CP3106 Project Report

Ray-supported high-performance distributed clustering algorithm

By

Wang Yifan

Department of Computer Science

School of Computing

National University of Singapore

AY2019/2020, Semester II

CP3106 Project Report

Ray-supported high-performance distributed clustering algorithm

By

Wang Yifan

Department of Computer Science

School of Computing

National University of Singapore

AY2019/2020, Semester II

Advisor: Assoc Prof Richard T. B. Ma

Deliverables:

Report: 1 Volume
Software: 1 USB

Abstract

The distributed system has become increasingly popular over the past few years due to the growth of AI applications. To meet the new and demanding systems requirement, a brand-new distributed computing framework – Ray is proposed in late 2017 to address this problem. With a unified interface that can express both task-parallel and actor-based computations, many users can use Ray API to build their own distributed applications. With two's years' growth, there are many libraries like TUNE, RLlib, and SGD for machine learning. And there are hundreds of projects using TensorFlow, Keras, and PyTorch on top of Ray to reach high performance distributed training. However, as for clustering algorithms, the classic AI applications, do not support well in the Ray System. Unlike Spark with complete highly efficient clustering wrappers, Ray has no clustering libraries. In this report, I build a whole MapReduce pipeline on top of Ray and implement K-Means – a classic clustering - by using Ray API. Furthermore, I modify the MapReduce according to the property of Ray, utilizing various optimization methods like K-Means++ and Elkan K-Means to reach high performance. Finally, I build a distributed clustering system that can be deployed on a cluster to do large-scale data clustering. To make a comparison, I also use Spark clustering libraries to see whether Ray can take Spark's place with better effects or Ray still has room for growth.

Subject Descriptors:

C.2.1 Network Architecture and Design
C.2.4 Distributed Systems
C.3 Special-Purpose and Application-Based Systems
C.4 Performance of Systems
F1.1 Models of Computation

Keywords:

clustering algorithm, distributed applications, MapReduce

Implementation Software and Hardware:

Python 3.7.6, Ray 0.8.4, NumPy 1.18.2, Pandas 1.0.1, sklearn 0.22.1, joblib 0.14.1, pyspark 2.4.5, Cython 0.29.15

Processor	2.4 GHz 8-CoreIntel Core i9
Memory	64 GB 2667 MHz DDR4
Graphics	AMD Radeon Pro 5500M 8 GB Intel UHD Graphics 630 1536 MB

Acknowledgement

First and foremost, I would like to show my deepest gratitude to my advisor, Prof Richard T. B. Ma, a respectable, responsible and resourceful scholar, who has provided me with valuable advice in every stage of the CP3106. With his instructions, I learn about plenty of knowledge about Ray, Spark and some principles of distributed systems.

I shall extend my thanks to Yuan Huang, my teammate of CP3106, for all his kindness and help. When I have some questions about Ray, he always provides some suggestions and patiently instructs me.

I would also like to thank all workers and administrators in the Ray community. With their help and encouragement, I finally stick to the distributed clustering algorithm on top of Ray and give up the idea of using complete libraries for machine learning.

Last but not least, I' d like to thank all my friends, especially my girlfriend, for their encouragement and support.

Table of Contents

Title ..... **Error! Bookmark not defined.**
Abstract iii
Acknowledgement iv
Table of Contents v
1 Introduction 1
1.1 Distributed System 1
1.2 Clustering 1
1.3 Ray 1
2 Study of Ray 3
2.1 Features 3
2.2 Programming Model 3
2.3 Framework 3
3 System Design Criteria 5
3.1 Feasibility and Critical factors 5
3.2 Software Requirement 5
3.3 Program Design 5
4 Implementation 9
4.1 Data Structure 9
4.2 Illustration of the Network Model 9
4.3 Algorithm Used 11
5 Testing Methodology 14
5.1 Parameter 14
5.2 Environment 14
5.3 Result 14
6 Conclusions 17
6.1 Summary 17
6.2 Limitations 17
6.3 Recommendations for Further Work 18
References v
Appendix A - Theory about Spark K-Means vi

1 Introduction

1.1 Distributed System

Nowadays, with the development of computer technology, the scale of computing tasks is gradually increasing. In the past, we will parallelize the tasks by using threads and processes in the local machine to improve the performance and efficiency of calculations. However, the method above is not competent to finish the task when there is a large amount of calculation. For example, as for reinforcement learning, it is necessary to simulate plenty of environments for testing at the same time, which includes millions of changing variables. Even if multiple threads are implemented on a single machine, it will not perform a highly efficient calculation and training. At this time, you need to use the distributed processing of computing tasks. In other words, distributed processing and computation is also a type of parallelization. It divides tasks and handles over multiple machines to process data in parallel and summarize the data to achieve high-performance computing competence that cannot be satisfied by a single machine.

With the emerging distributed computation, there are tons of data generated and should be stored during the period of data processing. Thus, the sharp increase in data size is also the root cause of the formation of distributed systems. At present, there are two ways to solve this problem. One is to expand vertically, which installs dozens of hard drives on one machine. The other is to expand horizontally, using multiple machines to store data dispersedly. The former will not be able to efficiently store large amounts of data because of the upper bound of the capacity of a machine. In the face of massive data currently, using multiple scattered machines to store data becomes essential. At the same time, with distributed computation, if multiple nodes read and store data to the same location concurrently, the efficiency will be affected by the performance of one single machine, such as network bandwidth, disk I / O, etc., which also forces the data to be decentralized storage to achieve high-efficiency data transmission operations.

1.2 Clustering

Around 2015, a large number of AI computing tasks have emerged. Clustering, as a traditional classification problem, exists in all aspects of human society, which includes K-Means, DBSCAN, etc. At the same time, due to the gradual increase in data scale, the clustering algorithm plays an important role in looking for principles for human society and exploring the nature of things. However, due to the increase in data size, a single machine cannot meet its computing requirements even if multiple threads are performed, so in recent years more and more people have begun to study distributed clustering algorithms. How to make clustering tasks applied to distributed systems becomes a hot topic. Since this problem involves data partition, result integration, and feedback updates, many clustering algorithms need to be redefined in detail. Therefore, in today's society, clustering algorithms based on distributed systems for massive data must be the direction for future research.

1.3 Ray

In recent years, as a dominant computing task, reinforcement learning and artificial intelligence training like autonomous driving performed not well on existing distributed frameworks such as MapReduce. At the same time, the essential mission of many AI tasks is to optimize the behavior

of AI in a large-scale simulation environment, which is completely different from Spark's original design intention. Secondly, in terms of resource scheduling, Spark only needs to start a calculation graph of a certain size for static calculation according to the availability of resources. Nevertheless, as for large-scale simulation, these calculation entities will generate a large number of intermediate states during the calculation. This huge behavior of immediate modification of the calculation graph is difficult to effectively support in Spark.

Ray, as an emerging distributed computing framework proposed in late 2017, is mainly oriented to large-scale machine learning and reinforcement learning. With almost two years' effort, Ray has completed the optimization of distributed computing for many machine learning tasks. Currently, Ray has three important libraries for machine learning. The first one is TUNE, scalable hyperparameter tuning, which integrates with various optimization libraries for any machine learning framework such as TensorFlow, PyTorch, Keras, etc. The second one is RLlib, scalable reinforcement learning, which offers both high scalability and a unified API for a variety of applications. The last one is RAY SGD, distributed training wrappers, which provides wrappers around PyTorch and TensorFlow for further data parallel training.

Although the intention of Ray mentioned in the paper is that Ray can address the system requirements AI applications impose, both in terms of performance and flexibility, many people including creators of Ray began to change the purpose of Ray, making it more general and suitable for all distributed task. As we all know, with two years' growth, Ray is still in the development stage. As a distributed framework that aims to replace Spark as a brand-new high-performance distributed system, there are many applications such as clustering that do not have good advanced API support. Therefore, specific API and optimization for special AI applications should be created for developers to build their own distributed tasks.

2 Study of Ray

2.1 Features

1. Call function distributed and asynchronously

In the Ray System, people can call function remotely, which means that the execution of a function can be assigned to not only a local machine but also a remote computer. This allows people to divide the huge task into small parts and assign them to many remote and distributed machines. Meanwhile, all functions in the Ray system are asynchronous. Once the function is called, it will return immediately even though the operation task has not been completed. With the benefit of this feature, all remote functions are non-blocking and the execution process cannot be affected by each other.

2. Memory Management

Unlike Spark using RDD, a special format to store and share data, Ray uses Object Store to store them and utilize Apache Arrow to realize high-performance shared memory storage. With this benefit, many tasks in the same node can share the memory, which can improve the efficiency of reading and storing data between tasks. Secondly, using object id and its value is more readable and explainable about data in this distributed system and it is more convenient for people to fetch data from different nodes.

2.2 Programming Model

There are two types of programming model in the Ray that user can use to build their own distributed application.

1. Task

In general, Task is the special function of Ray version. When program calls function in the function, Ray will assign it to another machine to execute remotely. Thus, this remote function is called Task. When one Task is invoked, it will return its result immediately, which is call future even though the execution of Task is not completed. The future, in this scenario, contains one or a list of Object id of its results. The remote function in the Ray is as stateless as a traditional function, which just processes the input and returns the result without no intermediate state.

2. Actor

Generally, the Actor is the special class of Ray version. Compared with class in Python, when an Actor is initialized, Ray will see it as a separate process and execute it remotely. All the methods in the Actor will be treated as Task. However, the most different part between these methods in an Actor and Task is that they execute on a stateful worker- Actor, which includes global states and can be modified by various methods within Ray.

2.3 Framework

The main feature of framework of Ray is master-worker, which can be shown as Fig. 2-1.

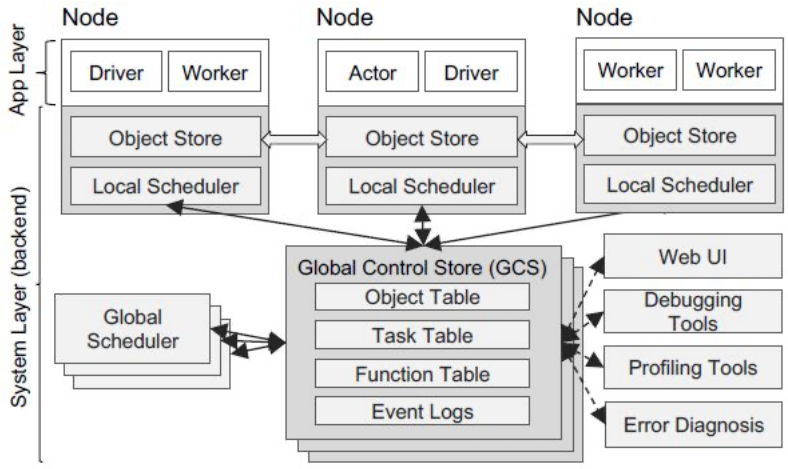

Figure 2-1

1. Node

There are three types of processes in the Node: Driver, Worker, and Actor. The Driver is responsible for task scheduling and every task will be executed on each worker. And the third type-Actor has control over the execution of the actor. In the meantime, to avoid overloading the global scheduler, a local scheduler on each node will schedule tasks locally. If one node overload, which means that tasks on this node exceed the requirements, this node will forward some tasks to the global scheduler to invoke it remotely.

2. Object Store

As mentioned above, each node has its shared object store which allows all workers on the same node to share data and memory storage. For each data, it will be stored as <key, value> and all workers can fetch them by using key and API provide by Ray. Most importantly, Ray has its own Object Manager called Raylet on every node, taking responsibility for managing object store and share data between different nodes.

3. Global Control Store

The Global Control Store, which is Redis-based, store much essential information about the whole distributed system. The first one is Object Table, as the name suggests, store the information of each store, such as object id and which node this data is stored. Secondly, Task Table shows the details of each task. Naturally, the task is special version of the process, so the Task Table will store information such as PID, which node executes this task, etc. Another purpose of the Global Control Store is that it will serve as a bridge between application Layer including several nodes and system layer like Global Scheduler and Debugging Tools.

3 System Design Criteria

3.1 Feasibility and Critical factors

Ray has its own flexible and simple API for users to easily build their own distributed system. Because of its simple wrapper and readable data format, it is feasible for me to build a Ray-supported distributed clustering algorithm and use various methods for optimization.

1. Simple Wrapper

In the document of Ray, developers set many simple demos for users to learn how to wrapper their python code to be distributed. As for remote function, people just use @ray.remote, which is decorator of Python to make an original function remote. This simple decorator can also be applied for Actor, which is special class of Ray version. With just one decorator, people can create their own remote function and Actor and use function.remote() to invoke them.

2. Data Format

As we all know, Ray uses Object Store to store its data, which contains object id and its value. With Ray's API, people can use ray.get() and ray.pull() to fetch and store data in the object store. Meanwhile, Ray has worked on the NumPy for a while, optimizing its performance on this distributed system. Therefore, convenient Ray API and easily handled NumPy, many people can transfer their application into Ray version.

3.2 Software Requirement

The Ray is built on Python, with C++ backend. Thus, the software requirement is only the proper version of Python, Ray, and other necessary packages. Here are some packages and their version utilized in my CP3106.

Python version: 3.7.6
Ray version: 0.8.4
NumPy version: 1.18.2, used for data structure and processing.
Pandas version: 1.0.1, used for reading and storing data.
sklearn version: 0.22.1, used for clustering algorithm in sklearn version
joblib version: 0.14.1, used for implementing sklearn in parallel
pyspark version: 2.4.5, used for clustering algorithm in Spark version
Cython version: 0.29.15, used for optimization when running python code

3.3 Program Design

In this report, I implement a whole MapReduce framework based on Ray API and test classic clustering algorithm: K-means on top of Ray built MapReduce. Then various methods are used to make the K-means a distributed clustering algorithm. To calculate more efficiently, I utilize many optimization algorithms on K-means. All contents are discussed in Program Design and Implementation parts

1. MapReduce

Map/Reduce is one of the Parallel computing models, whose pipeline can be shown as Fig. 3-1

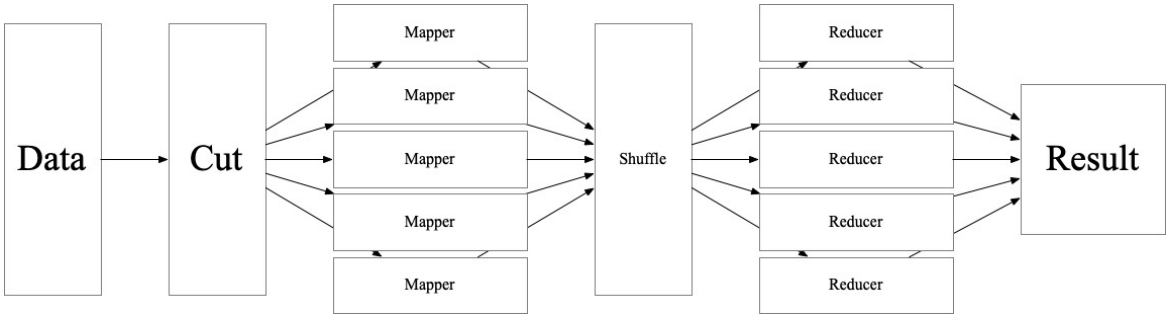

Figure 3-1

As we can see, firstly, data will be preprocessed into small parts, which is called cut. After getting many parts of data with key and value format, each part will be flowed into the mapper operator to get the intermediate state of keys and values. When doing shuffle, the same key of data from all mappers will flow into the same reducer. Each reducer will get one key with many different values. All reducers will do are that they will aggregate the result and output their results. Finally, we get a set including all results from different reducers.

There are two main parts of pipeline, Mapper and Reducer, which are responsible for plenty of calculation. Setting a correct calculation process and optimizing it are key parts of all pipeline.

2. K-means

The K-Means algorithm is an unsupervised clustering algorithm. It is relatively simple to implement and has a good clustering result, so it is widely used for large and continuous data. Many various optimization algorithms on top of Ray like K-means++, elkan K-Means algorithm, etc will be discussed in the Implementation part.
The K-means algorithm shows below.
Given data set D= $\{x_1, x_2, ...x_m\}$, Cluster number k and maximum number of iterations N as input.
Output is cluster partition C:

$$C = \{C_1, C_2, \dots C_k\}$$

1) Randomly select k samples from the data set D as the initial k centroid vectors $\{\mu_1, \mu_2, \dots \mu_k\}$
2) for n from 1 to N:
   a) Initialize cluster partition C to C_t :

$$C_t = \emptyset \ (\mathrm{t} = 1,2 \dots \mathrm{k})$$

   b) for i from 1 to m:
      calculate the distance between x_j and each $\mu_j (j = 1,2 \dots k)$:

$$d_{ij} = \left|\left|x_i - \mu_j\right|\right|_2^2$$

      assign the category λ_i corresponding to the smallest distance d_{ij} to the x_i and update

the C_{λ_i} :

$$C_{\lambda_i} = C_{\lambda_i} \cup \{x_i\}$$

c)for j from 1 to k:

recalculate new centroids for all sample points in C_j :

$$\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$$

d) if k centroid vectors are not update, output the cluster partition C:

$$C = \{C_1, C_2, \ldots C_k\}$$

An example shows below to illustrate to process of K-means.

1. Choose K samples randomly from dataset as initial centroid vectors (Fig. 3-2)

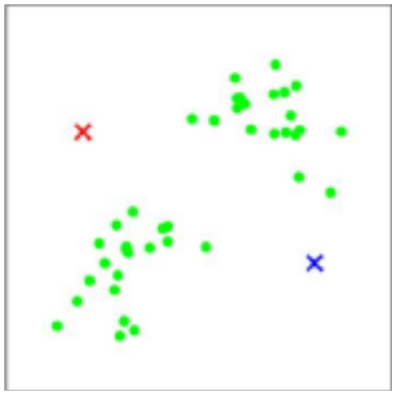

Figure 3-2

2. Assign each data point to the center point which is closest to it (Fig. 3-3)

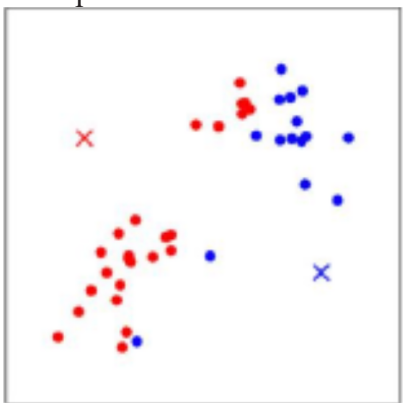

Figure 3-3

3. In each cluster, set average of all points as new center point (Fig. 3-4)

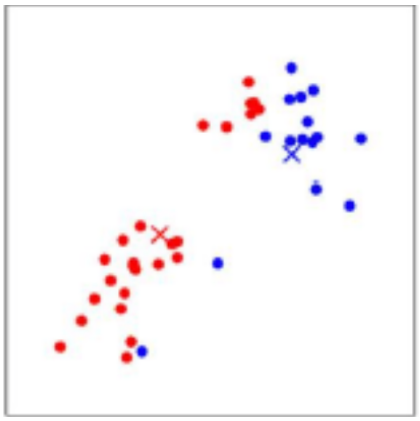

Figure 3-4

4. Assign each data point again to the new center point (Fig. 3-5)

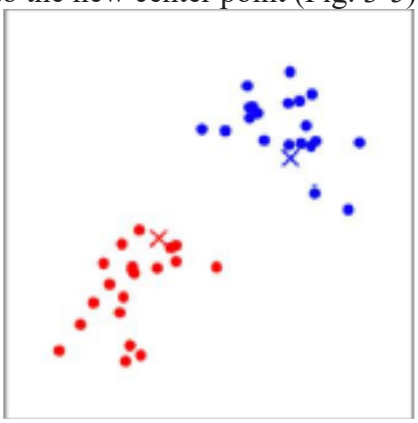

Figure 3-5

Loop the step 3 & 4 until all points in dataset don't be reassigned or reach max iterations (Fig. 3-6)

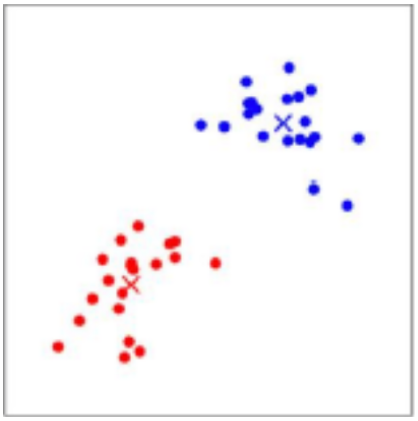

Figure 3-6

4 Implementation

4.1 Data Structure

To reach high-performance clustering algorithm, I use NumPy to implement all data type.
1. Dataset

To make a demo, I use Gowalla as the dataset, which is a location-based social networking website where users share their locations by checking-in. This dataset includes user, check-in time, latitude, longitude, and location id to represent all information. I only use latitude and longitude to make clustering because these two factors can be shown on the figure easily. After filtering out two dimensions (features) of the dataset, I transfer pandas to Numpy about the dataset. Each point including latitude and longitude can be represented by array.

2. Centroid Vectors

As for centroid vectors, I use 2-dimension array whose shape is (n_clusters, n). n_clusters represents the number of clusters and n means the feature of data set. For example, if the dataset is about location including latitude and longitude, the feature will be two.

3. clusterAssment

In the mapper operator, we should create an array named clusterAssment to store the clustering result of each point. The shape of arrayAssment array is (m, 2). m represents the number of samples each mapper operator deals with and 2 means the pair of index and distance. In the K-means algorithm, the most important step is to find which center is closest to each point. For example, if the point i is nearest to the center a with the minimal distance 5, the pair of this point will be (a, 5). For each dataset and its clusterAssment, the index will be the same. For example, the pair in clusterAssment of the first point in the dataset will also be the first one. With this benefit, we can easily use an integer to index all points and their pairs in the clusterAssment.

4. distMatrix

In the elkan K-Means algorithm, we should use distMatrix to store the distance between all center points. For example, if we want to have 5 clusters, we need to create a 5x5 matrix which stores the distance between center point 1 and center point 2, etc.

4.2 Illustration of the Network Model

Based on Pipeline of MapReduce discussed above, I make some changes according to the K-means algorithm and API of Ray to make it distributed clustering algorithm. The modified pipeline is shown as Fig. 4-1

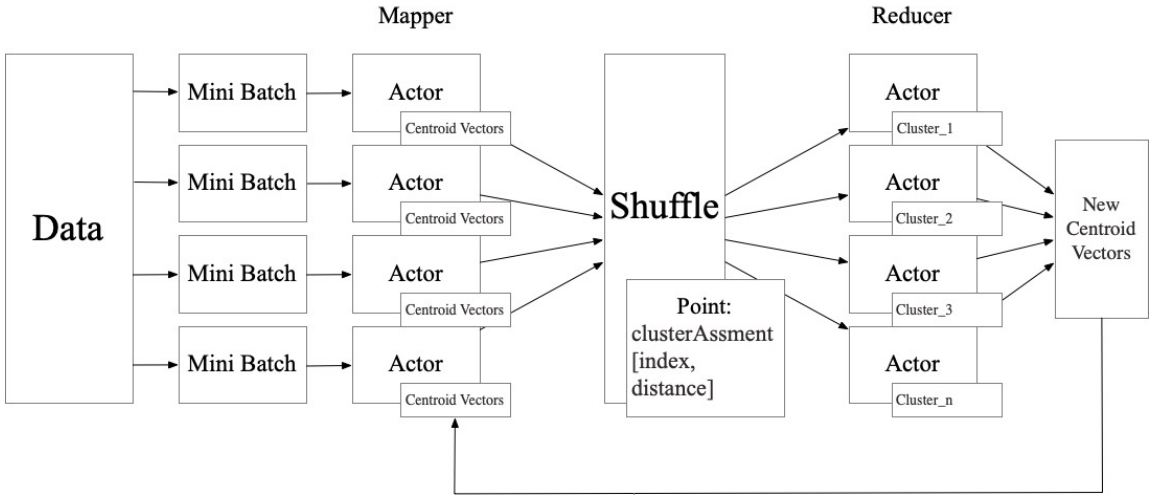

Figure 4-1

At the beginning, we filter out latitude and longitude of Gowalla check-in dataset and calculate the centroid vectors given clusters number k. Two methods can be utilized for generating centroid vectors. The first one is selecting k points randomly from dataset as initial centroid vectors. And the another one is K-means++ algorithm, which is discussed in the Algorithm Used part.

After getting initial centroid vectors, we split the dataset into many mini batch. In the traditional MapReduce, each batch will flow into the mapper operator to do the calculation. In this part, each mini batch will invoke an Actor in the Ray system as a mapper operator. And then we use a broadcast function to send the initial centroid vector to all actors. Given all requirements, each actor will process the data and generate the clusterAssment which includes index (which cluster each point belongs to) and distance between point and nearest centroid vector. In the Ray system, the execution between Actors is separate, so this mapper function can be highly efficient executed in parallel.

As for each Actor, it has a remote function called assign_cluster to deal with each point in the mini batch. In general, given k clusters and n points, we should calculate the distance kxn times sequentially to assign each point to its nearest centroid vector. This process is time-consuming, and many advanced algorithms are proposed to optimize this process, which is discussed in 4.3 Algorithm Used. As we all know, each remote function is treated as a separate process and will be executed on one worker. The issue here is that every task invocation has a non-trivial overhead (e.g., scheduling, inter-process communication, updating the system state) and this overhead dominates the actual time it takes to execute the task. Therefore, as it is a sequential program, it is necessary to implement some different data processes to make the program faster.

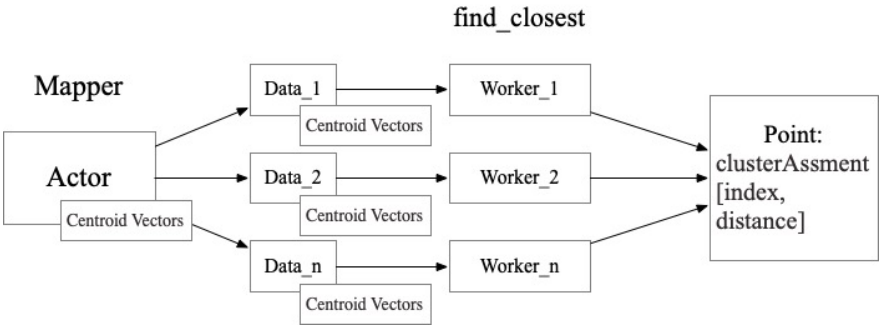

Figure 4-2

In Fig. 4-2, I implement a parallel calculation to amortize the remote invocation overhead. For each Mapper (Actor based), I split the dataset into n much smaller set with centroid vectors. And then I invoke n workers (separate processes) to deal with each small dataset. All I want to do is that although invoking workers can cost a bit, different workers can do distance calculations in parallel. Therefore, sequential work can be cut off into many parts and these parts will be executed in parallel. With this special treatment, the performance can be improved and the whole pipeline is more suitable for more machines to finish the distributed clustering algorithm.

Back to the main pipeline, after the mapper operators, we get all list of clusterAssment including every pair (index and distance) and information (latitude and longitude) of each point. In the reducer operator, we initiate as many actors as the clustering result. For example, if we generate 5 clusters, 5 actors will be invoked for reducers. For the reducer 1, it will get the point data from shuffle whose index is 1. The rest can be done in the same manner. Finally, the reducer 1 collects all points from cluster 1 and calculates the mean value of coordination (latitude and longitude) to generate a new center point.

With all reducers finishing calculation in parallel, we gain k new centroid vectors. In order to make a more precise result and finish iteration, we should compare the new one with old centroid vectors by some loss function and update the old one. With the new centroid vectors, we process the data again in a new iteration until we get the final result.

4.3 Algorithm Used

To make further optimization, I utilize some advanced algorithms to improve the performance of a distributed clustering algorithm.
1. K-means++

Although the traditional method, choosing k centroid vectors randomly, seems like fast and high efficiency, it won't perform well on convergency at a large dataset. Actually, with random choice, it will spend a lot of time on finding good centroid vectors. K-means++ offers one smart strategy when choosing k centroid vectors so that it will have a high efficiency in finding good results.
The K-means++ algorithm shows below.
1) Randomly select one point from the data set D as the first centroid vector μ_1
2) For every point x_i in the dataset, calculate its distance from the nearest cluster center of the selected centroid vectors:

$$D(x_i) = argmin \left|\left|x_i - \mu_j\right|\right|_2^2 \quad (j = 1, 2, \ldots k_{selected})$$

3) Select a new data point as the new centroid vector, the principle of selection is:
the larger the $D(x_i)$, the greater the probability of being selected as the centroid vector.
4) Repeat the step 2 and 3 until selecting k centroid vectors.

With the K-means++ algorithm, the distance between each centroid vector will be larger, which is good for convergency and efficiency of K-means distributed clustering algorithm.

2. Elkan K-Means

In the traditional K-Means algorithm, we need to calculate the distance from all sample points to all centroids in each iteration, which will be time-consuming. The Elkan K-Means uses the triangle property where the sum of the two sides is greater than or equal to the third side and the difference between the two sides is less than the third side to reduce the calculation of the distance.
The Elkan K-means algorithm shows below.
Given one point x and two centroid vectors μ_{j_1}, μ_{j_2}
if

$$2D(x, j_1) \leq D(j_1, j_2)$$

with triangle property, we know:

$$D(j_1, j_2) < D(x, j_1) + D(x, j_2)$$

then

$$2D(x, j_1) \leq D(j_1, j_2) < D(x, j_1) + D(x, j_2)$$

then

$$D(x, j_1) < D(x, j_2)$$

At this point we don't need to calculate $D(x, j_2)$ if we know $2D(x, j_1) \leq D(j_1, j_2)$. That is to say, we can reduce unnecessary distance calculation to reach high performance.

3. Spark K-Means

Spark has its own optimization about reducing calculation about distance. Unlike Elkan K-Means method using triangle property, Spark K-Means define a formula called lowerBoundOfSqDist to achieve goal.
The Spark K-means algorithm shows below.
Given one point x, one centroid vector μ and minimal distance between point x and first $\mu - 1$ centroid vectors:

if

$$lowerBoundOfSqDist(x, \mu) < \text{minimal distance}$$

then skip centroid vector μ and move on next centroid vector

Given the point (a1, b1), the lowerBoundOfSqDist(point) will be defined as:

$$lowerBoundOfSqDist(point) = (\sqrt{a1^2 + a2^2} - \sqrt{b1^2 + b2^2})^2$$
$$= a1^2 + a2^2 + b1^2 + b2^2 - 2\sqrt{(a1^2 + b1^2)((a2^2 + b2^2)}$$

This theory is proved at Appendix A. (see Appendix A for full details)

5 Testing Methodology

5.1 Parameter

To test the performance of the distributed clustering algorithm, I set the time of running 10 iterations as measurable indicators. As for the testing parameter, I set the scale of the dataset, the number of mappers and the number of workers dealing with the calculation of distance.
The constant parameter is the 10 iterations and 20 clusters.

5.2 Environment

1. Software

The Software requirement has been discussed in the 3.2 Software Requirement.

2. Hardware

Processor	2.4 GHz 8-CoreIntel Core i9
Memory	64 GB 2667 MHz DDR4
Graphics	AMD Radeon Pro 5500M 8 GB
	Intel UHD Graphics 630 1536 MB

5.3 Result

1. Size of Dataset, Mapper – Time

In this part, we will set different scales of the dataset and numbers of mappers to see which combination of these two factors can make the highest efficient clustering. The result can be shown in Fig. 5-1

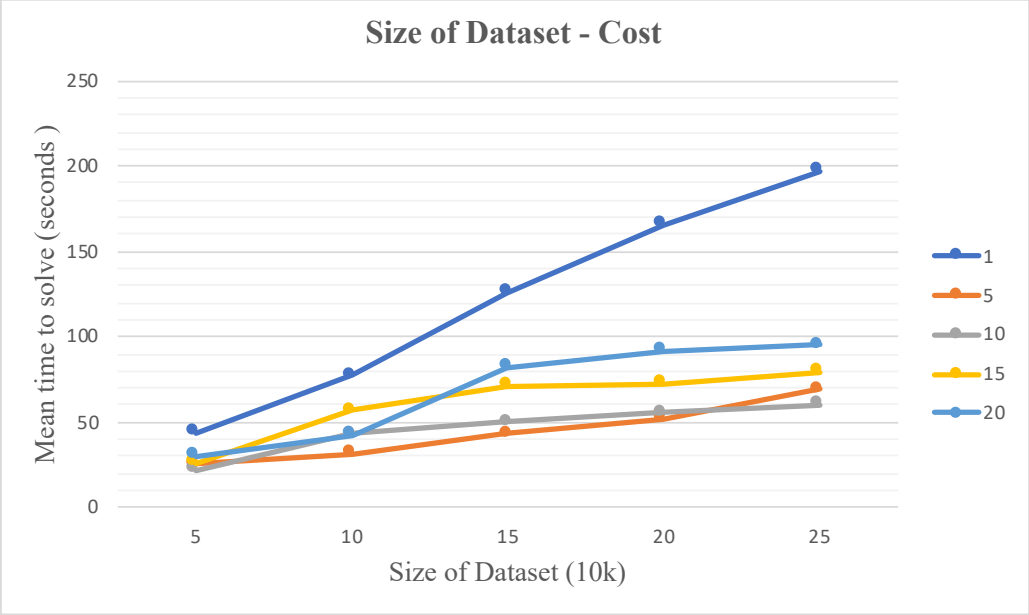

Figure 5-1

In Fig. 5-1, different colors represent a different number of mappers used in the system. As we can see, if the mapper is 1 (no split data), the cost will increase linearly. With the number of mappers increase, the efficiency will be improved. However, as we can see in the chart, too many mappers will invoke excessive actors and workers to deal with plenty of data. With the limitation of several workers can be invoked concurrently and the size of the object store, the efficiency will be decreased from 10 to 20. Different machines have different configurations. Therefore, people should take many experiments to find the best number of mappers to reach high performance.

2. Number of Workers, Size of Dataset – Time

In this part, we will set different scales of the dataset and numbers of workers shown in Fig. 5-2 to see whether the efficiency will be improved with the increasing numbers of workers. The result is shown in Fig. 5-2

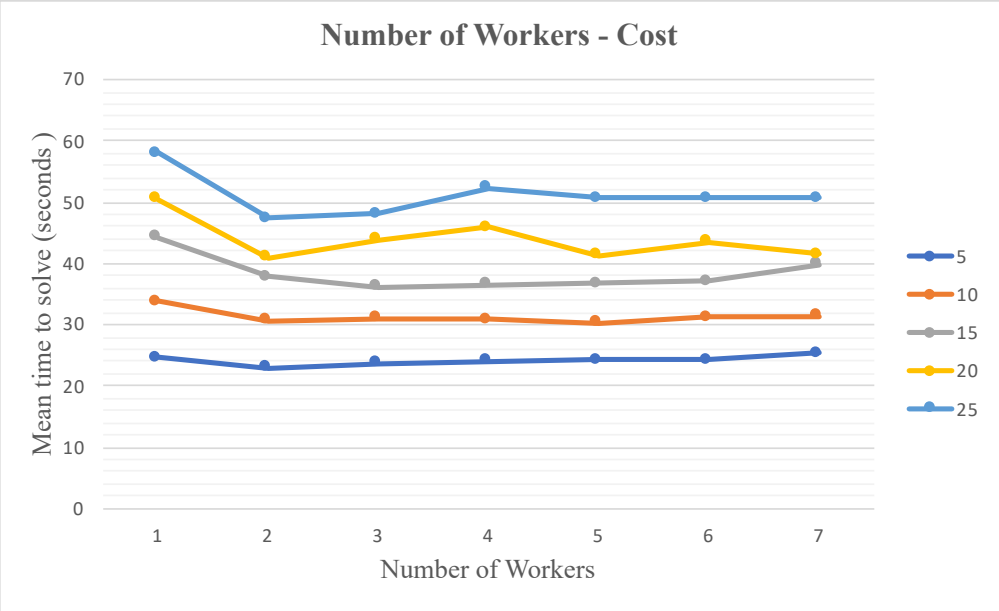

Figure 5-2

In Fig. 5-2, different colors represent a different scale of the dataset used in the system, whose units are 10k. From 1 worker to 2workers, the efficiency increases sharply. This is because 2 workers can share the computing tasks of 1 worker. However, with more workers joining in, the performance increases slightly and even decreases. As shown in my computer, I cannot create too many workers at the same time. Therefore, there is a queue with not finished tasks. Each task finds the idle workers to execute. All this action will shrink the efficiency of the Ray system, which leads to no good effects.

3. Spark, Sklearn, Ray

In this part, we will set different scales of the dataset and types of K-Means shown in Fig. 5-3 to make comparisons and see whether Ray performs well in distributed clustering. The result is shown in Fig. 5-3

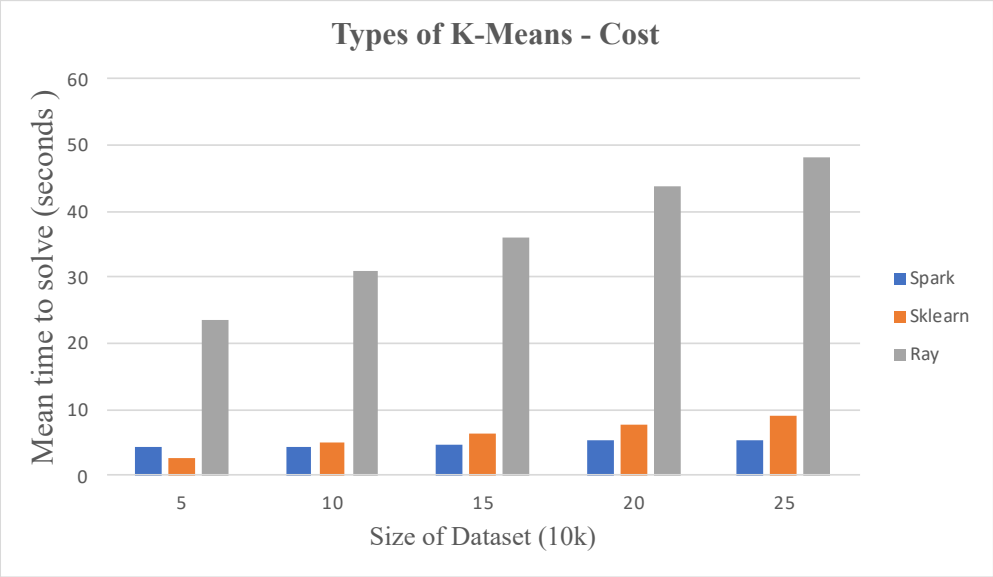

Figure 5-3

Unfortunately, although various methods for optimization have been implemented and the efficiency increases much than before, the result shows that the efficiency of Ray is slowest. As we all know, sklearn uses cpython to run python code fast and Spark uses RDD - a high performance of data type to improve the effectiveness. As for Ray, the cost of invoking actors and workers and the management of object stores can slower the system. Moreover, maybe I don't use the proper data type and a lot of data redundancy and worthless calculation can also make the system not efficient. Therefore, making a distributed clustering algorithm on top of Ray still have room for growth.

6 Conclusions

6.1 Summary

With sample wrappers and API of Ray, anyone can use their own distributed system, especially for machine learning. The initial intent of Ray is for Reinforcement learning, building a distributed training framework. With Ray's growth, many people are wondering whether Ray can take Spark's space, becoming a brand-new distributed computing framework for general purposes. In my report, I try one specific example – a distributed clustering algorithm to see the performance of Ray and compare it with Spark, which has a complete clustering API. With several optimizations and distributed strategies utilized, Ray can easily handle this question. However, Ray does not perform well in efficiency and has many aspects to optimize. Therefore, Ray is more like a key, opening the door to distributed applications for users. We all know that the future must be a distributed world, so this trend will never change. I firmly believe that Ray will become stronger and more robust with the maintenance of developers and users from all over the world.

6.2 Limitations

1. workers upper bound

When I run the code on the local machine, the console always shows that WARNING: 50 PYTHON workers have been started. This could be a result of using a large number of actors, or it could be a consequence of using nested tasks (see https://github.com/ray-project/ray/issues/3644) for some a discussion of workarounds. In the Ray system, each task will be executed on one worker. To some content, the execution of a task is the same as processes in Python. However, due to the distributed property of Ray, each task will be slower because every task invocation has to deal with many affairs (e.g., scheduling, inter-process communication, updating the system state). All these actions will lower the time and make the system not so efficient. What's more, Ray itself has some rules about how many workers can be executed at the same time, which means that some tasks cannot be executed in time because there are no idle workers. Although the configuration of the local machine is not very high and the Ray cannot perform well, I believe that if this distributes clustering algorithm can be implemented on the cluster (many machines connected with each other), there are no more workers upper bound problem.

2. object store upper bound

In the clustering pipeline, we have to calculate many variables and return millions of data. In the Ray system, all data will be stored in an object store and can be fetched by Ray API. However, with the limited size of the object store, we cannot store all data at the same time. In other words, if some workers run tasks concurrently and generate tons of data, whose size exceeds the size of the object store, some workers have to stop until there is available space in the object store. This limitation will severely affect performance when doing clustering. However, when deploying this distributed application on a cluster with a larger object store, this problem will be easily solved.

6.3 Recommendations for Further Work

1. optimize worker

As we discussed above, the execution of worker costs a lot. In the future, I should work on the source code of Ray about the worker, making some changes for specific purposes, to make it more efficient.

2. test on cluster

In my hardware environment, I just test on one machine with less object store size and configuration. Furthermore, in order to test the performance on distributed clustering deeply, I should focus on cluster which having more than one machine. With better configuration, I can do further tests and work based on Ray.

3. more clustering algorithm

To make a demo, I just test K-Means as a distributed clustering algorithm. In the future, I will test other clustering algorithms like DBSCAN, STING, etc.

In general, Ray, as an emerging distributed computing framework, has so many aspects to optimize and adjust. If Ray can solve all the problems of machine learning at high efficiency, more and more people will engage in it and build their own distributed application.

References

1. Lloyd, Stuart P. (1957). "Least square quantization in PCM". IEEE Transactions on Information Theory, VOL. IT-28, NO. 2, March 1982, pp. 129–137.

2. Arthur, D.; Vassilvitskii, S. (2007). "k-means++: the advantages of careful seeding". Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035.

3. B. Bahmani, B. Moseley, A. Vattani, R. Kumar, S. Vassilvitskii "Scalable K-means++" 2012 Proceedings of the VLDB Endowment.

4. Elkan, Charles (2003). "Using the triangle inequality to accelerate k-means" (PDF). Proceedings of the Twentieth International Conference on Machine Learning (ICML).

5. "MapReduce Tutorial". Apache Hadoop. Retrieved 3 July 2019.

6. Marozzo, F.; Talia, D.; Trunfio, P. (2012). "P2P-MapReduce: Parallel data processing in dynamic Cloud environments" (PDF). Journal of Computer and System Sciences. 78 (5): 1382–1402.

7. "Example: Count word occurrences". Google Research. Retrieved September 18, 2013.

8. Berlińska, Joanna; Drozdowski, Maciej (2010-12-01). "Scheduling divisible MapReduce computations". Journal of Parallel and Distributed Computing. 71 (3): 450–459.

9. Philipp Moritz et al. 2018. Ray: A Distributed Framework for Emerging AI Applications. In 13th USENIX Symposium on OSDI '18. 561-577.

10. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, HotCloud'10, pages 10--10, Berkeley, CA, USA, 2010. USENIX Association.

Appendix A - Theory about Spark K-Means

The proof is as follows.
Given two vectors:

$$X = \{x_1, x_2, \dots x_n\}$$
$$Y = \{y_1, y_2, \dots y_n\}$$

then

$$\sum_{i=1}^{n}(x_i - y_i)^2 \geq (\sqrt{\sum_{i=1}^{n}|x_i|^2} - \sqrt{\sum_{i=1}^{n}|y_i|^2})^2$$

calculate the norm of two vectors:

$$X.norm = \sqrt{\sum_{i=1}^{n}|x_i|^2}$$
$$Y.norm = \sqrt{\sum_{i=1}^{n}|y_i|^2}$$

We need to prove that

$$\sum_{i=1}^{n}(x_i - y_i)^2 \geq (X.norm - Y.norm)^2$$

this inequality means that if the minimum value of the distance between the current data point and the current center point has exceeded the previous optimal value, then the distance must be longer than the previous optimal value. Therefore, there is no need to loop n times to calculate Euclidean distance.
The inequality is proved below.

$$\sum_{i=1}^{n}(x_i - y_i)^2 = \sum_{i=1}^{n}|x_i - y_i|^2 \geq \sum_{i=1}^{n}(|x_i| - |y_i|)^2 = \sum_{i=1}^{n}(|x_i|^2 + |y_i|^2 - 2|x_i||y_i|)$$
$$= \left(\sqrt{\sum_{i=1}^{n}|x_i|^2}\right)^2 + \left(\sqrt{\sum_{i=1}^{n}|y_i|^2}\right)^2 - 2\sum_{i=1}^{n}|x_i||y_i|$$

Due to (Cauchy inequality)

$$\left(\sum_{i=1}^{n}|x_i||y_i|\right)^2 \leq \sum_{i=1}^{n}|x_i|^2 \sum_{i=1}^{n}|y_i|^2$$

Thus, formula above can be

$$\geq \left(\sqrt{\sum_{i=1}^{n}|x_i|^2}\right)^2 + \left(\sqrt{\sum_{i=1}^{n}|y_i|^2}\right)^2 - 2\sqrt{\sum_{i=1}^{n}|x_i|^2 \sum_{i=1}^{n}|y_i|^2}$$

$$= (\sqrt{\sum_{i=1}^{n}|x_i|^2} - \sqrt{\sum_{i=1}^{n}|y_i|^2})^2$$