Part 1: Azure Deployment and Testing

1. After you have finished with the Azure tutorial, measure the time it takes for the Assignment 3 query to run on Azure. What do you notice?

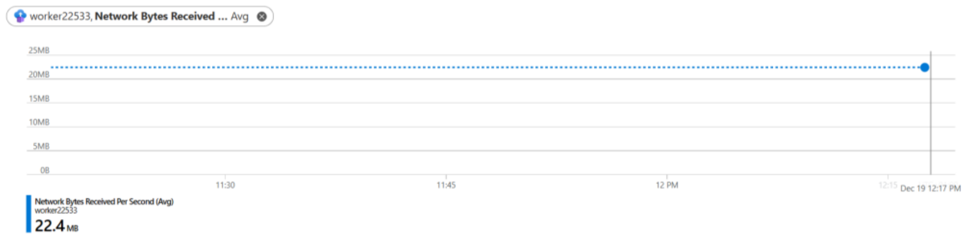
With only one worker, the task takes 17.199 seconds to be completed on Azure. It takes more time than running on a local machine.

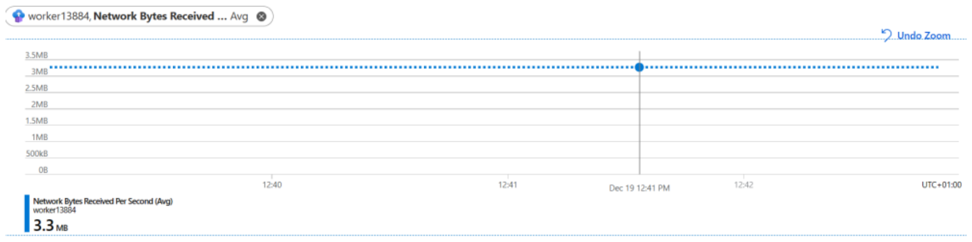
2. Go to the Azure monitoring panel for your containers: explain what is the bottleneck that increases query execution time. Include screenshots if needed.

The worker process finishes too quickly for Azure monitoring panel to capture multiple data points. However, it is obvious that the bottleneck is the network (see below).

3. Let's get faster: Pre-upload the data partitions and fileList inside Azure blob storage. Adapt your solution to read from there. What is the speedup you observe? How is it explained?

The number of network bytes received per second is 7 times bigger once we read the data from Azure Storage. This makes sense since container instances and storage instances are situated inside the same Azure region (West Europe). Here is a quote from Microsoft itself: "Each Azure region features datacenters deployed within a latency-defined perimeter. They are connected through a dedicated regional low-latency network. This design ensures that Azure services within any region offer the best possible performance and security."





Part 2: Managing Shared State - Design Questions

1. Give a brief description of your solution.

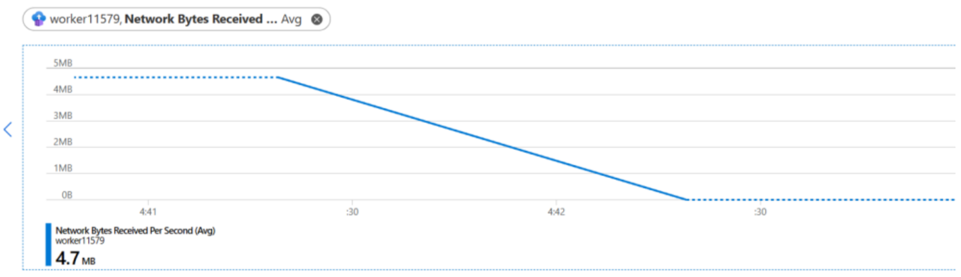
We developed a solution along the lines proposed in the tutorial session. As was the case with Assignment 3, data is partitioned into 100 files. Each file gets assigned to one of the workers who skims through it, extracts domains and counts their occurrence. As a result, each worker creates K files per partition. K is an adjustable parameter and represents the number of buckets domain set is partitioned into. Same domains get mapped to the same buckets, but each bucket can contain multiple domains. Each file is defined by partition and bucketID and consists of lines with domain name and count, separated by comma. During this phase, PARTITIONS_NO * K files are generated. Next, partial counts from all partitions need to be aggregated into a single count for each domain. Each bucket is assigned to one worker which should perform aggregation on its domains. Output of this phase are K files which contain key-value pairs representing their top 25 (or less) most visited domains. Finally, the coordinator goes through these K files to evaluate the global top 25 domains.

2. What was the query execution time in Azure? Include screenshots, logs if needed.

Using 4 workers with 4 buckets (subpartitions) on Azure took about 160 seconds. Here's the screenshot and a log for one of the workers:

LOG:

```
Worker 10.0.0.4 4242
Time: 44s
Task 1 load: 31
Task 2 load: 1
```



3. Which partitioning method did you choose for the calculation of the partial aggregates? Why?

We used hash partitioning. Given domain name, hash function returns unsigned integer. Bucket is determined by performing modulo K operation. All partial counts of each domain will be stored inside a single folder named aggr/bucket{bucketID}/. We chose this method because it is simple, and it spreads domains equally so that in the second phase each worker will have to process the same amount of data.

4. What number of subpartitions did you choose, and why? If you choose to create 100 subpartitions for each partition, is it a good choice? Is there a scalability limit?

The number of subpartitions (buckets) should be proportional to the number of workers we have. During the first phase, choice of number of subpartitions is not critical since the same amount of data is written. Yet, it is still better to have a smaller number of buckets since each bucket requires a separate write. The choice of subpartitions has a greater impact on the second phase. Here, it is obvious that the number of subpartitions should be at least as big as the number of workers. We argue that this is an optimal number as well. If there are more buckets than workers, we only get an overhead of extra communication with the coordinator (which includes worker asking coordinator for a new task, and coordinator assigning one to it). In addition, in the end, coordinator ends with more files to merge. This gives rise to a scalability limit. Creating 100 subpartitions is thus a bad choice unless the number of workers is of a similar scale. But here there is a good chance that the coordinator will become a bottleneck of the system since it would be cluttered with requests.

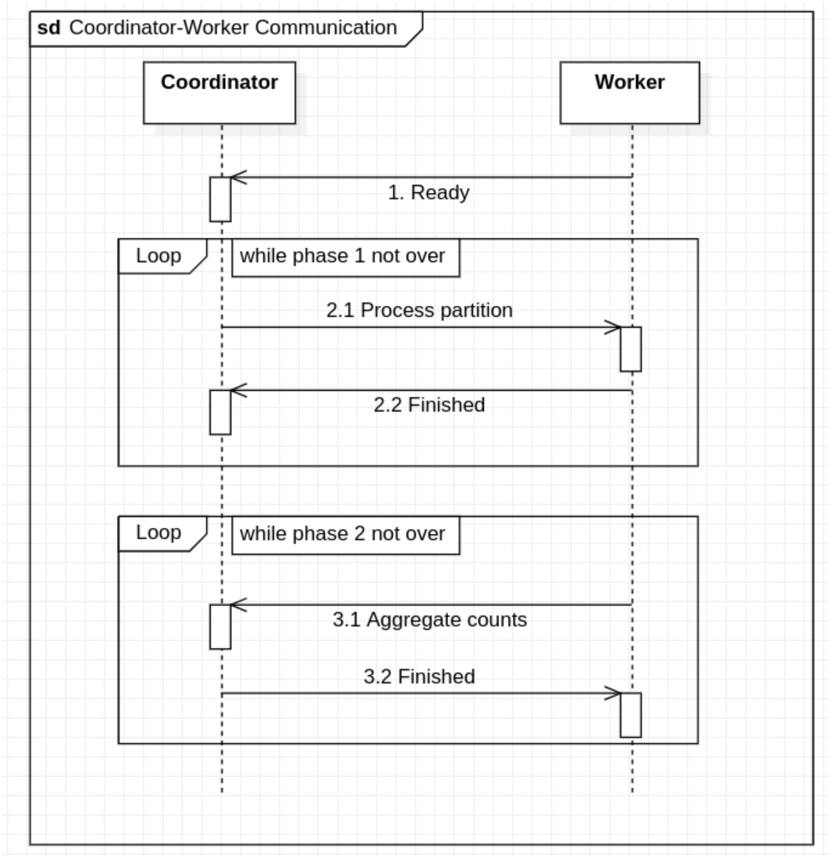
5. How does the coordinator know which partitions to send to the workers for the merging phase?

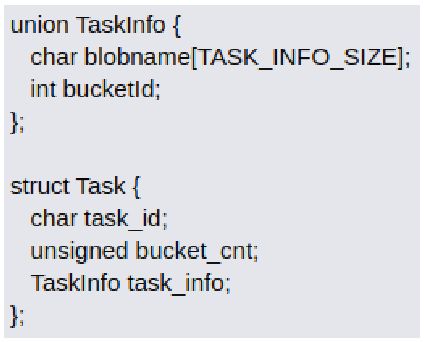
In the second, merging phase, coordinator sends workers IDs of the buckets that should be aggregated (K of them). There will be a folder named by bucket number. The worker goes through all the files in that folder and calculates the global count for the domains contained inside those files.

6. How do workers differentiate from the two tasks they have to perform (partial aggregation, merging subpartitions)?

For each task, the coordinator sends to workers a specific structure which includes information about the task (see below). These include a task ID which indicates to the worker what kind of task is expected to be done.

7. Give a brief sequence diagram to show how the coordinator and workers communicate. Add some details about the communication format.





```
union TaskInfo {
   char blobname[TASK_INFO_SIZE];
   int bucketId;
};

struct Task {
   char task_id;
   unsigned bucket_cnt;
   TaskInfo task_info;
};
```

Struct `Task` is a content of messages 2.1 and 3.1.

Each time coordinator assigns worker a task, it sends a `Task` structure over the network. First field `task_id` is used to distinguish between the tasks. The other two contain additional information which a worker needs to be able to successfully complete the task. Once it does, it sends back coordinator positive integer indicating everything went well.