Parallel Implementation of OPTICS Algorithm Based on Spark

Group 6

QIU Yaowen,20784389

WANG Qirui, 20786791

SHEN Yiqin, 20787501

BU Shihan, 20787953

Hong Kong University of Science and Technology

Big Data Computing (MSBD 5003)

---

1. Introduction and Motivation

Cluster analysis is nowadays a vital technique in database mining. For users, clustering can serve as an efficient data processing method to let users obtain deeper insights into the corresponding datasets in the way of visualization. Besides, it can also serve as a preprocessing technique in combination with other algorithms after detecting the groups of input dataset among databases. Moreover, this kind of technique can be widely applicated in large fields of areas such as anomaly detection.

A popular method is the hierarchical density-based clustering, DBSCAN (Density-Based Spatial Clustering of Applications with Noise). However, the performance of DBSCAN is sensitive to the setting of the initial input parameters. In this case, the parameter setting becomes particularly difficult in general cases without expert settings. Hence, OPTICS (Ordering Points to Identify the Clustering Structure) algorithm was proposed for non-sensitive parameter settings, which is friendly for general users when doing the clustering.

In the big data era, huge amount of data floods into database ceaselessly. In large dataset scenarios, there still exists inadaptation for OPTICS algorithm since its high complexity of temporal and spatial characteristic. With the enhancement of cloud and parallel computing, spark may provide us an effective method to solve the existing problem of OPTICS.

In this project, we implemented a Spark version of OPTICS algorithm. The original OPTICS algorithm is sequentially executed. We made full use of spark techniques such as Spark RDD, partition methods and parallelization designs for optimizing a well transformation structure of deriving the local result from each partition and combine these results to the global result. We made comparisons of performance based on our selected dataset and evaluation metrics. Furthermore, we indicated how spark version of OPTICS algorithm speed up the process of clustering in terms of reducing the time and space consumption.

2. OPTICS Algorithm Description

OPTICS, or Ordering points to identify the clustering structure, is a density-based clustering algorithm which improves the DBSCAN algorithm by reducing the sensitivity of input parameters. Before describing OPTICS, it is necessary to give definitions of terms used in OPTICS.

ϵ-neighborhood of a point p

A point p's ϵ-neighborhood are points within a radius of ϵ from p(including p).

Core point

If the number of points lying inside the ϵ-neighborhood of p is $\geqslant MinPts$, then p is a core point. In Fig 1, p is a core point.

Border point

A point q is a border point if q falls within the neighborhood of a core point but it is not a core point. In Fig 1, q is a border point.

Noise point

A point r is a noise point if it is neither a core point nor a border point. In Fig 1, s and r are noise points.

Directly Density-Reachability

A point q is directly density-reachable from a point p if p is a core point and q is in p's ϵ-neighborhood. In Fig 1, q is directly density-reachable from p.

Density-Reachability

A point q is density-reachable from a point p based on ϵ if there is a chain of point $p_1, p_2, \dots p_n$, and $p_1 = p, p_n = q$, and p_{i+1} is directly density-reachable from p_i. In Fig 1, s is density-reachable from p.

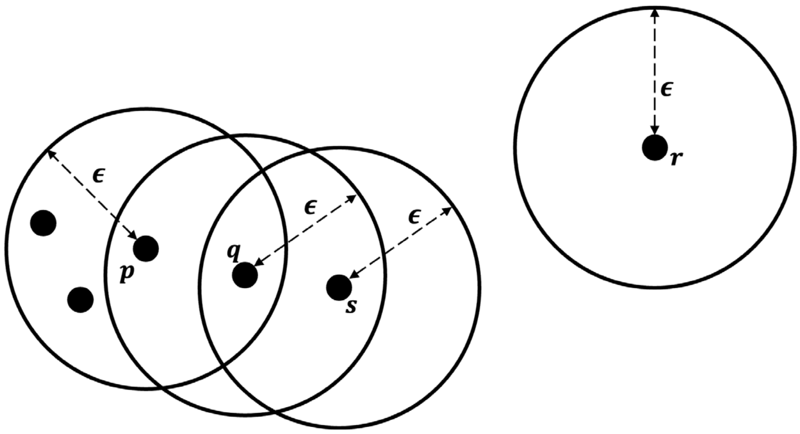

Figure 1: An illustration of core point, border point, and noise point, directly density-reachability, and density-reachability, when $MinPts = 4$. p is a core point, q is a border point, s and r is a noise point. q is directly density-reachable from p. s is density-reachable from p.

Core distance

For a point p and given $\epsilon, MinPts$, the core distance is defined as the minimum radius distance that makes p as a core point. Specifically, if p is not a core point, the core distance is undefined.

$$cd(p) = \begin{cases} \text{undefined} & |N_\epsilon(p)| < \text{ MinPts} \\ d\left(p, N_\epsilon^{MinPts}(p)\right) & |N_\epsilon(p)| \geq \text{Min } Pts \end{cases} \tag{1}$$

Reachability-distance

For a point p and point q, and given $\epsilon, MinPts$, the reachability-distance of q related to p is defined as the maximum value between the core distance of p and the distance between p and q.

Specifically, if p is not a core point, the reachability distance is undefined.

$$rd(q,p) = \begin{cases} \text{undefined} & |N_\epsilon(p)| < \text{ MinPts} \\ \max\{cd(p), d(p,q)\} & |N_\epsilon(p)| \geq \text{MinPts} \end{cases} \tag{2}$$

Note that every point p in the dataset has these two properties.

The objective of OPTICS is to output an ordered list of point in the dataset, and each point has its computed core distance and reachability distance. The input of OPTICS is a dataset consists of data points, with parameters ϵ and $MinPts$, while ϵ is default to infinity. First, the algorithm initializes the set of core points $\Omega = \emptyset$. Second, it traverses each point in the dataset and append all core points into Ω. Third, OPTICS randomly picks an object o in Ω to process. It marks o as processed and append it to the ordered list. Then, it computes the reachability-distance of each unvisited point in ϵ-neighborhood of o, and append them to a set of seeds $Seeds$ according to the reachability-distance. Later, OPTICS picks a seed s with smallest reachability-distance from $Seeds$, mark it as visited and append it to the ordered list. If s is a core point OPTICS will append all the unvisited neighbor of s to the $Seeds$ and re-compute the reachability-distance. Repeatedly process the objects in Ω and $Seeds$ until they are empty. Algorithm 1 demonstrates the pseudo code of OPTICS and Algorithm 2 is the pseudo code of update process.

Algorithm 1: OPTICS

```
Input: Dataset D, ϵ, MinPts
Output: OrderedList

//Find All Core points in D;
CorePoints = CorePointsQuery(D, ϵ,Minpts) ;
//Compute Core Distance for each Core Point;
CoreDists = ComputeCoreDists(D, ϵ,Minpts);
for each unprocessed point p in CorePoints do
    //Find all p's ϵ − neighbor, including p;
    N = RegionQuery(p,ϵ);
    mark p as processed;
    append p to the OrderedList;
    if N ⩾ MinPts then
        Seeds = empty priority queue;
        Update(N, p, Seeds, CoreDists);
        for each next q in Seeds do
            N' = RegionQuery(q,ϵ);
            mark q as processed ;
            append q to the OrderList;
            if N' ⩾ MinPts then
                Update(N',q, Seeds, CoreDists);
            end
        end
    end
end
return OrderedList
```

Algorithm 2: Update

```
Input: N, p, Seeds, CoreDists;

//Find Core Distance of p;
CoreDist = CoreDists[p];
for each o in N do
    if o is not processed then
        //Compute reachability − distance of p related to o;
        reach dist = max(CoreDist, dist(p,o));
        if reachability distance of o == NULL then
            reachability distance of o = reach dist;
            Insert (o, reach dist) into Seeds;
        else
            if reach dist < reachability distance of o then
                reachability distance of o = reach dist;
                Remove (o,reach dist) from Seeds;
            end
        end
    end
end
```

3. Spark Implementation and Parallelization

3.1 Ball Tree

Structured data improves our algorithm's efficiency. In this project, we applied ball tree to enhance computing speed. Ball tree is the binary tree which separate by hypersphere. Every node in ball tree partitions the data into two disjoint sets. If there is an intersection between two hyperspheres, points are decided with distance to the ball's center. Following algorithm describes how to construct a ball tree.

```
function construct_balltree is
    input: D, an array of data points.
    output: B, the root of a constructed ball tree.

    if a single point remains then
        create a leaf B containing the single point in D
        return B
    else
        let c be the dimension of greatest spread
        let p be the central point selected considering c
        let L, R be the sets of points lying to the left and right of the median along dimension c
        create B with two children:
            B.pivot := p
            B.child1 := construct_balltree(L),
            B.child2 := construct_balltree(R),
            let B.radius be maximum distance from p among children
        return B
    end if
end function
```

During searching process in OPTICS, ball trees use the property that, for any point outside the ball, the distance to any point inside ball is greater than or equal to the distance between the given point and ball's surface. Using this property, can improve searching when apply clustering algorithm on ball tree.

3.2 Core points, Core distance, and Reachability Distance

Compared to DBSCAN, the computation of core points is tricky. Since ϵ is default to infinity, all points in the data are core points. The only thing to do is assigning a list of indices of all data to variable which indicates core points.

The core distance of a point is defined as the distance between the point and $MinPts^{\text{th}}$ neighbor. From the ball tree structure, it is easily to search $MinPts^{\text{th}}$ neighbor for each point. Furthermore, the list of $MinPts$ neighbors of a point can be built during the search. The total complexity is O $((\frac{N}{p}) * (\log (\frac{N}{p}))$, where N is the number of points and p is the number of workers.

The reachability distance is a N*N matrix, where each element $rd(q,p)$ is defined as $\max\{cd(p), d(p,q)\}$, if ϵ is infinity. To make it in parallel, indexes of points can be partitioned, and each index can be mapped to a row of the matrix in each partition. Since a RDD cannot refer variables in other RDDs, the Core distance is collected as a python dictionary in driver and broadcasted to each worker. The total complexity is O $(\frac{N}{p} * N)$.

3.3 Update Seed

The update part is divided into two sections. The first one is seed update, which can be done in parallel, and the second one is seed pruning, which must be done in sequence. Recall the pseudo-code of update, each (key, value) pair will be updated if the value is smaller than the reachability distance of current point o. And the modification of one key-value pair does not influence others. This property allows the update of Seed done in parallel. Later, the seed should be sorted by its value in descending order. This can be implemented using sortBy() function in Spark easily. Figure 2 demonstrates the whole process of seed update.

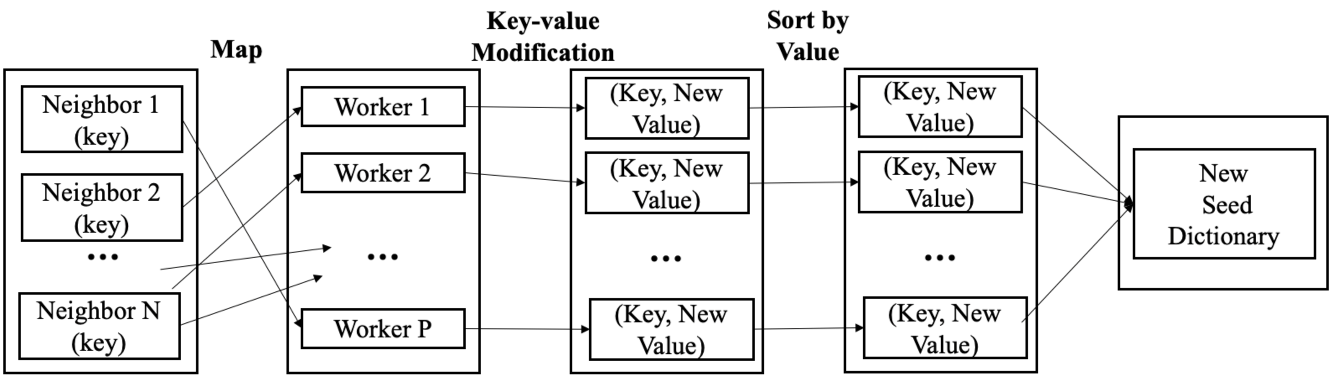

Figure 2: The process of updating Seed

However, after seed update, key with largest value will be pop and appended to the Order list. Then, it turns to next stage of seed update, until the seed is empty. Since the RDD of seed is dynamic and the terminated condition is related to seed itself, it is impossible to implement in parallel. Hence, we keep this part unchanged. Theoretically, the time complexity is reduce from O$(N * N)$ to O$(N * \frac{N}{p})$.

4. Experiment Results

4.1 Dataset description

In this part, we will test our program in different contexts, including some clustering benchmark datasets [3].

Clustering benchmark datasets [3] cover the different shapes or dimensions of node distribution, which can show the performance of our program in some extreme situations. In order to compare the speed between spark-versioned OPTICS and OPTICS under single machine, four clustering datasets with varies sizes are chosen. Table 1 demonstrates the statistics of each dataset.

Clustering Dataset			
Dataset	Node	Cluster	dimension
A1 set [3]	3,000	20	2
S1 set [3]	5,000	15	2
t4.8k[3]	8,000	6	2
Birch-set2 [3]	100,000	100	2

Table 1: Some clustering datasets may be tested in the project

4.2 Experiment settings

Since the OPTICS still requires $MinPts$, which has significant impact on the clustering result. It is time-costly to tune the $MinPts$ to fit each dataset. In addition, the project itself focus more on the performance of algorithm based on Spark, the $MinPts$ is set to 15 for each dataset, which does not mean it is the best parameter.

We build the spark cluster on Azure Databricks platform. Due to the limit of quota, a maximum number of 3 workers is allowed. Both driver and workers are Standard_DS3_v2 machine, with 14.00 GB RAM, an Intel Xeon E5-2673 v3 CPU with 4 Cores.

For each dataset, the number of partitions is set to 32,48, and 64 to test the relation between partitions and speed, respectively.

4.3 Experiment Result

Table 2 demonstrates the running time of Spark OPTICS under each setting of partition on each dataset. Compared to running time on single Standard_DS3_v2 machine, the running time on Spark is faster. The number of partitions also has impact on the speed. For A1 and S1 dataset, 48 partitions are faster than 32 partitions but slower than 64 partitions. A possible reason is the assignment of data to each partition cost more time on the driver program. The running time on t4.8k dataset are close under each partition while the case is not true on birch-set2. When the partition is set to 64, it is about half hour faster than the 32 partitions. The reason is most of time is spent on the calculation of reachability matrix and update, while the fixed time cost of assignment by the driver program is a small percentage of total running time. Since the number of nodes in Birch-set2 is 100,000, it is slow to run it in single machine, thus we stopped execution on the machine when the running time exceeds 2 hours.

Figure 3,4,5, and 6 demonstrates the visualization result of OPTICS algorithm using $MinPts = 15$ using the threshold selection principle. It is clear that $MinPts = 15$ works good on Birch-set2 but lead to lots of unexcepted outliers in another dataset.

Table 2: Running Time on each dataset

Dataset	Partitions			Time (Single Machine)
	32	48	64	
A1	51s	45s	47s	1m32s
S1	1m 30s	1m 26s	1m 26s	2m51s
T4.8k	3m 15s	3m 14s	3m 15s	5m47s
Birch-set2	1h 47min	1h 23min	1h 16min	>2h

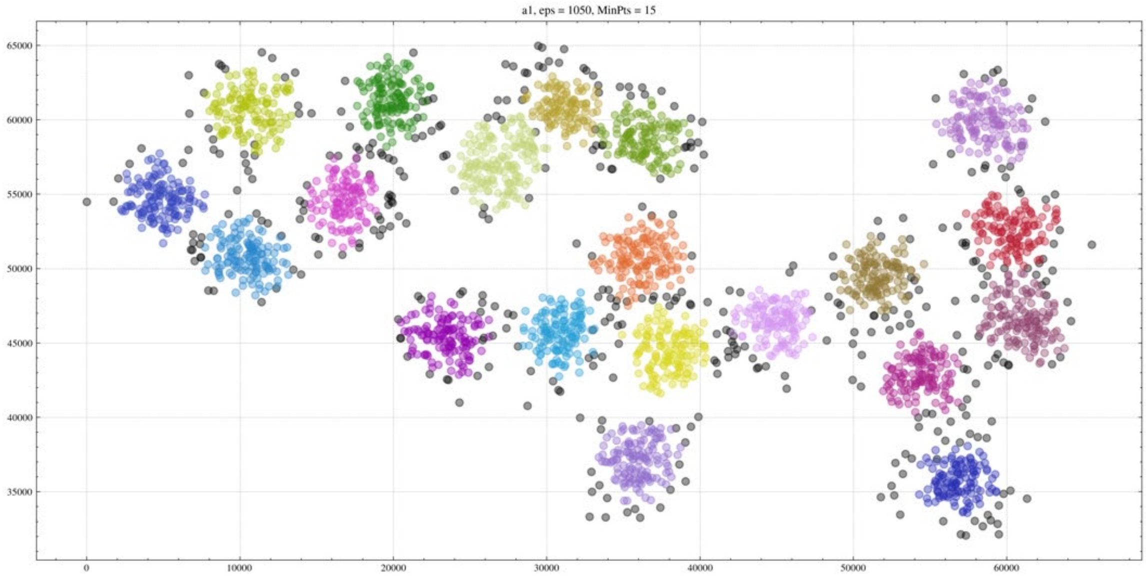

Figure 3: Visualization of OPTICS Cluster on A1 dataset

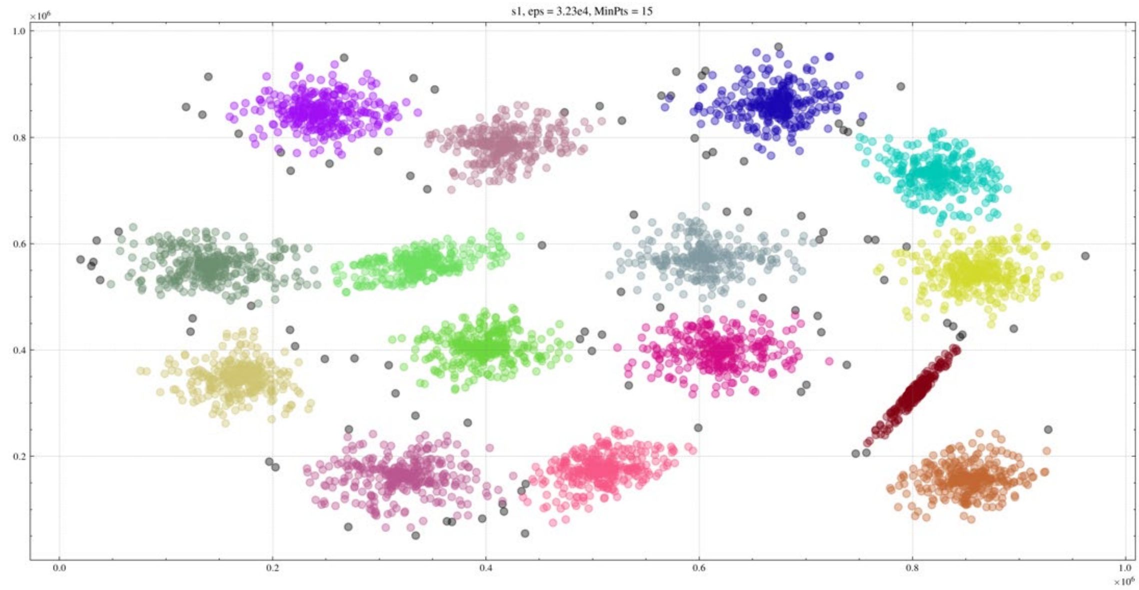

Figure 4: Visualization of OPTICS Cluster on S1 dataset

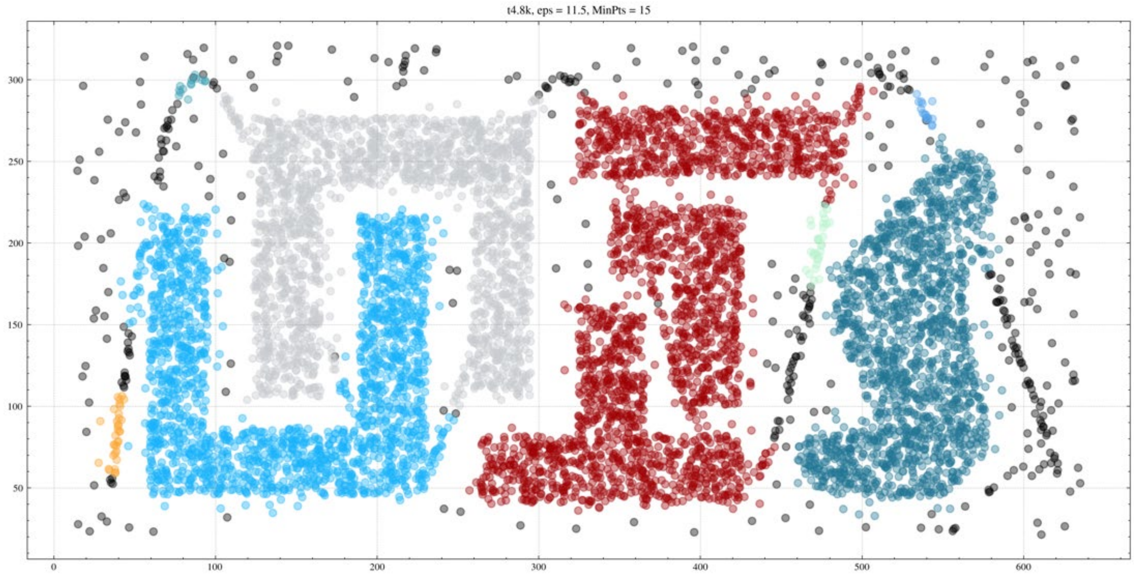

Figure 5: Visualization of OPTICS Cluster on t4.8k dataset

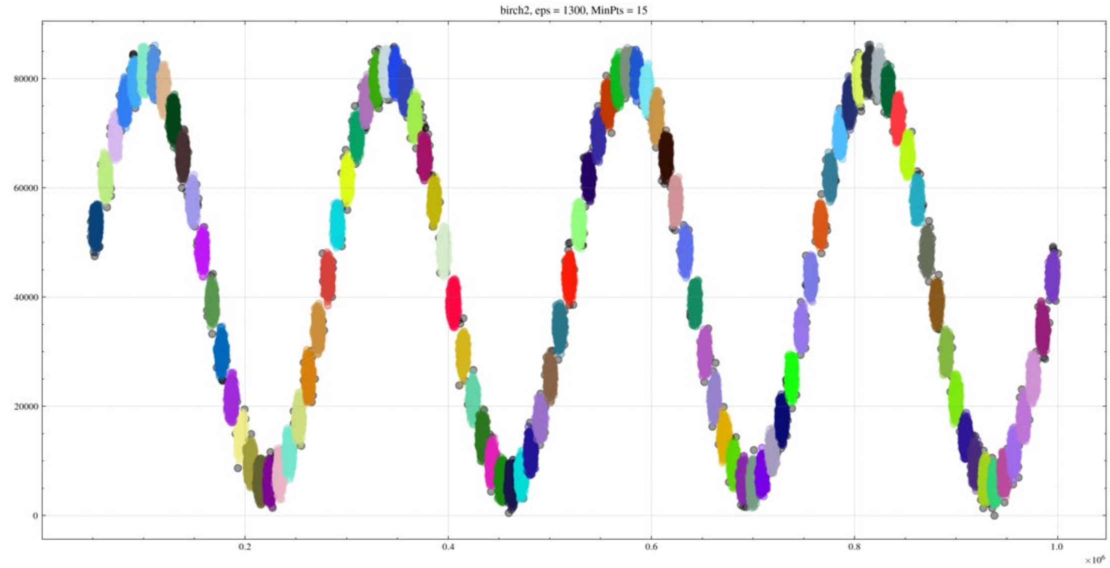

Figure 6: Visualization of OPTICS Cluster on Birch-set2 dataset

4.4 Result Analysis

Figure 7 is a snapshot of CPU and RAM usage when running the Spark OPTICS on S1 dataset with partition = 32. The average usage of CPU is stable around 50%. It means that around half of CPU resources is idle during the run.

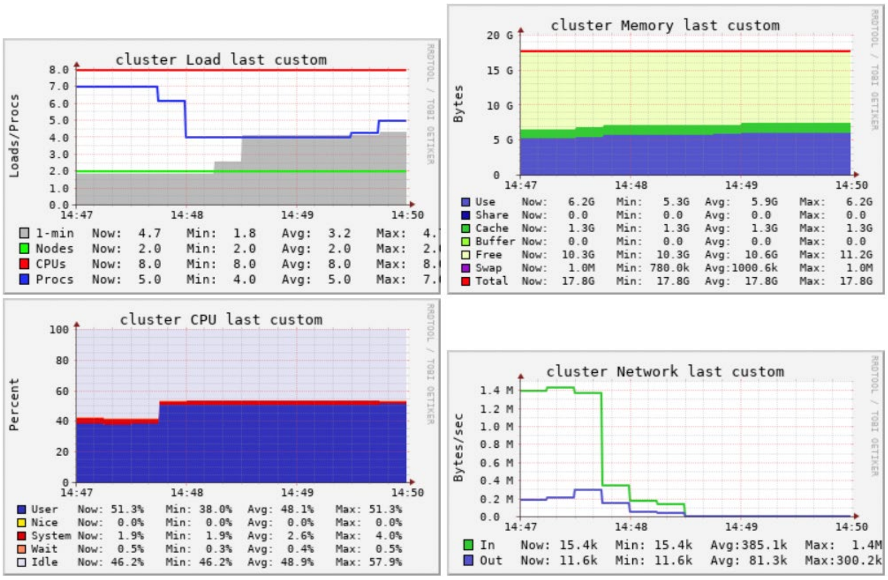

Figure 7: CPU and memory usage when running on S1 set with partition = 32

Figure 8 shows the percentage of time cost of the computation of reachability matrix, seed update, and others. “others” includes time spent on library import, load data, building ball tree, and core distance. For all dataset, the program spent most of time on computing reachability matrix. And with size increase on dataset, the portion of time spent on reachability matrix x tends to increase. In practical, the map function of r reachability matrix consists of two steps: the first part is to create a 1*n vector to store the row matrix; the second step is a for loop to fill the vector using the maximum value between core distance of given core point and distance between input data and given core point, when is done by calling a distance function. Since all points are core points in OPTICS, the operation of distance calculation and comparison exactly occurs $N * N$ times. It may explain the reason that the compute of reachability matrix spent most of time.

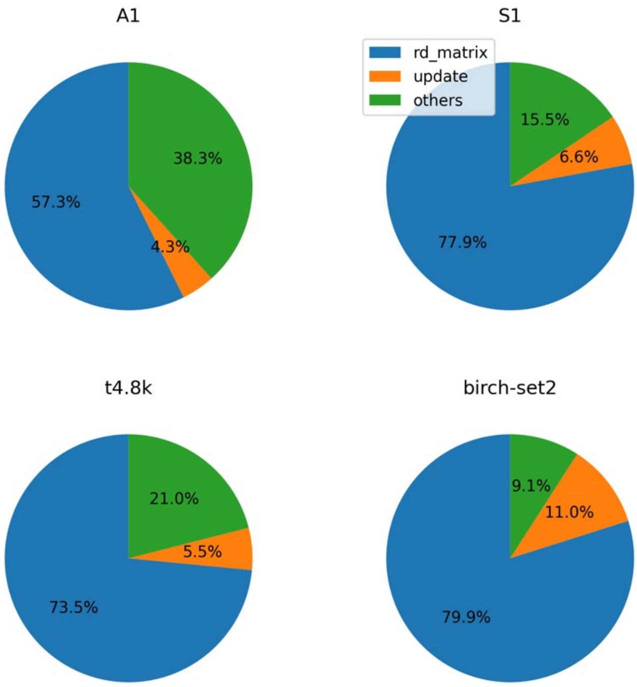

Figure 8: Percentage of time cost on reachability matrix, update function on each dataset

4. Future Works

In our project, we parallelize and compare the computational cost for building reachability distance matrix, seed update and other aspects of running time including computing core distance and constructing ball tree. Two-dimensional datasets were also utilized for comparison in our experiment. For further improvement, combination of reachability distance matrix with structure of ball tree can be considered to improve the efficiency of computational process. Moreover, for better performance of running on Spark, we can also do some optimization for memory usage based on reducing the number of collects, etc. In addition to using two-dimensional data only, the experimental accuracy of higher-dimensional data can be further performed and tested.

References

[1] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd* (Vol. 96, No. 34, pp. 226-231).

[2] Ankerst, M., Breunig, M. M., Kriegel, H. P., & Sander, J. (1999). OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod record*, *28*(2), 49-60.

[3] P. Fränti and S. Sieranoja. K-means properties on six clustering benchmark datasets. *Applied Intelligence*, 48 (12), 4743-4759, December 2018

[4] A. Reiss and D. Stricker. Introducing a New Benchmarked Dataset for Activity Monitoring. *The 16th IEEE International Symposium on Wearable Computers (ISWC)*, 2012.

[5] Liu, T.; Moore, A. & Gray, A. (2006). New Algorithms for Efficient High-Dimensional Nonparametric Classification. *Journal of Machine Learning Research*. **7**: 1135–1158.