A8: Distributed EC2 Sorting

Harshali Singh, Vishal Mehta, Akanksha Mishra, Saahil Singla

Design Decisions

1. **Partitioning the climate data:** We have used a python script *partition.py* to partition and distribute the climate data among all the running EC2 instances. We are sending equal chunks of complete input files to every instance. We do not consider partial files.
2. **Sorting Algorithm:** We have implemented *Parallel Sorting by Regular Sampling* (PSRS) sorting algortithm in this assignment. PSRS is a combination of a sequential sort, a load balancing phase, a data exchange and a parallel merge. It consists of four phases.
    - **Phase I: Sort Local Data** - Each processor is assigned a contiguous block of n/p items. The blocks assigned to different processors are disjoint. Each processor, in parallel, sorts their local block of items using sequential quicksort.
    - **Phase II: Find Pivots then Partition** - One designated processor gathers and sorts the local regular samples. p-1 pivots are selected from the sorted regular sample, at indices p + p; 2p + p; 3p+ p,… (p-1) + p . Each processor receives a copy of the pivots and forms p partitions from their sorted local blocks. A partition is contiguous internally and is disjoint from the other partitions.
    - **Phase III: Exchange Partitions** - In parallel, each processor i keeps the i th partition for itself and assigns the j th partition to the j th processor. For example, processor 1 receives partition 1 from all of the processors. Therefore, each processor keeps one partition, and reassigns p-1 partitions.
    - **Phase IV: Merge Partitions** - Each processor, in parallel, merges its p partitions into a single list that is disjoint from the merged lists of the other processors. The concatenation of all the lists is the final sorted list.
3. **Socket Program Implementation:** We send samples to Client.java which opens connection through sockets. Then we identify each clients' IP in variable 'myIP' which reads all the IPs from the 'iplist.txt' file. Thereafter,we send data from all the clients to one server in the form of 'String' and append keyword "splitForm" to identify that we are at phase:II of our program and we need to find pivots from this data.Then, we merge all the samples, sort them and find out Pivots. These pivots(number of processors - 1) are sent to all the clients. All clients(nodes) find out sections as per our phase-III and phase-IV. Now, each clients send their sections to the respective servers i.e all the First sections are sent to first server, all the second sections are sent to second server and so on.
Now, each server merge these sections and output a sorted list of these sections. Finally, with aws script we read all the sections which are obtained in output and merge them to get the final sorted list.
4. **Cluster Management:** We have developed 3 shell scripts to manage the EC2 instances which are as described below:
    - *start-cluster.sh* - This script starts N number of EC2 instances. N is specified in the argument. It parses the JSON response of *describe-instances* and retrieves the **Public-DNS** and **Public-IP** of all the running instances which writes to dns.txt & iplist.txt files respectively. The dns.txt file is read by *partition.py* file that partitions the input file to *dns* number of instances and iplist.txt file is read by all the instances to communicate among each other. It also transfers node.jar and iplist.txt to all the instances.
    - *sort.sh* - This script does ssh on all the running instances and executes the *node.jar* program by specifying input(climate data) and output.
    - *stop-cluster.sh* - This script fetches instance ids of all running instances and stops those instances.
    - *partition.py* - This script fetches the climate data from S3 and divides it into equal chunks of smaller input data set to be distributed to each instance. Each instance stores its part of the input data in an input folder locally which it uses to run the program. Hence, we pass 'input' as the input argument in sort.sh script.
    - *findmyip.sh* - This script is uploaded on each instance which finds out IP of that instance.
    - *sendtoS3.sh* - This script uploads the output from each instance to S3 bucket.

Challenges

1. We faced various challenges while designing and implementing the program:
    - *Transferring input data from S3 bucket to EC2 instances* - System asks for AWS credentials to perform this operation which we thought to automate. To accomplish this without having the system prompt for AWS credentials, we had to set IAM role to grant access to S3 bucket from EC2 instance in order to transfer and partition data from S3 to EC2.
    - *Communication between EC2 instances* - We were not able to establish communication between EC2 instances with socket programming initially. However, we were able to do so after many tweaks to the program and trying to run that among the instances.
    - *Network Issues* - We faced tons of network issues while connecting to EC2 instances via ssh.

- *Finding the correct AMI-ID to start the instances* - We did spend lot of time in searching the machine image and AMI ID to start the instances, in first few days. However, we figured out the problem and its solution after referring to AWS documentation.http://docs.aws.amazon.com/AWSEC2/latest/CommandLineReference/ec2-cli-launch-instance.html (http://docs.aws.amazon.com/AWSEC2/latest/CommandLineReference/ec2-cli-launch-instance.html)
- *Socket Programming* - We faced a lot of issues while implementing Network Manager in socket programming such as Connection Refused, Connection Reset, Broken Pipe. Earlier, we decided to adopt an approach in which the communication was carried out through ObjectInputStream and ObjectOutputStream which resulted in above mentioned errors. Later on, we managed to change the approach in which we passed Strings and hence we didn't have to deal with Input and Output Streams.

Top 10 values in data set

Following are the top 10 values in the data set sorted numerically on the "Dry Bulb Temp" field:

```
[FileData: [wban = 40604, yearMonthDay = 19960915, time = 1550, dryBulbTemp = 558.5],
FileData: [wban = 26627, yearMonthDay = 19980604, time = 1152, dryBulbTemp = 251.6],
FileData: [wban = 14847, yearMonthDay = 19960714, time = 1150, dryBulbTemp = 245.1],
FileData: [wban = 26442, yearMonthDay = 19980818, time = 1250, dryBulbTemp = 237.6],
FileData: [wban = 40604, yearMonthDay = 19971016, time = 1954, dryBulbTemp = 214.0],
FileData: [wban = 40504, yearMonthDay = 19971107, time = 0056, dryBulbTemp = 212.4],
FileData: [wban = 93115, yearMonthDay = 1991213, time = 0855, dryBulbTemp = 201.0],
FileData: [wban = 14780, yearMonthDay = 19991207, time = 1155, dryBulbTemp = 201.0],
FileData: [wban = 14611, yearMonthDay = 19991210, time = 1855, dryBulbTemp = 201.0],
FileData: [wban = 14780, yearMonthDay = 19991207, time = 1155, dryBulbTemp = 201.0]]
```

Study of Performance

We ran our code (node.jar) in 2 and 8 instances separately and found out their running time using time command as mentioned below:
Running the program on 2 instances: 39 min approx
Running the program on 8 instances: 15 min approx

Breakdown of Work

Vishal Mehta: Worked on building the scripts to start, stop the cluster, partition and sort the input data. Also, managed the instances on ec2.
Harshali Singh: Worked on developing the infrastructure to implement sample sort in Java. Also contributed in developing the scripts.
Saahil Singla: Worked on building the Network manager (Socket programming) and also contributed in testing the code on EC2 instances.
Akanksha Mishra: Worked and contributed in building the Network Manager and also spent lot of time in building the infrastructure for sample sort.

Team Contribution: Readme, Report, Running and testing code on EC2 instances, Resolving the challenges faced during implementation of entire assignment.

Conclusion

This problem helped us to understand the underlying shuffle and sort mechanism of MapReduce framework. This also helped us to understand how socket programming works in distributed environment. Moreover, it helped us to analyse the performance of running the sample sort on multiple clusters.

References

1. List of References:
    - Sample Sorting algorithm: http://csweb.cs.wfu.edu/bigiron/LittleFE-PSRS/build/html/PSRSalgorithm.html (http://csweb.cs.wfu.edu/bigiron/LittleFE-PSRS/build/html/PSRSalgorithm.html)
    - AWS Documentation:
        - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html#finding-an-ami-aws-cli (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html#finding-an-ami-aws-cli)
        - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/authorizing-access-to-an-instance.html (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/authorizing-access-to-an-instance.html)
        - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html#TroubleshootingInstancesConnecting
          (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html#TroubleshootingInstancesConnecting
        - http://docs.aws.amazon.com/codedeploy/latest/userguide/how-to-create-iam-instance-profile.html#getting-started-create-ec2-role-cli (http://docs.aws.amazon.com/codedeploy/latest/userguide/how-to-create-iam-instance-profile.html#getting-started-create-ec2-role-cli)