Kadapala Rakesh Reddy -M22AI608-Assignment-1 (Fractal - 3)- Algorithms for Big Data

Q1> To perform the specified tasks using MapReduce on the given weblog dataset, we'll design the algorithms for the map stage and reduce stage. We'll assume that each entry in the weblog dataset is represented as a tuple with fields (IP address, timestamp, URL, status code).

1. Count requests by IP address:

Map Stage:

For each entry in the weblog dataset, the mapper emits a key-value pair with the IP address as the key and the value of 1.

For each record in the input dataset: Emit intermediate key-value pairs based on the task:

Key: IP Address

Value: 1

Reduce Stage:

The reducer receives the key-value pairs from the mapper, where the key is an IP address and the values are a list of 1's representing the number of requests from that IP address. The reducer then sums up all the 1's for each IP address to get the total number of requests by that IP address.

For each key received in the intermediate key-value pairs: Initialize a variable `count` to 0.

For each value corresponding to the key : Increment `count` by the value.

Emit the final key-value pairs based on the task:

Key: IP Address

Value: count

2. Identify the top 10 IP addresses with the most requests:

Map Stage:

The mapper emits a key-value pair with the IP address as the key and the value of 1, similar to the previous task.

For each record in the input dataset: Emit intermediate key-value pairs based on the task:

Key: "TOP_10_IP"

Value: (IP Address, 1)

Reduce Stage:

The reducer receives the key-value pairs from the mapper, where the key is an IP address and the values are a list of 1's representing the number of requests from that IP address. The reducer keeps track of the top 10 IP addresses with the highest number of requests using a data structure (e.g., priority queue) to efficiently find the top 10.

For each key received in the intermediate key-value pairs: Initialize a variable `count` to 0.

For each value corresponding to the key : Add the value's count to the respective IP address in the dictionary

Emit the final key-value pairs based on the task:

Key: "TOP_10_IP"

Value: (IP Address, count)

3. Find the top 10 URLs that were requested the most:

Map Stage:

The mapper emits a key-value pair with the URL as the key and the value of 1, similar to the first task.

For each record in the input dataset: Emit intermediate key-value pairs based on the task:

Key: "TOP_10_URL"

Value: (URL, 1)

Reduce Stage:

The reducer receives the key-value pairs from the mapper, where the key is a URL and the values are a list of 1's representing the number of times that URL was requested. The reducer keeps track of the top 10 URLs with the highest request count using a data structure (e.g., priority queue).

For each key received in the intermediate key-value pairs: Initialize a variable `count` to 0.

For each value corresponding to the key : Increment `count` by the value.

Emit the final key-value pairs based on the task:

Key: "TOP_10_URL"

Value: (URL, count)

4. Discover the top 10 status codes returned:

Map Stage:

The mapper emits a key-value pair with the status code as the key and the value of 1.

For each record in the input dataset: Emit intermediate key-value pairs based on the task:

Key: "TOP_10_STATUS"

Value: (Status Code, 1)

Reduce Stage:

The reducer receives the key-value pairs from the mapper, where the key is a status code, and the values are a list of 1's representing the number of times that status code was returned. The reducer

keeps track of the top 10 status codes with the highest count using a data structure (e.g., priority queue).

For each key received in the intermediate key-value pairs: Initialize a variable `count` to 0.

For each value corresponding to the key : Increment `count` by the value.

Emit the final key-value pairs based on the task:

Key: "TOP_10_STATUS"

Value: (Status Code, count)