# Density Based Spatial Clustering Application with Nosie Implementation and Modification on Husky

**Student: LI Changji Aaron** **Supervising professor : Prof. James CHENG**

**The Chinese University of Hong Kong**

## Introduction

This study aims to implement Density Based Spatial Clustering Application with Noise (DBSCAN) on a data-parallel computing system, Husky[1], developed by group of Prof. James CHENG. Meanwhile, to improve the performance of DBSCAN, some variants of DBSCAN will be implemented and tested on Husky.

## Basic Concepts

- Eps ($\varepsilon$) : epsilon value, a positive real value;
- *MinPts* : a small positive constant positive;
- Density-approached : For two points, if the Euclidean distance between them is less then $\varepsilon$, these two points are density-approached (neighbors) to each other.
- *Core Point* : For a point $\wp$, if the number of points which are density-approached to $\wp$ is more than *MinPts*, $\wp$ will be called *Core Point*.
- *Border Point* : Non-core point and there exist at least one neighbor which is *Core Point.*
- *Noise* : Non-core point and non-border point.

## Method

Procedure of DBSCAN on Husky

1. Find all *Core Points* and their neighbors.
2. Merge the *Core Points* into their own cluster (MinHash)
3. Find *Border* and *Noise*

pDBSCAN[2]:

1. Partition data with a grid (cell length = $\varepsilon / \sqrt{d}$)
2. Find all *Core Cell* and their neighbors.
3. Merge the *Core Cell* into cluster.
4. Find *Border* and *Noise*

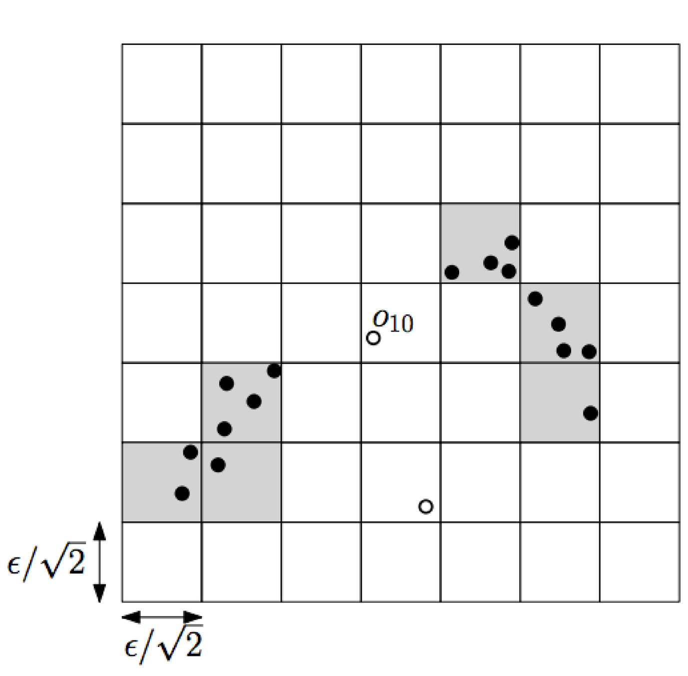


*Figure.1 pDBSCAN in a grid (2D)[]*

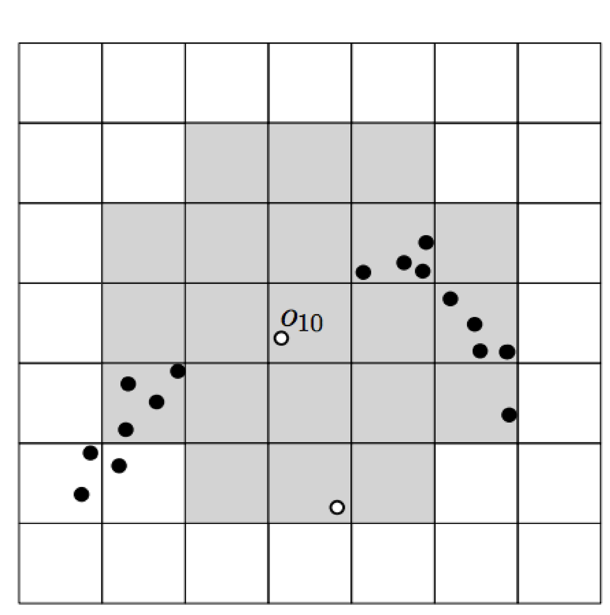


*Figure 2.Neighbor cells (in gray) of the cell of $o_{10}$ []*

## References

[1]F. Yang, J. Li and J. Cheng, "Husky: Towards a More Efficient and Expressive Distributed Computing Framework", PVLDB, 2016.
[2]J. Gan and Y. Tao, "DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation", SIGMOD, 2015.

## Result

**DBSCAN vs pDBSCAN:**

The datasets we used is HIGGS from UCI[]. We separate dataset into different size for testing the running time with different data size.

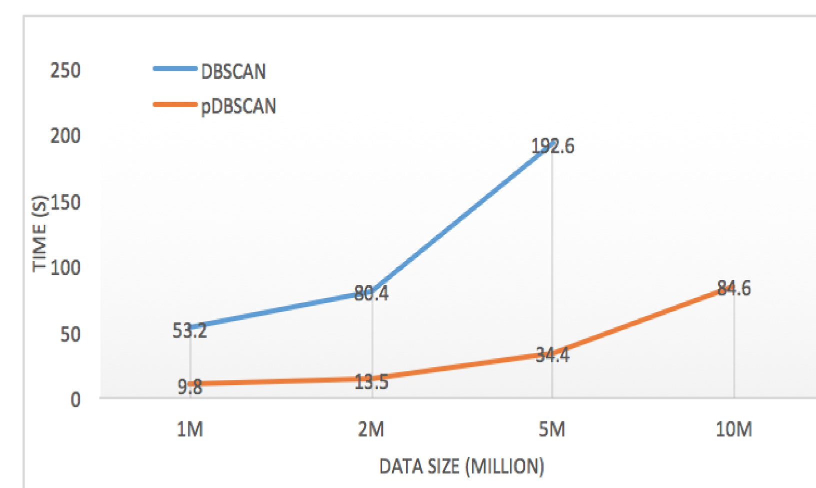


*Figure.3 Running time vs data size*

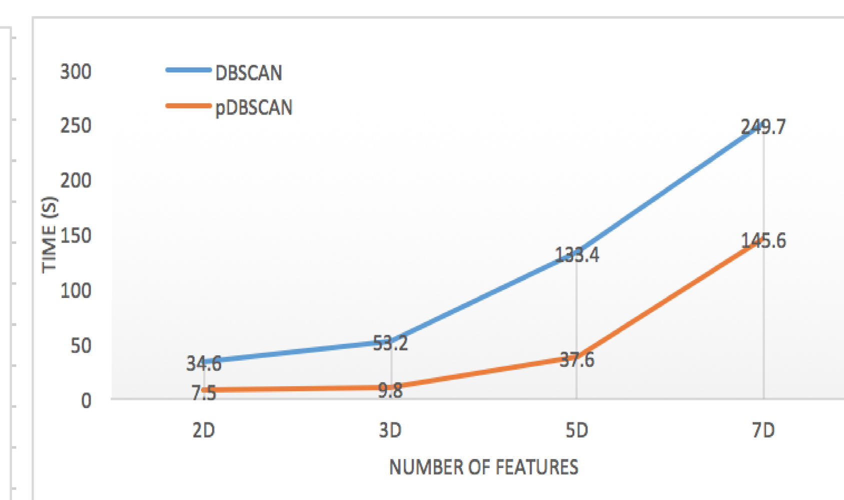


*Figure.4 Running time vs number of features*

The figure.3 shows that pDBSCAN is faster than DBSCAN and can handle larger data size. And figure.4 represents that the pDBSCAN can handle high dimensional data easier.

**Husky vs Spark:**

We compared pDBSCAN on Husky with DBSCAN on Spark whose algorithm is similar to pDBSCAN.

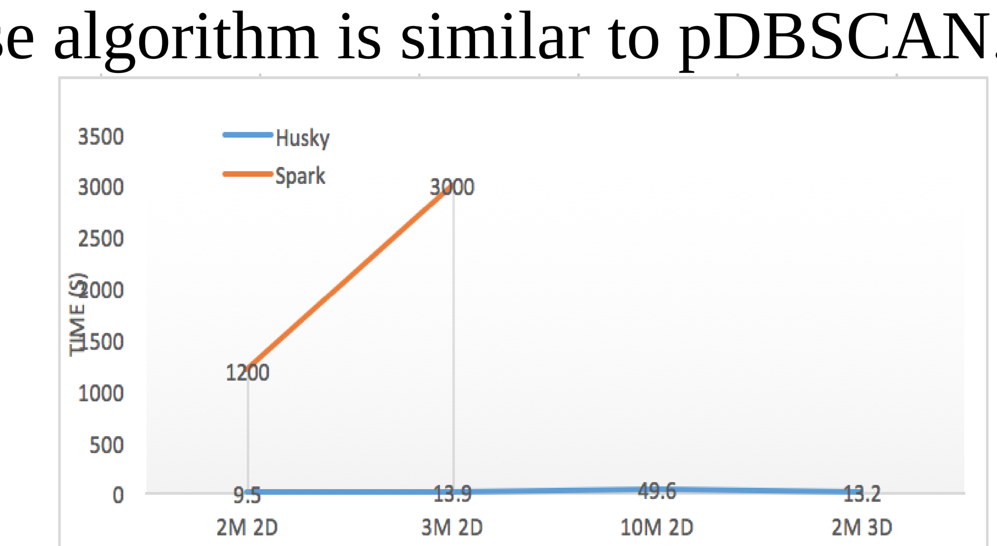


*Figure.5 Husky vs Spark*

Figure.5 tells us DBSCAN on Husky is faster than on Spark and it can handle larger and higher dimensional dataset.

## Discussion

- The basic DBSCAN can perform well because of the Husky and it also has many restrictions. And the pDBSCAN can handle more and perform better.
- There are also many other variants of DBSCAN which also can improve the performance such as approximate version of DBSCAN [2]. For approximate DBSCAN, the procedure of merge can be modified to O(n).
- However, because the parallel computing requires message communication a lot between each object, the running time complexity will be increased to $O(2^D)$ (D for number of dimensions) . Therefore, implementing the approximate DBSCAN on Husky and reaching the expected result is still a problem and will be the future work.

## Acknowledgements

I would like to express the deepest appreciation to my supervising professor and postgraduate mentor. Without their supervision and constant help this research would not have been possible.