Two-Phase Clustering of Large Datasets[1]

Najmeh Joze-Khajavi and Kenneth Salem
Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada
Email:kmsalem@uwaterloo.ca

Abstract

This paper addresses the problem of single-pass clustering of large, multi-dimensional datasets. A general two-phase approach to this problem is defined. In the first phase, an in-memory summary of the data set is constructed using a single pass through the data. The second phase then uses any clustering algorithm to cluster the summary data. Most clustering algorithms for large datasets are two-phase.

Two techniques for producing in-memory summaries are compared. The first is based on a previously-proposed data structure called a cluster-feature tree, or CF-tree. The second, simpler technique uses random sampling. Experiments with skewed artificial data sets are used to show that sampling produces clusters that are at least as accurate as those produced with CF-trees, and that sampling is much faster. An important issue when sampling is the sample size, which determines the amount of memory required during the first phase of two-phase clustering. It also controls a trade-off between clustering time and and clustering accuracy. This tradeoff is quantified, and guidelines for sample size selection are suggested.

1 Introduction

This paper addresses the problem of data clustering. Given a dataset of consisting of N points in a d dimensional metric space, and k, the desired number of clusters ($0 < k < N$), the problem is to partition the points into k subsets, or clusters, such that an error metric is minimized. In our case, the error metric is the root mean squared Euclidean distance between a data point and its cluster's mean.

[1]Technical Report CS-98-27, Dept. of Computer Science, University of Waterloo, November, 1998

Clustering can be viewed as a type of data mining. Each attribute of the dataset corresponds to a dimension of the data space, and tuples from the dataset correspond points in the space. The clustering algorithm attempts to identify groups of similar tuples in a non-uniformly populated space. Such information describes the multidimensional distribution of the tuples, and may be useful for such database system tasks as selectivity estimation.

Many clustering algorithms have been proposed, and they are briefly surveyed in Section 2. Until recently, most clustering algorithms were intended for datasets that fit in memory. Here, we focus on the problem of clustering datasets that are too large to fit in memory. In particular, we are interested in clustering algorithms that have the following two desirable properties. First, it should be possible to cluster using a single pass over the data in secondary (or tertiary) storage. Many clustering algorithms do not satisfy this condition because they are iterative and require multiple visits to points in the dataset. Second, it should be possible to control the amount of memory used by the algorithm. The algorithm should be able to operate with as much memory as it is given, although we expect that less memory may result in greater clustering error.

In this paper, we consider a simple two-phase clustering framework that has both of these properties. In the first phase, a single pass over the database is used to produce an in-memory summary of the dataset. The size of the summary can be controlled so that it will fit into the available memory. In phase two, the in-memory summary data is itself clustered using any clustering algorithm whose memory usage can be bounded. Several recently-proposed clustering algorithms, including BIRCH [14] and CURE [5], fit within this general framework. Besides its simplicity, one nice feature of the two-phase approach is that almost any existing clustering algorithm for memory-resident datasets can be used in the second phase.

This paper makes two contributions. First, it compares two techniques for producing summary data in the first phase. One technique is to build a *cluster feature tree* (CF-tree) in the available memory, as has been proposed in the BIRCH clustering algorithm.[14] The second is to draw a random sample from the dataset. We present empirical data that indicate that sampling is much faster than building the CF-tree, and results in clusterings of comparable quality. Sampling is also much simpler, in the sense that it is easier to implement and easier to control.

The amount of memory available for two-phase clustering affects both the quality of the resulting clusters and the clustering time. Our second contribution is a quantification of the tradeoff between memory size and clustering error when sampling is used in the first phase of the algorithm. Specifically, we develop a formula that can be used to estimate the amount of memory required to keep sampling-induced clustering error below a specified threshold. This can be used to determine the amount of memory required to use two-phase clustering without sacrificing cluster accuracy.

Our results also demonstrate that two-phase clustering can be very fast and accurate, particularly if sampling is used in phase one. For example, k-means clustering of one million three-attribute tuples in memory requires about 10 CPU-minutes on a SPARCstation 5. Two-phase clustering of the same data can be accomplished in about 15 CPU-seconds, with clustering error within a few percent of the error achieved using the full dataset. Two-phase techniques appear to scale well; we experimented with datasets one to two orders of magnitude larger than those used in previous

studies.

2 Related Work

Many clustering algorithms have been proposed. Several of the major classes of algorithms are described in Section 2.1. More detailed coverage of these algorithms can be found in [6, 7]. Most of these algorithms are intended for clustering memory-resident data. Several recently-proposed algorithms have been designed specifically for large datasets. These are discussed in Section 2.3.

Although clustering algorithms exist for many types of data, we will restrict ourselves in this paper to algorithms for metric data. It is possible to apply two-phase clustering to other types of data, such as categorical data, provided that both phases can accommodate it. Sampling is an example of a phase one summarization technique that applies equally well to metric and non-metric data.

2.1 Clustering In Memory

A widely-used class of clustering algorithms for metric data is based on the iterative k-means method.[6] Clusters are defined using a set of k cluster representatives, or centroids. Each representative is a point in the metric space in which the data is defined. A representative defines a cluster consisting of those points from the dataset that are closer to it than to any other representative.

A k-means algorithm starts with some initial set of representative points. Each iteration then consists of two steps First, each data point is assigned to the representative to which it is closest. Second, each representative is replaced by the centroid (mean) of the points that were assigned to it. The next iteration uses the newly calculated representatives. The algorithm can be terminated after a fixed number of iterations or after the resulting clustering error ceases to improve significantly. Each iteration of a k-means algorithm takes $O(kN)$ time, where N is the number of points being clustered, since the distance from each point to each centroid is computed. For large data sets (not memory resident), k-means algorithms require one scan of the data per iteration. Although k-means algorithms tend to converge quickly in practice [6], the number of iterations will normally be greater than one.

A second, similar class of algorithms is based on the iterative k-mediods method. In such algorithms, the cluster representatives are points from the dataset, rather than arbitrary points in the space. An initial set of k representatives, known as mediods, is first selected. As in the k-means method, each representative defines a cluster consisting of those points from the dataset that are closer to it than to any other representatives. Starting with the initial set of representatives, a k-mediods algorithm iteratively attempts to swap mediods with other points from the dataset, if such a swap reduces the clustering error. One such algorithm is PAM. An improved version of PAM, called CLARANS, uses a more sophisticated search strategy for determining which data points to swap with current mediods.[9] Both algorithms are intended for memory-resident data.

Other types of clustering algorithms include agglomerative, divisive, and mode-seeking algorithms. Agglomerative algorithms start with many small clusters, e.g., one cluster per data point, and iteratively combine similar clusters. Divisive algorithms do just the opposite, starting with a single cluster. Either type of algorithm may be halted when a specified number of clusters has been obtained, or when some other stopping criterion has been satisfied. Mode-seeking algorithms attempt to identify regions with a high point density.[6]. One potential advantage of such algorithms is that they may be able to identify irregularly shaped clusters. In contrast, k-mediod and k-mean algorithms are tend to identify regular, spherical clusters since each cluster is represented by a single central point.

2.2 Sampling

A second related topic is sampling, regarding which there is also a large literature. Two useful introductions are [1] and [2]. Of primary interest here are techniques for simple random sampling, i.e., sampling in which each element of the population has an equal chance of inclusion.

Reservoir sampling is a fast technique for generating simple random samples using a single sequential pass over the data.[13] We describe reservoir sampling further in Section 3.2 Techniques have also been proposed for generating simple random samples from index structures such as B-trees and hash files while minimizing disk operations.[10] Any of these techniques could be used in the first phase of two-phase clustering.

2.3 Clustering Large Datasets

There are several proposed clustering techniques for large datasets. Most are two-phase. That is, they first produce a compact in-memory representation of the dataset and then cluster that, rather than the original dataset.

Perhaps the first such algorithm was CLARA, which involves drawing a random sample and then applying PAM to the sample.[7] CLARA is not strictly a two-phase algorithm because this procedure is repeated several times, with the best result retained. Suggested guidelines for sample size and the number of iterations were based on empirical evidence obtained with very small data sets.

Several recently-proposed algorithms use sampling as the first phase. One proposal uses the CLARANS algorithm in the second phase.[4]. Another, called CURE, uses a second-phase algorithm that maintains multiple representative points per cluster.[5] The use of more than one representative distinguishes this algorithm from those described in 2.1 and allows it to identify non-spherical clusters.

BIRCH is a two-phase algorithm that does not use sampling in phase one.[14] Instead, it constructs an in-memory hierarchical summary of the full dataset. The BIRCH algorithm is described more fully in Section 3.1.

We are aware of one non-two-phase clustering algorithm for large data sets. DBSCAN is a mode-seeking algorithm that, like CURE, is capable of identifying irregularly shaped clusters.[3] DBSCAN

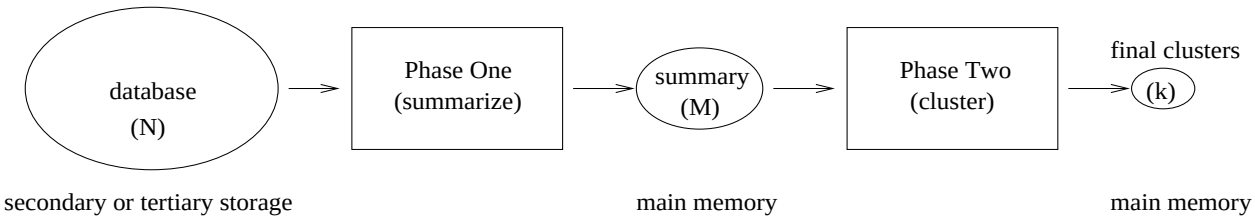

Figure 1: Two-Phase Clustering

assumes the existence of a multi-dimensional index on the dataset. The algorithm identifies clusters by attempting to expand known regions of high point density. During the expansion process, the index is used to support efficient retrieval of data points located near the known region. Unlike the two-phase algorithms described above, DBSCAN does not perform a single sequential scan of the dataset. Instead, the algorithm performs at most one index scan per data point. Thus, its I/O cost may be substantially higher than that of a two-phase algorithm for large datasets. It would be interesting to compare DBSCAN with a two-phase algorithm with similar capabilities, such as CURE, using large datasets. However, the purpose of this paper is to evaluate algorithms for use in the first phase of two-phase algorithms. For this reason, the DBSCAN will not be considered further here.

Also of note is recent work on the construction of histograms for one dimensional [12] and multidimensional [11] data. Although creating histograms is not the same as clustering, both techniques extract information about the data distribution. For large datasets, these histogram construction algorithms are two-phase, meaning that the data is sampled and then histograms are constructed using the sample. For the one-dimensional case, analytical results are available for determining appropriate sample sizes.[12]

3 Two-Phase Clustering

The two-phase clustering technique is summarized in Figure 1. The first phase uses a single pass over the dataset to create a memory-resident summary. The second phase generates k clusters from the memory-resident summary. The output of this phase is k cluster centroids.

A parameter M limits the memory used by the first phase. The value of M can be controlled to effect a tradeoff between clustering time and clustering accuracy, The second phase will require additional memory over and above M to store information about the clusters. The exact amount depends on the clustering algorithm that is used. Normally the amount of additional memory will be small relative to M and will depend only on the target number of clusters, k. For example, the k-means clustering algorithm we use requires space for two arrays, each of length k. For these reasons, we describe the total memory requirement as simply M.

In this paper we compare two techniques for producing the in-memory summary during the first phase. The first is to construct a cluster feature tree (CF-tree), first proposed as part of the BIRCH algorithm.[14] The second is to draw a random sample from the dataset. This latter approach is

used in CURE and in other algorithms. Both techniques require only a single sequential scan of the dataset. Both also allow the amount of memory, M, to be controlled. Sections 3.1 and 3.2 describe these techniques in more detail.

3.1 Cluster Feature Trees

A CF-tree is a height-balanced tree in which each node contains as many as B entries.[14] The tree has a maximum branching factor of B, since each entry includes a pointer to a child node. Each entry also contains sufficient information to describe the centroid, size (number of points), and intra-cluster variance of a cluster of data points. The locations of the individual points within the cluster are not stored. The entire tree represents a hierarchical clustering of all points from the dataset that have been inserted into the tree. Entries in leaf nodes represent clusters - every inserted point is part of exactly one such cluster. Entries in internal nodes represent meta-clusters that incorporate all of the points in the clusters in their sub-trees. That is, the clusters at one level of the tree are subsumed by clusters at the level above.

During phase one, a CF-tree is built in memory by scanning the database and inserting each point into the tree. Points are inserted by traversing the tree from root to leaf. At each node, the entry whose centroid is closest to the point is determined, and the search continues in that entry's sub-tree. When a leaf is reached, one of two things occurs. If the new point is close enough to one of the cluster centroids in leaf, then it is inserted into that cluster. This simply involves updating the cluster features for that entry in the leaf; all of the individual points that make up each cluster are not stored. If the point is not close enough to any centroid in the leaf, then a new entry (i.e., a new cluster) containing only the newly inserted point is created in the leaf. If the leaf is full, it is split, and the leaf's parent is updated. In general, this splitting procedure may continue all the way to the root node.

The key to the CF-tree is a threshold parameter, T. This parameter is used at the leaf nodes during insertion to determine whether the inserted point is close enough to one of the existing clusters in the leaf. If T is too small, the tree will grow too quickly and may eventually exhaust the available memory. If T is too large, the tree will grow too slowly, and the memory will not be fully utilized. This results in clusters that are unnecessarily large, resulting in over-summarization of the data (information loss).

Since the maximum size of the tree is constrained by M, there must be a procedure for recovering in case the CF-tree fills the available memory before all of the database points have been inserted. This is handled by increasing T and then rebuilding the tree using the new threshold. Rebuilding does not involve rescanning points that had already been inserted into the original tree. Instead, a new, empty tree is created, and the centroids of the leaves of the original tree are treated as (weighted) data points and are inserted into the new tree, which will hopefully be smaller than the original because of its larger threshold. Once this is complete, the process of inserting dataset points picks up where it left off, only now in the new tree.

The question of how to set T is a difficult one to answer, as is noted in [14]. The ideal value of

T depends on the distribution of points in the dataset, the size of the dataset, and the amount of memory available to hold the CF-tree. BIRCH starts with a small initial threshold (e.g., zero) and increases it each time the tree is rebuilt using a rather complex extrapolation heuristic based on a linear regression over the history of previous threshold values. In our CF-tree implementation, we used a similar but simpler threshold adjustment procedure. Specifically, we considered only the two most recent threshold values when performing the regression. The experiments reported in Section 4.5 address this issue further.

3.2 Random Sampling

Another way to summarize the database is to draw a random sample from it. In keeping with the constraints on the two-phase algorithm, this requires a sampling technique that can draw a random sample of a specified size (M) using only one pass through the dataset. Reservoir sampling algorithms do exactly this. The general idea of these algorithms is as follows. To draw a sample of size M, a "reservoir" of M items is maintained as the database is scanned. The contents of the reservoir always constitute a random sample of that portion of the database that has already been scanned. Initially, the reservoir contains the first M items from the database. When the scan reaches item i from the database ($i > M$), that item is added to the reservoir with probability M/i, displacing one randomly selected entry.

A straightforward implementation of this technique requires the generation of N random numbers, where N is the number of tuples in the dataset.[2] In our experiments we used an optimized version of the algorithm, called Algorithm X in [13], which we found to run somewhat faster than the straightforward algorithm. It requires fewer random numbers, though the expected CPU time is still $O(N)$. Further refinements to the technique, which result in sub-linear expected CPU time, are described in [13]. We did not implement them.

Note that the construction of a CF-tree requires that all of the points from the dataset be read. Sampling does not. However, our implementation of sampling does read the entire dataset sequentially from the disk, as does our implementation of the CF-tree. Thus, both techniques have the same I/O cost. In general, however, it may be possible to reduce the I/O cost of sampling, depending on how the data is blocked on secondary storage and on the sampling fraction. This issue is addressed in [10] for several types of database files.

3.3 Phase Two

Section 4 presents a series of experiments intended to measure the impact of the first phase on the overall run time and accuracy of two-phase clustering. For the purpose of these experiments, the same phase two algorithm was combined with each of the two phase one algorithms, so that the effect of phase one could be isolated. A k-means clustering algorithm due to Linde, Buzo, and Gray was used in phase two.[8]

[2] Actually, only $N - M$ are needed.

The k-means algorithm was chosen because it is simple, fast, and easy to understand, and because it is well-suited to the datasets to be clustered. It has no data-dependent parameters that require tuning. The k-means algorithm was also easily adapted to work with either of the two types of summary data from phase one. If sampling is used in phase one, the clustering input is simply the set of sampled points. If a CF-tree is used, the input consists of the centroids of the clusters from the leaf nodes of the final CF-tree. In this case, each input point has an associated weight, namely, the size of its cluster. Since the clustering algorithm expects unweighted input, we treat a point with weight w as w co-located, unweighted points.

The k-means algorithm chooses its initial cluster centroids randomly from the data space, using a uniform distribution. The algorithm terminates when the relative decrease in clustering error achieved by an iteration drops below a specified threshold. In all of our experiments, this threshold was set at one percent. Like most clustering algorithms, it is not guaranteed to converge to an optimal (minimum error) set of cluster centroids. In particular, the choice of initial cluster centroids may have an effect on the result. However, the algorithm does guarantee that the clustering error does not increase on successive iterations.

One appealing feature of this algorithm is that it is completely independent of the order in which its input data is presented. When combined with an order-independent phase one technique, like random sampling, the result is a clustering technique for large data sets that is insensitive to the input order.

More sophisticated clustering techniques do exist, and could be used in conjunction with either of the summarization algorithms. Some of these techniques were described in Section 2. The choice of an clustering algorithm for phase two depends primarily on the application: what types of clusters must be identified, and what level of performance is required? The choice may also be influenced by the type of summary data generated during phase one.

4 Performance

We ran a series of experiments designed to compare sampling and CF-tree construction as means of summarizing large datasets. Each experiment involved clustering a dataset using a two-phase algorithm consisting of either sampling or CF-tree construction in phase one, and k-means clustering in phase two. Because phase two is the same in both cases, observed differences in the overall performance of the two-phase algorithm can be attributed to the summarization technique used in phase one.

We chose to feed randomly-generated data to the these algorithms, primarily because this allowed us to control the properties of the data. Our experiments measured two properties of the clustering algorithms: clustering error and execution time.

Symbol	Default Value	Description
d	3	dimensionality of the data space
N	10^6	total number of points (tuples)
K	10^2	number of point clusters
σ_{max}	0.005	maximum cluster deviation

Figure 2: Database Generation Parameters

4.1 Properties of the Input Data

We implemented a database generator that produces random datasets using the parameters described in Table 2. The datasets consist of N points in a d-dimensional unit hypercube, that is, the attribute values in each of the d dimensions are considered to have been normalized to the range $[0, 1]$. Each point is placed in one of K clusters with an equal probability of placement in each. This results in variable cluster sizes, with a mean size of N/K. The cluster centroids are distributed randomly and uniformly in the data space. Once assigned to a cluster, each of the point's coordinates is chosen independently using a Gaussian distribution around the cluster centroid and with variance chosen randomly between 0 and σ^2_{max}. For a given cluster, the same variance is used in each dimension. The default parameter values shown in Table 2 were used in each experiment, unless otherwise indicated.

Figure 3 shows an example of a two dimensional dataset produced by our generator. The parameter values shown in Table 2 were used, except that $d = 2$ and N was scaled down to 10^4 so that the distribution could be plotted clearly. As the example illustrates, the datasets are non-uniform and exhibit varying degrees of clustering within the space.

4.2 Algorithm Implementation

We implemented both CF-trees and and reservoir sampling for phase one, and the k-means clustering algorithm described in Section 3.3 for phase two. Table 4 shows the parameters for these algorithms, and their default values. Both the database size, N, and the memory size, M, are measured in tuples.

When random sampling is used in phase one, the size of the sample drawn is M. When a CF-tree is used in phase one, we must determine the maximum number of CF-tree nodes that can fit into the available memory. Assuming that each attribute value occupies one word, then the total size of the memory in words is Md. Each entry in a CF-tree node requires $d + 2$ words, since a cluster feature consists of the linear sum of the points in the cluster (d words) plus the square sum of the points and the number of points (one word each) in the cluster.[14] Since each node consists of B entries, the maximum number of CF-tree nodes in a memory of size M is given by:

$$\frac{Md}{B(d+2)}$$

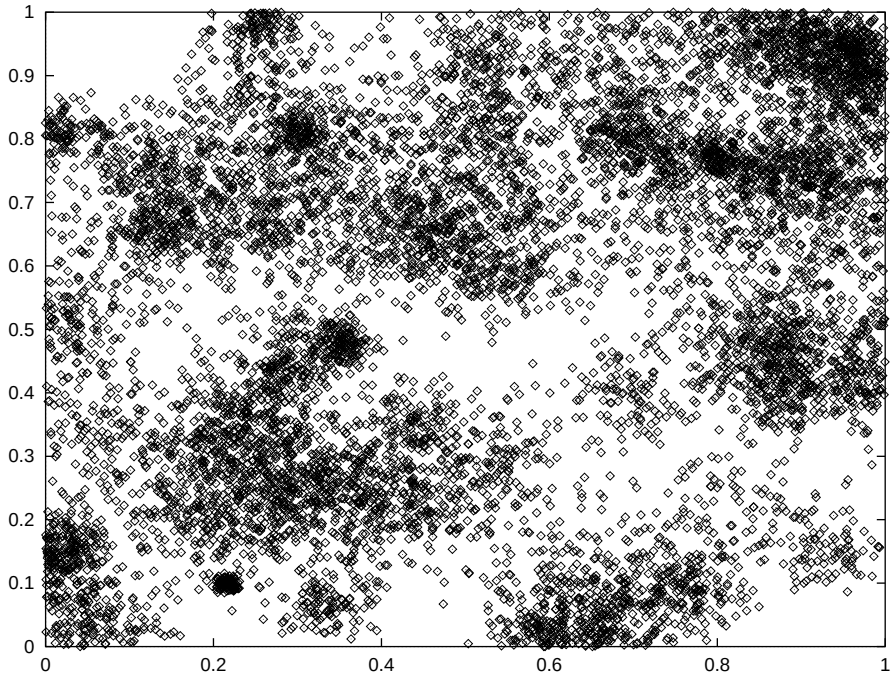

Figure 3: An Artificially Generated Dataset in Two Dimensions

Symbol	Default Value	Description
M	$50,000$	memory available for phase one (in tuples)
k	100	number of clusters produced by phase two
B	20	CF-tree branch factor (entries per node)
τ	0.01	termination threshold for K-means clustering

Figure 4: Clustering Algorithm Parameters

This simple analysis ignores the pointers necessary to link the CF-tree nodes into the tree, thus it somewhat overestimates the maximum size of a CF-tree in a memory of size M.

4.3 Experimental Method

All of the experiments ran on a Sun SPARCstation-5 running SunOS 5.5. All of the algorithms were implemented in C, and compiled using version 2.7 of the GNU compiler with optimization (-O2) enabled.

Each experiment consisted of running a two-phase clustering algorithm against a dataset that had been previously generated and stored in a file. For each experiment, we measured the CPU time required for each phase of the clustering algorithm. This includes the CPU time involved in such activities as reading the input data from the file system. We chose to measure CPU time rather than wall-clock time to avoid measuring the effects of background load on the experimental machine and the effects of file system buffering and prefetching of the database file. Note that both algorithms (CF-tree and random sampling) make a single sequential pass through the database file.

The clustering error is defined as the root mean square Euclidean distance between the database points and their nearest cluster centroids. We computed the clustering error using a second pass through the dataset after clustering was finished. The time required for this second pass was not included in the execution time measurement. Note that many cluster error metrics have been proposed. The choice of a metric ultimately depends on the intended application of the clusters. The error metric that we have used is a common application-independent choice. It is also well-suited to the k-means algorithm, which directly seeks to minimize the distances between cluster points and centroids. Others include cluster diameter (average point-to-point distance within the cluster) and various inter-cluster distance metrics.

4.4 Experiment One: Memory Size

The key issue for two-phase clustering is performance as a function of the amount of memory available for the first phase. In our first experiment, we ran both clustering algorithms with varying amounts of memory (M) while holding all other parameters at their default values. This experiment was repeated for ten randomly generated datasets, each generated using the parameters shown in Table 2.

The results of the experiment are shown in Figures 5 and 6. The curves show mean values computed over the ten datasets, with error bars at one standard deviation from the mean. In Figure 5 we have also plotted separate curves showing the mean CPU time for phase one only for each algorithm.

The results show that the sampling algorithm is both much faster than the CF-tree algorithm and at least as accurate. For example, when enough memory was available to hold half of one percent of the dataset (our default memory size), the CF-tree algorithm required more than 3 CPU-minutes to produce clusters of quality identical to those produced by the sampling algorithm in about 15 CPU-seconds, an order of magnitude speedup.

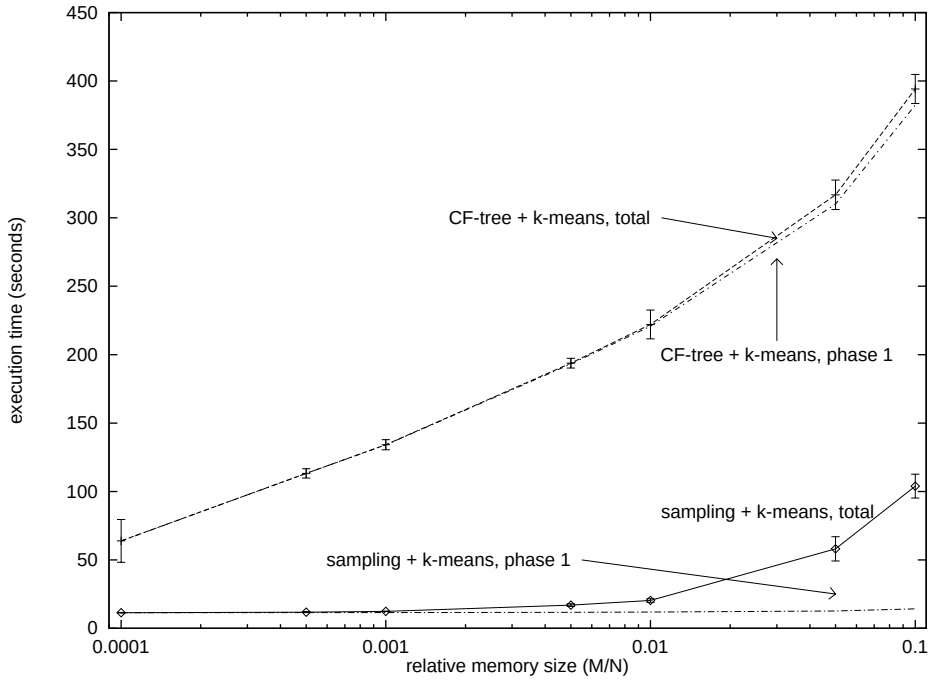

Figure 5: CPU Time Required for Two Clustering Algorithms

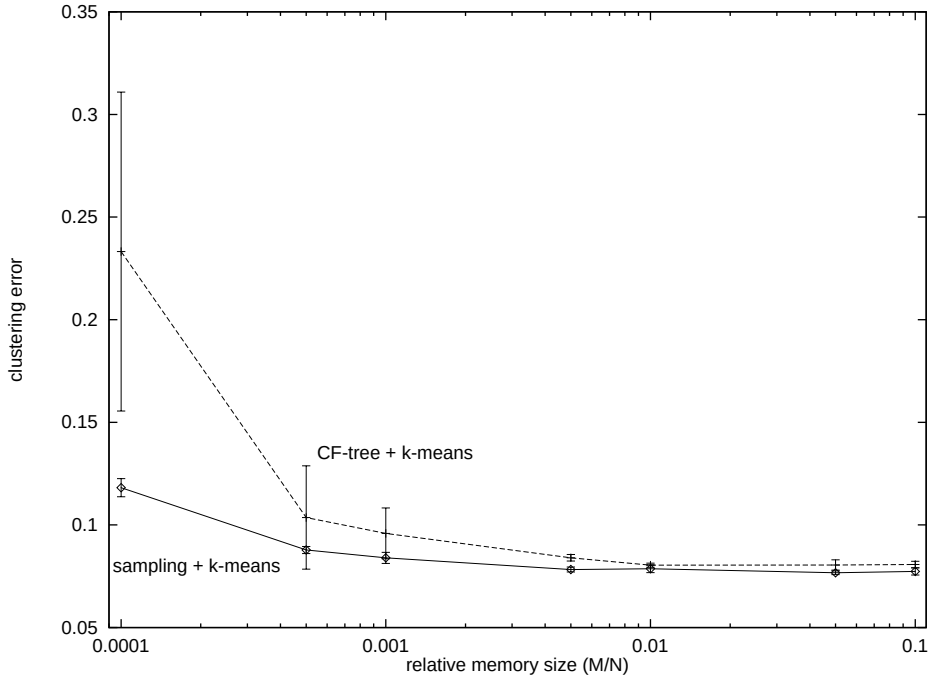

Figure 6: Clustering Error Produced by Two Clustering Algorithms

Figure 5 shows that the execution time of the CF-tree algorithm is due almost entirely to the construction of the CF-tree in the first phase. This time increases with the memory size, while time for sampling remains essentially flat. The CF-tree insertion algorithm involves many point-to-point distance calculations to determine the appropriate leaf node for each inserted point. These calculations are expensive in three dimensions and can be expected to increase in spaces of higher dimension. Sampling, on the other hand, requires no distance calculations.

Figure 6 shows that in terms of error reduction, the marginal rate of return of additional memory falls off sharply for both algorithms. When little memory was available, the CF-tree algorithm began to experience extreme variance from database to database. We found that this was due to the CF-tree's threshold adjustment heuristic, a subject to which we turn in the next experiment.

A comparison of Figures 5 and 6 shows that when the memory size becomes large the execution time of the sampling algorithm begins to increase (though it remains much faster than the CF-tree algorithm), with no corresponding decrease in clustering error. The increase is due to the k-means algorithm in phase two, which is faced with larger input when there is plenty of memory. This illustrates the need to control the amount of memory used by the sampling technique; more memory is not necessarily better. This is a subject to which we return in Section 5.

4.5 Experiment Two: Threshold Adjustment

The original BIRCH implementation used a complicated heuristic for adjusting the CF-tree threshold in case the available memory was exhausted.[14] We implemented a simpler version of that heuristic, but found that it resulted in the erratic behaviour shown in Figure 6 when the available memory was small.

Rather than trying to come up with a better heuristic, we sought to determine how the CF-tree algorithm would perform if it were given advance knowledge of an appropriate threshold to use. In a sense, this represents the performance we would expect from an ideal threshold adjustment heuristic, if such a heuristic could be created. For each of our ten test databases, and each memory size (the ideal threshold depends on both the data distribution and the amount of memory available), we used binary search to find a threshold value that would cause the CF-tree to just fill the available memory, with no rebuilding. This is a very time consuming process, though it is possible that ideal threshold values might be determined in some other way, perhaps using information about the distribution of points in the dataset. We then ran the CF-tree algorithm using the ideal threshold values and measured clustering error and execution time. The resulting clustering error is shown in Figure 7. The data for the sampling algorithm and the original CF-tree algorithm (from Figure 6) are also shown for comparison. The use of ideal thresholds had only a minor effect on the execution time of the algorithm, so we have not presented those data here.

Our data show that even under ideal conditions, the CF-tree algorithm results in clustering error comparable to what is achieved using sampling, unless the amount of memory available is very small. The use of ideal threshold values did eliminate most of the variance we observed when

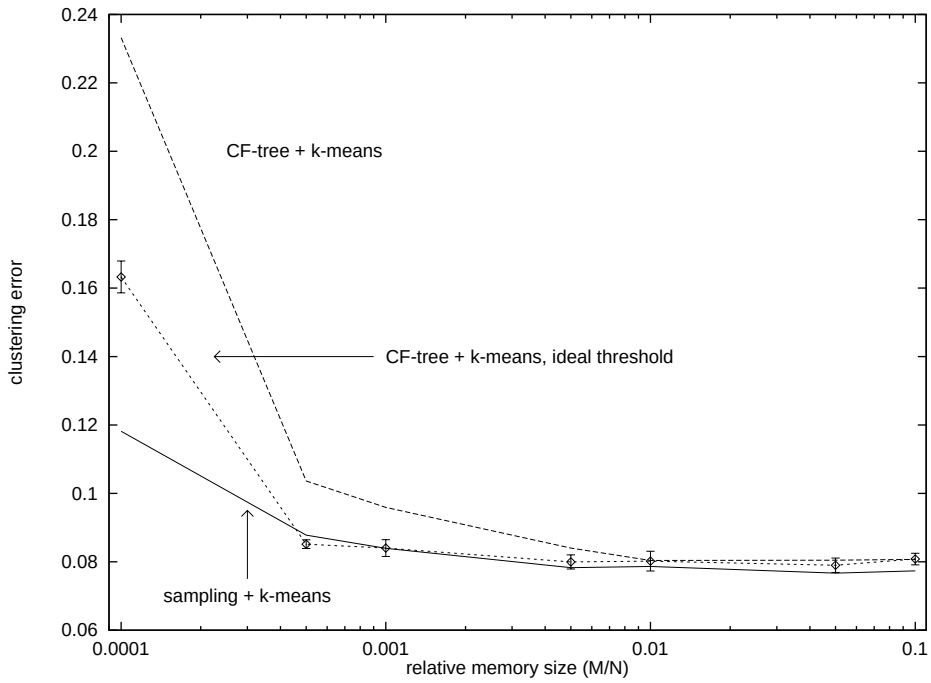

Figure 7: Clustering Error Using Ideal CF-Tree Thresholds

the threshold adjustment heuristic was used, as indicated by the shorter error bars. This indicates the importance of a good heuristic to the CF-tree algorithm. Fortunately, no such heuristic is required if sampling is used.

4.6 Experiment Three: Scalability

We ran experiments in which the database size was varied over several orders of magnitude while the remaining parameters, including memory size, were held at their default parameters. In particular, as the database was scaled up, the number of clusters (K) in the generated database remained constant. Figures 8 and 9 show the results. The curves show the mean clustering time and clustering error over ten randomly-generated datasets.

Since we are holding the memory size constant, phase two execution time remains essentially constant as the database changes size. Increases in total clustering time are due to phase one. Clearly, the key to clustering large databases quickly will be fast sampling. Neither our sampling implementation nor our CF-tree implementation are particularly efficient. We believe that the slopes of both curves could be reduced substantially with additional optimization.

Figure 9 demonstrates that the clustering error is essentially independent of the database size when sampling is used. Since the memory size (and hence the sample size) remained fixed as the database was scaled up, the sampling fraction effectively decreased as the database grew. This did not have a negative impact on the clustering error. The CF-tree algorithm did not fare as well on larger databases, and showed large variations from databases to database (not shown in Figure 9), again due to the vagaries of the threshold adjustment heuristic.

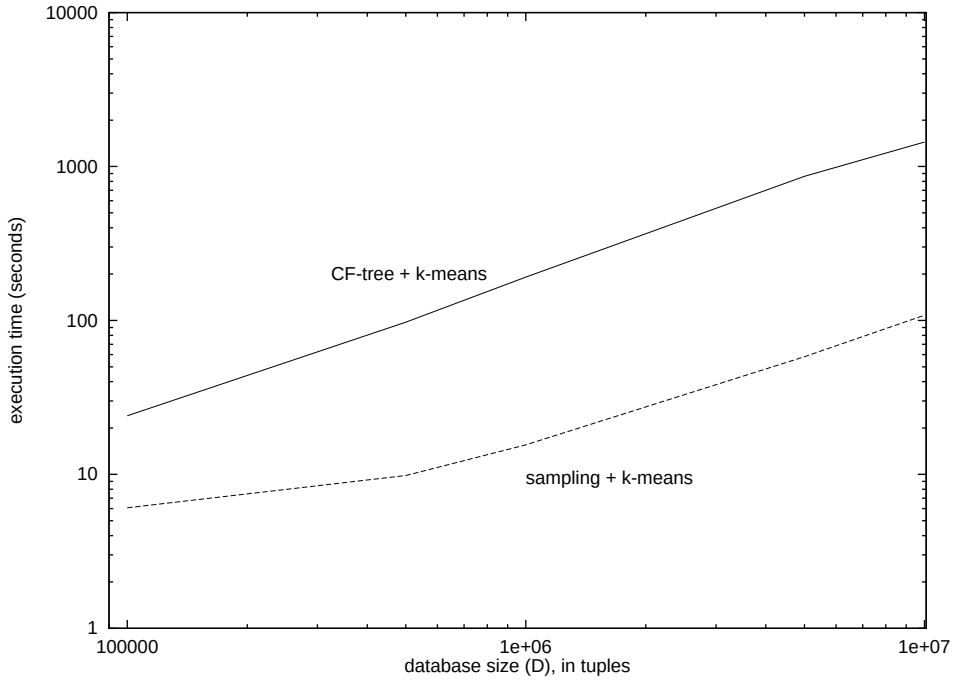

Figure 8: CPU Time for Clustering as a Function of Database Size

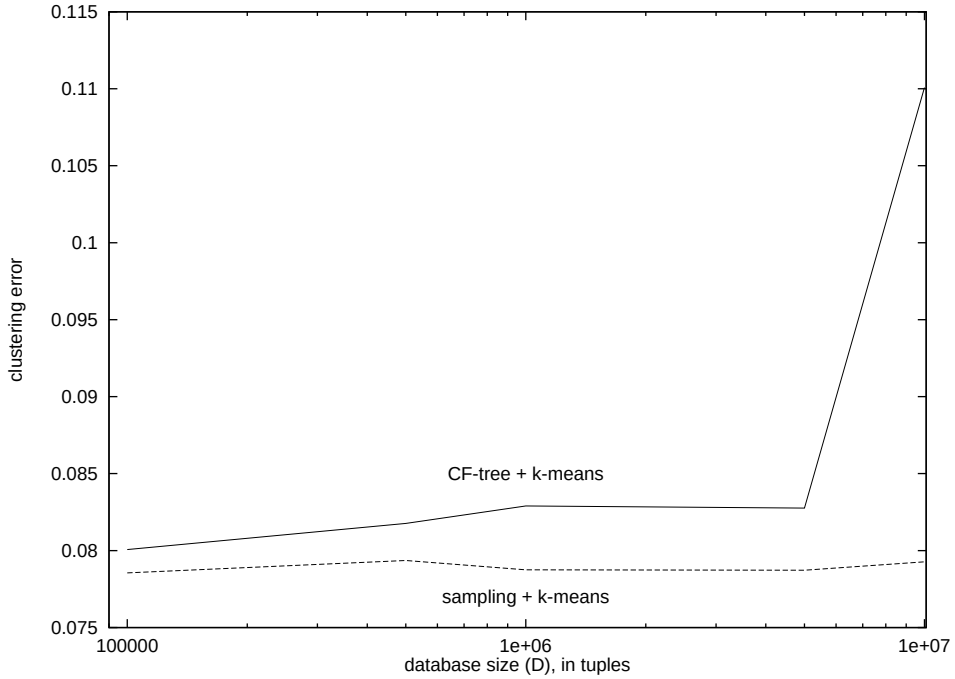

Figure 9: Clustering Error as a Function of Database Size

5 Controlling Memory Size

Although two-phase clustering may be significantly faster than clustering an entire dataset directly, the summarization performed in the first phase will generally introduce additional error in the resulting clusters. The amount of additional error depends on the amount of memory available to hold the summary data. As illustrated in Figures 5 and 6, insufficient memory results in larger errors, while too much memory increases clustering time without decreasing the error significantly.

We are interested in characterizing this additional error as a function of the memory size. Suppose that we run the k-means clustering algorithm against a complete dataset $\mathcal{D}$ of size N. For every point $x \in \mathcal{D}$, we define $\delta^2(x)$ to be the squared distance between x and the cluster centroid it is closest to. (Squared distances allow us to focus on the magnitude of the error, rather than its sign.) Next, suppose that we draw a random sample $\mathcal{S}$ of size M from the same dataset and then apply the k-means algorithm to the sample. For every point $x \in \mathcal{D}$, we define $\hat{\delta}^2(x)$ to be the squared distance between x and the closest centroid produced using the sampled data. We define the additional (squared) error resulting from sampling as

$$\varepsilon^2 = \frac{\sum_{x \in \mathcal{D}} \hat{\delta}^2(x) - \sum_{x \in \mathcal{D}} \delta^2(x)}{\sum_{x \in \mathcal{D}} \delta^2(x)} \tag{1}$$

We call ε the sampling error, or summarization error. Note that ε is dimensionless. It is the factor by which clustering error increases as a result of sampling, relative to the clustering error without sampling. For example, $\varepsilon = 0.1$ means that clustering error has increased by 10 percent as a result of sampling.

We will require the following notation. When the k-means algorithm is applied directly to the whole database, we will use c_i to denote the ith centroid ($1 \leq i \leq k$) produced by a given iteration of the algorithm, and $\mathcal{D}_i$ to denote the points in the ith cluster. Recall that $\mathcal{D}_i \subseteq \mathcal{D}$ consists of those points from the dataset that are closer to c_i than to any other centroid. The size and variance of $\mathcal{D}_i$ are N_i and σ_i^2, respectively.

For the case in which clustering is applied to sampled data, we will use $\hat{c}_i$ to denote the centroid of the ith cluster, and $\hat{\sigma}_i^2$ to represent its variance. The subset of the dataset that is closest to $\hat{c}_i$ will be written $\hat{\mathcal{D}}_i$. Finally, $\mathcal{S}_i = \hat{\mathcal{D}}_i \cap \mathcal{S}$ will represent that portion of $\hat{\mathcal{D}}_i$ that is included in the sample. The size of $\mathcal{S}_i$ is M_i.

We will make several additional assumptions to achieve a tractable analysis. Our main assumption will be that after every iteration of the k-means algorithm, $\mathcal{D}_i = \hat{\mathcal{D}}_i$. In other words, the database partitions produced by the clustering algorithm are the same, regardless of whether the database is sampled first.[3] In general this is not true. Sampling will perturb the cluster centroids, and even tiny movements of the centroids may cause tuples to jump from one cluster to another. However, we will assume that this effect is negligible if these sampling-induced perturbations are small. Under this assumption, we can rewrite our expression for sampling-induced error on a cluster

[3] This does not imply that the reported cluster centroids will be the same in either case, since the two-phase algorithm will calculate centroids based only on the sampled portion of the database.

by cluster basis. After every iteration of the k-means algorithm, the squared sampling error is given by:

$$\varepsilon^2 = \frac{\sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} \hat{\delta}^2(x) - \sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} \delta^2(x)}{\sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} \delta^2(x)}$$

Assuming for now that the the database is one dimensional, we can rewrite this as

$$\varepsilon^2 = \frac{\sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} (x - \hat{c}_i)^2 - \sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} (x - c_i)^2}{\sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} (x - c_i)^2} = \frac{\sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} (x - \hat{c}_i)^2 - \sum_{i=1}^{k} N_i \sigma_i^2}{\sum_{i=1}^{k} N_i \sigma_i^2}$$

Since there are many possible samples that might be drawn from the database, we will treat the sample centroids $\hat{c}_i$ as random variables. Hence, ε^2 is also a random variable. We wish to estimate the expected value of ε^2 over all possible samples of a given size M. The expected value of ε^2 can be written

$$E[\varepsilon^2] = \frac{\sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} E[(x - \hat{c}_i)^2] - \sum_{i=1}^{k} N_i \sigma_i^2}{\sum_{i=1}^{k} N_i \sigma_i^2}$$

If X is a random variable with mean μ_X and variance σ_X^2, the following is an approximation for the expected value of a twice-differentiable function $f(X)$.

$$E[f(X)] \approx f(\mu_X) + \frac{f''(\mu_X)}{2} \sigma_X^2$$

Applying this, we obtain:

$$E[\varepsilon^2] \approx \frac{\sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} [(x - \mu_{\hat{c}_i})^2 + \sigma_{\hat{c}_i}^2] - \sum_{i=1}^{k} N_i \sigma_i^2}{\sum_{i=1}^{k} N_i \sigma_i^2}$$

In Appendix A, we show that, under certain assumptions, $\mu_{\hat{c}_i} = c_i$ and $\sigma_{\hat{c}_i}^2 \approx \sigma_i^2 g(MN_i/N)$, where the function g is defined by

$$g(y) = e^{-y}[\mathrm{Ei}(y) - \ln(y)]$$

and Ei is the exponential-integral function, a well-known special function. Substituting and simplifying, we obtain

$$E[\varepsilon^2] \approx \frac{\sum_{i=1}^{k} N_i \sigma_i^2 g(MN_i/N)}{\sum_{i=1}^{k} N_i \sigma_i^2} \tag{2}$$

In general, sampling error depends on the cluster variances and on the distribution of cluster sizes. However, by considering the special case in which all cluster sizes are the same, the expression can be simplified even further. Substituting $N_i = N/k$ for all i, we are left with

$$E[\varepsilon^2] \approx g(M/k) \tag{3}$$

Note that this estimate is independent of both the cluster variances and the database size. The sampling error depends only on the total sample size M and the target number of clusters k. Intuitively, clusters that have high variance are both hard to cluster well and hard to sample. Though they have higher sampling error, they have higher clustering error as well. Because the

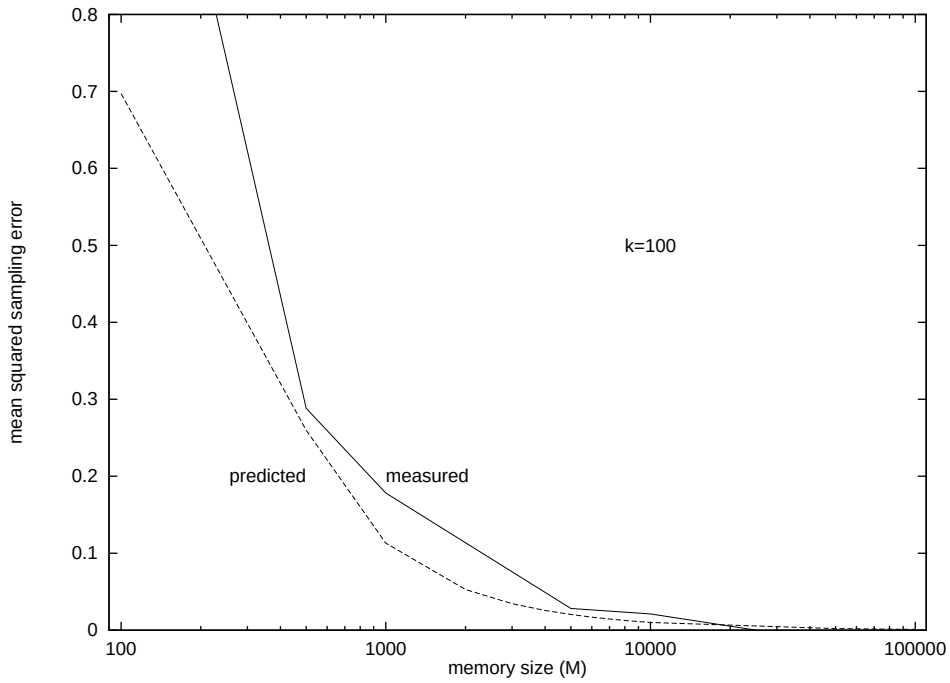

Figure 10: Predicted and Measured Sampling Error

sampling error is normalized relative to the clustering error, these effects cancel out. Conversely, "tight" clusters with low variance are both easy to cluster and easy to sample accurately.

For the special case in which the cluster sizes are all equal, the expected squared sampling error remains unchanged for datasets of dimension greater than one. In d dimensions, we can rewrite our definition of sampling error as

$$\varepsilon^2 = \frac{\sum_{x \in \mathcal{D}} \sum_{j=1}^{d} \hat{\delta}_j^2(x) - \sum_{x \in \mathcal{D}} \sum_{j=1}^{d} \delta_j^2(x)}{\sum_{x \in \mathcal{D}} \sum_{j=1}^{d} \delta_j^2(x)}$$

where the functions δ_j^2 and $\hat{\delta}_j$ measure squared distance in dimension j. Using the fact that

$$\sum_{x \in \mathcal{D}} \hat{\delta}_j^2(x) = (\varepsilon_j^2 + 1) \sum_{x \in \mathcal{D}} \delta_j^2(x)$$

where ε_j^2 is the sampling error in dimension j, and that $E[\varepsilon_j^2] \approx g(M/k)$, independent of the dimension, we can re-obtain Equation 3 for any value of d.

Figure 10 illustrates the function $E[\varepsilon^2]$ defined by Equation 3 and compares the sampling errors it predicts with those we observed in our experiments, for $k = 100$. To determine the observed errors, we first clustered each of our ten random databases (generated using the parameters in Figure 2) fully, without clustering. The measured sampling error was then determined using the actual clustering error (with sampling) and the error without sampling according to Equation 1. In the figure we plotted the mean observed squared sampling error over the ten databases.

Until the sample size becomes very small, the predicted and measured errors are close, with the prediction underestimating the measured error slightly. For very small sample sizes the underestimation is more significant. Under such conditions, the expected number of samples per

cluster becomes small, and the chance that the sample will completely miss a cluster becomes non-negligible. This violates an assumption that we made in estimating the statistics of the $\hat{c}_i$ in Appendix A. For larger sample sizes, the accuracy of the predictions suggests that our remaining assumptions were reasonable.

For $k = 100$, a sample size of about 2000 is sufficient to hold the predicted squared sampling error to about five percent. Since the predictions are low, a sample size several times larger, perhaps 3000 or 4000, is probably more prudent. Larger sample sizes do very little to further reduce the error.

For other values of k, we note from Equation 3 that the number of samples required to maintain constant sampling error increases in proportion to k. Thus, to find ten times as many clusters, one should use a sample ten times as large. From this, we conclude that for any desired k a sample size between $30k$ and $40k$ is probably adequate, independent of the database size. For other error levels, the appropriate multiplier for k can be found using Equation 3.

6 Discussion

The error analysis in Section 5 was specific to the k-means clustering algorithm that was used in phase two. The analysis was used to suggest guidelines for sample size selection. An obvious question is how dependent those guidelines are on the clustering algorithm. This question is difficult to answer definitively because there are many possible clustering algorithms. Furthermore, those algorithms tend to be difficult to analyze and compare because they are often iterative and complex.

Another way to provide sample size guidelines is to analyze the sampling procedure alone, ignoring the clustering algorithm used in phase two. The approach has the disadvantage that it cannot directly relate sample size to clustering error, as our analysis did. However, it can be used to characterize the sample itself. If the sample is "good", one can hope that a clustering algorithm applied to the sample in phase two will produce results similar to those that would have been produced had the entire dataset been clustered. Since the analysis does not consider the clustering algorithm, guidelines derived from it are weaker but may be more broadly applicable.

One such analysis is presented in [5]. In that analysis, a sample is considered to be good if it contains at least a specified fraction of the points from each of the dataset's k clusters. If f is the specified fraction, this assumption implies that the sample size should be at least fN for a dataset with N points. This is a very strong condition. If the database size doubles, the required sample size also doubles. Our experimental results suggest that it is not necessary to scale the sample with the database, at least for some kinds of clustering algorithms.

The equations in Appendix A can be used as the basis of a similar, but weaker, sampling-only analysis. Specifically, Equation 4 gives the expected (squared) distance between the centroid of a sampled cluster and the centroid of the full cluster. This expression can be used to determine the sample size required to keep the expected sampled cluster centroid close to the true centroid. If a "good" sample is defined as one in which the sampled centroids are close the to the true centroids, this expression can be used to determine the required sample size.

This is a much weaker definition of sample quality than the one used in [5]. However, it has the advantage that it leads to sample size requirements that do not scale with the database size. In fact, it leads to sample size guidelines very similar to those presented in Section 5. This is not too surprising since the analysis on which those guidelines were based was based in part on Equation 4. Similar but more complicated analyses can be developed for the sampling-induced error in other cluster statistics, such as variance. Stronger definitions of sample quality might insist on low sampling-induced error in several such statistics. We leave the exploration of appropriate definitions as a topic for future work.

7 Conclusion

We have described a general, two-phase approach to clustering large datasets in a single pass using limited memory. In the first phase, the dataset is summarized in memory, and in the second the summary data are clustered. Several proposed clustering algorithms for large datasets use this approach.

Using non-uniform, artificially-generated datasets, we compared two techniques for producing the summary data. One is based on CF-trees, as defined in the BIRCH clustering algorithm. The other is based on random sampling. We found sampling to be the superior performer. The sampling approach also has a number of other advantages. It is easy to understand and easy to implement. It can be used with almost any kind of input data, including categorical data. It is insensitive to the order in which the input data are presented. When combined with an order-insensitive in-memory clustering algorithm, such as k-means, this results in an order-insensitive two-phase algorithm for large datasets.

Sampling (or any summarization technique) increases cluster error. The amount of additional error depends on the sample size. We provided a formula that can be used to estimate the amount of memory required to make two-phase clustering produce clusters of about the same quality as (one-phase) clustering of the entire dataset. The estimate is independent of both the size and the distribution of the dataset.

An interesting potential application of this work is the use of cluster data to support multi-attribute selectivity estimation without the usual need for an attribute independence assumption. Recent work has begun to investigate multi-dimensional histograms for this purpose.[11] Cluster statistics may represent a useful alternative.

References

[1] Vic Barnett. *Elements of Sampling Theory.* Hodder and Stoughton, London, 1974.

[2] William G. Cochran. *Sampling Techniques.* John Wiley & Sons, third edition edition, 1977.

[3] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. Second Int. Conf. on Knowledge Discovery & Data Mining*, pages 226–231, 1996.

[4] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. A database interface for clustering in large spatial databases. In *Proc. First Int. Conf. on Knowledge Discovery & Data Mining*, pages 94–99, 1995.

[5] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 73–84, June 1998.

[6] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[7] Leonard Kaufman and Peter Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, Inc., 1990.

[8] Yoseph Linde, Andres Buzo, and Robert M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, COM-28(1):84–95, January 1980.

[9] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the International Conference on Very Large Data Bases*, pages 144–155, 1994.

[10] Frank Olken and Doron Rotem. Random sampling from database files: A survey. In Z. Michalewicz, editor, *Proc. Fifth Int'l Conf. on Statistical and Scientific Database Management*, pages 92–111, April 1990. Lecture Notes in Computer Science 420, Springer-Verlag.

[11] Viswanath Poosala and Yannis Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the International Conference on Very Large Data Bases*, pages 486–495, August 1997.

[12] Viswanath Poosala, Yannis Ioannidis, Peter Haas, and Eugene Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 294–305, June 1996.

[13] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.

[14] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 103–114, June 1996.

8 Appendix A

We are given a one-dimensional dataset $\mathcal{D}$ of size N that has been partitioned into k clusters. The ith cluster, $\mathcal{D}_i$, has size N_i, mean c_i and variance σ_i^2. A simple random sample $\mathcal{S}$ of size n is drawn from the dataset, and $\hat{c}_i$ is a random variable that represents the centroid of $\mathcal{S} \cap \mathcal{D}_i = \mathcal{S}_i$ over all possible samples. We wish to determine the mean $\hat{\mu}_i$ and variance $\hat{\sigma}_i^2$ of $\hat{c}_i$.

Note that $\mathcal{S}_i$ is a simple random sample of $\mathcal{D}_i$. Given n_i, the size of $\mathcal{S}_i$, we have

$$\hat{\mu}_i = c_i$$

and

$$\hat{\sigma}_i^2 = \frac{\sigma_i^2}{n_i}$$

assuming that the sample fraction n/N is small, which will normally be the case. However, n_i is not fixed, and will vary depending on how many of the cluster points happen to be included in a particular sample.

If the sample fraction is small, the cluster is small relative to the full dataset, and the full sample is not too small, the cluster sample size n_i will have approximately a Poisson distribution with parameter nN_i/N. This allows us to write:

$$\hat{\sigma}_i^2 \approx \sum_{j=1}^{N_i} \frac{\sigma_i^2}{j} Prob[n_i = j] \approx \sum_{j=1}^{N_i} \frac{\sigma_i^2}{j} \frac{e^{-nN_i/N}(nN_i/N)^j}{j!}$$

This expression has ignored the possibility that $n_i = 0$, i.e., that the sample misses the cluster entirely, since the sample variance is not defined in that case. Unless the cluster is extremely small, this is very unlikely.

This expression can be re-written in terms of the exponential-integral function $\mathrm{Ei}(x)$, a well-known special function defined as

$$\mathrm{Ei}(x) = \ln(x) + \sum_{j=1}^{\infty} \frac{x^j}{j(j!)}$$

Under our assumption that the sampling fraction is small, we obtain then

$$\hat{\sigma}_i^2 \approx \sigma_i^2 e^{-nN_i/N}[\mathrm{Ei}(nN_i/N) - \ln(nN_i/N)] \tag{4}$$