Introduction to Scalable Systems Assignment 1

Sandeep Kumar

September 22, 2024

1 Q1 Code Implementation and Complexity Analysis

Here is the C++ implementation of the function `groupCustomersByHashtags`:

```cpp
void groupCustomersByHashtags(fileIterator& hashtags, fileIterator& purchases, fileIterator& prices, int k, string outputFilePath)
{
    // Define a variable to store product to hashtags mapping
    unordered_map<int, set<string>> productHashtags;

    // Write a while loop to read the hashtags file and populate the productHashtags
    while (hashtags.hasNext()) {
        string line = hashtags.next();
        if(line.size() > 0){
            stringstream ss(line);
            int product;
            string temp;
            getline(ss, temp, ',');  // Read up to the first comma
            product = stoi(temp);    // Convert to integer
            while (getline(ss, temp, ',')) {
                // Convert temp to lowercase
                transform(temp.begin(), temp.end(), temp.begin(), ::tolower);
                productHashtags[product].insert(temp);
            }
        }
    }

    // Map customer ID to their hashtag frequency (i.e., interests)
    unordered_map<int, unordered_map<string, int>> customerInterests;

    // Step 2: Read the purchases and build each customer's interest list based on purchased products' hashtags
    while (purchases.hasNext()) {
        string line = purchases.next();
        if (line.size() > 0){
            stringstream ss(line);
            int customer, product;
            string temp;
            getline(ss, temp, ',');
            customer = stoi(temp);
            getline(ss, temp, ',');
            product = stoi(temp);
            for (const auto& hashtag : productHashtags[product]) {
                customerInterests[customer][hashtag]++;
           }
        }
    }

    // Step 3: For each customer, extract the top-k most frequent hashtags
    unordered_map<vector<string>, vector<int>, VectorStringHash> groupedCustomers;
    for (auto& customerInterestPair : customerInterests) {
        int customer = customerInterestPair.first;
        auto& interestMap = customerInterestPair.second;
        // Sort hashtags by frequency (descending order) and then lexicographically
        vector<pair<string, int>> freqHashtags(interestMap.begin(), interestMap.end());

        sort(freqHashtags.begin(), freqHashtags.end(), [](const auto& a, const auto& b) {
            if (a.second == b.second) return a.first < b.first;
            return a.second > b.second;
        });

        // Extract the top-k hashtags
        vector<string> topKHashtags;
        for (int i = 0; i < min(k, (int)freqHashtags.size()); ++i) {
            topKHashtags.push_back(freqHashtags[i].first);
        }
        // Sort the topKHashtags
        sort(topKHashtags.begin(), topKHashtags.end());

        // Group customers who share the same top-k hashtags
        groupedCustomers[topKHashtags].push_back(customer);
    }

    // Step 4: Write the grouped customers in CSV output file, each group in a new line, and customers in the group separated by commas
    for(const auto& temp : groupedCustomers){
        writeOutputToFile(temp.second, outputFilePath);
    }

    return;
}
```

Time Complexity: the overall time complexity of the function is:

$$O(H \cdot T \cdot \log T + P \cdot T + C \cdot T \log T) \tag{1}$$

Where:

- H is the number of products.
- T is the average number of hashtags per product.

- P is the number of purchases.
- C is the number of customers.

Space Complexity: the overall space complexity is:

$$O(H \cdot T + C \cdot T) \tag{2}$$

Where:

- $H \cdot T$ is the space required for storing product hashtags.
- $C \cdot T$ is the space required for storing customer interests.

Empirical Validation: from above time and space complexity analysis we can say that time and space complexity mainly depends on three variable 4 variable H, T, P and C.

- Varying H: We are considering the number of products (H) as a variable, while keeping the number of customers (C), the number of purchases (P), and the average number of hashtags per product (T) constant.

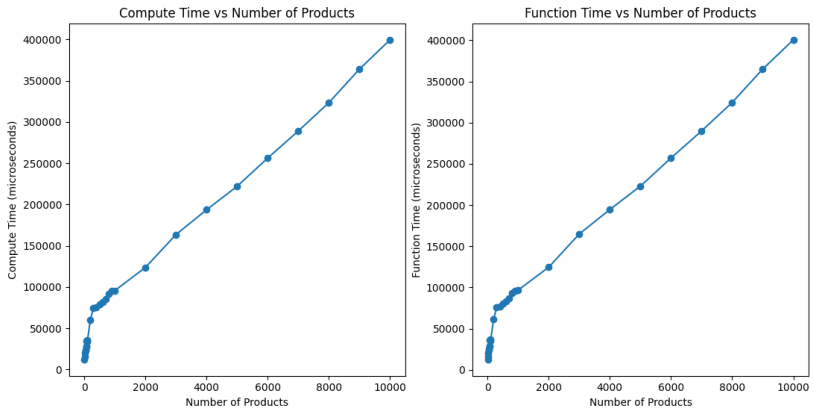

Fig1. (a) Compute time for different values of H, while keeping T = 30, C = 300, and P = 8C. (b) Function time for different values of H, while keeping T = 30, C = 300, and P = 8C.

The behaviour of compute time and function time is linear, as can be seen in Fig1 (a) and (b), and in [1] as T = 30, P = 8C and C = 300 are kept constant.

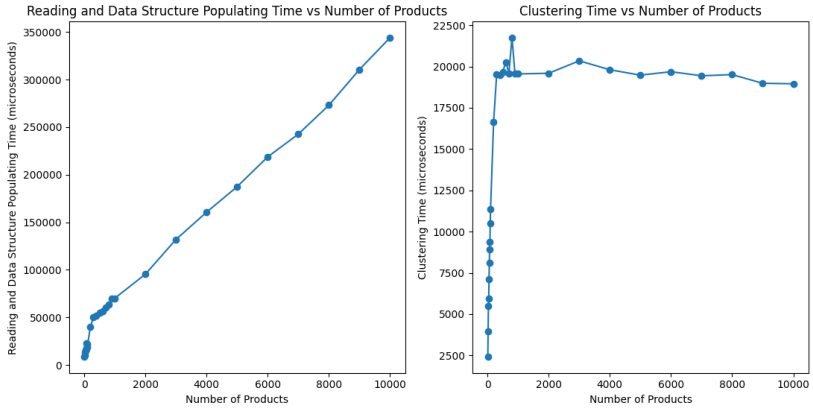

Fig2. (a) Reading and data structure populating time for different values of H, while keeping T = 30, C = 300, and P = 8C. (b) Clustering time for different values of H, while keeping T = 30, C = 300, and P = 8C.

Behaviour of reading and data structure populating time is linear, as it can be seen in Fig2 (a). The time complexity for reading and data structure populating is linear ($\mathcal{O}(H \cdot T + C \cdot T)$) as T = 30 and C = 300. time complexity for clustering time is constant, it can be seen in fig2 (b). the complexity for clustering time is constant($\mathcal{O}(P \cdot T)$) as T = 30 and P = 8C(C = 300)

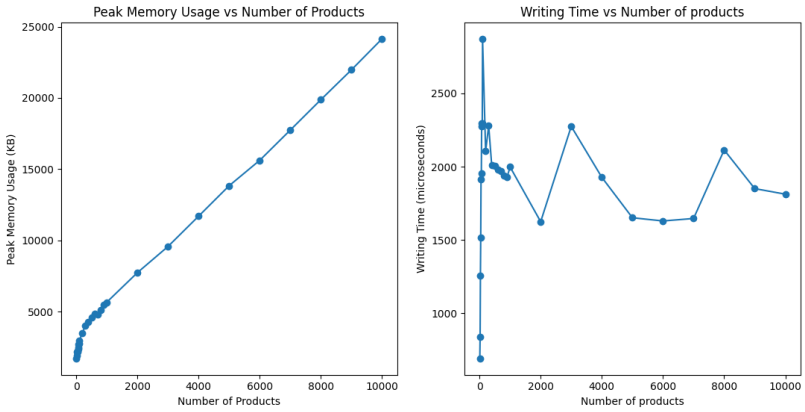

Fig3. (a) peak Memory Usage for different values of H, while keeping T = 30, C = 300, and P = 8C. (b) writing time for different values of H, while keeping T = 30, C = 300, and P = 8C.

Behaviour of peak memory usage is linear, as it can be seen in fig3. (a). peak memory usage is linear as space complexity($\mathcal{O}(H \cdot T + C \cdot T)$) as C = 300 and T = 30. writing time is constant, as it can be seen in fig3. (b), because lot of products will have same hashtags(as hashtags are limited and no of products is large) and no. of groups will become constant.

- Varying C: at P(P = 8C), T = 30, H = 300.

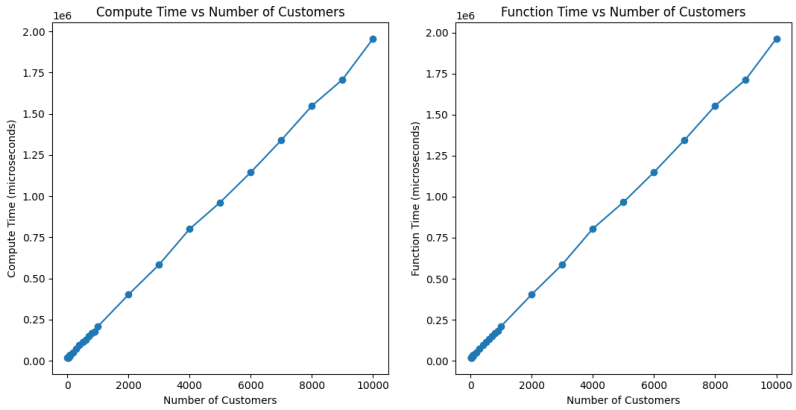

Fig1. (a) Compute time for different values of C, while keeping T = 30, H = 300, and P = 8C. (b) Function time for different values of C, while keeping T = 30, H = 300, and P = 8C.

The behaviour of compute time and function time is linear, as can be seen in Fig1 (a) and (b), and in [1] as T = 30, P = 8C and H = 300 are kept constant.

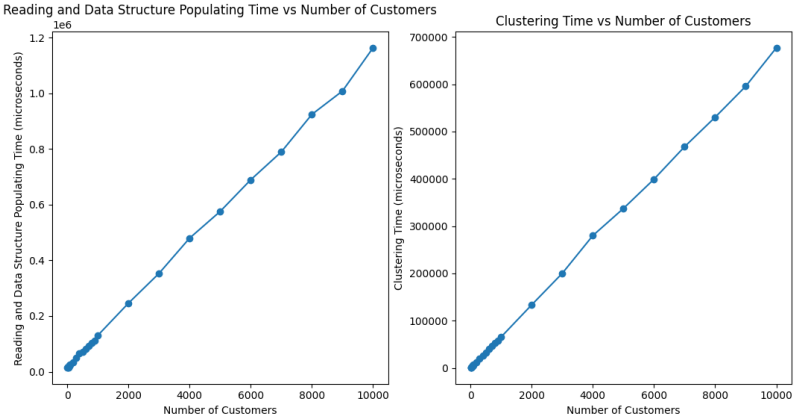

Fig2. (a) Reading and data structure populating time for different values of C, while keeping T = 30, H = 300, and P = 8C. (b) Clustering time for different values of C, while keeping T = 30, H = 300, and P = 8C.

Behaviour of reading and data structure populating time is linear, as it can be seen in Fig2 (a). The time complexity for reading and data structure populating is linear ($\mathcal{O}(H \cdot T + C \cdot T)$) as T = 30 and H = 300. clustering time is linear, it can be seen in fig2 (b). the complexity for clustering time is linear($\mathcal{O}(C \cdot T \log T)$) as T = 30.

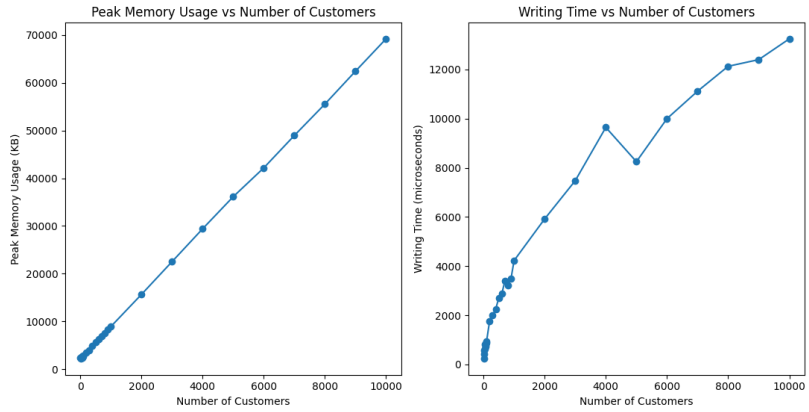

Fig3. (a) peak Memory Usage for different values of C, while keeping T = 30, H = 300, and P = 8C. (b) writing time for different values of C, while keeping T = 30, H = 300, and P = 8C.

Behaviour of peak memory usage is linear, as it can be seen in fig3. (a). peak memory usage is linear as space complexity($\mathcal{O}(H \cdot T + C \cdot T)$) as H = 300 and T = 30. writing time is approximately $c^{\frac{1}{2}}$, as it can be seen in fig3. (b).

- Varying T: at P(P = 8C), C = 300, H = 300.

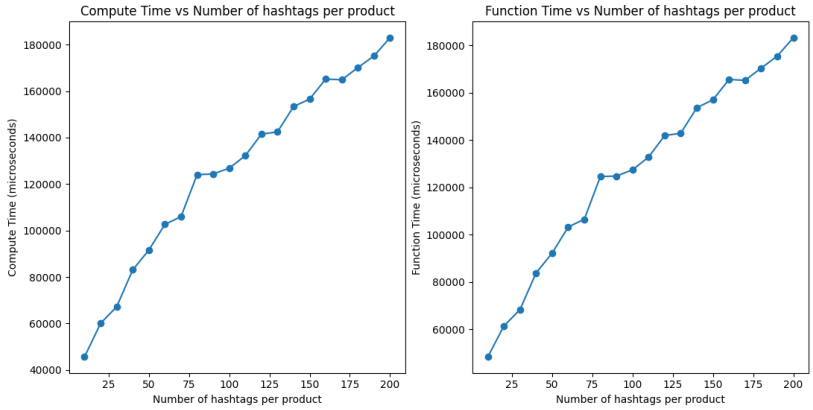

Fig1. (a) Compute time for different values of T, while keeping C = 300, H = 300, and P = 8C. (b) Function time for different values of T, while keeping C = 300, H = 300, and P = 8C.

The behaviour of compute time and function time is $\mathcal{O}(n \log n)$, as can be seen in Fig1 (a) and (b), and in [1] as C = 300, P = 8C and H = 300 are kept constant.

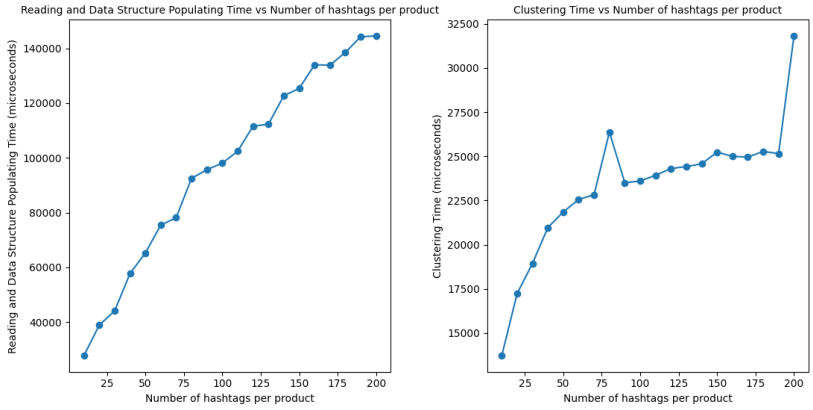

Fig2. (a) Reading and data structure populating time for different values of T, while keeping C = 300, H = 300, and P = 8C. (b) Clustering time for different values of T, while keeping C = 300, H = 300, and P = 8C.

Behaviour of reading and data structure populating time is linear, as it can be seen in Fig2 (a). The time complexity for reading and data structure populating is linear ($\mathcal{O}(H \cdot T + C \cdot T)$) as C = 300 and H = 300. clustering time is $\mathcal{O}(n \log n)$, it can be seen in fig2 (b). the complexity for clustering time is linear($\mathcal{O}(C \cdot T \log T)$) as C = 300.

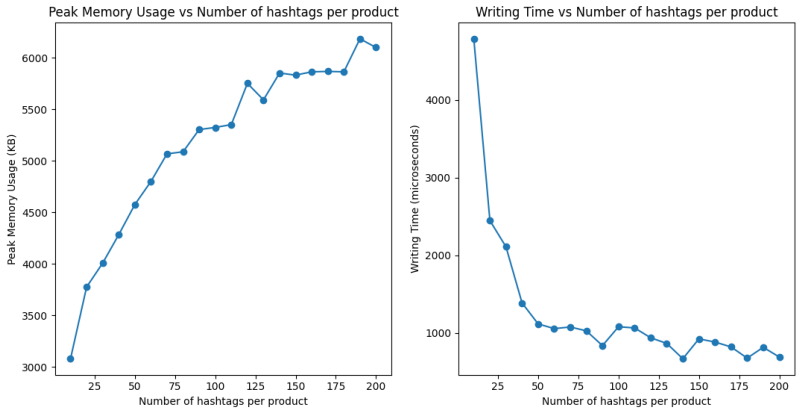

Fig3. (a) peak Memory Usage for different values of T, while keeping C = 300, H = 300, and P = 8C. (b) writing time for different values of C, while keeping C = 300, H = 300, and P = 8C.

The behavior of peak memory usage is linear, as can be seen in Fig. 3(a). Peak memory usage is linear with space complexity $\mathcal{O}(H \cdot T + C \cdot T)$, where $H = 300$ and $C = 300$. Writing time is approximately $\frac{1}{n}$, as shown in Fig. 3(b), it is because of every product have every hashtag(as hashtags are limited), therefore customer fall in less no of groups as T increases.

2 Q3 Code Implementation and Complexity Analysis

this question utilizes function `groupCustomersByHashtagsForDynamicInserts`:

Empirical Validation: in this case we need to analyze the time and space complexity of dynamic inserts.

$$TotalTimeComplexity = \mathcal{O}(H \cdot T + n \cdot T + P \cdot T + C \cdot (T \log T + k)) \tag{3}$$

where:

- H = number of hashtag lines
- T = average number of hashtags per product
- P = number of purchases
- C = number of customers
- n = number of new hashtag lines
- T = number of hashtags per entry
- k = top-k hashtags extracted per customer

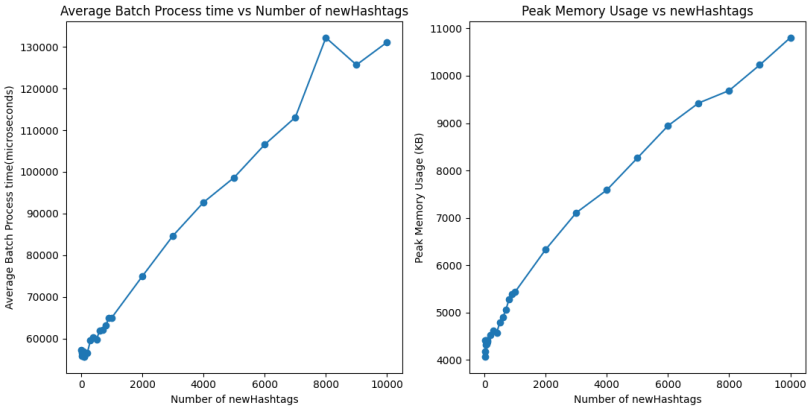

Fig1. (a) Average Batch process time for different value of n, while keeping T = 30, C = 300, H = 300, and P = 8C. (b) peak memory usage for different value of n, while keeping T = 30, C = 300, H = 300, and P = 8C.

The behaviour of Average Batch process time and peak memory usage is $\mathcal{O}(n \log n)$, as can be seen in Fig1 (a) and (b), and in [3] as T = 30,C =300, P = 8C and H = 300 are kept constant.

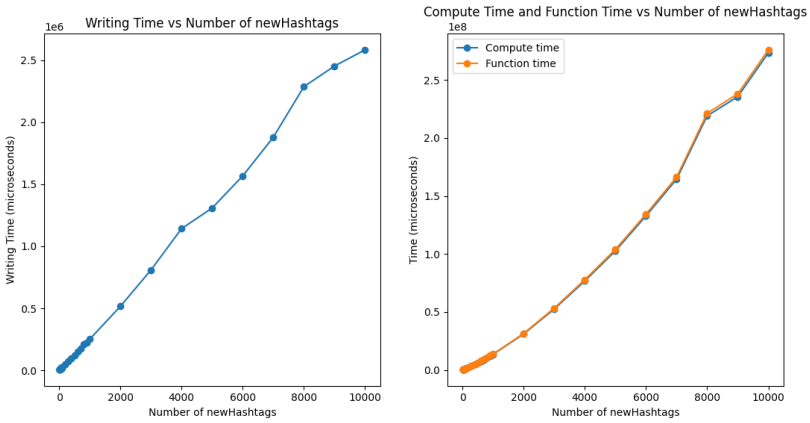

Fig2. (a) writing Time for different values of n, while keeping T = 30, C = 300, H = 300, and P = 8C. (b) compute time and function time for different values of n, while keeping T = 30, H = 300, C = 300 and P = 8C.

Behaviour of writing Time is linear, as it can be seen in Fig2 (a). The time complexity for compute time and Function time is $\mathcal{O}(n \log n)$ as T = 30, C = 300, P = 8C, H = 300.it can be seen in Fig1 (a), and in [3] as T = 30,C =300, P = 8C and H = 300 are kept constant.