Conquering the Divide: Continuous Clustering of Distributed Data Streams

Graham Cormode[1]
AT&T Labs–Research
graham@research.att.com

S. Muthukrishnan
Rutgers University
muthu@cs.rutgers.edu

Wei Zhuang
Rutgers University
weiz@cs.rutgers.edu

Abstract

Data is often collected over a distributed network, but in many cases, is so voluminous that it is impractical and undesirable to collect it in a central location. Instead, we must perform distributed computations over the data, guaranteeing high quality answers even as new data arrives. In this paper, we formalize and study the problem of maintaining a clustering of such distributed data that is continuously evolving. In particular, our goal is to minimize the communication and computational cost, still providing guaranteed accuracy of the clustering.

We focus on the k-center clustering, and provide a suite of algorithms that vary based on which centralized algorithm they derive from, and whether they maintain a single global clustering or many local clusterings that can be merged together. We show that these algorithms can be designed to give accuracy guarantees that are close to the best possible even in the centralized case. In our experiments, we see clear trends among these algorithms, showing that the choice of algorithm is crucial, and that we can achieve a clustering that is as good as the best centralized clustering, with only a small fraction of the communication required to collect all the data in a single location.

1 Introduction

Modern data acquisition systems are essentially automatic, distributed and continuous. For example, environmental sensors automatically gather their own sets of measurements that change over time; each soldier on the field has a series of location updates as they move around; multiple web/blog crawlers independently grow collections of gathered text and links; different mail servers collate their share of mails over time; multiple repositories of books, audio and video data evolve on their own; etc.

Formally, we have distributed sites $1 \dots m$. Each site i has a changing source of data S_i^t at time t. Not only may each source be large in size $|S_i^t|$, individual items in the sources may be high dimensional including audio, video or text. While it may indeed be desirable to put a database infrastructure atop these distributed sources for storing, indexing, accessing and searching all of the data, in many of

[1]Work done while the author was at Bell Labs.

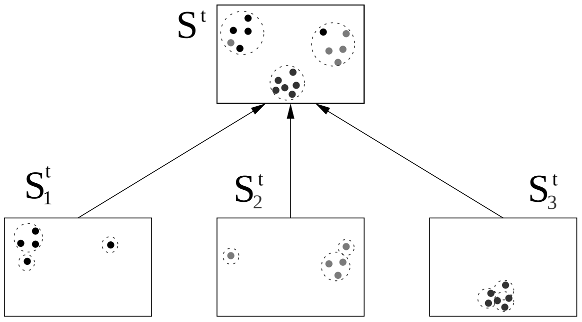


Figure 1. Continuous Distributed Clustering: $m = 3$ remote sites aim to find $k = 3$ clusters

the applications, the more immediate need is to *monitor* desirable properties of the union $\bigcup_{i=1}^m S_i^t$ of all the data distributed over the sites.

In this paper, we study a quintessential monitoring problem on continuously changing distributed sources, namely, *clustering*. More specifically, it is of great interest to *maintain* a clustering of the points in $S^t = \bigcup_{i=1}^m S_i^t$ at all times t, illustrated in Figure 1. Clustering is a fundamental problem in data engineering which lies at the interface of databases, data mining and machine learning. Irrespective of the nature of the data—text, spatial, or high dimensional vectors—dividing many points into a small number of representative clusters is a powerful way to understand the large quantity of data, and provide insights into its structure. To understand the challenge in monitoring a clustering over distributed sources, consider two straightforward "solutions":

1. At each time t, each site updates a central site with newly gathered points. The central site updates the clustering of $\bigcup_{i=1}^m S_i^t$ whenever a new point is seen.

This presents a communication and processing bottleneck. All data collected at the sites reach a central store and the total communication is proportional to the union of the data which is prohibitive in most modern applications.

2. Periodically, say every τ seconds, each site packages its data and sends it to the central site. The central site updates the clustering for $\bigcup_{i=1}^m S_i^t$, every τ seconds.

The first problem this poses is one of latency. Between the τ instants, the clustering could change radically, a serious drawback for monitoring applications. This also does not overcome the communication problem of collecting all the data at a central site.

This helps us formalize the problem further. Our problem is one of "conquering the divide": to design efficient methods to cope with dynamic data that is divided into pieces, minimize the overall communication between sites, and effectively perform clustering to yield clusters of comparable quality to a centralized clustering. Unlike some parallel computations, we do not have the luxury of designating how the data is shared between the participants; instead, we must be able to adapt to arbitrary divisions, and yet guarantee that the amount of communication needed is significantly less than would be required to share all updates with a single central site. Our methods must be able to monitor changes in the locally observed updates, and determine what communication is necessary, and when, in order to ensure the continued quality of the clustering.

Motivating Example. Underwater sensor networks are a particularly resource constrained setting because of physical conditions (reduced channel capacity, harsh environment) [2]. A typical problem is when m remote tracking stations are deployed in an underwater acoustic monitoring system. Each station keeps track of certain schools of fishes based on a given wave length, and reports the location information to a central base station. This station maintains a k-clustering of the schools so k attracting or dispelling acoustic devices can be deployed near the k center points to use the minimum energy to cover the whole region [20]. □

Our Contributions. Our main contributions are as follows:

1. We introduce and motivate the problem of clustering evolving distributed data, minimize the communication between sites and obtain clustering as good as the best centralized solution at any time.

2. We give the first algorithms for this problem, with accuracy guarantees. Our algorithms are easy to implement, and demonstrate a variety of tradeoffs between the amount of information that is shared and the guarantees on the quality of the clustering that arises. Our algorithms rely on trading of local clustering and global clustering in a careful way.

3. We perform a detailed experimental study, which shows that our proposed algorithms send only a small fraction of the data while generating a clustering that is as good as the best centralized clustering.

Outline. In the next section we give important background to our work and describe the model of Continuous, Distributed monitoring (Section 2.1) and prior work on clustering (Section 2.2). We then describe efficient algorithms for computing and merging k-center clusterings in Section 3. Our new algorithms for clustering in the continuous, distributed model are described in Section 4, and extensions described in Section 5. We give a thorough experimental evaluation in Section 6, and conclude in Section 7.

2 Preliminaries

2.1 Continuous Distributed Model

The Continuous Distributed model of computation has been refined over recent years through a sequence of papers [4, 26, 9, 7, 6]. It abstracts the key features of a broad variety of scenarios and identifies the principle dimensions. In the model there are a set of m different remote sites, each of which observes an update stream S_i^t defining the local data set. These sites cooperate with a distinguished node, known as the coordinator, to compute some function over the union of the streams $S^t = \cup_{i=1}^m S_i^t$. Communication is permitted between any pair of sites, but in several cases, the best results are found when every communication involves the coordinator node. The goal is that, at every instant, the coordinator has an accurate answer to a query, or set of queries, over S^t, even though the distributions and arrival rates at each site may vary widely. This accuracy is typically guaranteed in terms of some parameter ε.

We can compare algorithms in this model based on:
(1) The communication used by the algorithm, as a function of the size of the streams, $|S_i^t|$, and the accuracy ε.
(2) The amount of space required by each remote site and the coordinator: do the remote sites need to store their full streams, or can they keep a summary of much smaller size?
(3) How much information is shared between sites and the coordinator, if any, and how is this kept up to date?
(4) How is the communication charged for: the basic model assumes direct links between each pair, but can be extended to a hierarchical communication tree; costs can also vary in settings where broadcasting costs the same as sending a unicast message (modeling radio networks).

A variety of problems have been studied within this model, including: monitoring the top-k most frequent items [4]; tracking set expressions and duplicate resilient quantities [9, 8]; monitoring quantiles of a distribution [7]; and finding accurate sketch summaries of the data [6, 8]. In most cases, it is possible to write the communication cost of the protocol in terms of the accuracy parameter ε, establishing a trade-off between communication and accuracy. Note that the model abstracts away issues of message delay, synchronization and loss, by assuming that each message is delivered immediately. These assumptions can be removed using standard techniques of timestamping, acknowledgments and re-sending. Such concerns are removed from the model to focus on the main underlying computation.

Our focus in this work is on performing clustering in the Continuous, Distributed model. Thus, our streams S_i^t consist of points arriving at each site i, and our goal is to allow the coordinator to continuously maintain a clustering (description of the clusters) of S^t, the union of the S_i^t's. As is standard, we will focus on the features defined above: the communication cost, accuracy, and space required, as well as the communication model and information shared.

We do not make assumptions about the nature of the points to be clustered (i.e. whether they are text documents or high dimensional vectors etc.), but rather assume only that we have a distance function d which tells us the distance between any pair of points.

2.2 Prior Work on Clustering Algorithms

Clustering is an important problem in databases, mining, and beyond. We do not attempt to give a thorough survey of clustering work, since it spans over 150 years. Moreover despite the large volume of prior work, only a very small number of papers are relevant to our study on the *continuous, distributed* version of clustering.

Clustering can be defined informally as the problem of dividing the input data into clusters, such that all points in the same cluster are similar, and all points in different clusters are dissimilar. The definition can be formalized in various ways; however, many attempts to formulate a precise mathematical definition yield a problem that is NP-hard to optimize [21, 12]. Hence, most clustering work focuses either on efficient algorithms that give good results in practice, such as BIRCH [30], CURE [18] DBSCAN [11], k-means [23] and so on; or on giving guaranteed approximations to particular clustering optimization criteria, such as k-center (minimizing the maximum radius/diameter of any cluster) [16] and k-median (minimizing the sum of distances from each point to its cluster center) [3]. All of these methods assume full access to (static) data, and hence do not naturally adapt to the continuous distributed model.

Some prior work touches on aspects of our setting – for example, the DEMON project [15] considers the case when (centralized) data can be somewhat dynamic. In [29] and [10] the authors consider how to perform clustering on parallel processors, which shares some concerns with our setting. However, they have the freedom to preprocess and distribute the data to processors, whereas in our setting, the allocation of data to sites is fixed. From the Distributed Data Mining community (DDM), distributed clustering has been explored [13]. This motivated the problem of clustering distributed static data, and gave results based on collecting sufficient statistics for density based clustering algorithms from remote sites. [24] extends DBSCAN to a distributed setting, again assuming static data. Each one of these touches on some aspects of continuous, distributed clustering, but none fully solve this problem.

Also relevant to our study is work on clustering points that arrive as a data stream. Because such algorithms keep small memory state, this state can sometimes be used as a summary of the much larger data, and shared or merged with others to allow clustering of the union of streams. In [5], the "doubling algorithm" allows the k-center objective to be approximated over a single stream, up to a factor of 8, using merging techniques and storing only k points. We use similar ideas, but are able to get much better approximations with slightly more storage. The k-median objective was studied in [17], but much more space and much worse approximation factors make this algorithm impractical in our setting. Similarly, more recent work using ε-nets [19] and gridding techniques [22, 14] have addressed clustering in lower-dimensional spaces, but the complexity of the algorithms makes them unusable for our purposes. Prior work on clustering evolving data considered capturing historic trends at a central site [1], whereas our goal is to cluster the current set of point streams.

3 Centralized Clustering Algorithms

In this work we concentrate on the k-center objective for clustering, since this is a simple and popular criterion for clustering in arbitrary metric spaces. We first introduce the formal definition of k-center clustering, and then give algorithms that yield provably good clusterings.

Definition 1. *The* k-center *problem is, given a set P of n points and an integer $k < n$, to identify a set C of k points as centers. Given a point p and a set of centers C, we define its closest center as $C(p) = \text{argmin}_{c \in C}\, d(c, p)$, where d is a metric over the points. The* radius objective *is to minimize the radius, $\max_{p \in P} d(p, C(p))$. The* diameter objective *is to minimize the diameter of the clustering,* $\max_{p,q \in P, C(p)=C(q)} d(p, q)$.

The two objectives of radius and diameter are closely related. Since $d(p, q) \leq d(p, C(p)) + d(C(p), q)$, the diameter of a cluster is at most twice its radius (coinciding with our intuitions regarding radius and diameter of circles and spheres in Euclidean space). The methods we consider give similar guarantees for both radius and diameter objectives. The *cost* (or *quality*) of a clustering is either its radius or diameter (depending on context).

A clustering method may draw the centers as arbitrary points from the metric space (sometimes known as "continuous" clustering) or insist that $C \subset P$ ("discrete" clustering). We focus mostly on the latter version, since this is most natural when the points are from high dimensional metric spaces, where it is hard to create arbitrary points not in the input. Considering all $\binom{n}{k}$ possible ways of choosing k centers from n points, the optimal clustering is the one which minimizes the cost. Explicit search of this space takes time $\Omega(n^k)$, and the problem has been shown to be NP-hard [21, 12]. So we focus on guaranteed α-approximation algorithms: these are algorithms which return a clustering with a guarantee that the cost of the clustering is at most α the cost of the optimal clustering. The NP-hardness proof can be extended to show that it is NP-hard to guarantee any solution with $\alpha < 2$. Next, we will describe algorithms which obtain α-approximations with $\alpha = 2$ or α arbitrarily close to 2.

3.1 Furthest Point Algorithm

The Furthest Point algorithm [16] gives a guaranteed 2-approximation for both the radius and diameter measures. It begins by picking an arbitrary point as the first center, $C_1 = \{c_1\}$. Then find the $i+1$st center as the point that maximizes its distance from the first i centers, i.e.

$$c_{i+1} = \text{argmax}_{p \in P} \min_{c \in C_i} d(p,c)$$

and let $C_{i+1} = C_i \cup c_{i+1}$. After k iterations, one can show that C_k is a factor 2 approximation to the optimal clustering [16]. The algorithm is illustrated in Figure 2 (a).

The proof of approximation quality proceeds by considering a $k+1$st iteration, which produces a set of $k+1$ points all separated by a distance of at least D, implying that the optimal clustering must have diameter at least D. By the triangle inequality, the chosen clustering has diameter at most $2D$. A similar argument shows that the radius of the produced clustering is at most D, and any clustering of the points must have radius at least $D/2$. The algorithm requires k scans of the data in order to find the furthest points, so takes $O(kn)$ distance computations.

3.2 Parallel Guessing Algorithm

The correctness of the above algorithm relies on the fact that if there exists a set of $k+1$ points separated by at least $2R$, then R is a lower bound on the optimal radius, and a solution of cost $2R$ can be found. We now propose an alternative algorithm with a slightly worse approximation factor, but requiring only a single pass over the input points[2]. Suppose we knew R. We pick an arbitrary point as the first center, $C = \{c_1\}$, and for each point p in P compute

$$r_p = \min_{c \in C} d(p,c)$$

and test whether $r_p > R$. If it is, then set $C = C \cup \{p\}$. Since we do not know R in advance, we make multiple guesses of R as $(1+\frac{\varepsilon}{2}), (1+\frac{\varepsilon}{2})^2, (1+\frac{\varepsilon}{2})^3 ...$ and run the algorithm in parallel on each of these. We will bound the number of guesses made in terms of the ratio of the closest pair to furthest pair of points, $\Delta = \max_{p,q \in P} d(p,q) / \min_{p,q \in P, p \neq q} d(p,q)$. The algorithm is illustrated in Figure 2 (b).

Theorem 1. *The Parallel Guessing algorithm finds a $2+\varepsilon$ approximation to the k-center problem and stores at most $O(\frac{k}{\varepsilon}\log\Delta)$ points.*

Proof. First, observe that every guess R that generates k or fewer centers is a valid k clustering of P with radius less than R. The diameter of the clustering is at most $2R$, since by the triangle inequality, any p, q in the same cluster are separated by at most $2R$. Consider the optimal clustering of the points, which has a radius of R_{opt}. Any guess $R < R_{opt}$ will cause us to choose more than k centers, and so can tell that the guess of R was too small. If this were not the case, we would find a clustering of the points with radius less than R_{opt}, contradicting the optimality of R_{opt}.

For a guess of $R \geq 2R_{opt}$, then we will find k or fewer centers, and generate a valid k center clustering of P. Suppose we found more than k points from P such that each pair is separated by a distance of more than $2R_{opt}$. Then by the pigeonhole principle, at least two of these points must be placed in the same cluster, so *any* clustering will have a diameter greater than $2R_{opt}$, and hence must have a radius greater than R_{opt}, contradicting the claim that R_{opt} provided an optimal clustering. A guess of R so that $R_{opt} \leq R < 2R_{opt}$ can either allow us to find a valid clustering with k or fewer centers, or we may be unlucky, and find more than k centers from this guess. In the worst case, all guesses below $2R_{opt}$ may lead to more than k centers, and the next guess above $2R_{opt}$ may be as far above as possible. By our choice of guesses, we know that in this worst case, our guess cannot be bigger than $2R_{opt}(1+\frac{\varepsilon}{2})$. Thus, we guarantee to find a solution with radius at most $2+\varepsilon$ the optimal radius. Similarly, the algorithm gives a $2+\varepsilon$ approximation of the diameter cost.

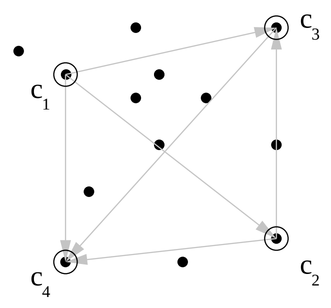
If the ratio $\max_{p,q} d(p,q) / \min_{p,q} d(p,q) = \Delta$, then the number of parallel guess made by the algorithm is $O(\log_{1+\varepsilon/2}\Delta) = O(\frac{1}{\varepsilon}\log\Delta)$ (since we never need a guess less than $\min_{p,q} d(p,q)$ or larger than $\max_{p,q} d(p,q)$). Any guess that identifies more than k points as centers is no good, and so can be dropped. Thus the space needed is that to store $O(\frac{k}{\varepsilon}\log\Delta)$ points. To process each new point, we compare it to each point stored at each level to find the closest center, so requiring $O(\frac{k}{\varepsilon}\log\Delta)$ comparisons. $\square$

Some important features of this algorithm are that it requires an amount of storage that is independent of n, the number of points; it depends only on the geometric properties of the point set; and it takes only a single pass through the point set. Thus it is suited for cases when there are a large number of points arriving in an incremental (streaming) fashion, since the algorithm can run incrementally as new points arrive. The output is sensitive to the order of arrival of the points (and to the point c_1 chosen as the first center), but the guarantee holds true irrespective of the order. A prior algorithm, due to Charikar *et al.* gave a similar result, storing only k points but guaranteeing a (much weaker) 8-factor approximation [5].

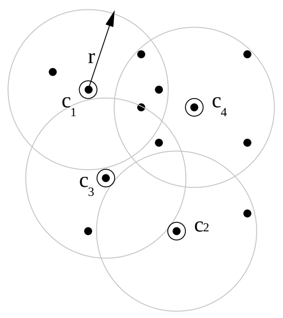
3.3 Merging Clusterings

We demonstrate that clusterings can be merged by running a clustering algorithm on the cluster centers.

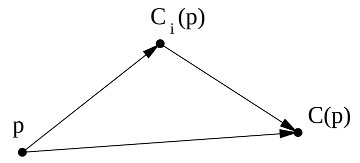
[2]As far as we know, this algorithm has not been formally described before, but related ideas are considered 'folklore' in the clustering world.



(a) In the Furthest Point algorithm, an arbitrary point is chosen as c_1, then subsequent centers are chosen to maximize their distance from existing centers.



(b) The Parallel Guessing algorithm guesses a radius R, and creates a new center every time a point is seen which is further than r from an existing center.



(c) In merging clusters, a point p with an initial center $C_i(p)$ has a final center $C(p)$. The distance $d(p, C(p))$ is bounded by the triangle inequality.

Figure 2. Figures illustrating Centralized k-Center Clustering Algorithms

Theorem 2. *Let a set of points P be divided into m (possibly overlapping) subsets $P_1 \dots P_m$. Let $C_i \subseteq P_i$ be the k centers of an α approximate k-center clustering of P_i. Running a β approximate k-center clustering on the union of the C_is guarantees the resulting k-centers C form an $(\alpha + \beta)$ approximation of the clustering for P.*

Proof. Let R_P denote the optimal radius of the clustering of P, and R_C the optimal clustering of the union of cluster centers C_i Then (see Figure 2 (c)):

$$\begin{aligned} d(p, C(p)) &\le d(p, C_i(p)) + d(C_i(p), C(p)) \\ &\le \alpha R_P + \beta R_C \le (\alpha + \beta) R_P \end{aligned}$$

The result relies on the fact that the cost of the optimal clustering on any subset of P has cost at most R_P. $R_C \le R_P$ follows immediately because $C \subseteq P$. □

4 Continuous Distributed Clustering

4.1 Merging Local Solutions

Our first algorithms maintain an accurate clustering by having each remote site keep a clustering of its local points, and only sending updates when this changes.

Local Algorithm based on Furthest Point (Local-FP). Each local site i maintains a Furthest Point clustering of its points, C, with radius R_i. When a new point p arrives, we measure the distance between p and its closest center, $C(p)$. If $d(C(p), p) \le (1 + \frac{\varepsilon}{2})R_i$, then we know that the local clustering is still good for this new point: since R_i is at most twice the optimal radius, adding this extra point increases the radius to at most $2 + \varepsilon$ times the optimal radius. Hence we take no further action. If this condition is not true, then a reclustering is forced. We recompute a furthest point clustering on all the points held by the local site, and inform the coordinator of the new cluster centers. The coordinator can then recompute its clustering as the clustering of the union of the centers from each site. By Theorem 2 above, this guarantees a $2 + (2 + \varepsilon) = 4 + \varepsilon$-approximation to the optimal cost. In order to allow the reclustering, each site must retain all its points. Each reclustering requires $O(k)$ points to be sent to the coordinator, but since it is not clear how many communications there will be, we do not have a closed form for the total communication cost.

A natural improvement is to ensure that each site knows a better lower bound on the clustering cost, such as the maximum over all local clusterings and the global clustering, $R_{max} = \max(R_{global}, \max_i R_i)$ (where R_{global} denotes the cost of the clustering found by the coordinator). We can delay reclustering until a point more than $(1 + \frac{\varepsilon}{2})R_{max}$ from its closest center is seen, instead of $(1 + \frac{\varepsilon}{2})R_i$. This maintains the same $4 + \varepsilon$ accuracy guarantee, but may degrade the actual quality of clustering seen in practice. We keep sites informed of R_{max} either by eagerly broadcasting the value when it changes, or by informing sites lazily, when they contact the coordinator with a new clustering.

Theorem 3. *Algorithm Local-FP guarantees that at any time the coordinator has a k-center clustering of the input which is a $4 + \varepsilon$ approximation to the optimal clustering.*

Local Algorithm based on Parallel Guessing (Local-PG). Each local site i maintains the Parallel Guessing algorithm over its points. It shares with the coordinator the current "good" guess and associated centers: this is the smallest guess R_i such that there are no more than k centers for site i. When this guess is no longer good (a new point arrives which is at least R_i from all k current centers), then we move to the next larger guess $R_i(1 + \frac{\varepsilon}{2})$, and communicate this guess and its centers to the coordinator. If a current good guess has fewer than k centers and a new center is opened, this new center is sent to the coordinator. The coordinator merges the most recent cluster centers it has received from each site using an appropriate algorithm, such as Furthest Point. By appealing to Theorem 2, we ensure a $2 + (2 + \varepsilon) = 4 + \varepsilon$ accurate solution. Each site needs space to store only $O(\frac{k}{\varepsilon} \log \Delta)$ points, and takes $O(\frac{k}{\varepsilon} \log \Delta)$ point comparisons per new arrival. Further, each remote site sends up to k centers for each guess, and works through at most $O(\frac{1}{\varepsilon} \log \Delta)$ guesses. Over m sites, the total communication cost is therefore at most $O(\frac{km}{\varepsilon} \log \Delta)$ points.

As in the previous algorithm based on furthest point, we can try to pre-empt communication by ensuring that each site knows that over all remote sites, the largest good guess is $R_{global} = \max_i R_i$. This shows that R_{global} is a lower bound on the best possible clustering over the whole point set, and so we can immediately drop all guesses less than R_{global}, and consequently reduce communication. Again, this reduction in communication may come at the cost of observed accuracy; the guarantee still remains $4 + \varepsilon$.

Theorem 4. *Algorithm Local-PG guarantees the coordinator always has a $4+\varepsilon$-approximation to the optimal clustering. The communication is bounded by $O(\frac{km}{\varepsilon} \log \Delta)$ data points, and stores $O(\frac{k}{\varepsilon} \log \Delta)$ points at each remote site.*

4.2 Maintaining Global Solutions

Our next set of algorithms are based on running a centralized algorithm in a distributed way. This turns out to give tighter approximation bounds, close to the best possible.

Global Algorithm Based on Furthest Point (Global-FP) The first approach is to take the furthest point algorithm and run it in a distributed fashion. The algorithm operates in two phases: computing a new clustering in $O(k)$ rounds of clustering, and then monitoring that this remains a good solution as new points arrive. To build a clustering, we assume that the coordinator knows some point from the input, c_1. It broadcasts this point to all sites, which each return the furthest point from their input from c_1. The coordinator can then process these m responses, and identify which is truly the furthest from c_1, and broadcast this back as c_2. Sites respond with the point that maximizes its distance from c_1 and c_2. The process can continue for $k + 1$ rounds of communication, until the coordinator has picked a set of k centers C and found R, the radius of the clustering. By the protocol, the remote sites also know these centers and r. They then enter the monitoring phase, where they observe new points, and for each new point p, the find $C(p)$, and determine whether $d(p, C(p)) > (1 + \frac{\varepsilon}{2})R$. If not, then p is stored but no other action is needed; if the bound is broken, then the coordinator must initiate reclustering. The protocol guarantees that the current clustering is always a $2 + \varepsilon$ approximation of the optimal clustering. Each reclustering requires $O(mk)$ points to be sent. However, there are drawbacks: the remote sites must retain all their input points in order to correctly perform reclustering; and no non-trivial bound has been shown on the communication cost.

Theorem 5. *Algorithm Global-FP guarantees a $2 + \varepsilon$ approximation to the optimal clustering at the coordinator.*

Global Algorithm Based on Parallel Guessing (Global-PG). The final approach has the coordinator site and all remote sites maintain multiple guesses of the optimal radius. For each active guess R, the coordinator maintains a set of at most k points C_R whose pairwise distance is at least R. Whenever a remote site observes a new point p for an active guess R, such that $d(p, C_R(p)) > R$, it sends p to the coordinator. The coordinator must deal with sequentialization and duplication of messages from remote sites, and announce when a new point is to be added to C_R. The communication cost can be bounded: in the worst case, all m remote sites simultaneously observe a new point p for guess R, and send it to the coordinator, who picks one and announces it back (Note that if more points arrive at remote sites after sending to the coordinator, they must buffer them and wait for the response before proceeding). For each guess, this can happen k times, and there are at most $O(\frac{1}{\varepsilon} \log \Delta)$ guesses, hence the total cost is $O(\frac{km}{\varepsilon} \log \Delta)$ points sent. The coordinator always knows which guess is currently good, and so the result is guaranteed to be a $2 + \varepsilon$ approximation. As in the local version, each new point that arrives must be compared to $O(\frac{k}{\varepsilon} \log \Delta)$ others.

Theorem 6. *Algorithm Global-PG guarantees a $2 + \varepsilon$-approximation to the optimal clustering at the coordinator, while using at most $O(\frac{km}{\varepsilon} \log \Delta)$ communication and storing $O(\frac{k}{\varepsilon} \log \Delta)$ points at each site.*

4.3 Summary of Properties of the Algorithms

Algorithm	Guarantee	Space	Communication
Local-FP	$4+\varepsilon$	$O(n)$	—
Local-PG	$4+\varepsilon$	$O(\frac{k}{\varepsilon} \log \Delta)$	$O(\frac{km}{\varepsilon} \log \Delta)$
Global-FP	$2+\varepsilon$	$O(n)$	—
Global-PG	$2+\varepsilon$	$O(\frac{k}{\varepsilon} \log \Delta)$	$O(\frac{km}{\varepsilon} \log \Delta)$

This table summarizes properties of the four algorithm types. Global algorithms have better accuracy guarantees, while the PG-based algorithms have smaller space requirements and communication guarantees. From the table, it seems the global-PG variation is the method of choice, however in our experimental evaluation later we see that this algorithm does not always achieve the best combination of communication cost and clustering quality in practice.

5 Extensions

5.1 Dynamic Point Sets

In full generality, our methods naturally extend to when the data arriving is dynamic. That is, in addition to arrivals of points, old points may get deleted or moved; or we may only be concerned with a sliding window of recent points. In this setting, it is not straightforward to maintain small space: if many points are deleted so only k points remain, these k points *must* be returned to give an approximation to the optimal clustering, since the optimal cost is zero. Hence, we focus on variations of clustering algorithms which store all current points. These can be maintained in the presence of deletions, moves, or sliding windows.

Our insert-only methods based on global or local solutions all build a clustering, monitor that it is still a good clustering, and then recluster when needed. The same approach works in the presence of deletions. Recall that the furthest point algorithm finds a set of $k+1$ points that form a proof to the quality of the clustering. So long as none of these $k+1$ points is deleted, then we know that the cost of the clustering does not decrease. Thus we can run the same algorithms as before, but additionally monitor when one of these $k+1$ centers is deleted, and recluster. For non-adversarial sequences of transactions (i.e. when deletions are not deliberately targeted at the centers), reclusterings due to deletions will be relatively rare.

Situations where points move can be modeled as the deletion of a point and reinsertion at its new location; similarly, in the sliding window case, when a point is no longer within the sliding window, this can be treated as a deletion of that point. Hence, our methods naturally extend to these cases. In the full version of this paper we show experiments demonstrating the communication cost of these extensions remains low while accuracy remains high.

5.2 Variable Numbers of Clusters

So far, we have assumed that k is fixed up front, before any points are seen. In full generality, we would like to be more flexible. A useful feature of our algorithms is that they can accept any given k' after processing the points, and return a guaranteed approximation to the k'-center problem, provided $k' \leq k$. This can be seen by studying the centralized algorithms on which we base our distributed algorithms. The Furthest Point algorithm only uses k to determine when to halt. Hence, the k' clustering is just the first k' centers of the k clustering. Similarly, the Parallel Guessing algorithm uses k to determine when a guess is too small. So, given a set of parallel guesses, the k' clustering is the smallest guess that generates at most k' centers.

In the distributed setting, algorithms which merge local solutions can take the same local solutions, and run a k'-center clustering algorithm on the same (k center) solutions to find k' centers. Since the cost of the clustering is monotone increasing as k' decreases, the accuracy remains guaranteed. Algorithms that maintain a global solution have to extract the k' centers from the k centers, as indicated above.

5.3 1-center in Geometric Spaces

For the case of 1-center (equivalently, minimum enclosing ball), the above algorithms all coincide to the trivial solution of picking an arbitrary point from the input. The triangle inequality ensures that the radius of the ball enclosing all points centered at an arbitrary point is at most twice the radius of the smallest ball that encloses all points. In general we would like to do better, especially for applications such as monitoring mobile users. In such cases, where the points reside in low-dimensional Euclidean space, we can offer much better guarantees on the accuracy, by picking a center that is not necessarily from the input (i.e. continuous clustering). We outline solutions for two dimensional Euclidean space, other low-dimensions are similar.

Our algorithms reduce the problem of 1-center in two dimensions to multiple instances of the problem in one dimension. The one-dimensional case is quite simple: the 1-center is defined by the midpoint of the smallest and greatest points seen so far, which can be easily maintained exactly or approximately in a distributed setting. So in two dimensions, each site takes each input point and projects it onto lines spaced uniformly at multiples of some angle α through the origin. For each line, we track the maximum and minimum projection onto the line, and a point that achieves that projection. We allow these projections to be approximated: we only replace a maximum or minimum point with a new point if it increases the *extent* e_i (distance between maximum and minimum point on projection i) by a $(1+\beta)$ factor, where $0 \leq \beta < \varepsilon$.

Theorem 7. *Computing the Minimum Enclosing Ball of the points stored by this algorithm is a $(1+\varepsilon)$-approximation to the 1-center of the point set, where $\varepsilon = \frac{\alpha^2/8+\beta}{1-\alpha^2/8}$.*

We can use this structure as the basis of our algorithms for maintaining 1-center in a continuous distributed way. If each remote site ensures that the coordinator has approximate extents of projections on regularly spaced directions, the coordinator can merge these (by taking the max and min point in each direction) to get good approximate extents for the whole point set, and hence compute a $(1+\varepsilon)$-approximate 1-center. We outline two possible variations:
(1) Sites send approximate (upto $(1+\beta)$) extents ($\beta < \varepsilon$).
(2) Sites only send extents when they have changed by more than a $(1+\beta)$ fraction of the diameter of the point set.

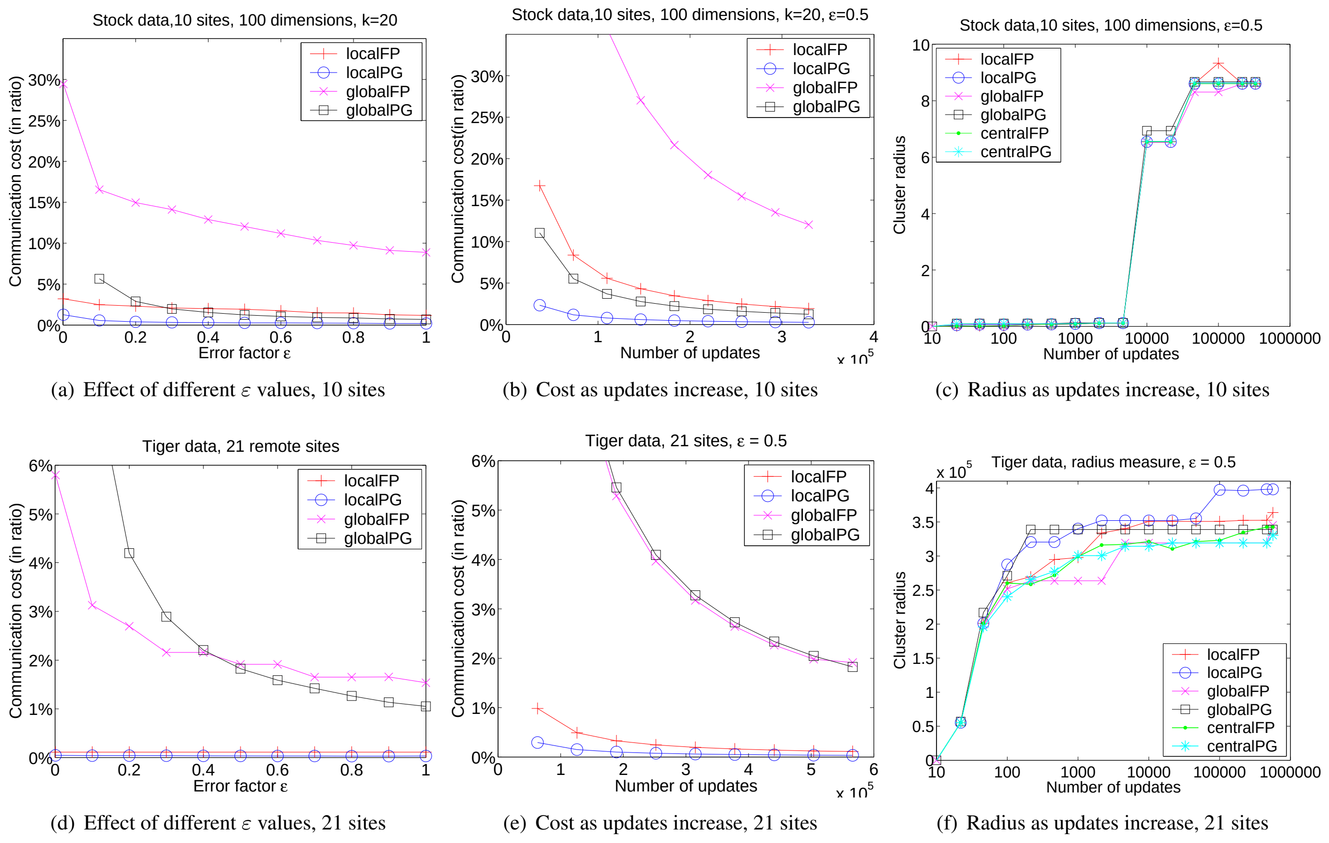
These algorithms can be shown to give accurate answers, and we can analyze the communication cost:

Lemma 1. *The worst case communication cost of algorithms (1) and (2) is $O(\frac{\log \Delta}{\varepsilon^{3/2}})$.*

Our experimental results in the full version of this paper show that these algorithms save up to four orders of magnitude in communication compared to naively sending every point to the coordinator.

6 Experimental Study

We performed a thorough experimental study of algorithms, in terms of the communication cost of performing



(a) Effect of different ε values, 10 sites

(b) Cost as updates increase, 10 sites

(c) Radius as updates increase, 10 sites

(d) Effect of different ε values, 21 sites

(e) Cost as updates increase, 21 sites

(f) Radius as updates increase, 21 sites

Figure 3. k-center clustering: (a) – (c) 100-D stock series data. (d) – (f) 2-D US Census (Tiger) data

distributed clustering, and the dependence on parameters such as the accuracy ε, number of sites and number of clusters, and the algorithm used. We implemented simulations of all four algorithms and their variants in C. We performed experiments on a mixture of low and high dimensional data, and measured the communication cost of each run in terms of the total number of bytes transmitted by the algorithm. This includes all points and counts transmitted, but does not include other header information, as this will vary from system to system. We present these costs as ratios to the naive cost of sending every local point to the coordinator site for centralized clustering. Experiments were performed on a 3.2GHZ machine with 1GB RAM. We did not explicitly compare the time costs, since our simulations are not fully optimized. However, we observed that in all cases, the time cost was very small, processing millions of high dimensional data points in a few seconds.

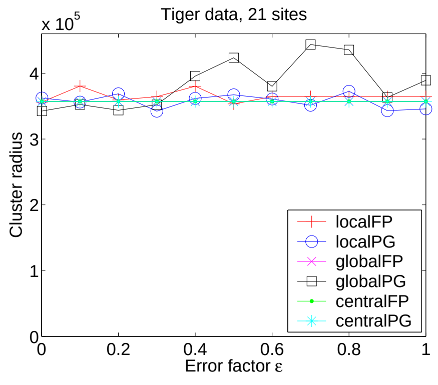
6.1 Data Sets and Methodology

We ran our experiments on a mixture of real and synthetic data sets. We used Euclidean distance between points, although our methods work for any metric. For brevity, we focus primarily on two real data sets, and describe but do not plot experiments on other data.

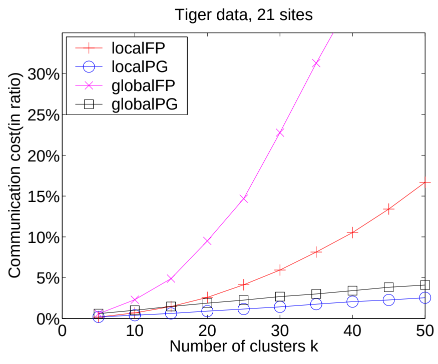
US Census 2005 (Tiger [28]) data. This data set consists of extracts of selected geographic and cartographic information from the Census Bureau's TIGER (Topologically Integrated Geographic Encoding and Referencing) database. We selected all the (latitude,longitude) pairs for New Jersey as 2-D data points. It contains about 600K points, divided into 21 counties. By default, we set the number of sites m to 21, and each site was allocated all points from one county. The default number of clusters was $k = 20$.

Stock price series [27]. The stock data series consists of the price for a single stock taken at frequent intervals over a six year period. In total, this defines a time series with over n =330K values. From this, we create a high-dimensional data set by sliding a window of width d over the series to produce a sequence of $(n - d)$ d-dimensional points.[3] For the experiments shown here, we used $d = 100$, sufficient to show the effect of high dimensional data (where the size of the points is significantly larger than other data that may be transmitted). Each point is randomly assigned to a site.

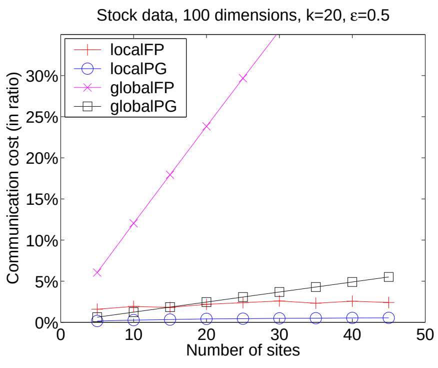
[3]Although the result of clustering such a set of points may not always be meaningful, it remains a good test of the clustering algorithms.



(a) Radius and error, 20 clusters



(b) Effect of varying number of clusters, 21 sites



(c) Effect of varying number of sites, 20 clusters

Figure 4. k-center clustering: varying varying accuracy, number of clusters, and number of sites

Synthetic Data [25]. We also ran experiments on a synthetic aperiodic time series consisting of 1 million entries. We again made this into high-dimensional data points by using a sliding window of default size $d = 100$.

6.2 Experimental Results

We implemented the improvements to the local algorithms outlined in Section 4.2, of (lazily) sharing lower bounds on the radius of the clustering as R_{global} or R_{max}, after observing that this helped the performance of the algorithms without affecting the clustering quality.

Fixed number of sites and number of clusters. Our first set of experiments focused on how the communication cost and quality of clustering varied between algorithms as a function of the error parameter ε and size of the input data. Figures 3(a) and 3(d) show that the communication cost of all algorithms are only a small fraction of the cost of sending all updates to the coordinator, and that this decreases as ε is increased. On both the high and low dimensional data, the relative performance of the algorithms is the same: the global algorithms are more expensive than the local ones, and the parallel guessing (PG) better than their furthest point (FP) counterparts. We saw this relationship on all experiments, and in all cases, the Local-PG algorithm obtained the lowest communication cost.

In Figure 3(b), Global-PG significantly outperforms the cost of Global-FP, by several times, whereas in the corresponding experiment on Tiger data (Figure 3(e)) the two have similar cost. This turns out to be a function of the data ordering: the Tiger data is arranged so that each site sees data that is locally clustered (each site sees points from the same county). When we randomized the allocation of points to sites, we observed that the Global-PG cost stayed about the same, but the Global-FP cost increased by a factor of about 4, suggesting that the FP algorithm is much more sensitive to the point distribution. These plots show that as more points are seen, the communication cost of this clustering is very small: the cost of Local-PG on the Tiger data is 0.1% of sending the full data, and 0.5% on the stock data. Similar results were observed on synthetic data.

Dependence on accuracy parameter ε. We should like to be sure that the local methods, while offering much lower communication cost, are not doing so at the expense of the quality of the clustering. We measure the clustering quality by computing the radius of the observed clustering. Although we do not know what is the optimal clustering, we can compare to the radius of an algorithm which takes the full data and runs a centralized clustering algorithm on it: either furthest point (central-FP) or parallel guessing with a very small value of ε (central-PG). It is perhaps surprising that in our experiments (Figures 3(c) and 3(f)) we saw close similarity between the performance of all our algorithms in radius (the worst case analysis allows for up to a factor of two worse between the local and global algorithms)[4]. On synthetic data with $\varepsilon = 0.5$ (not shown), we observed some cases where local algorithms had radius upto 20% higher than centralized algorithms — an appreciable difference, but still significantly within the $2 + \varepsilon$ guarantees.

We see in more detail how the quality depends on ε in Figure 4(a): as ε increases, we see more variability in the radius of the clustering output. But even for the larger values of ε, most methods (notably, the local algorithms) are very close to the centralized algorithms (which obtain almost identical results to each other).

Varying number of clusters and sites. The relative ordering of the algorithms remained the same over different numbers of clusters and numbers of sites. Figure 4(b) shows that as the number of clusters increases, the FP algorithms increase the communication cost much more quickly than the PG algorithms. Fitting a curve to these trends shows that the dependency of both local and global PG algorithms is linear

[4]We show these plots on a log-scale, since the radius of the clustering typically stabilizes after seeing only a small fraction of the points.

in k, the number of clusters, while the dependency of both FP algorithms are quadratic. The quality of the curve fitting (the r^2 coefficient) is 0.99 in all cases, indicating a very accurate fit. This implies that the analytic bounds for the PG algorithms in Section 4, which showed linear dependency in k, are quite tight, and that an analytic $O(k^2)$ dependency may follow for the FP algorithms. The effect of increasing the number of sites is shown in Figure 4(c). Here, it is the two global algorithms that show similar, clearly defined trends: both Global-FP and Global-PG show a cost that is strictly linear in m, the number of sites, with r^2 values indistinguishable from 1 (perfect fit). For the local algorithms, the trend is less clear: at worst, it is linear or sublinear. These results are in line with the analysis of Section 4, which guarantees worst-case linear dependency on m.

7 Conclusions

We have introduced the problem of continuous, distributed clustering, and given a selection of algorithms, based on the paradigms of local vs. global computations, and furthest point or parallel guessing clustering. In our experimental evaluation, the combination of local and parallel guessing gave the least communication cost: typically only a fraction of one percent of the cost of sending all the data to a central site. The resultant clustering was very similar in quality to a centralized clustering, and the space required is independent of the size of the input. This emphasizes that in distributed monitoring scenarios, it is frequently preferable to track each site locally, and combine the results at the coordinator site. It remains to extend this work to other settings, and other clustering algorithms. For some methods, such as k-means, it is easy to see how to create global instantiations, but less clear how to form accurate local versions. It will also be of interest to study other popular clustering algorithms, such as BIRCH, DBSCAN and others.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for clustering evolving data streams. In *VLDB*, 2003.

[2] I. F. Akyildiz, D. Pompili, and T. Melodia. Underwater acoustic sensor networks: Research challenges. *Ad Hoc Networks*, 3(3):257–279, May 2005.

[3] V. Arya, N. Garg, R. Khandekat, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Computing*, 33(3):544–562, 2004.

[4] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, 2003.

[5] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *STOC*, 1997.

[6] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

[7] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD*, 2005.

[8] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate resilient aggregates on data streams. In *ICDE*, 2006.

[9] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set-expression cardinality estimation. In *VLDB*, 2004.

[10] I. Dhillon and D. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-scale Parallel Data Mining*, 2000.

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[12] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *STOC*, 1988.

[13] G. Forman and B. Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explorations*, 2(2):34–38, 2000.

[14] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. In *STOC*, 2005.

[15] V. Ganti, J. Gehrke, and R. Ramakrishnan. DEMON: Mining and monitoring evolving data. In *ICDE*, 2000.

[16] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(2-3):293–306, 1985.

[17] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS*, 2000.

[18] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *SIGMOD*, 1998.

[19] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *STOC*, 2004.

[20] C. Holden. Sound Sightings. *Science*, 313(5788):777, August 2006.

[21] W. Hsu and G. Nemhauser. Easy and hard bottleneck location problems. *Disc. Appl. Math.*, 1:209–215, 1979.

[22] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *STOC*, 2004.

[23] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[24] E. Januzaj, H.-P. Kriegel, and M. Pfeifle. Towards effective and efficient distributed clustering. In *Workshop on Clustering Large Data Sets (ICDM)*, 2003.

[25] E. J. Keogh and M. J. Pazzani. Pseudo periodic synthetic time series. `http://kdd.ics.uci.edu/databases/synthetic/synthetic.html`

[26] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, 2003.

[27] SFR-USD tickwise stock data set. `http://www-psych.stanford.edu/~andreas/Time-Series/Data/`

[28] Topologically Integrated Geographic Encoding and Referencing system (TIGER), US census bureau, 2005. `http://www.census.gov/geo/www/tiger/`

[29] X. Xu, J. Jäger, and H.-P. Kriegel. A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery*, 3:263–290, 99.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD*, 1996.