Distributed MapReduce:

Calculate K-Means on Large Clusters

HangYee Wong (r9i0b)
Kailun Hong (s2d0b)
Qiushan Zhao (n0g9)
Yukang (Ken) Zhang (u2z8)
Zixuan Yin (p4d0b)

Introduction

Inspired by a pressing demand of corporations for building a global market model without giving up their data privacy, we decide to build a distributed system based on the MapReduce programming model developed by Google and apply it to K-means clustering (Dean & Ghemawat, 2008). K-means clustering is an unsupervised machine learning method. The goal of this algorithm is to find clusters in the unclassified data, with the number of clusters represented by the variable K (Trevino, 2016). The algorithm proceeds by alternating between assigning each data point to one of K groups and calculate new means of the new clusters. The algorithm has converged when the assignments no longer change.

In our project, we treat a company providing data and computation power as a resource node. The system trains a global model using data from all resource nodes in the network. Upon user program request, the system classifies the user provided data into k categories using that model. The system is expected to gracefully survive node failures, as well as incorporate various optimizations such as load balancing and backup tasks in order to further boost the performance.

Nodes' Types

There are three types of nodes in our system:

- **User Programs:** Send request to master for classifying a user provided dataset.
- **Master Server:** Monitor the states of Resource Nodes in the network and assign tasks for them.
- **Resource Nodes:** Join the network with their own training dataset that will contribute to the training of global model, and work on map and/or reduce tasks assigned by the master while connected.

Key Assumptions

- We assume that the master server is always online, does not fail, and does not misbehave.
- We assume that there is a process to check nodes' credibility in the real world (See in the Security and Data Privacy section).

Azure Usage

- Azure VMs:
    - The entire system is deployed on Azure VMs
- Azure Cosmos DB:
    - Our system uses Azure Cosmos DB to store the classify data and result. The benefit of using a globally distributed database is that our network can take advantage of the its locations and availability to reduce our network load and latency because the heavy data transferring happens on the Azure distributed database system network instead of our system.

Implementations

We describe the implementations of the three types of nodes and communication topology in this section.

- **Client**
  The client sends requests made up of k (desired number of clusters) and the credentials to access the data to be classified to the master server (Users are expected to upload the data onto the database before running the client program).

- **Master Server**
  The master server is responsible for accepting and managing requests from clients. If multiple clients are connected to the master server and send many requests to it, the master server queues all the requests and process it based on the queue.

  When the K supplied by a client does not match the K value of the current global model, the master server will retrain the global model through iterations of map and reduce phases. During the entire training phase, master server runs K-Means with 50 randomly initialized means, each requires approximately 15 iterations for the model to converge. Multiple initializations are required in order to find the model that produces the highest accuracy because K-Means algorithm is sensitive to initializations.

Training Phase
During the map phase, the master server contacts all registered resource nodes and informs them of the number of clusters K, and randomly initialized means or existing means depending on if this is the first iteration. The details of the communication can be found on the “Map Phase” of the ShiViz diagram on the last two pages. Resource nodes are authorized against a list of public keys stored on the server upon registration. At this phase, since nodes failures only slightly affect the precision of the trained model, failed tasks are reassigned.

After all resource nodes completed, master server starts reduce phase of the current iteration. The details of the communication can be found on the “Reduce Phase” of the ShiViz diagram. It splits the tasks evenly and sends each resource node the tasks they should work on and a list of IP addresses of the nodes to contact for retrieving the corresponding data. Failed tasks or unfinished tasks of failed nodes are reassigned to idle nodes.

Classifying Phase
Once global model is trained, the server splits the classifying request from the client into tasks and assigns them to resource nodes. The details of the communication can be found on the “Classify Phase” of the ShiViz diagram on the last two pages. For this phase, there is a load balancing mechanism in place, so unlike the previous phases, tasks are assigned based on resource nodes’ performance instead of splitting them evenly. The performance of a resource node is defined by the amount of time taken for the resource node to complete the classifying request dividing by the number of tasks assigning to it during the last classifying phase. The nodes are splitted to two tiers depending on their performance. The faster 50% of the resource nodes is assigned twice amount of the tasks of the slower ones. Similar to the training reduce phase, failed tasks or unfinished tasks of failed nodes will be reassigned to idle nodes.

As we recognize a possible scenario where some nodes may take unusually long to process the tasks and slow down the overall process, hence significantly prolonging the overall execution time, we use backup tasks to preempt these stragglers. In the training reduce phase and the classifying phase, after more than 80% of the total tasks are complete, the master server assigns the remaining 20% of the total tasks to the nodes that have already completed their own tasks.

- **Resource Node**
  Resource nodes follow instructions of the master server to execute different functions.

  Training Phase
  During the training map phase,the server behaves the same as we described before. Each resource node reads their local train dataset and applies the map function for training phase to each data entry.
  After applying map function, the resource node obtains the three K key-value pairs, which are sum (the sum of the data entries that belongs to the cluster), count (the number of the data entries), local k-mean error (sum of the distances between each data entries and the mean). Upon completion, resource nodes store key-value pairs to the local disk.

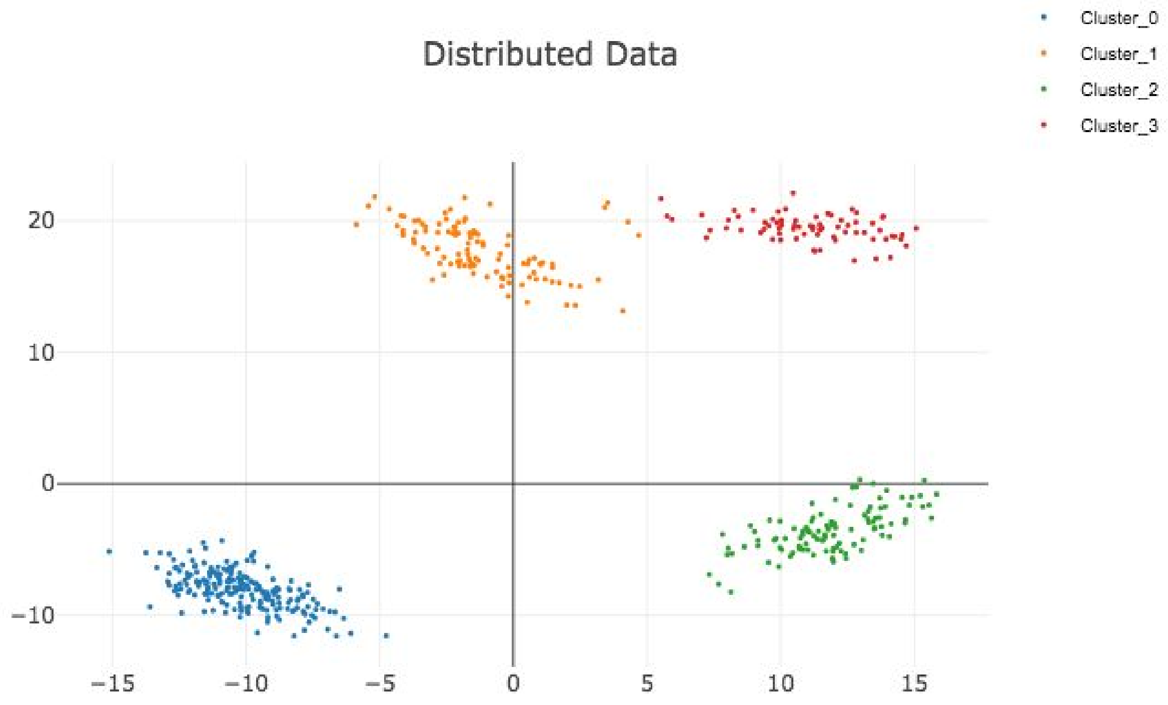
  During the training reduce phase, each resource node receives a list of keys from the master server. The assigned keys are the cluster numbers that the resource node is responsible for calculating the global means. Then, it contacts other resource nodes listed by the server and retrieves local sum, count, and error that are computed during the training map phase by them.

  After gathering all the values, the resource node computes the global sum by adding up all the local sums, as well as the global count by adding up all the local count. Then, it computes the new mean by dividing the global sum by the global count, and calculates the new k-mean error by summing up all the local K-mean errors. After calculation, it returns the new global mean and k-mean error to the master server.

  Classifying Phase
  Each resource node receives the indexes of data entries assigned, global means, and database credentials from the Master server. Then, it retrieves the data entries from the database and applies the classifying function to each data entry. The classifying function takes one data entry and the global means as inputs, computes the distance between the data entry and the global means, classifies the data entry to the cluster with the nearest mean, and then uploads the result to the database.

Evaluations

We prove the correctness of our system by comparing the means of our global model to the result of running Scikit Learn ‘s K-Means library, which is a widely used machine learning package implemented in python. We also visualize our distributed model by plotting the categorized result which is expected to have K properly separated clusters.



The graph above is the example of classification result with K = 4

Limitations

Backup tasks introduce the possibility that one task might be worked on by multiple workers. Ideally, once a task is finished, workers working on that task should be notified to stop. However, our current implementation does not terminate duplicated tasks’ processes but only adopts the first result returned and ignore the rest. It is potentially a waste of processing power but for the benefits of less I/Os, because this way we are able to flexibly combine tasks and send them in batches.

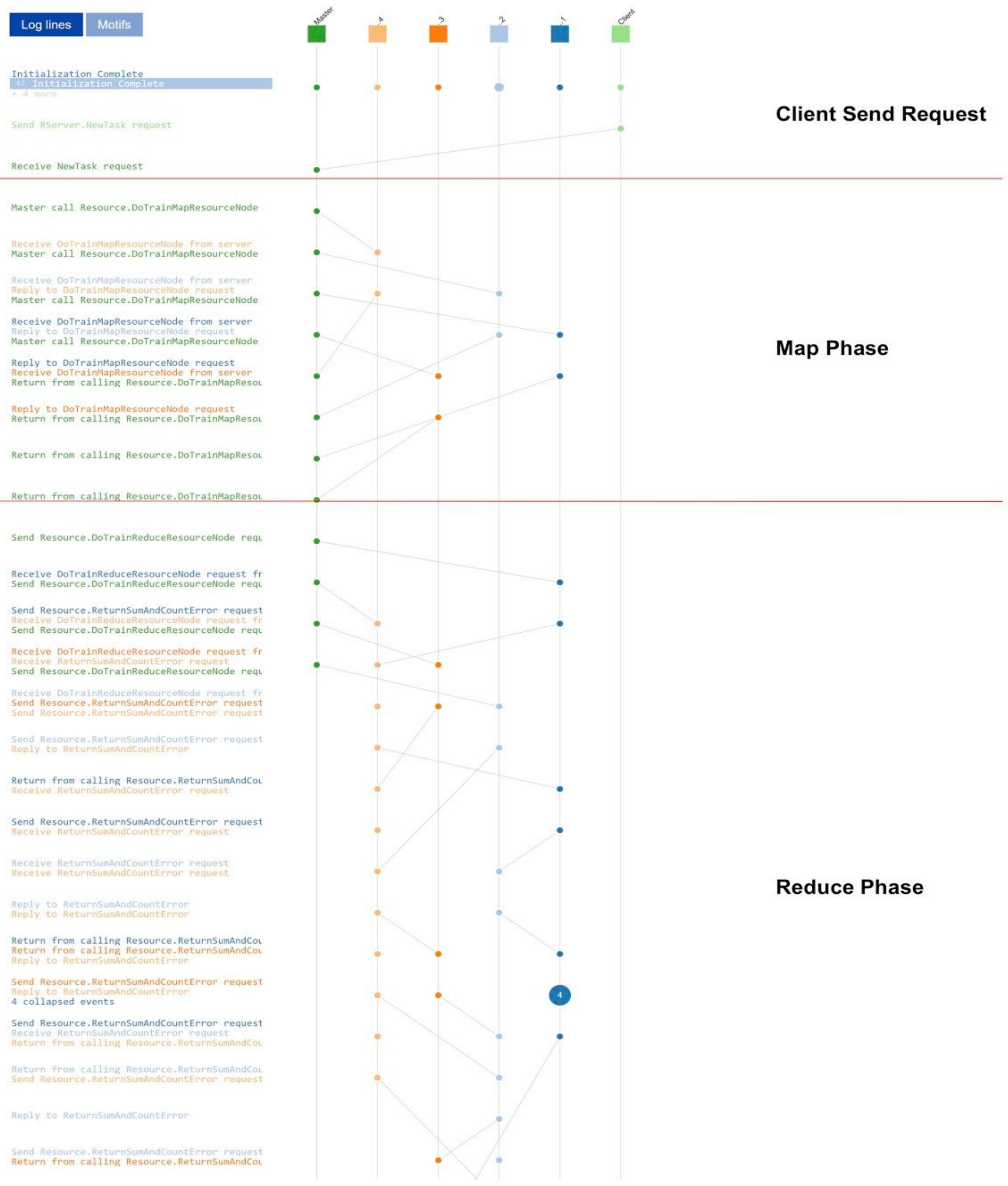
The lack of authentication against the server on the resource node’s side exposes a vulnerability of the system. Malicious nodes could potentially disguise as the server and send tasks to the resource nodes. However, since in the map phase of training and classifying is always done locally and no other nodes can access raw data, the ramification of potential security hazard is mitigated. With that being said, it is still possible for the malicious node to occupy all resources nodes with random tasks.

The system’s efficiency will be undermined if the resource node does not have substantial amount of data to contribute and/or the network latency among the nodes are high. In this case, the improvement in the precision of trained model and increase in the aggregated computation power might not outweigh the communication cost.

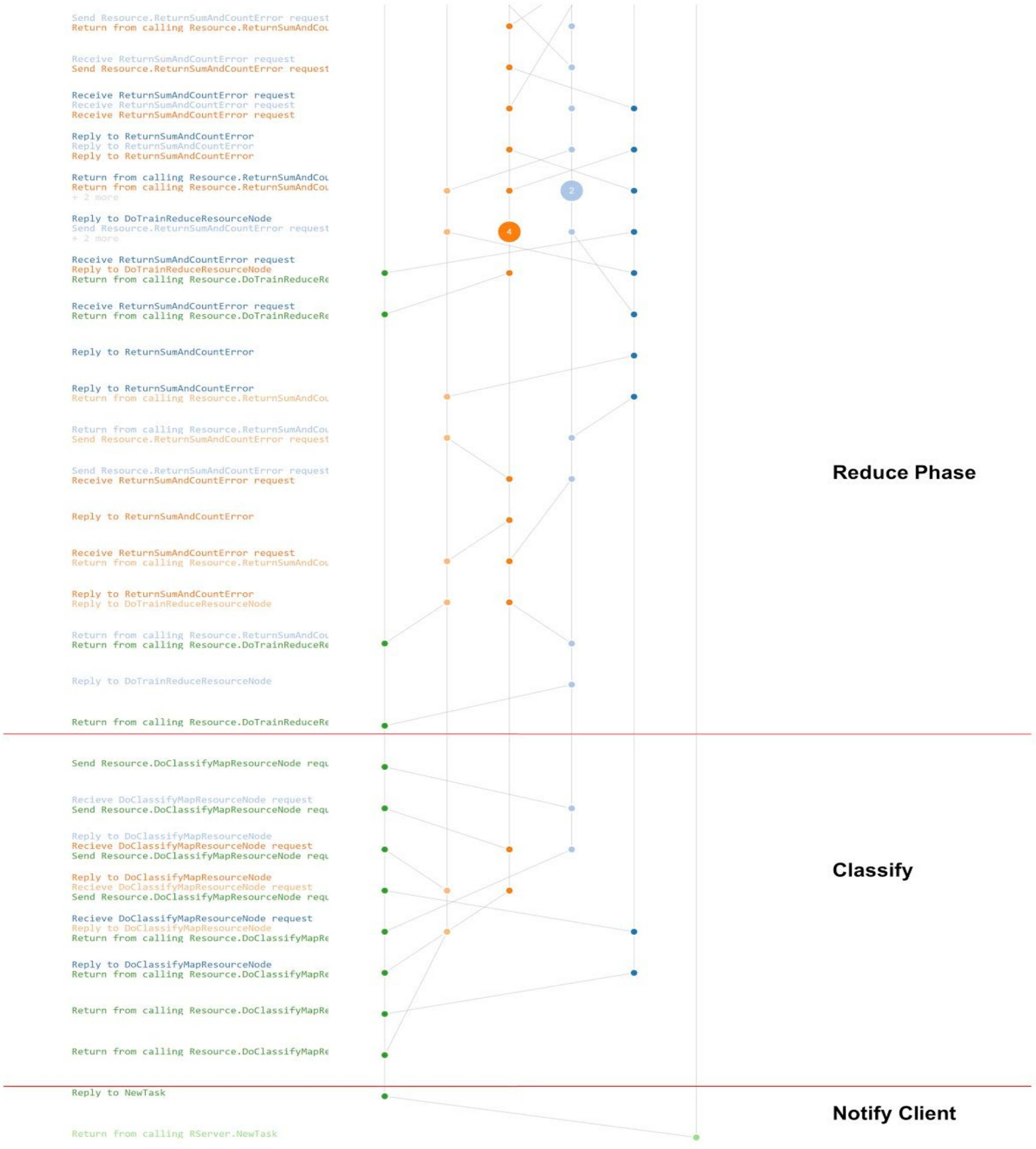
Completeness of the Project

We deem our project as completely finished. The normal operation, survival situation, and nodes joining and leaving the system are properly working at the time when we write the report. Also, we completed the bonus EC1: Add support for GoVector and ShiViz to your system. We do not foresee a significant amount of changes from now till our demo date. However, we may need to fix some small bugs as we test the system.

The Shiviz diagram is shown below:



```
Initialization Complete
 47 Initialization Complete
+ 4 more

Send RServer.NewTask request

Receive NewTask request

Master call Resource.DoTrainMapResourceNode

Receive DoTrainMapResourceNode from server
Master call Resource.DoTrainMapResourceNode

Receive DoTrainMapResourceNode from server
Reply to DoTrainMapResourceNode request
Master call Resource.DoTrainMapResourceNode

Receive DoTrainMapResourceNode from server
Reply to DoTrainMapResourceNode request
Master call Resource.DoTrainMapResourceNode

Reply to DoTrainMapResourceNode request
Receive DoTrainMapResourceNode from server
Return from calling Resource.DoTrainMapResou

Reply to DoTrainMapResourceNode request
Return from calling Resource.DoTrainMapResou

Return from calling Resource.DoTrainMapResou

Return from calling Resource.DoTrainMapResou

Send Resource.DoTrainReduceResourceNode requ

Receive DoTrainReduceResourceNode request fr
Send Resource.DoTrainReduceResourceNode requ

Send Resource.ReturnSumAndCountError request
Receive DoTrainReduceResourceNode request fr
Send Resource.DoTrainReduceResourceNode requ

Receive DoTrainReduceResourceNode request fr
Receive ReturnSumAndCountError request
Send Resource.DoTrainReduceResourceNode requ

Receive DoTrainReduceResourceNode request fr
Send Resource.ReturnSumAndCountError request
Send Resource.ReturnSumAndCountError request

Send Resource.ReturnSumAndCountError request
Reply to ReturnSumAndCountError

Return from calling Resource.ReturnSumAndCou
Receive ReturnSumAndCountError request

Send Resource.ReturnSumAndCountError request
Receive ReturnSumAndCountError request

Receive ReturnSumAndCountError request
Receive ReturnSumAndCountError request

Reply to ReturnSumAndCountError
Reply to ReturnSumAndCountError

Return from calling Resource.ReturnSumAndCou
Return from calling Resource.ReturnSumAndCou
Reply to ReturnSumAndCountError

Send Resource.ReturnSumAndCountError request
Reply to ReturnSumAndCountError
4 collapsed events

Send Resource.ReturnSumAndCountError request
Receive ReturnSumAndCountError request
Return from calling Resource.ReturnSumAndCou

Return from calling Resource.ReturnSumAndCou
Send Resource.ReturnSumAndCountError request

Reply to ReturnSumAndCountError

Send Resource.ReturnSumAndCountError request
Return from calling Resource.ReturnSumAndCou
```



```
Send Resource.ReturnSumAndCountError request
Return from calling Resource.ReturnSumAndCou

Receive ReturnSumAndCountError request
Send Resource.ReturnSumAndCountError request

Receive ReturnSumAndCountError request
Receive ReturnSumAndCountError request
Receive ReturnSumAndCountError request

Reply to ReturnSumAndCountError
Reply to ReturnSumAndCountError
Reply to ReturnSumAndCountError

Return from calling Resource.ReturnSumAndCou
Return from calling Resource.ReturnSumAndCou
+ 2 more

Reply to DoTrainReduceResourceNode
Send Resource.ReturnSumAndCountError request
+ 2 more

Receive ReturnSumAndCountError request
Reply to DoTrainReduceResourceNode
Return from calling Resource.DoTrainReduceRe

Receive ReturnSumAndCountError request
Return from calling Resource.DoTrainReduceRe

Reply to ReturnSumAndCountError

Reply to ReturnSumAndCountError
Return from calling Resource.ReturnSumAndCou

Return from calling Resource.ReturnSumAndCou
Send Resource.ReturnSumAndCountError request

Send Resource.ReturnSumAndCountError request
Receive ReturnSumAndCountError request

Reply to ReturnSumAndCountError

Receive ReturnSumAndCountError request
Return from calling Resource.ReturnSumAndCou

Reply to ReturnSumAndCountError
Reply to DoTrainReduceResourceNode

Return from calling Resource.ReturnSumAndCou
Return from calling Resource.DoTrainReduceRe

Reply to DoTrainReduceResourceNode

Return from calling Resource.DoTrainReduceRe

Send Resource.DoClassifyMapResourceNode requ

Recieve DoClassifyMapResourceNode request
Send Resource.DoClassifyMapResourceNode requ

Reply to DoClassifyMapResourceNode
Recieve DoClassifyMapResourceNode request
Send Resource.DoClassifyMapResourceNode requ

Reply to DoClassifyMapResourceNode
Recieve DoClassifyMapResourceNode request
Send Resource.DoClassifyMapResourceNode requ

Recieve DoClassifyMapResourceNode request
Reply to DoClassifyMapResourceNode
Return from calling Resource.DoClassifyMapRe

Reply to DoClassifyMapResourceNode
Return from calling Resource.DoClassifyMapRe

Return from calling Resource.DoClassifyMapRe

Return from calling Resource.DoClassifyMapRe

Reply to NewTask

Return from calling RServer.NewTask
```

Sources

Dean, J., & Ghemawat, S. (2008). *MapReduce: Simplified data processing on large clusters*. New York: ACM.10.1145/1327452.1327492

Trevino, A. (2016). Introduction to K-means Clustering. Retrieved March 08, 2018, from https://www.datascience.com/blog/k-means-clustering