Distributed Local Outlier Detection in Big Data

Yizhou Yan*
Worcester Polytechnic Institute
yyan2@wpi.edu

Lei Cao*
Massachusetts Institute of Technology
lcao@csail.mit.edu

Caitlin Kuhlman
Worcester Polytechnic Institute
cakuhlman@wpi.edu

Elke Rundensteiner
Worcester Polytechnic Institute
rundenst@wpi.edu

ABSTRACT

In this work, we present thefi rst distributed solution for the Local Outlier Factor (LOF) method – a popular outlier detection technique shown to be very effective for datasets with skewed distributions. As datasets increase radically in size, highly scalable LOF algorithms leveraging modern distributed infrastructures are required. This poses significant challenges due to the complexity of the LOF definition, and a lack of access to the entire dataset at any individual compute machine. Our solution features a distributed LOF pipeline framework, called DLOF. Each stage of the LOF computation is conducted in a fully distributed fashion by leveraging our invariant observation for intermediate value management. Furthermore, we propose a data assignment strategy which ensures that each machine is self-sufficient in all stages of the LOF pipeline, while minimizing the number of data replicas. Based on the convergence property derived from analyzing this strategy in the context of real world datasets, we introduce a number of data-driven optimization strategies. These strategies not only minimize the computation costs within each stage, but also eliminate unnecessary communication costs by aggressively pushing the LOF computation into the early stages of the DLOF pipeline. Our comprehensive experimental study using both real and synthetic datasets confirms the efficiency and scalability of our approach to terabyte level data.

KEYWORDS

Local outlier; Distributed processing; Big data

ACM Reference format:
Yizhou Yan, Lei Cao, Caitlin Kuhlman, and Elke Rundensteiner. 2017. Distributed Local Outlier Detection in Big Data. In *Proceedings of KDD ’17, Halifax, NS, Canada, August 13-17, 2017,* 10 pages.
https://doi.org/10.1145/3097983.3098179

1 INTRODUCTION

Motivation. Outlier detection is recognized as an important data mining technique [3]. It plays a crucial role in many wide-ranging applications including credit fraud prevention, network intrusion detection, stock investment, tactical planning, and disastrous weather forecasting. Outlier detection facilitates the discovery of abnormal phenomena that may exist in the data, namely values that deviate significantly from a common trend in the data [12].

One popular outlier detection method, the Local Outlier Factor (LOF) [7], addresses challenges caused when data is skewed and outliers may have very different characteristics across data regions. Traditional outlier detection techniques such as distance [14] and neighbor-based methods [5] tend to fail in such cases, because they assume that the input dataset exhibits a uniform distribution. Thus they detect outliers based on the *absolute density* of each point (the distance to its neighbors). LOF is able to better detect outliers in real world datasets which tend to be skewed [18], outperforming other algorithms in a broad range of applications [3, 15].

LOF is a complex multi-phase technique. It detects outliers by identifying unusual phenomena *in relation to* other data observations around them. Specifically, a point p is considered to be an outlier if its *local density* significantly differs from the *local density* of its k nearest neighbors (kNN). To determine this, a number of intermediate values must be computed for p and its kNN. The *k-distance* and *reachability distance* values are used to compute the *local reachability density* (LRD) of each point, and in turn this is used to compute an *outlierness score* for p, denoted as the *LOF score*.

Unfortunately, the centralized LOF algorithm [7] can no longer satisfy the stringent response time requirements of modern applications, especially now that the data itself is inherently becoming more distributed. Therefore, the development of distributed solutions for LOF is no longer an option, but a necessity. Nevertheless, to the best of our knowledge, no distributed LOF work has been proposed. In this work we focus on designing LOF algorithms that are inherently parallel and work in virtually any distributed computing paradigm. This helps assure ease of adoption by others on popular open-source distributed infrastructures such as MapReduce [1] and Spark [10].

Challenges. Designing an efficient distributed LOF approach is challenging because of the complex definition of LOF. In particular, we observe that the LOF score of each single point p is determined by many points, namely its k nearest neighbors (kNN), its kNN’s kNN, and its kNN’s kNN’s kNN – in total $k + k^2 + k^3$ points. In a distributed system with a shared nothing architecture, the input dataset must be partitioned and sent to different machines. To identify for each point p all the points upon which it depends for LOF computation and send them all to the same machine appears to be a sheer impossibility. It effectively requires us to solve the LOF problem before we can even begin to identify an ideal partition.

*Authors contributed equally to this work.

KDD ’17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098179

One option to reduce the complexity of distributed LOF computation is to calculate LOF in a step by step manner. That is,first only the kNN of each point are calculated and materialized. Then, the kNN values are used in two successive steps to compute the *LRD* and *LOF* values respectively. For each of these steps, the intermediate values would need to be *updated* for the next step of computation. In a centralized environment such data can be indexed in a global database table and efficiently accessed and updated. In a shared-nothing architecture common for modern distributed systems, even if it were possible to efficiently compute the kNN of each data point, no global table exists that can accommodate this huge amount of data nor support continuous access and update by many machines. Therefore an effective distributed mechanism must be designed to manage these intermediate values stored across the compute cluster.

Proposed Approach. In this work, we propose thefi rst distributed LOF computation solution, called DLOF. As foundation, we first design a distributed LOF framework that conducts each step of the LOF computation in a highly distributed fashion. The DLOF framework is built on the critical *invariant observation*. Namely, in the LOF computation process of point p, although each step requires different types of intermediate values, these values are always associated with a *fixed* set of data points. Leveraging this observation, our *support-aware assignment* strategy ensures the input data and required intermediate values are co-located on the same machine in the computation pipeline.

Second, we propose a data-driven approach named *DDLOF* to bound the *support points*, or potential kNN, of the core points in each data partition $\mathbb{P}_i$. DDLOF effectively minimizes the number of support points that introduce data duplication, while still guaranteeing the correctness of kNN search. DDLOF defeats the commonly accepted understanding in the literature that efficient distributed algorithms should complete the analytics task in as few rounds as possible [2]. It instead adopts a multi-round strategy that decomposes kNN search into multiple rounds, providing an opportunity to *dynamically* bound each partition using data-driven insights detected during the search process itself. DDLOF reduces the data duplication rate from more than 20x the size of the original dataset in the state-of-the-art approach to 1.

Moreover, based on the crucial *convergence* observation, we succeed to further enhance *DDLOF* by introducing our *early termination* strategy, henceforth called *DDLOF-Early*. Instead of calculating the LOF score step by step, *DDLOF-Early* aggressively pushes the LOF computation into the early stage of the pipeline and completes the LOF computation of any point as early as possible. Eliminating the points that have succeeded to acquire its LOF at the earliest possible stage of the DLOF process reduces both communication and computation costs. Therefore *DDLOF-Early* succeeds to scale the LOF technique to the true big data realm.

Contributions. The key contributions of this work include:

• We propose *thefirst* distributed LOF approach inherently parallel and deployable on virtually any distributed infrastructure.

• We design a multi-step pipeline framework called *DLOF* that by leveraging our *invariant observation* computes LOF scores in a highly distributed fashion.

• Our data-driven strategy *DDLOF* effectively minimizes the number of support points using insights derived from the multiphase search process itself.

• Driven by the *convergence observation*, we optimize the DDLOF solution by aggressively applying an *early termination* mechanism to reduce the communication and computation costs.

• Experiments demonstrate the effectiveness of our proposed optimization strategies and the scalability to terabyte level datasets.

2 PRELIMINARIES

Local Outlier Factor (LOF) [7] introduces the notion of *local outliers* based on the observation that different portions of a dataset may exhibit very different characteristics. It is thus often more meaningful to decide on the outlier status of a point based on the points in its neighborhood, rather than some strict global criteria. *LOF* depends on a single parameter k which indicates the number of nearest neighbors to consider.

Definition 2.1. The **k-distance** of a point $p \in D$ is the distance $d(p, q)$ between p and a point $q \in D$ such that for at least k points $q' \in D - p$, $d(p, q') \leq d(p, q)$ and for at most k-1 points $q' \in D$, $d(p, q') < d(p, q)$.

The k points closest to p are the *k-nearest neighbors* (kNN) of p, and *k-distance* of p is the distance to its kth nearest neighbor.

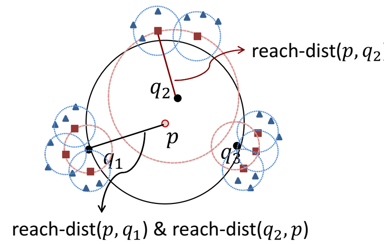


Figure 1: LOF definition

Definition 2.2. Given points $p, q \in D$ where $q \in kNN(p)$, the **Reachability Distance** of p w.r.t. q is defined as:

$$reach\text{-}dist(p, q) = max(k\text{-}distance(q), d(p, q)).$$

If one of p's kNN q is far from p, then the *reach-dist* between them is simply their actual distance. As shown in Fig. 1, the *reach-dist(p,q_1)* is the actual distance between p and q_1. On the other hand, if q is close to p, then the *reach-dist* between them is the *k-distance* of q. The red line in Fig. 1 shows the *reach-dist(p,q_2)*, which is the *k-distance* of q_2. The *reachability distance* introduces a smoothing factor for a stable estimation of the local density of p.

Definition 2.3. Given points $p, q \in D$, where $q \in kNN(p)$, the **Local Reachability Density (LRD)** of p is defined as:

$$LRD(p) = 1/\left(\frac{\sum_{q \in knn(p)} reach\text{-}dist(p, q)}{\|k\text{-}neighborhood\|}\right).$$

The local reachability density *LRD(p)* is the inverse of the average reachability distance of p to its neighbors. *LRD* values of each point and its neighbors are then used to compute *LOF*.

Definition 2.4. Given points $p, q \in D$, where $q \in kNN(p)$, the **Local Outlier Factor (LOF)** of p is defined as:

$$LOF(p) = \left(\frac{\sum\limits_{q \in knn(p)} \frac{LRD(q)}{LRD(p)}}{||k\text{- } neighborhood||} \right).$$

Informally, *LOF(p)* is the ratio of the average density of p's neighbors to the density of p. *LOF* scores close to 1 indicate "inlier" points, and the higher the LOF score, the more the point is considered to be an outlier. As shown in Fig. 1, p is considered to be a local outlier since the density of p is much smaller than the average density of p's neighbors.

3 DISTRIBUTED LOF FRAMEWORK

As described in Sec. 2, the LOF score of p is determined by its kNN q, its kNN's kNN q', and its kNN's kNN's kNN q'' – in total $k + k^2 + k^3$ points. These points, essential for detecting the LOF status of p, are called the **support points** of p. Within each machine in the compute cluster, data is distributed among machines according to some partitioning criteria, and only part of the dataset can then be accessed locally. There is thus a high chance that the support points of p are not available locally. Calculating the LOF score of p may thus require access to data assigned to numerous different machines.

However, popular distributed infrastructures such as MapReduce [8] and Spark [10] do not allow machines unrestricted pairwise exchange of data. Intuitively this problem could be solved if we could design a partitioning mechanism which **assigns p and all its support points to the same machine**. Unfortunately, given datasets exhibiting different distribution characteristics throughout, it is difficult to predict the distance between p and its kNN. In a dense region, p might be close to its kNN, while in a sparse area the distance between p and its kNN could be very large. Worst yet, to compute the LOF score of p, we not only have to predict the location of its direct kNN, but also that of its indirect kNN (its kNN's kNN and so on). This is extremely difficult if not impossible. Moreover, this would introduce extremely high data duplication as support points need to be copied to many machines.

Proposed Step by Step Processing Pipeline. To tackle this complexity, we introduce our Distributed LOF framework, *DLOF*, that adopts a *step by step* conceptual processing pipeline for LOF computation. It is logically a 3-step pipeline composed of: **Step 1**: K-distance Computation. By Def. 2.1, the kNN and *k-distance* of each point are calculated and materialized as intermediate values; **Step 2**: LRD Computation. By Def. 2.2 the *reachability distances* of each point p to its kNN q are computed using the *k-distances* of p and q from step 1. At the same time, the LRD value of p, the average reachability distance of p to its kNN q, can be naturally derived and materialized; **Step 3**: LOF Computation. *LRD* values materialized in the second step are utilized to compute thefinal *LOF* scores.

Intermediate Value Management. In these three steps, each point requires access not only to its corresponding intermediate values, but also those associated with numerous other points. These intermediate values have to befirst *updated, maintained*, and then *made available* to other points in the next step of the computation. It is important to note that this augmented data is now bigger than the initial raw data itself. Given a big dataset, it is not feasible to store the intermediate values of all points in one single machine nor to support concurrent access by many machines. Instead, the intermediate values have to be stored across and updated by many different machines. An effective intermediate data management mechanism must not only guarantee the correctness of updates performed simultaneously by multiple writers, but also efficiently support retrieval requests by many different readers. Otherwise locating and retrieving these intermediate values would risk being as expensive as re-calculating them from scratch. *DLOF* effectively solves the intermediate value management problem based on our *invariant observation* below.

Invariant Observation. In the *LOF* computation process of a point p, although each step requires different types of intermediate values, these intermediate values are *only* related to the *direct* kNN of p. More specifically, to calculate the *LRD* value of p, we only need the *k-distance* of each of its kNN. Similarly when calculating the LOF score of p, only the *LRD* of each of its kNN is required. Therefore although eventually the LOF score of p is determined by both its direct kNN and indirect kNN, in the step-by-step LOF computation pipeline, *p only needs to directly access* the intermediate values of its direct kNN in each step. Given a point p, as long as the intermediate values associated with the kNN of p are correctly updated and passed to the machine that p is assigned to in the next step, the LOF score of p can be correctly computed *without having to be aware of its indirect kNN.*

Support-aware Assignment Strategy. Our DLOF framework leverages the above *invariant observation* by employing a *support-aware assignment* strategy to solve the intermediate data management problem. The strategy assigns two roles to each point which imply certain responsibilities. One role comes with "write access", responsible for the update of the intermediate values, while the other role requires only "read-only" access to distribute intermediate values across consumers. *This separation of concerns makes the complex problem of distributed intermediate data management tractable.*

Suppose the original input dataset D has been partitioned into a set of disjoint data partitions. The support-aware assignment strategy assigns two different roles, namely the *core point* and the *support point* role, to each point p based on its relationship to different data partitions. p is called a *core point* of a partition $\mathbb{P}_i$ if p falls into $\mathbb{P}_i$. Each point p is the *core point of one and only one partition.* Furthermore, p may be needed by other partitions $\mathbb{P}_j \neq \mathbb{P}_i$ as a support point when p could potentially be a nearest-neighbor of some core point in $\mathbb{P}_j$ based on our invariant observation. We note that p could be a *support point of multiple partitions.* As shown in Fig. 2, p_1 is a core point of $\mathbb{P}_1$, while being a support point of $\mathbb{P}_2$. Then p_1 has $\mathbb{P}_1$ as its core partition and $\mathbb{P}_2$ as its support partition.

After deciding upon the roles for each point p, our *support-aware assignment* strategy assigns each p to both its core partition and its many support partitions. This is called the *assignment plan* of p. By this, the core points of each partition will be grouped together along with all their support points into the same partition. *DLOF* can now conduct thefi rst step of the LOF computation process to calculate the kNN and *k-distance* of the core points of each

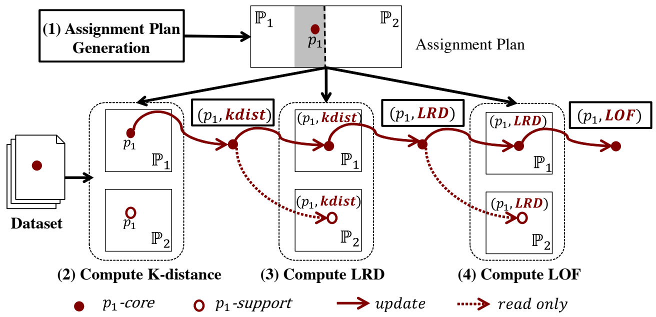


Figure 2: Support-aware Assignment Strategy

partition in parallel since each core point is now *co-located with all its potential neighboring points in the same partition.*

At the end of this step, each machine will enrich the core points with their intermediate values: its kNN, *k-distances* and the IDs of their core and support partitions, which correspond to the assignment plan. Results are spilled out to the local disk as shown in Fig. 2. Although a point p serves two roles and potentially many copies of p have been produced, only in its core point role would p need to be "enriched" (updated) in each step. For this reason, each point is correctly updated by exactly one machine.

In the next step of the LRD computation, the system will again retrieve each (now enriched) core point and assign it to both its core partition and support partitions based on the *same assignment plan* that has already been made available and encoded with each point in the last step. In other words, the *assignment plan* effectively serves as a *distributed index* to read back all support points for LRD computation of core points. Furthermore, when points are retrieved to serve as core or support points, they are being replicated from the *just updated core points only*. Therefore all support points of p have their associated kNN and *k-distances* values computed in the last step. By Def. 2.3, p is now guaranteed to have sufficient information needed to calculate its LRD value.

The same process is repeated in thefi nal step of *DLOF*. Each of the three steps of the LOF computation process, including the read and update of the intermediate results, is fully distributed.

4 DATA-DRIVEN DISTRIBUTED LOF

As shown in Sec. 3, the key of *DLOF* is to produce an *assignment plan* that, for any point *p*, determines both its core partition and its support partitions. This is equivalent to defining a *supporting area* for each partition $\mathbb{P}_i$, denoted as $\mathbb{P}_i$*.suppArea*. All points falling into the supporting area are potentially the kNN of at least one point of partition $\mathbb{P}_i$. In Fig. 2, the area highlighted in gray represents the supporting area of partition $\mathbb{P}_2$. Now the problem of producing an assignment plan is mapped into the problem of partitioning data into disjoint partitions and then defining the boundary of the supporting area for each partition, namely *support-aware partitioning*.

The key challenge is how to determine an effective boundary of the supporting area for each partition. Ideally the supporting area should be as small as possible to limit the points which must be duplicated and transmitted multiple times to other partitions as support points. A large number of support points introduce heavy communication costs which often, if not always, are the dominant costs of a distributed approach [2]. Therefore we model the effectiveness of a support-aware partitioning method using the notion of a *"Duplication Rate"*, which refers to the average number of replicas that must be created for each input data point.

Definition 4.1. Given a dataset D and a distributed algorithm A for computing *LOF* scores for all points in D, the **duplication rate** $dr(D, A) = \frac{|Rec(D,A)|-|D|}{|D|}$, where $|\, D \,|$ represents the cardinality of D and $|\, Rec(D, A) \,|$ the cardinality of the data records produced by the partitioning of Algorithm A.

The goal of support-aware partitioning thus is to minimize the duplication rate while ensuring the correctness of LOF computation. In [17] a pivot-based method for kNN join is proposed that partitions the input points based on their distances to a set of selected *pivots*. It then utilizes these distances to predict a bound on the support points for each partition such that a kNN search can subsequently be completed in a single map-reduce job.

This method bounds the support points conservatively, based on the furthest possible kNN of all points in a partition. This corresponds to a *safe* but *worst case* estimation, leading to a large number of replicas and in turn a high duplication rate larger than 20. As our experimental studies on real data demonstrate, the lost opportunity cost outweighs the benefit gained from [17] forcing the kNN search to be conducted in a single map-reduce job. Adapting this *pivot-based* approach to our DLOF framework, our experiments confirm the pivot-DLOF approach cannot even handle datasets larger than 1G (Sec. 6).

Based on the above analysis, we now propose an alternative approach *DDLOF* (Data-Driven Distributed LOF) that significantly reduces the duplication rate. It succeeds to achieve our important milestone to scale LOF to terabyte level datasets. *DDLOF* consists of two components, namely *Equal-cardinality Partitioning* and *Data-Driven kNN Search*.

The equal-cardinality partitioning of *DDLOF* partitions the domain space of D into n disjoint grid partitions $\mathbb{P}_i$ such that $\mathbb{P}_1 \cup \mathbb{P}_2 \cup \ldots \cup \mathbb{P}_n = D$. Each grid partition contains a *similar number of data points*, in spite of having different grid sizes. This ensures the balanced workload across different machines.

Definition 4.2. A **grid partition** $\mathbb{P}_i$ is a d-dimensional rectangle $\mathbb{P}_i = (\mathbb{P}_i^1, \mathbb{P}_i^2, \ldots, \mathbb{P}_i^d)$, where $\mathbb{P}_i^m$ denotes an interval $[l_i^m, h_i^m]$ representing the domain range of $\mathbb{P}_i$ along dimension m and $1 \leq m \leq d$.

Data-driven kNN search is the key component of DDLOF. It no longer aims to complete the kNN search within one single map-reduce job. Instead it utilizes one step without data duplication to gain insights from the kNN search process itself. These insights are then leveraged to dynamically determine the upper bound k-distance for each partition in the next step. This method is based on two key ideas. One, in thefi rst step of kNN search, the *local k-distance* of each p_i in the core partition $\mathbb{P}_i$ can be acquired. As we will show later this distance is sufficient to determine whether p_i can locally determine its kNN in $\mathbb{P}_i$ without needing to examine any remote support point. Two, if support points are still required, the *local k-distance* can aid us as a data-driven guide to determine p_i's supporting area. In practice we found the local k-distance is aways close to thefi nal k-distance. Therefore the supporting area generated by utilizing the local k-distance tends to be much smaller than the supporting area of the pivot-based method bounded by worst case estimation [17]. *This leads to a 20-fold reduction in the duplication rate.* With a low duplication rate and balanced workload, *DDLOF* now scales to dataset sizes in the terabytes as confirmed in our experiments (Sec. 6.4).

4.1 Data-Driven kNN Search

The data-driven *k*NN search is decomposed into two parts, namely a separate *core partition* and *supporting area kNN search.*

Core Partition *k*NN Search. Thefi rst phase performs an initial *k*NN search within each local grid partition. For each core point p_i, the k closest points in $\mathbb{P}_i$ - the so called "local *k*NN" of p_i - are found. Since the "actual k-distance" of p_i discovered in the whole dataset D cannot be larger than its local k-distance, the actual *k*NN of p_i are guaranteed to be located at most local k-distance away from p_i. Intuitively if point p_i is in the middle of partition $\mathbb{P}_i$, very possibly its actual *k*NN may not fall outside of the partition (Fig. 3(a)). However, if p_i is at the edge of $\mathbb{P}_i$, its actual *k*NN may fall in adjacent partitions. In this case the local *k*NN of p_i might not be its actual *k*NN.

Fig. 3(b) depicts a point p_i in $\mathbb{P}_i$ and a circular area with radius r, which is determined by the local k-distance of p_i. This circle bounds the distance from p_i to its local *k*NN. We can see that a sufficient supporting area for partition $\mathbb{P}_i$ must cover the area of the circle which falls outside of partition $\mathbb{P}_i$. This can be accomplished by extending the boundaries of $\mathbb{P}_i$ in each dimension to form partition $\hat{\mathbb{P}}_i$ that includes $\mathbb{P}_i$ and its supporting area illustrated by the area shown in grey. It is obvious that any point outside this gray area is at least r far away from p_i and thus cannot be in the *k*NN of p_i. Given a point p_j in partition $\mathbb{P}_j$, to determine whether p_j is a support point of p_i, we need to be able to examine whether p_j falls in $\hat{\mathbb{P}}_i$.

Next we illustrate how to decide the boundaries of area $\hat{\mathbb{P}}_i$ utilizing a two dimensional grid partition $\mathbb{P}_i$ shown in Fig. 3(b), without loss of generality. Each dimension of $\mathbb{P}_i$ has two boundaries corresponding to the lowest and highest values on this dimension. Here we utilize l_i^1 and h_i^1 to denote the two boundaries on thefi rst dimension of $\mathbb{P}_i$. In order for $\hat{\mathbb{P}}_i$ to cover the supporting area determined by the local k-distance of p_i, two (denoted as h_i^1 and l_i^2) out of its four boundaries of the original core partition $\mathbb{P}_i$ are extended. Since the circle does not overlap the other two boundaries (l_i^1 and h_i^2) of the partition, they do not need to be extended. The extended boundaries have the following property: the shortest distance from point p_i to any extended boundary is identical to r. This distance can be divided into two pieces. Take boundary l_i^2 as an example. The distance from point p_i to the extended boundary of $\hat{\mathbb{P}}_i$ can be treated as the sum of the distance from point p_i to boundary l_i^2 and the **extended distance** $Ext(l_i^2)$ (define in Def. 4.3).

Definition 4.3. Suppose the *local k-distance* of $p_i \in \mathbb{P}_i$ is r. Then **the extended distance** of p_i is $Ext(x) = max\{0, r - dist(p_i, x)\}$ where $dist(p_i, x)$ denotes the smallest distance from p_i to boundary x of $\mathbb{P}_i$.

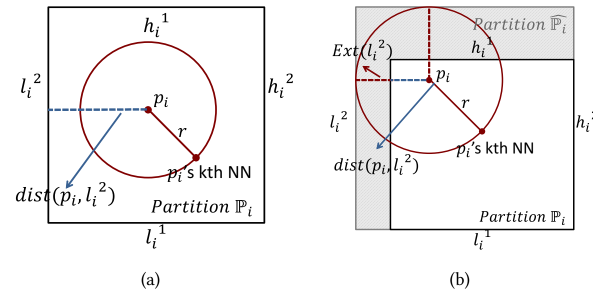


Figure 3: DDLOF: Supporting Area.

Extended distances describe how much further the boundaries of the original partition $\mathbb{P}_i$ have to be expanded to form the boundaries for the new partition $\hat{\mathbb{P}}_i$ augmented with the supporting area. Lemma 4.4 shows how to utilize it to determine boundaries of $\hat{\mathbb{P}}_i$.

LEMMA4.4. *Given a d-dimensional rectangular partition* $\mathbb{P}_i = (\mathbb{P}_i^1, \mathbb{P}_i^2, ..., \mathbb{P}_i^d)$, *where* $\mathbb{P}_i^m$ *represents an interval* $[l_i^m, h_i^m]$ *along dimension m, suppose the local k-distance of* $p_i \in \mathbb{P}_i = r$, ***then the actual kNN of*** p_i, *kNN*(p_i), ***is guaranteed*** *to be discovered in* $\hat{\mathbb{P}}_i = (\hat{\mathbb{P}}_i^1, \hat{\mathbb{P}}_i^2, ..., \hat{\mathbb{P}}_i^d)$. *Here* $\hat{\mathbb{P}}_i^m$ *denotes an interval* $[\hat{l}_i^m, \hat{h}_i^m]$, *where* $\hat{l}_i^m = l_i^m - Ext(l_i^m)$ *and* $\hat{h}_i^m = h_i^m + Ext(h_i^m)$.

PROOF. To prove Lemma 4.4, wefi rst prove that for any given point $p_j \notin \hat{\mathbb{P}}_i$, $dist(p_j, p_i) > r$. Here we denote the domain value of p_j as $p_j(p_j^1, p_j^2, ..., p_j^d)$. If $p_j \notin \hat{\mathbb{P}}_i$, then there must exist a dimension m that $p_j^m > \hat{h}_i^m$ or $p_j^m < \hat{l}_i^m$ holds.

Suppose $p_j^m > \hat{h}_i^m = h_i^m + Ext(h_i^m)$. Since $dist(p_j, p_i) \geq |p_j^m - p_i^m|$, then $dist(p_j, p_i) > h_i^m + Ext(h_i^m) - p_i^m = (h_i^m - p_i^m) + Ext(h_i^m) = dist(p_i^m, h_i^m) + Ext(h_i^m)$ (1).

If $dist(p_i^m, h_i^m) \geq r$, then $dist(p_j, p_i) > r$ according to Equation (1). If $dist(p_i^m, h_i^m) < r$ then $dist(p_j, p_i) > dist(p_i^m, h_i^m) + Ext(h_i^m) = r$ based on the definition of the extended distance. Therefore in either case $dist(p_j, p_i) > r$ holds.

The condition of $p_j^m < \hat{l}_i^m$ can be proven in a similar way. Due to space restrictions, we omit the proof here.

Since for any point $p_j \notin \hat{\mathbb{P}}_i$ $dist(p_j, p_i) \geq r$, then any point p_j out of $\hat{\mathbb{P}}_i$ will not be *k*NN of p_i, because there are at least k other points (local *k*NN of p_i) closer to p_i than p_j. Lemma 4.4 is proven. □

The corollary below sketches how to utilize Lemma 4.4 to determine whether the local k-distance of p_i is its **actual k-distance**.

COROLLARY4.5. *If* $Ext(l_i^m) = Ext(h_i^m) = 0$ *for any* $m \in \{1, 2, ..., d\}$, *then the* ***local k-distance*** *of* p_i *is guaranteed to be its* ***actual k-distance****.*

Once we have acquired the *individual supporting areas* for each point p_i in partition $\mathbb{P}_i$, it is trivial to derive the supporting area of the overall partition $\mathbb{P}_i$ that covers the *k*NN for all points in $\mathbb{P}_i$. This area adopts the maximum h_i^m of each partition $\hat{\mathbb{P}}_i$ as thefinal h_i^m and the minimum l_i^m of each partition $\hat{\mathbb{P}}_i$ as thefinal l_i^m. Since our skew-aware partitioning makes each partition $\mathbb{P}_i$ roughly uniform, most of the points have a similar *local k-distance*. Therefore we

Algorithm 1 Core Partition kNN Search.

1: **function** CoreKNNSearch(int k)
2: **for** p_i $\in$v-list $\in$ $\mathbb{P}_i$ **do**
3: p_i.kNNs= SearchKNN(p_i,v-list,k)
4: p_i.kdistance=max(dist(p_i.kNN,p_i))
5: p_i.supBound = CalcSupportBound(p_i,p_i.kdistance))
6: **if** p_i.supBound==0 **then**
7: p_i.type= 'Y' ▷ Canfi nd actual kNN
8: **else**
9: p_i.type= 'N' ▷ needs update kNN
10: supportBound($\mathbb{P}_i$) = max(supportBound($\mathbb{P}_i$), p_i.supBound)

expect that the supporting area of the whole partition $\mathbb{P}_i$ is not much larger than the *supporting areas* of individual points.

Algorithm 1 shows the procedure of this core partition kNN search. Atfi rst, each point searches for its local kNN within the local core partition $\mathbb{P}_i$ and gets its local k-distance (Lines 3-4). Then the supporting area of each point is bounded by applying Lemma 4.4. The supporting area of $\mathbb{P}_i$ is acquired as the spatial max-union of the supporting areas of all points (Line 10). In the meantime, each point is classified as either being able tofi nd its actual kNN by applying Corollary 4.5 or not (Lines 6-9). Finally, the "local" kNN as well as the point's kNN status (actual kNN found) are attached to each point and written out to HDFS. Given a point p, it will be assigned to partition $\mathbb{P}_i$ as a support point if p falls in the augmented partition $\hat{\mathbb{P}}_i$ but not in the original $\mathbb{P}_i$. Therefore in this task the assignment plan of each point can be naturally derived as the input of the supporting area kNN search task.

Supporting Area kNN Search. The supporting area kNN search corresponds to the k-distance computation step of the DLOF framework. As explained in Sec. 3, each point p is assigned to both its own partition as core point and to several partitions as a support point based on the assignment plan generated by the core partition kNN search. The previously computed local kNN attached to each core point are fully reused. In other words, the supporting area kNN search will only be conducted on those points that have *not yet fully acquired their actual kNNs*. If a core point p_i does not have its actual kNN, the local kNN of p_i will be parsed and stored in a priority queue *tempKNN*(p_i). The points in this structure are sorted in descending order by their distances to p_i. Then only the support points will be searched. If one support point p_s of p_i is closer to p_i than at least one point in *tempKNN*(p_i), p_s is inserted into *tempKNN*(p_i), and the top point removed. This process proceeds until all support points are examined. Then the remaining points in *tempKNN*(p_i) correspond to the actual kNN of p_i. This way, any duplicate kNN search between a core partition and a supporting area search is completely avoided.

Overall Process of DDLOF. As shown in Fig. 4 DDLOF contains four phases, namely *preprocessing, k-distance computation, LRD computation and LOF computation* that can be realized infi ve map-reduce jobs. The preprocessing phasefi rst utilizes one map-reduce job to divide the domain space into equal cardinality grid partitions. Then the *k-distance computation phase* is composed of two map-reduce jobs corresponding to the core partition and supporting

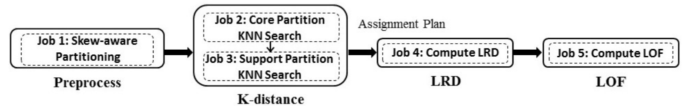


Figure 4: Overall Process of Data-driven DLOF

area kNN searches respectively (Sec. 4.1). In the core partition kNN search job, each mapper assigns points to the corresponding partitions based on the grid partitions generated in the previous job. No support point is produced in this job. Each reducer computes for each point its local k-distance which is then utilized to generate the support-aware assignment plan (Algorithm 1). In the supporting area kNN search job, each mapper reads in the points and generates partitions containing both core and support points based on the assignment plan. In this stage, each reducer then computes the final kNN for the points which did not acquire their actual kNN during the earlier core partition kNN search. After that, the *LRD computation phase* is conducted in one map-reduce job. Each mapper reads in and assigns points to the corresponding partitions based on the assignment plan embedded within each point. Then each reducer conducts the LRD computation. Thefinal *LOF computation phase* follows a similar process computing LOF scores from LRD values.

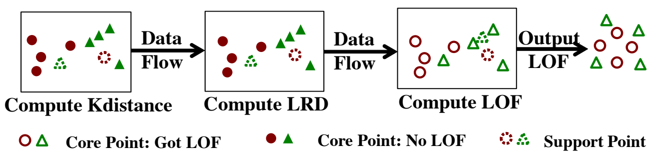
5 DATA-DRIVEN DLOF WITH EARLY TERMINATION

As shown in Fig. 5(a), based on the DLOF framework proposed in Sec. 3, DDLOF must transmit all core points as well as their respective support points throughout three phases of the LOF process flow. As our experiments in Sec. 6 confirm, even with the significant improvement of the duplication rate in DDLOF, it still incurs high communication costs, especially when handling large datasets. Unfortunately, often, if not always, the communication costs are the dominant costs of a distributed approach [2]. Therefore it is critical to minimize these communication costs.

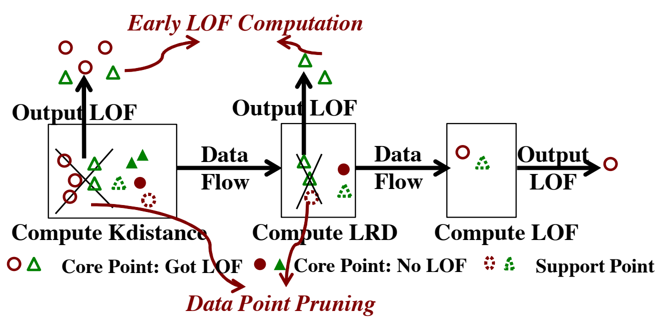
To accomplish this, we now enhance our DDLOF method with an *early termination* strategy, henceforth called *DDLOF-Early*. The key idea of *DDLOF-Early* is that instead of computing LOF scores by strictly following the LOF pipeline, we now aim to complete the computation process at the individual point granularity level instead of in unison synchronized among all points. Put differently, we aggressively push the LOF score computation into the earliest step possible. The points that have already acquired their respective LOF scores and do not serve a support point role for any partition are eliminated from the processfl ow. This reduces the communication costs as well as the intermediate value maintenance I/O costs due to reduced data transmission rates.

Convergence Observation. The effectiveness of *DDLOF-Early* rests upon our convergence observation on the LOF score computation. Namely, given a grid partition, most of the points can compute their *LOF* scores without the assistance of any support point.

First, we observe that although in theory the kNN definition does not satisfy the commutative nor transitive property, in practice data points tend to be neighbors of each other. That is, if a point A is in the kNN set of a point B, then B tends to be in the kNN set of point A, although this is not theoretically guaranteed. Furthermore, although the LOF score of point p depends on both direct and indirect kNN of p as shown in Sec. 3, layers of related points tend to converge to a small region within grid partition $\mathbb{P}$ instead of being spread across the entire domain space, since data points tend to be neighbors of each other. In practice, only a relatively small number of data points located at the edge of each partition need access to points in other partitions. This is empirically confirmed by our experiments on a rich variety of real world datasets (Sec. 6). For example, in the Massachusetts portion of the OpenStreetMap dataset which contains 30 million records, 98.6% of data points can compute their exact LOF scores without any support from other partitions.



(a) Framework for DDLOF



(b) Framework for DDLOF-Prune

Figure 5: Comparison of DDLOF and DDLOF-Prune

As shown in Fig. 5(b), *DDLOF-Early* incorporates two new optimization methods compared to the original *DDLOF* approach, namely, (1) *early LOF computation* and (2) *data point pruning*. These two techniques can be seamlessly plugged into each step of the DDLOF method (Sec. 4), including the core partition kNN search, supporting area kNN search, and the *LRD* computation stage (the reduce phase of the corresponding map-reduce job).

Early LOF Computation. In the core partition kNN search, points are classified into one of two states - either with or without their actual kNN (k-distance) by applying Corollary 4.5. The early LOF computation is conducted only on points that have already acquired their actual k-distances (those with kNN complete status) as shown in Algorithm 2. All points of this status (Lines 4-5) are maintained in a "GotKdist" list. Given such a "ready" point p, we then evaluate if p can also acquire its *LRD* value locally by checking if all p's kNN are also in the "GotKdist" list. If so, p's LRD value will be computed, and p inserted into another list "GotLrd" (Lines 10-11). Then a similar evaluation and computation process will be conducted for thefinal

Algorithm 2 Early LOF Computation

```
1:  k ← number of nearest neighbors
2:  function EarlyLOFComputation(key partition-id,v-list[p1,..., pm])
3:      for pi ∈v-list do
4:          if hasTrueKNNs(pi) then
5:              GotKdist.add(pi)
6:          else
7:              pi.type = "NONE"
8:      for pi ∈GotKdist do
9:          if CanCalLRD(pi,GotKdist) then
10:             pi.lrd = CalLRD(pi,GotKdist)
11:             GotLrd.add(pi)
12:         else
13:             pi.type = "KNN"
14:     for pi ∈GotLrd do
15:         if CanCalLOF(pi,GotLrd) then
16:             pi.lof = CalLOF(pi,GotLrd)
17:             pi.type = "LOF"
18:         else
19:             pi.type = "LRD"
```

LOF score computation. Points that have acquired their LOF score will be marked as "completed".

Data Point Pruning. Although this early LOF computation determines the LOF score of a point p as early as possible, p cannot simply be eliminated from the distributed LOF pipeline even if p has already acquired its own LOF score. Instead, p might still be required in the LOF computation of other points for two reasons. First, p in partition $\mathbb{P}_i$ may be one of the kNN of some adjacent point q that is also in $\mathbb{P}_i$ and has not yet acquired its *LOF* value. Second, p may be a support point of any other partition $\mathbb{P}_j$. In the first case, since both p and q are located in the same partition and therefore on the single machine, we can easily check whether q is marked as "completed". In the second case, in the LOF computation processfl ow each core point p constantly maintains a list of partitions *sup-list* for which it is a support point. Therefore this case can be evaluated by checking whether the *sup-list* attached to p is empty.

In all other cases, p can be ***eliminated*** from the processfl ow. Therefore this pruning strategy significantly reduces the communication costs as confirmed by our experiments in Sec. 6.4.

6 EXPERIMENTAL EVALUATION

6.1 Experimental Setup & Methodologies

Experimental Infrastructure. All experiments are conducted on a Hadoop cluster with one master node and 24 slave nodes. Each node consists of 16 core AMD 3.0GHz processors, 32GB RAM, 250GB disk. Nodes are interconnected with 1Gbps Ethernet. Each server runs Hadoop 2.4.1. Each node is configured with up to 4 map and 4 reduce tasks running concurrently, sort buffer size set to 1GB, and replication factor 3. All code used in the experiments is made available at GitHub: https://github.com/yizhouyan/DDLOFOptimized.

Datasets. We evaluate our proposed methods on two real-world datasets: OpenStreetMap [11] and SDSS [9]. *OpenStreetMap*, one of the largest real datasets publicly available, contains geolocation data from all over the world and has been used in other similar research work [17, 20]. Each row in this dataset represents an object like a building or road. To evaluate the robustness of our methods for diverse data sizes, we construct hierarchical datasets of different sizes: *Partial Massachusetts* (3 million records), *Massachusetts* (30 million records), *Northeast of America* (80 million records), *North America* (0.8 billion records), up to the *whole planet* (3 billion).

The *OpenStreetMap* data for the entire planet contains more than 500GB of data. To evaluate how our proposed methods perform on terabyte level data we generate two datasets 1TB and 2TB respectively based on the *OpenStreetMap* dataset. More specifically, we generate the 2TB dataset by moving each point vertically, horizontally, and also along both directions to create three replicas. That is, given a two dimensional point $p(x, y)$ where $0 \leq x \leq d_x$ $0 \leq y \leq d_y$, three replicas $p'(x + d_x, y)$,$p''(x, y + d_y)$ and $p'''(x + d_x, y + d_y)$ are generated. Then the 1TB dataset (6 billion records) is generated by extracting half of data in half of the domain from the 2TB dataset (12 billion records). In our experiment two attributes are utilized, namely *longitude* and *latitude* for distance computation.

Sloan Digital Sky Survey (*SDSS*) dataset [9] is one of the largest astronomical catalogs publicly accessible. The thirteenth release of SDSS data utilized in our experiments contains more than 1 billion records and 3.4TB. In this experiment we extract the eight numerical attributes including ID, Right Ascension, Declination, three Unit Vectors, Galactic longitude and Galactic latitude. The size of the extracted dataset is 240GB.

Metrics. First, we measure the total ***end-to-end execution time*** elapsed between launching the program and receiving the results – a common metric for the evaluation of distributed algorithms [17, 20]. To provide more insight into potential bottlenecks, we break down the total time into time spent on key phases of the MapReduce workflow, including *preprocessing*, *k-distance* calculation, *LRD* calculation, and *LOF* calculation. Second, we measure the ***duplication rate*** of each method as defined in Def. 4.1.

Algorithms. We compare (1) baseline *PDLOF*: adapts the pivot-based partitioning method in [17] to our DLOF framework; (2) *DDLOF*: data-driven distributed LOF in Sec. 4; (3) *DDLOF-Early*: data-driven distributed LOF with early termination in Sec. 5.

Experimental Methodology. We evaluate the **effectiveness and scalability** of our algorithms. In all experiments, the same kNN search algorithm is applied to eliminate the influence of the various kNN search algorithms and indexing mechanisms. The input parameter k of LOF isfi xed as 6 which in [7] is shown to be effective in capturing outliers. Based on our experimental tuning we apply the most appropriate partition number to each algorithm on each dataset.

6.2 Evaluation of Elapsed Execution Time

We evaluate the breakdown of the execution time of the three algorithms usingfi ve OpenStreetMap datasets and the SDSS datatset.

OpenStreetMap Datasets. Fig. 6 shows the results on the OpenStreetMap datasets. *PDLOF* is only able to process the Partial Massachusetts dataset. This is due to the high duplication rate of the pivot-based approach producing some extremely large partitions that cannot be accommodated by a single compute node. Our two data-driven algorithms *DDLOF* and *DDLOF-Early* scale to the Planet – the whole OpenStreetMap dataset. This performance gain results from the small duplication rate of their partitioning methods (Sec. 6.3), which significantly reduces communication costs. It also reduces the computation costs of the kNN search, since each reducer must only search for the kNN of its core points within a small area.

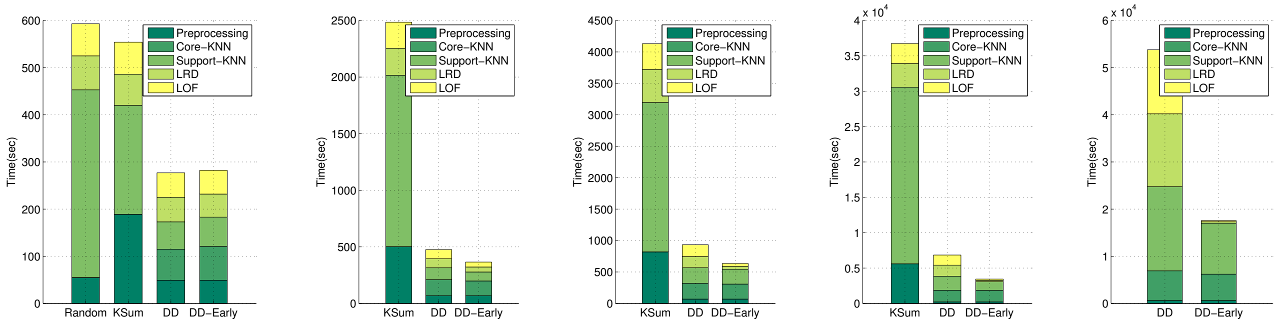
As for the two data-driven algorithms, although *DDLOF* is slightly better on the total time consumption than *DDLOF-Early* on the small Partial Massachusetts dataset, *DDLOF-Early* beats *DDLOF* in all other cases as the dataset gets larger. Since these two approaches share the same preprocessing phase, the difference here comes from other phases, namely the *k-distance*, *LRD* and *LOF* computation phases. *DDLOF-Early* is more expensive than *DDLOF* during the *k-distance* phase, because it not only computes kNN, but also aggressively computes the *LOF* scores whenever possible. However at the later LRD and LOF computation phase, *DDLOF-Early* succeeds to outperforms *DDLOF*. It is up to three times faster in total execution time, especially when the dataset scales to the whole planet (Fig. 6(e)), because *DDLOF-Early* eliminates data points from the workfl ow that were able to acquire their LOF scores during the inner kNN search. Therefore both communication and computation costs are reduced.

SDSS Dataset. Tab. 1 demonstrates the results on the eight dimensional SDSS dataset. Since the pivot-based algorithm cannot handle a dataset of this size, Tab. 1 only shows the results of *DDLOF* and *DDLOF-Early*. Similar to the OpenStreetMap data results, the preprocessing and core partition kNN search phases take about the same time, while *DDLOF-Early* significantly outperforms *DDLOF* in other phases. In total execution time, *DDLOF-Early* is 2 times faster than *DDLOF*. This experiment shows that our data-driven approach can scale to large datasets with eight dimensions.

6.3 Evaluation of Duplication Rate

Next we evaluate the duplication rates of all 3 algorithms using the same data and setting as in Sec. 6.2. Since the duplication rates are identical for both data-driven methods, we only show DDLOF.

Fig. 7 shows the results on the OpenStreetMap datasets. PDLOF has much higher duplication rate than *DDLOF* – up to 21 on a relatively small dataset. This explains why *PDLOF* cannot even handle the Massachusetts dataset (30 million records). *DDLOF* instead has very low duplication rates – around 1 for all small datasets and around 2 for the largest planet dataset. This is expected because *PDLOF* bounds the supporting area based on worst case estimation (Sec. 4), while our data-driven kNN search in *DDLOF* utilizes the "local" *k-distance* generated in the core partition kNN search to bound supporting areas (Sec. 4.1). This bound is much tighter than the worst case upper bound of *PDLOF*. This explains *DDLOF*'s superiority. Moreover, *DDLOF* has a duplication rate slightly larger than 1 even on the large eight dimensional SDSS dataset (Tab. 1), while *PDLOF* methods fail on data at this scale.



(a) Partial Massachusetts (b) Massachusetts (c) Northeast (d) North America (e) Planet

Figure 6: Evaluation of End-to-end Execution Time

Table 1: Experimental Results of DDLOF and DDLOF-Early (SDSS/1TB/2TB)

Methods	Dataset	End-to-end Execution Time Costs (sec)						Duplication
		Preprocessing	Core-KNN	Support-KNN	LRD	LOF	Total	
DD	SDSS	464	7899	32059	7796	6598	54816	1.5863
DD-Early	SDSS	464	8454	20245	404	385	29952	1.5863
DD	1T	1059	11076	56282	25371	23940	117728	2.0474
DD-Early	1T	1059	11987	31964	793	743	46546	2.0474
DD-Early	2T	1909	27824	61858	711	489	92791	2.1514

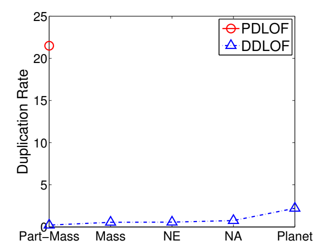


Figure 7: Duplication Rate for Varying Size of Datasets.

6.4 Scalability Evaluation

We utilize the 1TB and 2TB data described in Sec. 6.1 to evaluate the scalability of *DDLOF* and *DDLOF-Early* to terabyte level data. As shown in Tab. 1, although both algorithms scale to the 1TB dataset, *DDLOF-Early* is three times faster than *DDLOF*. This is due to the fact that the *early termination* strategies (Sec. 5) reduce communication costs by eliminating points that already acquired their LOF scores. In particular *DDLOF-Early* eliminates 71.3% (1T) and 76.6% (2T) points after the core partition kNN search. However, the *DDLOF* method fails on 2T dataset. In the core partition kNN search phase, *DDLOF* enriches all core points with their local kNN and spills this information out to HDFS. This produces extremely large intermediate data that causes system failure due to network congestion when traveling over the network during the shuffle operation in the next phase. *DDLOF-Early* is able to handle this 2T dataset, since it does not maintain the kNN information for the points which have already acquired their LOF scores.

7 RELATED WORK

Distributed Outlier Detection. *To the best of our knowledge, no distributed LOF algorithm has been proposed to date.* In [16], Lozano and Acunna proposed *a multi-process LOF algorithm on one single machine.* All processes share the disk and main memory and therefore can access any data in the dataset at any time. This way, the processes can communicate with each other without introducing high communication costs. Clearly this approach cannot be adapted to popular shared-nothing distributed infrastructures targeted by our work. Here, each compute node only has access to partial data and communication costs are often dominant.

Bhaduri et al. [6], proposed a distributed solution for distance-based outlier detection [19], which depends on nearest-neighbor search. Their algorithm requires a ring overlay network architecture wherein data blocks are passed around the ring allowing the computation of neighbors to proceed in parallel. Along the way, each point’s neighbor information is updated and distributed across *all* nodes. A central node maintains and updates the top-n points with the largest kNN distances. Their strategies are not applicable to shared nothing infrastructures lacking a central node.

Other Related Distributed Analytics Techniques. In [17] a pivot-based method is introduced for kNN-join to partition the two to-be-joined datasets. Given a partition $\mathbb{P}_i$ in dataset D_1 the distances between the points and the corresponding pivots are utilized to bound the partitions $\mathbb{P}_j$ in the other dataset D_2 which could possibly produce join results with $\mathbb{P}_i$ in D_1. We adapt this method as our baseline PDLOF approach by replacing its bounding rule with our customized rule. However, as we demonstrate, this method [17] cannot even handle 1GB dataset (Sec. 6.3) due to the high duplication rate, while our *DDLOF* related methods work for TB datasets.

In [4, 13], distributed approaches for density-based clustering and spatial joins are introduced. These approaches employ the general notion of “support” to ensure each machine can complete its task by replicating boundary points. Both problems have a proximity threshold as input parameter that determines the boundary points. In our more complex distributed LOF context, no such explicit user-provided criteria exists for bounding the support points. Instead, the support points have to be bounded dynamically by exploring the data. Deriving a sufficient yet tight bound to determine the support points for each partition is a unique challenge addressed in our work.

8 CONCLUSION

In this work, we propose thefi rst distributed solution for Local Outlier Factor semantics (*LOF*) – a popular technique to detect outliers in skewed data. Innovations include a step-by-step framework that computes LOF scores in a fully distributed fashion, a data-driven partitioning strategy to reduce the duplication rate from a rate of 20 down to 1, and an early termination mechanism to minimize the communication costs. Our experimental evaluation shows the efficiency and scalability of our solution to terabyte datasets.

9 ACKNOWLEDGEMENT

This work is supported by NSF IIS #1560229, NSF CRI #1305258, and Philips Research.

REFERENCES

[1] 2015. Apache Hadoop. https://hadoop.apache.org/. (2015).
[2] Foto N. Afrati, Anish Das Sarma, Semih Salihoglu, and Jeffrey D. Ullman. 2013. Upper and Lower Bounds on the Cost of a Map-Reduce Computation. *PVLDB* 6, 4 (2013), 277–288.
[3] Charu C. Aggarwal. 2013. *Outlier Analysis*. Springer.
[4] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel H. Saltz. 2013. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *PVLDB* 6, 11 (2013), 1009–1020.
[5] Stephen D. Bay and Mark Schwabacher. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*. 29–38.
[6] Kanishka Bhaduri, Bryan L. Matthews, and Chris Giannella. 2011. Algorithms for speeding up distance-based outlier detection. In *KDD*. 859–867.
[7] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and JÃűrg Sander. 2000. LOF: Identifying Density-based Local Outliers. In *SIGMOD*. ACM, 93–104.
[8] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
[9] Kyle S. Dawson et.al. 2016. The SDSS-IV Extended Baryon Oscillation Spectroscopic Survey: Overview and Early Data. *The Astronomical Journal* 151, 2 (2016), 44. http://stacks.iop.org/1538-3881/151/i=2/a=44
[10] Matei Zaharia et.al. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX*. 2–2.
[11] Mordechai Haklay and Patrick Weber. 2008. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing* 7, 4 (2008), 12–18.
[12] Douglas M. Hawkins. 1980. *Identification of Outliers*. Springer. 1–188 pages.
[13] Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. 2011. MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce. In *ICPADS*. 473–480.
[14] Edwin M. Knorr and Raymond T. Ng. 1998. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *VLDB*. 392–403.
[15] Aleksandar Lazarevic, Levent Ertöz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. 2003. A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection.. In *SDM*. SIAM, 25–36.
[16] Elio Lozano and Edgar Acuña. 2005. Parallel Algorithms for Distance-Based and Density-Based Outliers. In *ICDM*. 729–732.
[17] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. 2012. Efficient processing of k nearest neighbor joins using mapreduce. *PVLDB* 5, 10 (2012), 1016–1027.
[18] Gustavo Henrique Orair, Carlos H. C. Teixeira, Ye Wang, Wagner Meira Jr., and Srinivasan Parthasarathy. 2010. Distance-Based Outlier Detection: Consolidation and Renewed Bearing. *PVLDB* 3, 2 (2010), 1469–1480.
[19] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient Algorithms for Mining Outliers from Large Data Sets. In *SIGMOD*. 427–438.
[20] Chi Zhang, Feifei Li, and Jeffrey Jestes. 2012. Efficient parallel kNN joins for large data in MapReduce. In *EDBT*. 38–49.