Efficient Distributed Density Peaks for Clustering Large Data Sets in MapReduce

Yanfeng Zhang
Computing Center
Northeastern University, China
Email: zhangyf@cc.neu.edu.cn

Shimin Chen
Institute of Computing Technology
Chinese Academy of Sciences
Email: chensm@ict.ac.cn

Abstract—Density Peaks (DP) is a recently proposed clustering algorithm that has distinctive advantages over existing clustering algorithms. It has already been used in a wide range of applications. However, DP requires computing the distance between every pair of input points, therefore incurring quadratic computation overhead, which is prohibitive for large data sets. In this paper, we study efficient distributed algorithms for DP. We first show that a naive MapReduce solution (Basic-DDP) has high communication and computation overhead. Then, we propose LSH-DDP, an approximate algorithm that exploits Locality Sensitive Hashing for partitioning data, performs local computation, and aggregates local results to approximate the final results. We address several challenges in employing LSH for DP. We leverage the characteristics of DP to deal with the fact that some of the result values cannot be directly approximated in local partitions. We present formal analysis of LSH-DDP, and show that the approximation quality and the runtime can be controlled by tuning the parameters of LSH-DDP. Experimental results on both a local cluster and EC2 show that LSH-DDP achieves a factor of 1.7–70× speedup over the naïve Basic-DDP and 2× speedup over the state-of-the-art EDDPC approach, while returning comparable cluster results. Compared to the popular K-means clustering, LSH-DDP also has comparable or better performance. Furthermore, LSH-DDP could achieve even higher efficiency with a lower accuracy requirement.

I. INTRODUCTION

Clustering is a common technique widely used in many fields, including data mining, machine learning, information retrieval, image processing, and bioinformatics. Density Peaks (DP) is a new clustering algorithm proposed in 2014. Due to its effectiveness, DP has already been employed in a wide range of applications, such as neuroscience [2], geoscience and remote sensing [3], molecular biology [4], computational biophysics [5], image processing [6], and computer vision [7].

Given a set of points, DP computes two metrics for every point \(p\): (i) \(\rho\), the local density, which is the number of points within a specified distance from \(p\); and (ii) \(\delta\), the minimum distance from \(p\) to other points with higher densities. It is observed that the center of a cluster sees the highest local density among its neighbor points, and has a relatively large distance from other points with higher densities. Therefore, cluster centers can be determined by identifying the points with both high \(\rho\) and high \(\delta\).

Compared with previous clustering algorithms, DP has the following advantages. First, DP does not require a priori knowledge about the point distribution. In many well-known algorithms, such knowledge is important for choosing good algorithm parameters (e.g., the number of clusters in K-means [8], \(\varepsilon\) and \(minPts\) in DBSCAN [9]). In comparison, the clustering results of DP have been shown to be robust against the initial choice of algorithm parameters. Second, DP supports arbitrarily shaped clusters. Its effectiveness does not rely on the distribution of the data. Third, DP is deterministic. It always computes consistent cluster results, while many clustering algorithms (e.g., K-means and EM clustering [10]) may converge to different local minimums with different initial iterative states. Last but not least, \((\rho, \delta)\) provides a two-dimensional representation of the input point data, which can be in very high dimensions. It is straightforward to visualize \((\rho, \delta)\) in a 2D decision graph. From the graph, users can gain new insights into the data distribution and intuitively determine cluster centers.

While DP is attractive for its effectiveness and its simplicity, the application of DP is limited by its computational cost. In order to obtain \(\rho\) and \(\delta\), DP computes the distance between every pair of points. That is, given \(N\) points in the input data set, DP’s computational cost is \(O(N^2)\). As a result, it can be very time consuming to perform DP for large data sets.

In this paper, we study efficient distributed algorithms for DP so that this promising clustering algorithm can be more broadly used. In particular, we design distributed DP algorithms in MapReduce, which is one of the most popular big data processing paradigms today.

Challenges for Distributed DP. In a baseline implementation (Basic-DDP, Basic Distributed DP), we compute \(\rho\) and \(\delta\) values in two subsequent MapReduce jobs. The two jobs have similar computation procedures: The Map and the shuffling stages are mainly used to send and prepare the input data, while the Reduce stage performs the actual computation of \(\rho\) and \(\delta\), respectively. However, the algorithm has to shuffle every point to every other points and compute distances of all pairs of points, incurring quadratic computation and communication cost. Such cost will be prohibitive for large data sets that have millions of points, which are becoming more and more common in the big data era.

We consider approximate algorithms in order to reduce the computation and communication cost of distributed DP. We observe that DP takes advantage of the local characteristics (such as local density) of the data points for clustering. Therefore, it is natural to employ Locality-Sensitive Hashing (LSH) [11] to partition the input data so that closer points are more likely to be assigned to the same partition. Typically,
an LSH-based algorithm performs local computation within 
each partition, and then aggregates the local results from all 
partitions to obtain the final approximate results.

There are several challenges in employing LSH for DP. 
The first challenge is the computation of $\delta$. While the local 
density $\rho$ is a local property, $\delta$ is the the minimum distance 
to other points with higher $\rho$. Given a point $p$, it is possible 
that other points with higher $\rho$ are far away from $p$ and thus 
do not reside in $p$’s local partition. The second challenge is to 
provide guarantees for approximation accuracy of $\delta$ and $\rho$. It 
would be nice if LSH parameters such as the number of hash 
functions and the number of local partitions can be derived 
from the approximation accuracy target specified by the user. 
Finally, the LSH parameters may also impact the runtime of 
the solution. Therefore, it is important to study the tradeoff 
between approximation quality and efficiency.

Our Solution: LSH-DDP. To address the above challenges, 
we propose an approximate algorithm for DP, called LSH- 
DDP (LSH based Distributed DP). Specifically, we exploit 
the fact that cluster centers have both high $\rho$ and high $\delta$. 
Therefore, given a point $p$, if we cannot find another point 
with higher $\rho$ in the local partition, then we will consider $p$ 
as a candidate cluster center. Moreover, we analyze LSH-DDP 
and prove the approximation accuracy guarantees for $\rho$ and $\delta$. 
Based on this analysis, we derive the relationship between LSH 
parameters and the expected approximation quality. Finally, 
we evaluate the accuracy and performance of LSH-DDP by 
comparing LSH-DDP with Basic-DDP using real-world data 
sets with up to 11.6 million data points. Experimental results 
show that compared to Basic-DDP, LSH-DDP obtains very 
similar clustering results, while achieving up to 70x speedup.

Contributions. The contributions of the paper are threefold: 
First, we propose LSH-DDP, an efficient distributed algorithm 
that approximates $\rho$ and $\delta$ values in the DP algorithm. Second, 
we present formal analysis of LSH-DDP. Given a specific 
result quality requirement, users can tune the parameters 
to balance between effectiveness and efficiency. Finally, We 
conduct extensive experiments on real data sets. Experimental 
results demonstrate that LSH-DDP achieves a factor of 1.7– 
70x speedup over the naïve Basic-DDP and 2x speedup over 
the state-of-the-art EDDPC approach, while returning compa-
rable cluster results. Compared to the widely used K-means 
clustering, LSH-DDP has comparable or better efficiency.

The remainder of the paper is organized as follows. Section 
II describes background on DP and MapReduce. Section III 
introduces the basic MapReduce implementation of DP as 
baseline. Section IV proposes and analyzes our LSH-based 
approximate solution. Section V discusses parameter tuning. 
Section VI reports the experimental results. Section VII dis-

II. PRELIMINARIES

In this section, we review the standard DP algorithm and 
the MapReduce framework.

A. DP Algorithm

Density Peaks (DP) Cluster [1] is a novel clustering algo-

The local density $\rho_i$ of data point $i$ is computed as 

$$\rho_i = \sum_j \chi(d_{ij} - d_c)$$  \hspace{1cm} (1)

where $\chi(x) = 1$ if $x < 0$ and $\chi(x) = 0$ otherwise, and $d_c$ is 
called the cutoff distance. That is, $\rho_i$ is equal to the number 
of data points within the cutoff distance $d_c$.

The $\delta_i$ distance of data point $i$ is computed as 

$$\delta_i = \min_{j | |\rho_i| > \rho_i} (d_{ij})$$  \hspace{1cm} (2)

It is the minimum distance from point $i$ to any other point 
whose local density is higher than that of point $i$. Suppose 
$j = \arg\min_{|\rho_j| > \rho_i} (d_{ij})$. We say that point $i$ is assigned 
to point $j$, and point $j$ is referred to as the upslope point of point 
i. If point $i$ has the highest density among all data points, i.e., 
$i = \arg\max_i \rho_i$, then we set $\delta_i = \max_j (d_{ij})$. This point is 
called the absolute density peak.
Figure 1 illustrates the process of DP clustering through a concrete example. Figure 1a shows the distribution of a set of data points. Figure 1b depicts the corresponding density contour view based on the local density $\rho$ of each point. The warmer the color, the higher the density. Clearly, the peaks of the density mountains (a.k.a. density peaks) correspond to the cluster centers. Then we compute $\delta$. For a normal point $i$ on the slope of a mountain, the closest point that has higher density than $i$ is the next upslope point on the same mountain. This holds for all the points except the density peaks, who will be assigned to points on other higher mountains. This process forms an assignment chain as shown in Figure 1d, where the height of each point indicates its density $\rho$. Therefore, the density peaks are distinguished from other points as they have the highest local density $\rho$ and a large $\delta$. A point $i$ is depicted on a decision graph by using $(\rho_i, \delta_i)$ as its x-y coordinate. Then the density peaks can be identified as outliers in the top right region of the decision graph. Given the selected density peaks (cluster centers), it is straightforward to follow the assignment chain of a point to determine the density peak and the corresponding cluster that it belongs to.

DP requires the computation of pair-wise distances. A sequential implementation can be improved with the following techniques: (1) For computing $\rho$, we can employ the triangle inequality to filter unnecessary distance computations. (2) For computing $\delta$, we can first sort the points according to descending $\rho$ values. To compute $\delta_i$, we only need to consider $i$’s distance to the points ahead of point $i$. Note that these techniques are orthogonal to our proposed techniques and can be easily employed in the sub-tasks of the distributed computation in this paper.

B. MapReduce

MapReduce [12] is a popular big data processing framework using shared-nothing clusters. In MapReduce, input data are represented as key-value pairs. A user-defined Map function takes an input key-value pair and produces a set of intermediate key-value pairs. The MapReduce system then groups these intermediate key-value pairs by their keys and shuffles them to the user-defined Reduce function. The Reduce function takes an intermediate key and a list of its corresponding values as input and computes the final output key-value pairs. An algorithm often requires a series of MapReduce jobs. The output of the previous MapReduce job is processed as the input to the next job. The processing logic can be specified in a driver program, which is executed on the master node of the MapReduce system.

III. Baseline Method

In this section, we describe a basic MapReduce implementation of distributed DP, Basic-DDP. We analyze its cost and then discuss improvement opportunities.

A. Basic Strategy

As described in Section II-A, DP performs clustering by computing the $\rho$ and $\delta$ values for every data point. Therefore, the two main steps in Basic-DDP are the computation of $\rho$ and $\delta$ in a distributed fashion. The resulting $\rho$ and $\delta$ values are often much smaller than the input data size. Consequently, Basic-DDP selects the density peaks and performs cluster assignment in a centralized step.

In the following, we describe the four steps of Basic-DDP: a preprocessing step for choosing $d_c$, (using MapReduce), two key steps for computing $\rho$ and $\delta$ values (using MapReduce), and the final centralized step. Table I lists the notations used in this paper.

Preprocessing Step: Choosing $d_c$. The cutoff distance $d_c$ is a key parameter in DP. $d_c$ specifies the meaning of local in the computation of the local density $\rho$ in Equation (1). While the DP paper [1] shows that varying $d_c$ (by a factor of 20) produces mutually consistent results, we still need to choose a reasonable $d_c$ without a priori knowledge of the input data. As a rule of thumb, one can choose $d_c$ so that the average number of neighbors is around 1%-2% of the total number of points in the data set [1]. Suppose the distances between all pairs of points $D_{SEQ} = \{d_{12}, d_{13}, \ldots, d_{21}, \ldots\}$ are known, the 1% or 2% position of the ascending ordered set $Ord_a(D_{SEQ})$ can be approximately seen as $d_c$.\footnote{Please refer to the implementation code of [1].}

Considering that distributed sorting is an expensive task, we rely on sampling and run a preprocessing MapReduce job. The map() function samples and sends point pairs to a single reducer. The single reduce() function first computes the distances of all received point pairs and then chooses $d_c$ based on the sorted distance values.

Step 1: Computing $\rho$. As shown in Equation 1, the computation of $\rho_i$ requires to know the distance values between point $i$ and all other points. In a straightforward strategy, the map() function invocation for point $i$ will generate $N$ intermediate key-value pairs: $\forall j \in \mathbb{P}, (j, i:\text{point}_i)$. After shuffling and grouping, the reduce() function will be provided with $(i, \{j:\text{point}_j | \forall j \in \mathbb{P}\})$. It computes the distance between point $i$ and any point $j$, i.e., $\{d_{ij} | \forall j \in \mathbb{P}\}$. Given the pre-computed $d_{ij}$, $\rho_i$ can be obtained according to Equation (1). This straightforward strategy incurs a very high communication cost. The amount of data shuffled is $N$ times the input data size, where $N$ is the total number of points, typically a very large value.

Basic-DDP employs an alternative strategy that performs blocking to save the shuffle cost. The points set $S$ is partitioned into $n$ disjoint subsets, i.e., $S = \bigcup_{1 \leq k \leq n} S_k$, where $S_k \cap S_l = \emptyset (\forall k \neq l)$ and $\mathbb{P}_k$ contains the point ids of $S_k$. The

<table>
<thead>
<tr>
<th>Table I. Notations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>the set of points</td>
</tr>
<tr>
<td>$\mathbb{P}$</td>
<td>the set of point ids</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>the set of $\rho$ values, $\Omega = {\rho_i</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>the set of $\delta$ values, $\Delta = {\delta_i</td>
</tr>
<tr>
<td>$P(S)$</td>
<td>an LSH partition of $S$</td>
</tr>
<tr>
<td>$N$</td>
<td>the number of points in $S$</td>
</tr>
<tr>
<td>$n$</td>
<td>the number of subsets</td>
</tr>
<tr>
<td>$i$ or $j$</td>
<td>the point id</td>
</tr>
<tr>
<td>$k$ or $l$</td>
<td>the subset index</td>
</tr>
<tr>
<td>$c$</td>
<td>the cluster id</td>
</tr>
<tr>
<td>$u_i$</td>
<td>point $i$’s upslope point id</td>
</tr>
</tbody>
</table>
block partitioning is performed by the `map()` function. Since the distance matrix is symmetric, Basic-DDP will compute only the distance values in the upper triangular matrix to avoid redundant computation. Therefore, it sends each subset $S_k$ only to $\{S_l | k \leq l \leq n\}$ rather than all the subsets. The `reduce()` function is then applied to each pair of subsets $(S_k, S_l)$, where $k < l \leq n$, or the diagonal subsets $S_k$. Based on distance computation, for $(S_k, S_l)$, `reduce()` outputs two sets $\Omega_i = \{\rho|v_i \in P_k\}$ where $\rho_i = \sum_{j \in P_i} \chi(d_{ij} - d_c)$ and $\Omega_f = \{\rho|v_i \in P_l\}$ where $\rho_i = \sum_{j \in P_i} \chi(d_{ij} - d_c)$. Similarly, for the diagonal subsets $S_k$, it outputs $\{\rho|v_i \in P_k\}$ where $\rho_i = \sum_{j \in P_i} \chi(d_{ij} - d_c)$. Finally, Basic-DDP runs another MapReduce job to combine the results of $\rho_i$ for all $1 \leq l \leq n$, i.e., $\rho = \sum_{i=1}^n \rho_i$.

**Step 2: Computing $\delta$.** As shown in Equation 2, the computation of $\delta_i$ requires to know all points’ density values $\Omega = \{\rho|v_i \in P\}$ as well as the distance values from $i$ to all points $j$, i.e., $\{d_{ij}|v_j \in P\}$. For the former, Basic-DDP uses the output of Step 1. For the latter, however, it may not be a good idea to store and reuse the precomputed distance matrix because the matrix can be very large and it can incur significant I/O cost on the underlying distributed file system (i.e., HDFS). Therefore, Basic-DDP recomputes distances.

The computation of $\delta$ values is similar to the blocking method for computing $\rho$ values. Given the entire density values set $\Omega = \{\rho|v_i \in P\}$, for a pair of $(S_k, S_l)$, the `reduce()` function computes the distance values $\{d_{ij}|v_i \in P_k, v_j \in P_l\}$, and outputs two sets $\Delta_i = \{\delta_i|v_i \in P_k\}$ where $\delta_i = \min_{j \in P_i} \chi(d_{ij} - d_c)$ and $\Delta_f = \{\delta_i|v_i \in P_l\}$ where $\delta_i = \min_{j \in P_i} \chi(d_{ij} - d_c)$. Besides, each point $i$’s upslope point $u_i = \arg \min_{u_i \in P_k \cup P_l} \rho (d_{in})$ is also recorded along with its $\delta_i$ value. Obviously, $\delta_i$ is not the final result. Another MapReduce job is required to select the smallest $\delta_i$ value among the candidates $\delta_i$, $l = 1, 2, \ldots, n$, i.e., $\delta_i = \min \delta_i$, and to record its corresponding upslope point.

**Step 3: Centralized Density Peaks Selection and Point Assignment.** We assume that the set $(\Omega)$ of $\rho$ values and the set $(\Delta)$ of $\delta$ values can fit into the main memory of a single machine\(^3\). Therefore, Basic-DDP computes a decision graph in the centralized fashion. Figure 1c shows an example decision graph. Users have the flexibility to select density peaks on the decision graph. Note that it is possible to design certain criteria for choosing the peaks automatically. However, we believe it is better to retain this user-algorithm interaction, since the visualized reference (i.e., decision graph) provides users with an opportunity to better understand the data and choose the preferred clustering result. This is a key feature that distinguishes DP from other clustering algorithms (e.g., Kmeans and DBSCAN), which require users to face the challenge of specifying key algorithm parameters in advance. Given the chosen density peaks (i.e., cluster centers), Basic-DDP follows the upslope point for each point to assign it to a cluster, as illustrated in Figure 1d.

---

\(^3\)Suppose $\rho$ is represented as a 32-bit integer and $\delta$ is represented as a double precision floating number. Then, one billion points require 12 GB main memory, which can easily fit into a mid-range server today.

---

**B. Cost Analysis and Improvement Opportunities**

From the above description, we see that the most expensive steps in Basic-DDP are the computation of $\rho$ (Step 1) and $\delta$ (Step 2). The blocking technique still has to send every point $\lceil \frac{n+1}{2} \rceil$ times during the shuffling phase in Step 1 as well as in Step 2. This incurs significant shuffle overhead especially when the point set $S$ is large. Moreover, Basic-DDP computes $\frac{N(N+1)}{2}$ distances in both Step 1 and Step 2. The computational cost is quadratic with respect to the total number of points $N$.

To improve performance, an ideal strategy is to partition $S$ into $n$ disjoint subsets $\{S_k|1 \leq k \leq n\}$ such that the $\rho$ and $\delta$ computation could be self-contained within each partition. First, distances are computed only inside a partition, $\{d_{ij}|v_i \in P_k, v_j \in P_k\}$. Second, to guarantee the correctness of $\rho$, the subset $S_k$ must contain each point $i$’s $d_c$-length neighbors. We have to put more points into every $S_k$ to form $S''_k$, i.e., $P_k = \bigcup \{j|v_j \in P_k, d_{ij} < d_c\}$. Note that, $S''_k \cup S''_l$ is typically not empty. Recent technology in KNN search [13] or triangle inequality might be employed to select additional points to include in each $S''_k$. However, the $\delta$ computation becomes infeasible. Each point $i$ in $S_k$ is only aware of the distance to the subset $S''_l$ of points, i.e., $\{d_{ij}|v_j \in P''_k\}$. The points with higher density are likely not in $S''_k$. Copying all the higher density points will incur excessive shuffle cost.

While the above ideal partitioning approach does not work, it inspires us to develop an alternative approximate solution, as will be detailed in the next section.

**IV. LSH BASED APPROACH WITH QUALITY CONTROL**

In this section, we propose an approximate distributed algorithm, LSH-DDP, for DP. Intuitively, a locality preserving partition strategy is desirable for DP. This is because the computation of $\rho_i$ is based on the neighbors within a distance of $d_c$ from point $i$, and the computation of $\delta$ looks for the nearest point with higher density. Hence, closer points play a more important role in the computation. As suggested by the name, LSH-DDP leverages Locality-Sensitive Hashing...
(LSH) [14] to partition points so that closer points are more likely to be assigned to the same partitions.

Figure 2 overviews LSH-DDP. To improve approximation accuracy, we partition the point set \( S \) using \( M \) LSH partition layouts, \( \mathcal{P}_1(S), \mathcal{P}_2(S), \ldots, \mathcal{P}_M(S) \). An LSH partition layout \( \mathcal{P}_m(S) \) is a partition of the data space. \( S \) is split into multiple partitions such that \( \mathcal{P}_m(S) = S_{m1}^m \cup S_{m2}^m \cup \ldots \), where \( S_{m1}^m \cap S_{m2}^m = \emptyset (\forall k \neq l) \). With a larger \( M \), it is more likely that points that are close will collide in the same partition in one of the \( M \) partition layouts.

As shown in Figure 2, LSH-DDP computes the distances of pairs of points within each partition \( S_{m_i}^m \), and derives a set \( \hat{\Omega}_{m_i}^{m_i} \) of approximate \( \rho \) values within partition \( S_{m_i}^m \). The computation on multiple \( \mathcal{P}_m(S) \) can be performed in parallel. Then, LSH-DDP aggregates the multiple approximations, \( \hat{\Omega}_{m_i}^{m_i} \), to obtain more accurate results, \( \hat{\Omega} \). The approximation of \( \delta \) values follows the same strategy. With the previously approximated \( \hat{\rho} \) values, LSH-DDP finds the slope point \( u_i \) for each point \( i \) and computes \( \delta_i \) within each partition. The multiple approximations, \( \hat{\Delta}_{m_i}^{m_i} \), are further aggregated to obtain more accurate results, \( \Delta \).

A. Step 1: LSH Partition

**LSH Background.** The Locality-Sensitive Hashing (LSH) function has the property that points that are closer to each other have a higher probability of colliding than points that are farther apart [14]. It has been widely adopted in solving approximate nearest neighbor search problem [15], [16], [17], [18], [19]. LSH is formally defined as follows.

**Definition 1. (LSH)** Given a distance \( r \), an approximation ratio \( c \) and two probability values \( P_1 \) and \( P_2 \), a hash function \( h : S \rightarrow U \) is called \( (r, cr, P_1, P_2) \)-sensitive if for any \( p_i, p_j \in S \)

- If \( d_{ij} \leq r \) then \( P[r | h(p_i) = h(p_j)] \geq P_1 \),
- If \( d_{ij} > cr \) then \( P[r | h(p_i) = h(p_j)] \leq P_2 \).

We pick \( c > 1 \) and \( P_1 \geq P_2 \). With these choices, nearby points (i.e. those within distance \( r \)) have a greater chance of being hashed to the same value than points that are far apart (i.e. those at a distance greater than \( cr \) away).

The commonly used LSH function for Euclidean distance is as follows [11]

\[
h(p) = \frac{a \cdot p + b}{w}
\]

where \( a \) is a \( d \)-dimensional random vector, each entry of which is chosen independently from a p-stable distribution [20], \( b \) is a real number chosen from \([0, w]\), and \( w \) is also a real number representing the width of the LSH function.

The distance-preserving property of LSH allows us to partition the set of points based on their hash values. If two points \( i \) and \( j \) are hashed to the same bucket, we know that \( i \) and \( j \) are close to each other with certain confidence. Therefore, we can assign them to the same partition. However, it is possible that two distant points happen to be hashed to the same bucket according to Equation (3). To reduce such false positives, a group \( G \) of \( \pi \) hash functions \( G = (h_1, h_2, \ldots, h_\pi) \) are employed. That is, only points sharing all the \( \pi \) hash values are placed in the same partition. Thus, each point \( i \) is labeled with \( G(p_i) = [h_1(p_i), h_2(p_i), \ldots, h_\pi(p_i)] \), which is considered as a partition id. Multiple partitions are formed and assigned to multiple workers\(^3\) for parallel processing. The resulting data partition is referred to as an LSH partition \( \mathcal{P} \).

The formal definition of LSH partition is as follows:

**Definition 2. (LSH Partition)** Given a set of points \( S \), and a group of hash functions \( G = (h_1, h_2, \ldots, h_\pi) \), an LSH partition is obtained by hashing every point \( p_i \in S \) using \( G \) and assigning \( p_i \) to the partition as identified by hash key \( G(p_i) = [h_1(p_i), h_2(p_i), \ldots, h_\pi(p_i)] \). The point set \( S \) is accordingly split into multiple disjoint subsets, i.e., \( \mathcal{P}(S) = S_1 \cup S_2 \cup \ldots \), where \( S_1 \cap S_2 = \emptyset, \forall k \neq l \).

However, it is also possible that points that are close happen to be hashed to different partitions, especially when \( \pi \) is large, incurring false negatives. To reduce the number of false negatives, we employ a combination of \( M \) hash groups, \( (G_1, G_2, \ldots, G_M) \). That is, the point set is partitioned in \( M \) different ways. Suppose by applying a hash group \( G_m \), we obtain an LSH partition \( \mathcal{P}_m(S) = S_{m1}^m \cup S_{m2}^m \cup \ldots \), where \( S_{m1}^m \cap S_{m2}^m = \emptyset, \forall k \neq l \). Similarly, by applying \( M \) groups of hash functions \( (G_1, G_2, \ldots, G_M) \), we will have \( M \) LSH partitions \( (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M) \) of the set \( S \). For example, Figure 3 illustrates two possible LSH partitions of point set \( S \).

We achieve multiple LSH partitions in the map phase. The map() function invocation on a point \( p_i \) computes \( M \) hash keys \( G_1(p_i), G_2(p_i), \ldots, G_M(p_i) \) and then sends the intermediate key-value pairs, \( (G_1(p_i), p_i), (G_2(p_i), p_i), \ldots, (G_M(p_i), p_i) \), to reducers. Each reduce() function will receive a subset \( S_{m_k}^m \) of points under a certain LSH partition \( \mathcal{P}_m(S) \). In this way, \( M \) LSH partitions are created.

B. Step 2: Approximating \( \rho \)

**Local Computation of \( \hat{\rho}_m^m \).** For a certain LSH partition \( \mathcal{P}_m(S) \), a subset \( S_{m_k}^m \) is shuffled to a reduce() function. The reduce() function first computes the distances between any pairs of points in \( S_{m_k}^m \). Then it computes a density value \( \hat{\rho}_m^m \) for each point \( i \), i.e., \( \hat{\rho}_m^m = \sum_{j \neq i} \chi(d_{ij} - d_i) \).

However, \( \hat{\rho}_m^m \) is not necessarily equal to \( \rho_i \). As shown in Figure 3a, point \( p_2 \) in LSH partition layout 1 is located near

\(^3\)That is, each hash bucket is assigned to a distributed computation unit, e.g. a reducer. Here, we use the term worker for ease of exposition.
the border line between $S_1$, $S_4$, and $S_5$. The computation of $\hat{\rho}_1^m$ is limited only to the points that are in $S_1$. However, it is clear that a large number of points that are close to $p_2$ are located in $S_4$ and $S_5$. Therefore, $\hat{\rho}_2^m < \rho$. The use of multiple hash groups mitigates the problem. As shown in Figure 3b, all $p_2$’s $d_c$-length neighbors reside in the same partition as $p_2$. Therefore, $\hat{\rho}_2^m = \rho$.

To study the probability of $\Pr[\hat{\rho}_i^m = \rho_i]$, we give the following two lemmas.

**Lemma 1.** Given a point $p_i$ and an LSH function $h(p_i) = \lfloor \frac{a \cdot p_i + b}{w} \rfloor$, for the points $\{p_j | j \in \mathbb{P}, d_{ij} \leq d_c\}$, the probability that all these points are hashed to the same bucket is as follows:

$$P_\rho(w, d_c) = \Pr[h(p_i) = h(p_j), \forall j \in \mathbb{P}, d_{ij} \leq d_c] \geq 1 - \frac{4d_c}{\sqrt{2\pi w}}.$$  

**Proof:**

Figure 4 depicts the idea of the proof intuitively. Let us consider a number line, where each point is a real number. $y_i = a \cdot p_i + b$ is a point on the number line. By floor dividing $w$, the number line is divided into a sequence of $w$-width slots. According to the LSH function, all the points in the same $w$-width slot share the same hash key. The points that are close to $p_i$ are all hashed to the points close to $y_i$ on the number line. The position of $y_i$ is important. If $y_i$ is close to the center of the slot, it is more likely that all $d_c$-length neighbors of $p_i$ are in the same slot.

According to the definition of $p$-stable distribution [11], given a $d$-dimensional random vector $a$ each entry of which is chosen independently from a standard gaussian distribution $\mathcal{N}(0, 1)$, for two points $p_i$ and $p_j$, the distance between their projections $|a \cdot p_i - a \cdot p_j|$ is distributed as $d_{ij} \cdot x$, where $x$ is the absolute value of a standard gaussian random variable. Therefore, for any $p_j$ where $d_{ij} < d_c$, we have $\max_j |y_i - y_j| = \max_j |a \cdot p_i - a \cdot p_j| < d_c \cdot x$.

Moreover, $y_i = a \cdot p_i + b$ is uniformly distributed in a certain slot. To ensure that $y_i$ and all its $d_c$-length neighbors are in the same slot, $y_i$ has to be located in the interval of $\lfloor ax + d_c \cdot x, (\alpha + 1)w - d_c \cdot x \rfloor$ for some $\alpha$, as shown in Figure 4. The probability that $y_i$ resides in such an interval is $\frac{1 - 2d_c \cdot x}{w}$. The probability density function of the absolute value of the standard gaussian distribution is $f_p(x) = \frac{2\pi^{-\frac{1}{2}}}{\sqrt{x^2 + \pi^2}^\frac{1}{2}}$, where $x > 0$. Therefore, the probability becomes $1 - \frac{2d_c \cdot x}{w} = \int_0^\infty (1 - \frac{2d_c \cdot x}{w})f_p(x)dx$, and a further calculation shows that the probability is $1 - \frac{4d_c}{\sqrt{2\pi w}}$.

**Lemma 2.** For an LSH partition $P_m$ with $\pi$ hash functions, we have $\Pr[\hat{\rho}_i^m = \rho_i] \geq P_\rho(w, d_c)$.

**Proof:**

According to Definition 2, a group of $\pi$ LSH functions $G = (h_1, h_2, \ldots, h_\pi)$ is applied to each point. Only points sharing all the $\pi$ hash values are placed in the same partition. Because each LSH function is independently and randomly selected according to Equation (3), we have:

$$\Pr[\hat{\rho}_i^m = \rho_i] = \Pr[G_m(p_i) = G_m(p_j), \forall j \in \mathbb{P}, d_{ij} \leq d_c] = \prod_{i=1}^\pi \Pr[h_i(p_i) = h_i(p_j), \forall j \in \mathbb{P}, d_{ij} \leq d_c] \geq P_\rho(w, d_c)^\pi.$$  

**Aggregation of Multiple $\hat{\rho}_i^m$.** The point set is partitioned in $M$ LSH partition layouts ($P_1, P_2, \ldots, P_M$). Accordingly, for each point $i$, we will obtain $M$ approximate density values ($\hat{\rho}_1^m, \hat{\rho}_2^m, \ldots, \hat{\rho}_M^m$). These candidates (that are retrieved from multiple distributed reducers) are aggregated in the second MapReduce job. Since $\hat{\rho}_i^m \leq \rho_i$, we choose $\hat{\rho}_i = \max_m \hat{\rho}_i^m$. We hope that the aggregate value is closer to the exact value.

Employing $M$ LSH partitions reduces the chances that a point’s $d_c$-length neighbors reside in different partitions. As a result, it reduces the number of false negatives, and thus significantly increases $\Pr[\hat{\rho}_i = \rho_i]$.

**Theorem 1.** With $M$ LSH partitions ($P_1, P_2, \ldots, P_M$), we have $\Pr[\hat{\rho}_i = \rho_i] \geq 1 - \left[1 - P_\rho(w, d_c)^\pi\right]^M$.

**Proof:**

$\hat{\rho}_i^m \leq \max_m \hat{\rho}_i^m \leq \hat{\rho}_i$. If $\max_m \hat{\rho}_i^m \neq \rho_i$, then $\forall m = 1, \ldots, M, \hat{\rho}_i^m \neq \rho_i$. From Lemma 2, $\Pr[\hat{\rho}_i^m = \rho_i] \geq P_\rho(w, d_c)^\pi$. Since $G_m(1 \leq m \leq M)$ is independently and randomly generated, we have the following:

$$\Pr[\hat{\rho}_i = \rho_i] = 1 - \prod_{i=1}^\pi (1 - \Pr[\hat{\rho}_i^m = \rho_i]) \geq 1 - \left[1 - P_\rho(w, d_c)^\pi\right]^M.$$  

**C. Step 3: Approximating $\delta$**

The computation of $\delta$ depends on $\rho$ values. Therefore, after Step 2, LSH-DDP associates each point $p_i$ with its approximate $\hat{\rho}_i$ value. Then, Step 3 follows the same idea as approximating $\rho$. LSH-DDP partitions the points using $M$ LSH layouts $P_1, P_2, \ldots, P_M$ with the map() function. Then it performs local computation for $\delta_i^m$ values as follows.
Local Computation of $\hat{\delta}_m^i$. Let us consider a reduce() function working on a partition $S_m^i$ in a certain LSH partition layout $P_m$. LSH-DDP computes the distances between all pairs of points in $S_m^i$. Then, using the approximate density $\hat{\rho}_i$, we approximate $\hat{\delta}_m^i = \min_{j \in m, \hat{\rho}_i > \hat{\rho}_j} d_{ij}$ for any $i \in m$. For the point with the highest density in $S_m^i$, i.e., point $i = \arg \max_{j \in m, \hat{\rho}_i} \hat{\rho}_i$, we set $\hat{\delta}_m^i = \infty$.

However, even though the approximated $\hat{\rho}_i$ were exactly equal to $\rho_i$, $\hat{\delta}_m^i$ might not be equal to $\delta_i$, since the computation is constrained within a subset of points. For example, in LSH partition layout 1 as shown in Figure 5a, since point $p_2$'s real upslope point resides in a different partition, the local $\delta$ approximation returns a wrong result, an incorrect upslope point on another density mountain. Fortunately, in LSH partition layout 2 as shown in Figure 5b, $p_2$ and its upslope point are assigned in the same partition, and the correct $\delta_2$ can be computed.

Assume $\hat{\rho}_i = \rho_i$, we study the probability of $\Pr[\hat{\delta}_m^i = \delta_i]$ in the following lemmas. First, based on the property of LSH and $p$-stable distribution (refer to Datar’s paper [11]), we have Lemma 3.

**Lemma 3.** Given a point $p_i$ and an LSH function $h(p_i) = \lfloor z + p_i k \rfloor$, suppose $p_i$'s upslope point is $p_{ui}$ (if exist) and $d_{ui}$, is the distance from $i$ to $u_i$ (i.e., $d_{u_i} = \delta_i$), we have

$$P_{\delta_i}(d_{u_i}, w) = \Pr[h(p_i) = h(p_{ui})] = \int_{0}^{w} \frac{1}{d_{u_i}} f_p(x) (1 - \frac{x}{w})dx = 2 \cdot \text{norm} (\frac{w}{d_{ui}}) - 1 - \frac{2d_{ui}}{\sqrt{2\pi w}} (1 - e^{-\frac{w^2}{2d_{ui}^2}}),$$

where $f_p(x)$ denotes the probability density function of standard gaussian distribution, and $\text{norm}(\cdot)$ is the cumulative distribution function (cdf) for a random variable that is distributed as $N(0,1)$.

**Lemma 4.** Suppose point $p_i$'s real upslope point is $p_{ui}$ (if exist), by a certain LSH partition $P_m$ with $\pi$ hash functions, we have $\Pr[\hat{\delta}_m^i = \delta_i] = P_{\delta_i}(d_{ui}, w)^\pi$.

**Proof:** Point $i$’s upslope point $u_i = \arg \min_{j \in m, \hat{\rho}_i > \hat{\rho}_j} d_{ij}$, so that $\hat{\delta}_m^i = \min_{j \in m, \hat{\rho}_i > \hat{\rho}_j} d_{ij}$. On the other hand, suppose $i \in \mathbb{P}_m$, we have $\hat{\delta}_m^i = \min_{j \in \mathbb{P}_m, \hat{\rho}_i > \hat{\rho}_j} d_{ij}$ and the approximate upslope point $\hat{u}_i = \arg \min_{j \in \mathbb{P}_m, \hat{\rho}_i > \hat{\rho}_j} d_{ij}$. Since we assume $\hat{\rho}_i = \rho_i$ and $\hat{\rho}_j = \rho_j$, the probability $\Pr[\hat{\delta}_m^i = \delta_i] = \Pr[\hat{u}_i = u_i]$ is equal to the probability that point $i$ and $u_i$ are hashed to the same bucket $S_m^i$, i.e., $\Pr[i \in \mathbb{P}_m, u \in \mathbb{P}_m] = \Pr[G_{m}(p_i) = G_{m}(p_{ui})]$. Due to the fact that each LSH function is independently and randomly selected according to Equation (3), the probability $\Pr[\hat{\delta}_m^i = \delta_i]$ is correspondingly as follows:

$$\Pr[\hat{\delta}_m^i = \delta_i] = \prod_{t=1}^{\pi} \Pr[h_t(p_i) = h_t(p_{ui})] = P_{\delta_i}(d_{ui}, w)^\pi.$$  

**Aggregation of Multiple $\hat{\delta}_m^i$.** For each point $i$, we will obtain $M$ approximate values $(\hat{\delta}_1^i, \hat{\delta}_2^i, \ldots, \hat{\delta}_M^i)$ in various LSH layouts. We hope that at least one of them is equal or close to the exact value. According to Equation (2), the smallest one is more likely to be the exact $\delta_i$. Therefore, we aggregate these approximate $(\hat{\delta}_1^i, \hat{\delta}_2^i, \ldots, \hat{\delta}_M^i)$ in a MapReduce job and set $\delta_i = \min_m \hat{\delta}_m^i$. Similar to $\rho$ approximation, the probability $\Pr[\delta_i = \delta_i]$ is enlarged as follows.

**Theorem 2.** Given a point $i$’s upslope point $u_i$, with $M$ LSH partitions $(P_1, P_2, \ldots, P_M)$, we have $\Pr[\delta_i = \delta_i] = 1 - [1 - \Pr[\delta_i = \delta_i)]^M = 1 - [1 - \Pr[\delta_i = \delta_i)]^M$.

From Theorem 2, we can see that the probability $\Pr[\delta_i = \delta_i]$ depends $d_{ui}$, or $\delta_i$, i.e., the distance from point $i$ to its “nearest” neighbor with higher density. Generally speaking, as shown in Figure 1d, $d_{ui}$ is small for most points and therefore the probability $\Pr[\delta_i = \delta_i]$ is correspondingly high. However, this is not true if point $i$ is distant from its upslope point (i.e., $d_{ui} = \delta_i$ is large). This leads to a very interesting situation. $\delta_i$ is more accurate for smaller $\delta_i$ but inaccurate for larger $\delta_i$.

We leverage the characteristic of DP to deal with the inaccurate $\delta_i$. Since the density peaks have large $\delta_i$, they are distant from each other and are unlikely to be hashed to the same bucket under a locality-preserving hash function. Therefore, LSH-DDP may wrongly recognize these density peaks as the absolute density peak in a partition and therefore assign $\delta_i = \infty$. (The infinite $\delta_i$ will be rectified as the finite $\max \delta_i$ value before drawing them on the decision graph.) Interestingly, this will not affect the cluster assignment significantly. An accurate approximation gives an abnormally large $\delta_i$, while an inaccurate approximation results in an infinite value. In either case, the points will probably be chosen as density peaks in the density peak selection step. This effect is confirmed in our experiments (see Section VI-C). In fact, there is an unexpected side benefit—The infinite $\delta_i$ makes it easier to identify the density peaks in the decision graph.

**D. MapReduce Implementation and Cost Analysis**

To sum up, the MapReduce implementation of LSH-DDP consists of four MapReduce jobs (for approximating $\rho$ and $\delta$) and a centralized program (for density peak selection and point assignment). The first job performs LSH partition (Map1) and local computation of $\hat{\rho}_m^i$ (Reduce1). The second job aggregates the $\rho_m^i$ values (Reduce2). Similarly, LSH partition (Map3) and local computation of $\hat{\delta}_m^i$ (Reduce3) are carried out in the third job. The fourth job aggregates and the $\hat{\delta}_m^i$ values (Reduce4).

For $M$ LSH partition layouts, LSH-DDP shuffles $M$ copies of each point in the first and third job, respectively. It also aggregates $\Omega^m, m = 1, 2, \ldots, M$ in the second job and $\Delta^m, m = 1, 2, \ldots, M$ in the fourth job. Therefore, the total shuffle cost is $2M \cdot |S| + M \cdot |\Omega| + M \cdot |\Delta|$.

LSH-DDP avoids the computation of the entire distance matrix. It breaks down the matrix into multiple small sub-matrices and performs distance computation within every sub-matrix. Hence, our approach reduces the number of distance measurements but pays the extra cost for computing LSH values.
V. PARAMETERS TUNING

To launch LSH-DDP, there are three parameters to be determined, the number of hash groups (i.e., $M$), the number of hash functions in each group (i.e., $\pi$), and the width of hash function (i.e., $w$). A reasonable selection of these parameters is crucial to approximation accuracy and performance. The determination of these three parameters is an optimization problem. The optimization problem takes two factors into account: the accuracy of result and the cost (including shuffle cost and computational cost). In this section, we discuss the parameter determination with a certain accuracy expectation.

A. Problem Formulation

Accuracy. The LSH-DDP algorithm makes the cluster assignment for each point based on their approximated $\hat{\rho}$ and $\hat{\delta}$ values. The accuracy of each point assignment should be $\Pr[\hat{\rho}_i = \rho_i] \cdot \Pr[\hat{\delta}_i = \delta_i]$. However, according to Theorem 2 the accuracy of an approximated $\hat{\delta}_i$ greatly depends on the real $\delta_i$, which is unknown in advance. We would like to only analyze the accuracy of the approximated $\hat{\rho}$ values. According to Theorem 1, we define the expected accuracy as follows:

$$A(w, \pi, M) = 1 - \left[1 - P_\rho(w, d_c)^\pi\right]^M,$$

(5)

where $P_\rho(w, d_c)$ is defined in (4) and $d_c$ is fixed.

Shuffle Cost. To simplify the analysis, we only consider the major shuffle cost, including (a) the shuffle cost of $S$ (i.e., $M \cdot |S|$) and (b) the shuffle cost of distance values (i.e., $\sum_{m=1}^{M} \sum_{k=1}^{K_m} |D_{mk}|$, where $D_{mk}$ is the pair-wise distance matrix of subset $S_k$ and $K_m$ is the number of partitions in the $m$th LSH layout). Since in general either $|\Omega|$ or $|\Delta|$ is much smaller than $|S|$ and $\sum_{m=1}^{M} \sum_{k=1}^{K_m} |D_{mk}|$, the shuffle cost of $\hat{\rho}$ values set $\Delta$ and $\delta$ values set $\Omega$ can be ignored. Suppose $N_k = |S_k|$, the shuffle cost (or the size of shuffled data) is

$$C_s(w, \pi, M) = M \cdot |S| + \sum_{m=1}^{M} \sum_{k=1}^{K_m} (N_k^m)^2 \cdot e,$$

(6)

where $e$ is the entry size of $D_{mk}^m$ (e.g., $e = 8$ for double numbers). We further have the expected shuffle cost:

$$E[C_s(w, \pi, M)] = M \cdot \left(|S| + \sum_{k=1}^{K} N_k^2 \cdot e\right),$$

(7)

where $K$ is the expected number of partitions, $N_k$ is the expected size of a partition, and $\sum_{k=1}^{K} N_k = N$.

Computational Cost. The computations occur in the LSH partition, the distance calculations, and $\hat{\rho}$ and $\hat{\delta}$ approximations. Among them, the distance calculations are the most costly. We consider the number of distance calculations as the computational cost, which are from the generations of distributed distance matrices $D_{mk}^m$, $\forall k, 1 \leq m \leq M$ where $D_{mk}^m$ is a $N_k^m \times N_k^m$ matrix. Therefore, the expected computational cost can be represented as

$$E[C_c(w, \pi, M)] = E\left[\sum_{m=1}^{M} \sum_{k=1}^{K_m} (N_k^m)^2\right] = M \sum_{k=1}^{K} N_k^2,$$

(8)

where $\sum_{k=1}^{K} N_k = N$.

We aim to minimize both the shuffle cost and the computational cost while satisfying a certain accuracy guarantee. This is a multi-objective optimization problem. Apparently, this multi-objective optimization problem can be transformed to a single objective optimization problem by unifying the costs into time cost. Suppose the ratio of the time unit for shuffling each byte to the time unit for each distance calculation is $\mu$.

The single optimization problem can be described as follows:

$$\begin{align*}
\text{min.} & \quad \mu \cdot M \cdot \left(|S| + \sum_{k=1}^{K} N_k^2 \cdot e\right) + M \sum_{k=1}^{K} N_k^2 \\
\text{s.t.} & \quad 1 - \left[1 - P_\rho(w, d_c)^\pi\right]^M \geq \text{required value}
\end{align*}$$

(9)

We can see that the parameters $w, \pi, M$ play a key role in solving this optimization problem.

B. Analysis of Parameter Variations

From Theorem 1, it is obvious that the accuracy increases with the increase of $M$ and $w$, and with the decrease of $\pi$.

For both shuffle and computational cost, it is obvious that they increase with the increase of $M$ and the increase of the sum of squares $\sum_{k=1}^{K} N_k^2$, where $\sum_{k=1}^{K} N_k = N$. The value of $\sum_{k=1}^{K} N_k^2$ depends on the data distribution and affected by $w$ and $\pi$. We do not make any assumption of data distribution and would like to only study the relationship between $\sum_{k=1}^{K} N_k^2$ and the parameters $w$ and $\pi$. Intuitively, small $w$ leads to narrow partition, and large $\pi$ leads to a fine partition of the space. That is, small $w$ and large $\pi$ lead to a large number of small $N_k$ and probably small $\sum_{k=1}^{K} N_k^2$. Therefore, the shuffle and computational cost should decrease with the decrease of $M$ and $w$, and with the increase of $\pi$.

Based on the above analysis, we see that the impacts of the three parameters on the expected accuracy and on performance are reverse. As a result, there is a tradeoff between approximation accuracy and performance.

In our implementation, users are allowed to launch the approximate algorithm with a certain accuracy confidence level $A$. Users also need to specify the two positive integer parameters $M$ and $\pi$. Given $M, \pi$, to satisfy the expected accuracy requirement $A$, we can find the minimal feasible LSH partition width $w$ by solving Equation (5). In Section VI-E, we will empirically show the effect of these parameters.

VI. EXPERIMENTAL EVALUATION

In this section, we evaluate the accuracy and performance of our proposed LSH-DDP algorithm.

A. Experimental Setup

Machine Configuration. The experiments are performed both on our local cluster of machines and on EC2 cloud. Our local cluster contains 5 machines (1 master and 4 slaves), each equipped with an Intel i5-4690 3.3G 4-core CPU, 4GB memory, running Hadoop 1.2.1. The EC2 cluster consists of 64 m1.medium instances.
Table II. Data Sets

<table>
<thead>
<tr>
<th>data set</th>
<th># instances</th>
<th># dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation</td>
<td>788</td>
<td>2</td>
</tr>
<tr>
<td>S2</td>
<td>5000</td>
<td>2</td>
</tr>
<tr>
<td>Facial</td>
<td>27,936</td>
<td>300</td>
</tr>
<tr>
<td>KDD</td>
<td>145,751</td>
<td>74</td>
</tr>
<tr>
<td>3Dspatial</td>
<td>434,874</td>
<td>4</td>
</tr>
<tr>
<td>BigCross500K</td>
<td>500,000</td>
<td>57</td>
</tr>
<tr>
<td>BigCross</td>
<td>11,620,300</td>
<td>57</td>
</tr>
</tbody>
</table>

B. Clustering Quality of DP vs. Previous Algorithms

Before evaluating LSH-DDP, we would like to understand DP’s advantages over previous clustering algorithms. Figure 8 (a) depicts the ground truth. Figure 8 (b)–(f) compare the clustering results of DP and four previous representative algorithms, including agglomerative hierarchical cluster (connectivity-based), K-means (centroid-based), EM (distribution-based), DBSCAN (density-based). Table III lists the key features of the clustering algorithms.

The input parameter $d_c$ of DP_Cluster is estimated as the 2% position of the ascending ordered distance set (see Section III-A). For the algorithms that take the number of clusters $k$ as the input parameter, $k$ is set to the number of clusters in the ground truth. DBSCAN’s input parameter $\varepsilon$ is configured to be equal to $d_c$, and the minimum number of points in a cluster is set to 1.

Figure 8 shows the results for the Aggregation data set, which is a shaped data set. In addition, we compare the algorithms using 7 other shaped data sets and see similar trends. For space limitation, we focus only on the Aggregation data set here. There are seven clusters in the ground truth. The hierarchical and the DBSCAN algorithms correctly identify three clusters, but make mistakes for the other clusters. The two algorithms cannot easily separate clusters that are close to each other. On the other hand, K-means and EM can correctly identify four clusters, while they work poorly for non-oval shapes. In contrast, DP correctly identifies all the seven clusters, achieving the best clustering results.

C. Accuracy of LSH-DDP

In order to visualize the cluster result, we run Basic-DDP and LSH-DDP on a medium sized 2D data set, S2. In LSH-DDP, the expected accuracy $A$ is set to 0.99. We set $M = 10$, $\pi = 3$ accordingly.

Figure 7(a) and (b) show the decision graphs for Basic-DDP and LSH-DDP, respectively. The decision graph of Basic-DDP is generated using the computed accurate $(\rho, \delta)$ values, while the decision graph of LSH-DDP is drawn using the approximate $(\hat{\rho}, \delta)$ values. We show a possible selection of peaks on the two decision graphs (i.e. all points that satisfy $\rho > 14$ and $\delta > 40$).

We see that the number of peaks chosen in LSH-DDP is the same as in Basic-DDP. Their $\rho$ values are roughly the same, while the $\delta$ values of some points in LSH-DDP appear at the top of the chart. What happens is that for such a point, LSH-DDP cannot find any points with larger $\rho$ in the LSH generated local partition. As a result, the computation wrongly regards the point as the absolute density peak and assigns the maximum distance to its $\delta$. This is expected as discussed in Section IV-C. However, for the purpose of clustering, it is sufficient to identify such points as candidate cluster centers. Consequently, the cluster result looks good. As shown in Figure 8, the cluster results of Basic-DDP and LSH-DDP are almost the same. Differences exist only at boundary points and/or for deciding whether a set of points should be clustered at a finer granularity.

We further evaluate the accuracy of LSH-DDP for the large real world data sets. Since the cluster result of a large multi-dimensional data set cannot be easily visualized, we focus on measuring the accuracy of $A$. We define two metrics $\tau_1$ and $\tau_2$ to characterize the accuracy of the approximation.

$$\tau_1 = \frac{1}{N} \sum \left( \frac{|i|}{\hat{d}_i, P_i = p_i} \right)$$

is the fraction of correctly approximated $\rho$ values. Larger $\tau_1$ means that the $\hat{\rho}$ of more points are approximated correctly. When every $\hat{\rho}$ is approximated correctly, $\tau_1 = 1$. $\tau_2 = 1 - \frac{1}{N} \sum |\hat{\rho} - \rho|$. It is 1 minus the normalized absolute error. Hence, the smaller the error, the larger the $\tau_2$. When the error approaches 0, $\tau_2$ grows to 1.

We have run experiments for all the four real world data sets and see similar results. For space limitation, we focus on BigCross500K in Figure 9. On the x-axis, we vary the expected accuracy $A$. Given a $A$, we set the LSH-DDP parameters accordingly and then run the algorithm. The resulting $\tau_1$ and $\tau_2$ are reported in Figure 9(a) and (b), respectively. From the figures, we see that both $\tau_1$ and $\tau_2$ increase as the expected accuracy $A$ increases. Both metrics approach 1 as $A$ approaches 1. Note that the definition of $\tau_1$ corresponds to the accuracy target. It is clear that $\tau_1$ points reside closely around the diagonal line. This shows that LSH-DDP has successfully realized the accuracy target as specified by $A$.

D. Performance of LSH-DDP

Runtime. We run Basic-DDP and LSH-DDP on four data sets, i.e. Facial, KDD, 3Dspatial, and BigCross500K on the local cluster of machines. The parameters of LSH-DDP are set as follows: $A = 0.99, M = 10, \pi = 3$, and the block size parameter of Basic-DDP is set as 500. As shown in Figure 10(a), LSH-DDP is dramatically better than Basic-DDP, achieving 1.7–24x speedups. Moreover, the larger the data set size, the more benefit LSH-DDP brings. The 24x speedup is achieved for the largest data set. To understand the performance benefit, we delve into the communication cost and the computation cost in the following.
TABLE III. KEY FEATURES OF VARIOUS CLUSTERING ALGORITHMS

<table>
<thead>
<tr>
<th></th>
<th>iterative</th>
<th>cluster shape assumption</th>
<th>predefined # of clusters</th>
<th>complexity</th>
<th>embarrassingly parallel</th>
<th>interactivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>hierarchical</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>$O(n^3)$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>k-means</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>$O(n \cdot k \cdot I)$</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>EM</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>$O(n \cdot k \cdot I)$</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>$O(n^2)$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>DP</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>$O(n)$</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Shuffle Cost. Figure 10(b) compares the total amount of data shuffled in the MapReduce jobs of Basic-DDP and LSH-DDP. Basic-DDP has to send every point to every other point using the Map and the shuffle stages. In contrast, LSH-DDP sends only the local results computed from LSH local partitions, thereby avoiding the quadratic communication cost. As shown in Figure 10(b), LSH-DDP reduces the amount of shuffled data by 5–87x compared to Basic-DDP. Since the amount of shuffled data in Basic-DDP grows quadratically, LSH-DDP sees larger savings for larger data sets.

Computational Cost. Figure 10(c) reports the number of distance measurements computed in Basic-DDP vs. LSH-DDP. The computation cost of Basic-DDP grows quadratically, while LSH-DDP sees only linear growth. Consequently, the savings of LSH-DDP grow as the input data set size increases. We see a 1.7–6.1x savings for computational cost.

Comparison to EDDPC. EDDPC [21] is a recently work on parallelizing DP algorithm. It leverages Voronoi diagram and careful data replication/filtering to reduce the huge amount of useless distance measurement cost and data shuffle cost. Rather than approximation, EDDPC will return the exact $\delta$ and $\rho$ values. We also compare our approach with EDDPC for clustering the BigCross500K data set. The results are listed in Table IV. We can see that LSH-DDP requires less runtime and much less shuffled data though higher number of distance measurements (i.e., $\# \text{dist.}$). Note that, LSH-DDP will result in even higher efficiency with lower accuracy requirement.

Clustering Large Data Set on EC2. In order to see the performance of LSH-DDP for very large data sets in large scale distributed environments, we run the algorithms on the BigCross data set with 11.6 million points on an EC2 virtual cluster of 64 m1.medium instances. Basic-DDP takes 91.2
hours to complete, while LSH-DDP takes only 1.3 hours. Compared to Basic-DDP, LSH-DDP achieves a 70x speedup.

Additionally, we run the popular K-means algorithm for 100 iterations on the BigCross data set. Figure 11 shows the runtime of K-means after every iteration. We also compare the runtime of LSH-DDP with K-means. We see that the runtime of LSH-DDP roughly corresponds to the 24-th iteration in K-means. In some certain situations, K-means takes more iterations to converge. It is also possible to lower the accuracy requirement to speedup LSH-DDP further. Therefore, LSH-DDP has comparable performance with K-means.

E. Effect of LSH Parameters

We study the effect of LSH-DDP’s parameters $M$ and $\pi$. We run LSH-DDP on the BigCross500K data set on our local cluster of machines. We set $\mathcal{A} = 0.99$, then vary $M$ and $\pi$. Figure 12(a) and (b) report the impact of the parameters on the runtime and the accuracy metric $\tau_2$, respectively. As shown in Figure 12(a), when $\pi = 3$, the runtime increases as $M$ increases. However, this is not true for larger $\pi$. When $\pi = 20$, the trend actually reverses. This is because that the workload is quite skewed when $M$ is small and $\pi$ is large, leading to degraded performance. Figure 12b shows the impact of the choice of parameters on $\tau_2$. When $M$ is less than 5, $\tau_2$ is unexpectedly low and this could reduce the quality of the clustering result. On the other hand, when $M$ is larger than 5, $\tau_2$ is stable, achieving 99% accuracy for almost all cases. Taking both runtime and accuracy into consideration, we recommend to set $M = [10, 20]$ and $\pi = [3, 10]$.

VII. RELATED WORK

Clustering Techniques. Previous clustering algorithms include connectivity based clustering (e.g., hierarchical clustering [22]), centroid-based clustering (e.g., k-means [8]), distribution-based clustering (e.g., EM clustering [10]), and density-based clustering (e.g., DBSCAN [9]). As described in Section 1, Density Peaks [1] is a newly proposed clustering algorithm. DP has several distinctive advantages over previous clustering algorithms: It does not require a priori knowledge, it supports arbitrarily shaped clusters, it is deterministic, and it provides a 2D representation to visualize the input data. As a result, DP has already been employed in a wide variety of applications [2], [3], [4], [5], [6], [7]. Moreover, researchers in the AI community are interested in extending DP in various aspects [23], [24], [25], [26], [27]. In this paper, we propose and evaluate LSH-DDP, an efficient distributed DP algorithm. While we focus on the original DP, we believe that it is feasible to modify our solution to support variants of DP.

MapReduce Parallelization of Sequential Algorithms. As a popular distributed programming paradigm, MapReduce has been used in parallelizing a wide range of algorithms for processing big data. This includes text processing [28], crowdsourcing [29], kNN join [30], nonnegative matrix factorization (NMF) [31], and spatial data query [32]. In this paper, we also choose MapReduce as the programming model for distributed DP algorithms.

All-Pair Computation in MapReduce. The computation of $\rho$ and $\delta$ is related to a set of problems where computation is required for all pairs of input data elements. Kiefer et al. reduced the communication overhead of all-pair computation by using replication of set elements to enable partitioning, and by aggregating the results gathered for different copies of an element [33]. Ture et al. presented an LSH-based scalable MapReduce implementation of the sort-based sliding window algorithm for extracting pair-wise similarity [34]. In this paper, we also employ LSH in our solution. Note that Ture et al.’s work cannot be applied since $\rho$ and $\delta$ are not similarity measurements.

Approximate Algorithms using LSH. The LSH method was first proposed by Datar [11]. Since its introduction, LSH has been used to optimize a wide range of applications. Stupar et al. exploited LSH to answer kNN query [35]. Zhang et al. extended this work to solve kNN join problem [30]. Liu et al. employed LSH to optimize distributed graph summarization [36]. Yu et al. supported scalable content-based music retrieval through LSH [37]. Pillutla gave an approximate algorithm for distance based outlier detection using LSH [38]. We also employ LSH in our solution. As discussed in Section 1, there are several challenges in applying LSH to DP. We leverage

\footnote{Note that the number of iterations to convergence depends on the underlying data, the initial states, and the stop criterion in K-means.}
the characteristics of DP to deal with the fact that some δ values cannot be directly approximated in local partitions. We present formal analysis of LSH-DDP, and show that the approximation quality and the runtime can be controlled by tuning the parameters of LSH-DDP.

VIII. CONCLUSION

In this paper, we present an efficient distributed algorithm LSH-DDP for Density Peaks clustering. We perform theoretical analysis of LSH-DDP, which allows users to specify the expected approximation accuracy. Compared to the naïve MapReduce implementation (Basic-DDP), LSH-DDP significantly reduces the amount of shuffled data and the computational cost, thereby achieving a factor of 1.7–70x speedups when clustering large real-world data sets. It also achieves 2x speedup over EDDPC with a very high accuracy requirement. Compared to the popular K-means clustering algorithm, LSH-DDP has comparable or better performance. In conclusion, LSH-DDP is a promising solution that makes DP algorithm feasible for clustering large real-world data sets. We hope that LSH-DDP will help introduce the benefits of DP to a larger number of real-world application problems in the big data era.

REFERENCES


