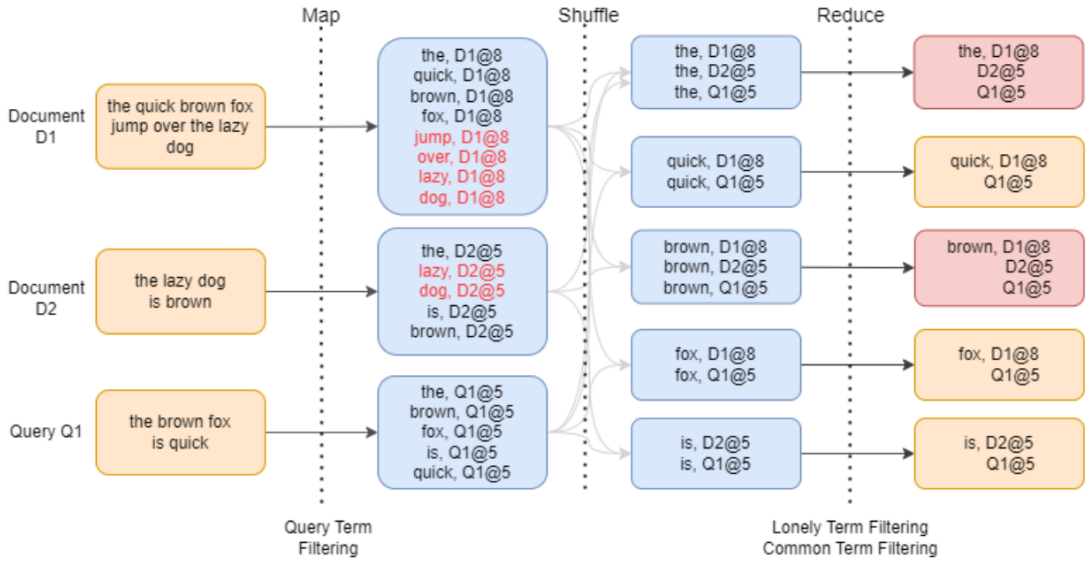
3.3.3 Ví dụ minh họa

Để mô tả rõ hơn về 2 giai đoạn MapReduce, nhóm sử dụng ví dụ minh hoạt gồm 2 tài liệu D1: “the quick brown fox jump over the lazy dog", D2: “the lazy dog is brown" và câu truy vấn Q1: “the brown fox is quick".

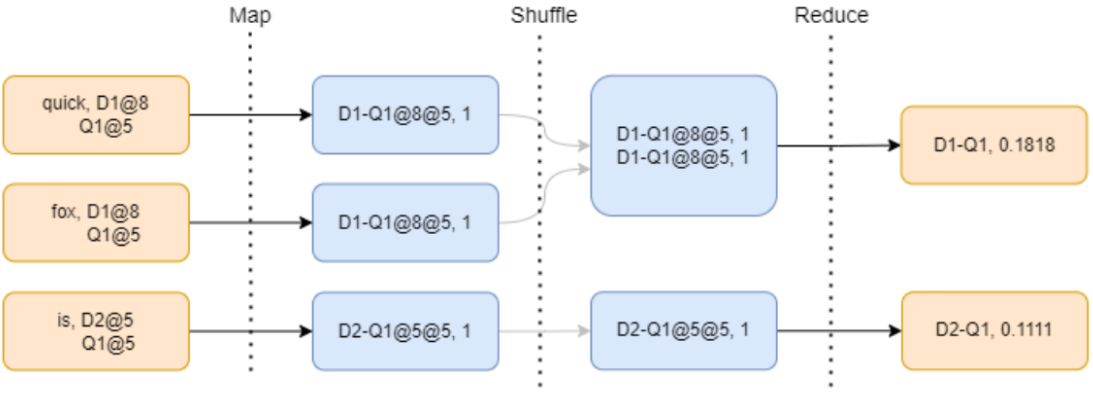
Hình 2 minh họa giai đoạn Mapreduce-1 với đầu vào là các tài liệu và câu truy vấn. Các mapper sẽ thực hiện đọc dữ liệu đầu vào, trích xuất các term có trong tài liệu. Bước Query Term FIltering được thực hiện để loại bỏ các term chỉ xuất hiện trong các tài liệu mà không xuất hiện trong câu truy vấn. Trong ví dụ mình họa, các term "jumps", "over", "lazy", "dog" chỉ xuất hiện trong tài liệu D1, D2 mà

không xuất hiện trong câu truy vấn sẽ bị loại bỏ và mapper sẽ xuất ra các cặp key, value trung gian của các term còn lại. Sau đó, các reducer sẽ nhận bộ key và các value tương ứng với key đó, áp dụng Lonely Term Filtering để loại bỏ các term chỉ xuất hiện trong câu truy vấn, áp dụng Common Term Filtering để loại bỏ các term xuất hiện trong tất cả các tài liệu và câu truy vấn. Trong ví dụ mình họa, term "the", "brown" xuất hiện trong tất cả các tài liệu và truy vấn nên sẽ bị loại bỏ vì không mang thông tin phân biệt các tài liệu. Các key, value của chỉ mục đảo ngược sẽ được xuất ra từ các term còn lại bao gồm [[quick, [D1@8, Q1@5]], [fox, [D1@8, Q1@5], [is, [D2@5, Q1@5]]].

Hình 3 mình họa giai đoạn MapReduce-2. Đầu tiên, các mapper sẽ xuất ra các cặp dự tuyển [[D1-Q1@8@5, 1], [D1-Q1@8@5, 1], [D2-Q1@5@5, 1]. Bước shuffle sẽ nhóm các key lại với nhau và sắp xếp theo thứ tự. Sau đó, các reducer nhận các cặp key, value đã sắp xếp, tổng hợp, và tính độ tương tự, kết quả như sau: [[D1-Q1, 0.1818], [D2-Q1, 0.1111]]



Hình 2: *Bước MapReduce-1 của phương pháp dựa trên độ đo Jaccard*



Hình 3: *Bước MapReduce-2 của phương pháp dựa trên độ đo Jaccard*

4 Đánh giá

4.1 Tập dữ liệu Gutenberg

4.1.1 Giới thiệu chung

Tập dữ liệu Gutenberg là một bộ sưu tập gồm hàng chục ngàn cuốn sách, tác phẩm văn học kinh điển, các tác phẩm phiên bản tự do và chưa bản quyền của nhiều tác giả nổi tiếng. Được chia sẻ công khai và miễn phí, tập dữ liệu Gutenberg được sáng lập bởi dự án Gutenberg của Cộng đồng Quốc tế với mục tiêu làm cho các tác phẩm văn hóa truyền thống trở nên dễ dàng tiếp cận cho mọi người trên toàn thế giới.

Tập dữ liệu Gutenberg có thể được tải về miễn phí từ trang web của Dự án Gutenberg https://www.gutenberg.org/. Tập dữ liệu được cung cấp dưới nhiều định dạng khác nhau, bao gồm định dạng văn bản thô, HTML, EPUB và Kindle, giúp cho các nhà nghiên cứu, nhà phát triển và người đam mê tiếp cận dễ dàng. Tập dữ liệu bao gồm sách trong nhiều ngôn ngữ khác nhau, tuy nhiên đa số là tiếng Anh. Mỗi cuốn sách được định danh bằng một ID số duy nhất, có thể được sử dụng để truy cập sách dưới nhiều định dạng khác nhau.

Tập dữ liệu Gutenberg là một tài nguyên quý giá cho các nhà nghiên cứu và nhà phát triển làm việc với xử lý ngôn ngữ tự nhiên và phân tích văn bản. Nó cung cấp một bộ sưu tập văn bản lớn và đa dạng có thể được sử dụng cho nhiều tác vụ khác nhau, bao gồm mô hình hóa ngôn ngữ, phân loại văn bản, phân tích tâm trạng và nhiều hơn nữa. Vì tập dữ liệu thuộc phạm vi công cộng, nó có thể được sử dụng và phân phối một cách tự do mà không gặp bất kỳ hạn chế pháp lý nào.

Chúng em quyết định sử dụng tập dữ liệu Gutenberg để thực hiện đánh giá hiệu suất và so sánh giữa hai phương pháp xử lý dữ liệu: tuần tự và sử dụng Hadoop MapReduce. Bằng cách sử dụng tập dữ liệu này, chúng em sẽ đánh giá cả khả năng xử lý của hệ thống khi đối mặt với một lượng lớn văn bản với độ phức tạp và đa dạng cao. Và chúng em tin rằng việc sử dụng nó làm dữ liệu benchmark sẽ mang lại những kết quả có ý nghĩa trong việc đánh giá và so sánh hiệu suất của các giải thuật và phương pháp xử lý dữ liệu.

4.2 Thử nghiệm giải thuật chạy tuần tự và MapReduce

Để thử nghiệm giải thuật tìm kiếm tương tự bằng cách so sánh độ tương đồng Jaccard theo hướng tuần tự và chọn đường dẫn đến tập dữ liệu gutenberg bao gồm 335 text files. Ở cả hai giải thuật chúng em đều sử dụng chung 1 câu tìm kiếm để có thể nhận được kết quả trực quan nhất.

Search term: "The Online World does not cover any specific area of the world"

4.2.1 Giải thuật tuần tự

Việc đánh giá thử nghiệm giải thuật tuần tự được chạy được tiến hành trên hệ thống máy tính được trang bị bộ xử lý Intel Core i7, RAM 32GB và chạy Python 3.10.

Listing 1: Jaccard Similarity Tuần Tự

```
00001.txt - 0.00917431 - 0.0133(s)
00002.txt - 0.00305998 - 0.0781(s)
00003.txt - 0.01149425 - 0.0118(s)
00004.txt - 0.00320256 - 0.0468(s)
00005.txt - 0.00416667 - 0.0480(s)
.....
05098.txt - 0.00089240 - 0.6129(s)
05099.txt - 0.00079673 - 1.0066(s)
05288.txt - 0.00323102 - 0.0431(s)
05292.txt - 0.00278552 - 0.0517(s)
05293.txt - 0.00314713 - 0.0452(s)
```

Tổng thời gian chạy: 235 giây

4.3 Giải thuật MapReduce

Việc đánh giá thử nghiệm giải thuật tuần tự được chạy được tiến hành trên hệ thống Hadoop với 1 master và 3 workers/slaves. Mỗi worker được cấp cấu hình 8 GB Memory và được chạy trên môi trường Hadoop(Streaming) 2.7.2 và Python 3.6

Listing 2: Jaccard Similarity Hadoop MapReduce

```
00001.txt - 0.00917431
00002.txt - 0.00305998
00003.txt - 0.01149425
00004.txt - 0.00320256
00005.txt - 0.00416667
.....
05098.txt - 0.00089240
05099.txt - 0.00079673
05288.txt - 0.00323102
05292.txt - 0.00278552
05293.txt - 0.00314713
```

Tổng thời gian chạy: 474 giây

4.4 Đánh giá

Dựa vào kết quả trên, ta có thể thấy rằng thời gian thực thi bằng MapReduce có thể thấy là lâu hơn đáng kể so với phương pháp tuần tự. Nguyên nhân chính của sự chênh lệch này là do phương pháp sử dụng MapReduce yêu cầu hai giai đoạn chính: xây dựng chỉ mục ngược(inverted index) và sau đó tính toán độ đo Jaccard. Mặc dù việc xây dựng inverted index có thể tạo thêm chi phí thời gian ban đầu, nhưng giá trị của nó nằm ở khả năng tái sử dụng cho các lượt tra cứu tiếp theo, giúp tối ưu hóa hiệu suất trong các tác vụ đòi hỏi truy cập dữ liệu lặp lại.

Thêm vào đó, một lý do khác cần được xem xét là bản chất của máy tính đang được sử dụng trong thử nghiệm. Hiện tại, máy tính này chỉ có 8 CPU và 32 GB bộ nhớ. Do đó, việc tạo ra một lượng lớn các node-slave cho hệ thống Hadoop có thể bị hạn chế. Sự hạn chế này có thể tạo ra một tình trạng mà hiệu suất của Hadoop MapReduce không thể được tận dụng tối đa do số lượng node-slave cũng như cấu hình không đủ tốt. Điều này làm tăng thời gian thực hiện và có thể làm giảm hiệu suất so với phương pháp tuần tự trên một máy tính với cấu hình tương đương

Ở đây em có thử thêm câu tìm kiếm dài hơn khoảng 500 chữ. Khi ta xem chi tiết thời gian chạy của 2 giai đoạn Map Reduce ta sẽ có được bảng sau:

Map Reduce Stage	Term 1	Term 2
1. Build inverted index	462	458s
2. Jaccard similarity	12s	16s

Bảng 1: *So sánh khi tăng số chữ đầu vào*

5 Kết luận

Tổng kết lại, bài báo cáo về việc so sánh giữa hai phương pháp xử lý độ đo Jaccard, bằng phương pháp tuần tự và phương pháp sử dụng Hadoop MapReduce, đã cung cấp cái nhìn sâu sắc về hiệu suất của chúng trên tập dữ liệu phong phú từ thư viện Gutenberg. Kết quả thử nghiệm cho thấy rằng, mặc dù phương pháp sử dụng MapReduce có thể đòi hỏi thời gian thực hiện lâu hơn do yêu cầu xây dựng inverted index và tính toán độ đo Jaccard, nhưng lại mang lại những lợi ích đáng kể về khả năng mở rộng và tái sử dụng dữ liệu.

Trong bối cảnh máy tính hiện tại với sự phổ biến của công nghệ đám mây, việc mở rộng quy mô cho Hadoop MapReduce trở nên đơn giản và linh hoạt hơn bao giờ hết. Công nghệ đám mây cung cấp khả năng linh hoạt cao, cho phép chúng ta mở rộng cụm xử lý một cách dễ dàng thông qua việc tăng số lượng node-slave mà không gặp phải các hạn chế về tài nguyên cục bộ. Ngược lại, phương pháp tuần tự thì dường như hiệu quả hơn với cấu hình máy tính nhất định, tận dụng tối đa tài nguyên có sẵn.

Tuy nhiên, không thể phủ nhận rằng sự lựa chọn giữa hai phương pháp phụ thuộc nhiều vào bối cảnh cụ thể của hệ thống và yêu cầu công việc. Trong trường hợp cần xử lý lượng dữ liệu lớn và có khả năng mở rộng, Hadoop MapReduce trở thành một lựa chọn hợp lý, đặc biệt khi khả năng tái sử dụng chỉ mục ngược có thể giúp tối ưu hóa hiệu suất trong các tác vụ tra cứu lặp lại.

Một trong những hạn chế quan trọng ở bài báo cáo này là bản thân tập dữ liệu thử nghiệm chưa đủ lớn để đánh giá hoàn toàn sức mạnh và yếu điểm của cả hai thuật toán trong điều kiện tối ưu nhất. Trong môi trường thử nghiệm của chúng em, chúng em đã cố gắng tối ưu hóa các yếu tố như kích thước dữ liệu và tài nguyên máy tính, nhưng không tránh khỏi việc rằng còn nhiều yếu tố khác, chẳng hạn như đặc tính cụ thể của các tập dữ liệu, môi trường mạng, và cấu hình Hadoop MapReduce, có thể ảnh hưởng đến kết quả thực tế.

Tài liệu

[1] Trong Phan, Josef Küng, and Tran Dang. *An Efficient Similarity Search in Large Data Collections with MapReduce*, pages 44–57. 01 2014.

[2] Trong Nhan Phan. Approaches for fast similarity search with mapreduce, 2016.