Parallel and Efficient Hierarchical k-Median Clustering

Vincent Cohen-Addad[*]
Google Research
cohenaddad@google.com

Silvio Lattanzi[*]
Google Research
silviol@google.com

Ashkan Norouzi-Fard[*]
Google Research
ashkannorouzi@google.com

Christian Sohler[†]
University of Cologne
csohler@uni-koeln.de

Ola Svensson[*]
EPFL
ola.svensson@epfl.ch

Abstract

As a fundamental unsupervised learning task, hierarchical clustering has been extensively studied in the past decade. In particular, standard metric formulations as hierarchical k-center, k-means, and k-median received a lot of attention and the problems have been studied extensively in different models of computation. Despite all this interest, not many efficient parallel algorithms are known for these problems. In this paper we introduce a new parallel algorithm for the Euclidean hierarchical k-median problem that, when using machines with memory s (for $s \in \Omega(\log^2(n + \Delta + d))$), outputs a hierarchical clustering such that for every fixed value of k the cost of the solution is at most an $O(\min\{d, \log n\} \log \Delta)$ factor larger in expectation than that of an optimal solution. Furthermore, we also get that in for all k simultanuously the cost of the solution is at most an expected $O(\min\{d, \log n\} \log \Delta \log(\Delta dn))$ factor bigger that the corresponding optimal solution. The algorithm requires in $O\left(\log_s(nd \log(n + \Delta))\right)$ rounds. Here d is the dimension of the data set and Δ is the ratio between the maximum and minimum distance of two points in the input dataset. To the best of our knowledge, this is the first *parallel* algorithm for the hierarchical k-median problem with theoretical guarantees. We further complement our theoretical results with an empirical study of our algorithm that shows its effectiveness in practice.

1 Introduction

Clustering is a central tool in any large scale machine learning library. The goal of clustering is to group objects into subsets so that similar objects are in the same groups while dissimilar objects are in different groups. There are many different definitions of clustering depending on the description of the objects, the similarity measure between objects, as well as the application. While it is impossible to find the perfect formulation for a clustering problem ([38], see also [19]), the metric version of the problem has received a lot of attention over the last decades and is a central topic in unsupervised learning research.

Here we focus on one fundamental metric clustering problem: the hierarchical Euclidean k-median problem. In this formulation, the input objects are described by vectors and the distance (dissimilarity) between objects is measured by their Euclidean distance. More formally, given a set $P \subseteq \mathbb{R}^d$, the objective of the hierarchical k-median problem in its classic formulation [42] is to return an ordering

[*]Equal contribution

[†]Work was partially done while author was visiting researcher at Google Research, Switzerland.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

$c_1, \ldots, c_n$ of points and a set of n nested partitions $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of P (i.e., for any $1 \leq i \leq n$, the $(i-1)$-partition is obtained by merging two clusters in the i-partition) such that the following objective is minimized: the maximum over all $1 \leq k \leq n$ of the ratio of $\sum_{i=1}^{k} \sum_{p \in \mathcal{C}_i} \|p - c_i\|_2$ to the optimum unconstrained k-median cost, where $C_k = \{c_1, \ldots, c_k\}$ is the set of centers and $\mathcal{P}_k = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ a k-partition. In this paper, we will provide a slightly different guarantee: instead of minimizing the maximum ratio, we show that for every fixed k this ratio is $O(\min\{d, \log n\} \log \Delta)$ in expectation. Our results extend to a slightly weaker bound that holds simultanuously for all k. We also remark that the above problem formulation implies a similar bound for the variant of the problem where we do not require an ordering of the centers but instead choose the optimal center for each cluster of the partition.

Hierarchical clustering, thanks to its ability in explaining the nested structures in real world data, has been extensively studied in the computer science literature [35, 34, 29, 53, 14, 20, 51, 55, 7]. The main focus of previous work has been on sequential solutions for the problem and so they are difficult to apply on large data sets. For this reason, several papers recently proposed scalable hierarchical clustering algorithms [35, 34, 29, 53, 7, 49, 43]. Nevertheless most prior work has focused only on scaling the single-linkage algorithm (in fact, efficient MapReduce and Spark algorithms are known for this problem [35, 34, 7, 58]) for which no objective function is clearly specified.

Another closely related area of research focuses on the k-median and k-means "flat" clustering problems in the distributed setting where several important results are known [5, 9, 24, 44, 3, 15, 30]. The main idea behind these algorithms is to compute in a distributed way some form of small summary of the data (such as coresets). However, all algorithms based on this framework use at least $k + n^{\epsilon}$ memory per machine (often significantly more) to solve the problem in $O(\frac{1}{\epsilon})$ rounds. Unfortunately, this is too much in our setting where we want to solve the problem also for $k \in \Theta(n)$ (in fact to solve the hierarchical clustering problem we need to compute a solution even for $k = n - 1$). To overcome this problem, in a recent related work [40] Lattanzi et al. provide a parallel algorithm for the hierarchical Euclidean k-median problem by dividing the data in two clusters iteratively. Unfortunately, this approach does not provide any theoretical guarantees for the problem (in the experimental section we actually show that it may return low quality solutions) and its round complexity is $O(\log n)$.

Our Results. In this work we design the first distributed algorithm for the hierarchical Euclidean k-median problem with provable guarantees. In particular, our algorithm returns an approximate solution to the problem using memory $s \in \Omega(\log^2(\Delta + n + d))$, $O\left(\log_s(nd \log(n + \Delta))\right)$ parallel rounds, and $O(nd/s)$ machines, where Δ is the ratio between the maximum and minimum distance of two points. Note that this implies that our algorithm runs in a constant number of rounds if $s \in n^{\delta}$ for any constant $\delta > 0$. Interestingly, our algorithm is easily implementable in standard parallel frameworks such as MapReduce [22], Hadoop [56], Spark [59] and Dryad [33] etc. and we analyze it in the standard Massive Parallel Computation (a. k. a. MPC) model [36, 28, 10]. Furthermore, we complement our theoretical analysis with an experimental study in distributed setting where we show that our algorithm is significantly faster than parallel hierarchical [40] and flat solutions [5, 15, 24] for the k-median problem.

Properties, extensions and limitations. *Work efficiency.* An interesting aspect of our algorithm is that the total running time of our algorithm across all machines is almost linear. In particular, the total running time of our algorithm is $\widetilde{O}(nd)$ so it has almost optimal work efficiency[3]. To the best of our knowledge, our algorithm is also the first approximation algorithm having almost linear running time for the high-dimensional Euclidean k-median problem (at the expense of having an expected $O(\min\{d, \log k\} \log \Delta)$-approximation factor).[4].

Round Lower Bound. We note here that Theorem 4 in [12] can be easily extended to the Euclidean k-median problem[5] and so to the hierarchical Euclidean k-median problem. Thus, no distributed

[3] In this section, for the sake of simplicity, we use $\widetilde{O}$ notation to hide both $\log n$ and $\log \Delta$ factors. The exact term are provided in the following sections.

[4] We remark that for general metric spaces there is a lower bound of $\Omega(nk)$ total work to achieve a constant approximation guarantee for our problem [46]. Furthermore, we note that finding a constant-factor approximation algorithm for the Euclidean k-median problem in almost linear time would imply an asymptotically better approximation algorithm for the closest pair problem which is a long standing open problem in algorithm design. For low-dimensional Euclidean space, near-linear time algorithms are known [17, 18, 39]

[5] The exact same construction and proof work also for the Euclidean k-median problem.

algorithm with memory s can output any approximate solution in less than $\log_s n$ rounds. So our memory-round complexity trade-off is asymptotically optimal.

Simplicity and Efficiency. Our algorithm is extremely simple to implement. In fact, our results show that it is possible to approximate the hierarchical Euclidean k-median problem just by using a sorting algorithm and few aggregation operations. For the same reason, our algorithm is also very parallelizable and efficient.

Practicality. Besides our theoretical guarantees our algorithm is also very efficient in practice. We run distributed experiments and we show that the algorithm is more accurate and efficient than previous parallel algorithms. Furthermore, for $k = 10$ our algorithm already outperforms the classic distributed algorithms for flat k-median[6] [5, 15, 24] by a factor of 3. For larger k the gap becomes even larger (for example for $k = 1000$ our algorithm is an order of magnitude faster)[7].

The large k setting. In a recent paper [12] the authors showed for the first time that it is possible to compute an approximate solution for the Euclidean k-means problem using parallel memory significantly smaller than k. In particular they provide a bi-criteria algorithm that approximates the solution of the k-means problem within a factor $O((\log n \log\log n)^2)$ using $O(k \log k \log n)$ centers, memory-per-machine $s \in \Omega(d \log n)$, and $O(\log_s n)$ parallel rounds. Our algorithm proves that this is also possible for the k-median problem. Furthermore our algorithm returns a solution that is not bi-criteria solving an open question in [12].

Limitations. Our result has two main limitations. First, it relies on the existence of a good tree embedding that can be computed in a few MPC rounds. As we show in this paper, it is indeed possible to compute a tree embedding of a d-dimensional Euclidean point set with expected distortion $O(d \log \Delta)$ in a constant number of rounds. While there exists a sequential $O(\log n)$-distortion tree embedding construction of point sets of size n of arbitrary metric spaces [26], it remains an interesting open problem as to how this could be implemented in a constant number of rounds. Second, the expected approximation factor of our algorithm is $O(\min\{d, \log n\} \log \Delta)$ and the expectation is for every fixed k. Nevertheless, in our experimental analysis, we show that in practice the quality of our solution is comparable to state-of-the-art algorithms while being much faster.

For additional related work we refer the reader to Appendix A.

2 Preliminaries

For two points p, q we use $\|p - q\|$ to denote their Euclidean distance. For a point p and set $C \subseteq \mathbb{R}^d$ we define $\textsc{Dist}(p, C) = \min_{c \in C} \|p - c\|$, i.e., the distance from p to its closest point in C. For two partitions P_1, P_2 we say that P_1 is nested in P_2 if P_2 can be obtained from P_1 by merging two or more parts of P_1. Given a point set $P \subseteq \mathbb{R}^d$, a set of centers $C = \{c_1, \ldots, c_k\}$, and a partition $\mathcal{P}$ of P into k parts $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$, its k-median cost is $\textsc{Cost}(P, C, \mathcal{P}) = \sum_{i=1}^{k} \sum_{p \in \mathcal{P}_i} \textsc{Dist}(p, c_i)$. The points in C are called the *cluster centers*. We let $\textsc{Opt}(P, k) = \min_{C \subseteq \mathbb{R}^d, |C|=k} \min_{\mathcal{P}, |\mathcal{P}|=k} \textsc{Cost}(P, C, \mathcal{P})$ denote the cost of an optimal k-median solution.

In this paper, we focus on a slight variation of the classic Hierarchical Euclidean k-median problem[8], which for a given point set $P \subseteq \mathbb{R}^d$ asks to find an (ordered) sequence of centers $C = \{c_1, \ldots, c_n\}$ together with a collection of n nested partitions $\Pi = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ of P, such that $\mathcal{P}_i$ partition the entire space and contains i parts. We say that an algorithm for the Hierarchical Euclidean k-median problem has an approximation guarantee of α if, for every $k = 1, \ldots, n$, the output C, Π defines an α-approximate solution for k-median. That is, for every $k \in \{1, 2, \ldots, n\}$ $\textsc{Cost}(P, \{c_1, \ldots, c_k\}, \mathcal{P}_k) \leq \alpha \cdot \textsc{Opt}(P, k)$. Similarly, if the algorithm is randomized then we say that it has an approximation guarantee of α if the guarantee holds in expectation over the random hierarchical clustering C, Π output by the algorithm: for every $k \in \{1, 2, \ldots, n\}$, $\mathbf{E}[\textsc{Cost}(P, \{c_1, \ldots, c_k\}, \mathcal{P}_k)] \leq \alpha \cdot \textsc{Opt}(P, k)$.

[6] Note that flat clustering is a sub-problem of hierarchical Euclidean k-median problem, in fact our algorithm solve the problem for all k at the same time

[7] Note that the large k has many practical applications in spam and abuse [48, 52], near-duplicate detection [31], compression or reconciliation tasks [50]

[8] We note here that our results naturally extend to the Prefix Euclidean k-median problem [45], where clusters are not required to be nested.

For simplicity of presentation, here we assume that $P \subseteq \{0, \ldots, \Delta\}^d$ and then we explain how to remove this assumption in our theoretical analysis in Appendix. The k-median problem can also be formulated for discrete metric spaces $M = (P, \textsc{Dist}_M)$, where P is a finite set of points and $\textsc{Dist}_M$ is a metric. For a subset $C \subseteq P$ of the points, we also let $\textsc{Dist}_M(p, C) = \min_{c \in C} \textsc{Dist}_M(p, c)$ be the distance from the point p to its closest point in C. The objective of the metric k-median problem is then to find a subset $C \subseteq P$ of size k such that $\textsc{Cost}_M(P, C) = \sum_{p \in P} \textsc{Dist}_M(p, C)$ is minimized. The hierarchical k-median problem is analogously defined.

A (discrete) metric space $(P, \textsc{Dist}_T)$ is called a *tree metric*, if there exists a positively weighted tree $T = (P, E, w)$ such that for all pairs $p, q \in P$ we have that $\textsc{Dist}_T(p, q)$ equals the shortest path distance between p and q in T. We will be mostly interested in tree metrics that are defined by *restricted hierarchically well separated trees*. [9]

Definition 2.1 *A* restricted ℓ-hierarchically well separated tree (RHST) *is a positively weighted rooted tree such that all the leafs are at the same level, all edges at the same level have the same weight, and the length of the edges decreases by a factor of ℓ on every root to leaf path.*

Throughout the paper, we consider RHSTs with $\ell = 2$, i.e., 2-RHSTs. We therefore simplify notation and sometimes write RHST for 2-RHST. In the next section, we describe how to embed our point set P in Euclidean space into the leaf of a RHST. Formally, a metric embedding between two metric spaces $(P, \textsc{Dist})$, $(P', \textsc{Dist}')$ is an injective mapping $f : P \to P'$. To simplify notation and to be consistent with this definition (and the definition of a metric) we consider instances that contain no two points within distance 0. However, all our arguments and algorithms generalize to instances that may have several identical points.

MPC model. We design algorithms for the MPC model [36, 28, 10] that is considered de-facto the standard theoretical model for large-scale parallel computing. Computation in MPC proceeds in synchronous parallel *rounds* over multiple machines. Each machine has memory s. At the beginning of a computation, data is partitioned across the machines. During each round, machines process data locally. At the end of a round, machines exchange messages with a restriction that each machine is allowed to send messages and also receive messages of total size s. The efficiency of an algorithm in this model is measured by the number of rounds it takes for the algorithm to terminate and by the size of the memory of every machine. In this paper we focus on the most challenging and practical regime of small memory. In particular, we only assume to have $s \in \Omega(\log^2(n + \Delta + d))$.

3 A Work Efficient Sequential Algorithm

In this section we introduce a work efficient sequential algorithm and then in the next section, we show how to parallelize it in the MPC model. From a high level perspective our algorithm is based on embedding the input point set into the leaf of a RHST and then solving the problem optimally for the embedded points. In order to obtain the embedding, we choose a (standard) random quad-tree embedding of the input points into a hierarchically well separated tree. We then show that a simple greedy algorithm solves the problem optimally on the induced tree metric. Unfortunately, a naive implementation of our greedy algorithm would result in large parallel and sequential running time. So we modify our greedy algorithm and we show that it can be implemented using only few aggregation operations combined with sorting.

3.1 Quadtree Embedding into a 2-RHST

The first step of our algorithm is to embed the points in a restricted 2-hierarchically separated tree. Interestingly, we observe that using standard embedding techniques it is possible to embed all the points in the datasets in a 2-RHST by incurring only a small distortion. Furthermore, in this construction every point can compute its own position in the embedding independently just by knowing its own coordinated and a random shift r that is applied to all the points.

Given that the construction of the embedding is standard we defer it to Appendix F. Now we present formally the main property of the embedding that will be useful for our algorithm. Before

[9] Our definition is slightly more restricted than the standard notion of ℓ-HST (see, for example, [6]), because we require that all edges at the same level have the same cost (rather than all descendants of a node) and that the cost decreases by the same factor.

stating the property we need to introduce some additional notation. For two points p, q in the RHST T, we use $\textsc{Dist}_T(p,q)$ to denote their (shortest-path) distance. Similarly we let $\textsc{Dist}_T(p,C) = \min_{c\in C} \textsc{Dist}_T(p,c)$. We now define $\textsc{Cost}_T(P,C) := \sum_{p\in P} \textsc{Dist}_T(p,C)$ to be the cost of a set of centers C with respect to the tree metric. We use $\textsc{Opt}_T(P,k) := \min_{C\subseteq\mathbb{R}^d,|C|=k} \textsc{Cost}_T(P,C)$ to denote the cost of optimal solution with respect to a given RHST T. Recall that $\textsc{Opt}(P,k)$ denotes the cost of an optimal solution with respect to the original Euclidean distances. Now we are ready to state the main result of this section whose proof is presented in Appendix F.

Theorem 3.1 *Let $P \subseteq \{0,\ldots,\Delta\}^d$ be a point set. There exists a procedure that constructs a 2-RHST tree T in time $O(nd\log\Delta)$ such that for the its optimum solution $C^*_{T,k}$ using k centers, we have $\mathbf{E}[\textsc{Cost}(P,C^*_{T,k})] = O(d\cdot\log\Delta)\cdot\textsc{Opt}(P,k)$. Furthermore all the input points are mapped to leaves of the RHST. In addition, we have that $\mathbf{E}[\max_k \frac{\textsc{Cost}(P,C^*_{T,k})}{\textsc{Opt}(P,k)}] = O(d\cdot\log\Delta\log(dn\Delta))$.*

3.2 Optimal Algorithms for k-Median on 2-RHST

In this subsection, we design two optimal sequential algorithms for hierarchical k-median on a 2-RHST metric. This combined with our 2-RHST embedding will give us an efficient approximation algorithm. We first show that a simple greedy algorithm (see Algorithm 1) finds the optimum solution. The greedy algorithm chooses, at each step, the point that leads to the largest decrease in the cost. Then we modify the algorithm to be more amenable to parallel implementation and to run in almost linear time.

3.2.1 A Greedy Algorithm for 2-RHST

We start by providing the pseudo-code for our algorithm in Algorithm 1 and state the main property of our greedy algorithm (whose proof is deferred to Appendix G).

Theorem 3.2 *For any set of points $P \subseteq \{0,\ldots,\Delta\}^d$ and distance function $\textsc{Dist}_T$ defined by an 2-RHST T, Algorithm 1 returns an optimum solution for the hierarchical k-median problem.*

Proof.(Sketch) First observe that by the cluster assignment in line (8) the returned partitions $\{\mathcal{P}_1,\ldots,\mathcal{P}_n\}$ are nested. It remains to show that for every k, the partition $\mathcal{P}_k$ with centers $c_1,\ldots,c_k$ is optimal. The proof is by induction on the height of the tree T. For the base case, when T is of height 0, the statement is clear: In that case the instance consists of a single point so $\textsc{Cost}_T(P,S) = 0$ if $S \neq \emptyset$.

For the inductive step we make two fundamental observations. First, we observe that in a 2-RHST metric the cost of a solution can be decomposed as a sum of the cost of individual subtrees. Second, we also show that the cost of each subtree is roughly independent of the choice made by the algorithm in other subtrees. So combining these two facts, with the fact that greedy is optimal for all the subtrees of
we can conclude our inductive arguments. □

Algorithm 1 GREEDY alg. for hier. k-median on 2-RHST

Input: Set of points P, cost function $\textsc{Cost}_T$ defined
by a 2-RHST T
1: Set $S_0 \leftarrow \emptyset$
2: Set $P_0 \leftarrow \{P\}$
3: Label all internal nodes of the RHST as unlabelled
4: **for** $i = 1$ to n **do**
5: Let $c_i = \operatorname{argmin}_{x\in P} \textsc{Cost}_T(P, x\cup S_{i-1})$
6: Label the highest unlabelled ancestor of c_i with c_i
7: S_i is obtained by adding c_i to S_{i-1}
8: Define $\mathcal{P}_i$ as the clustering obtained by assigning all points to the cluster centered at their closest labelled ancestor.
9: **end for**
Output: return $c_1,\ldots,c_n,\mathcal{P}_1,\ldots,\mathcal{P}_n$

3.2.2 Quasi Linear Time Algorithm

In this subsection we show how to change the previously presented algorithm (Algorithm 1) to obtain an almost linear-time algorithm that can be easily parallelized. The algorithm will output a list of centers and an implicit description of a list of nested partitions. The list of centers as well as the partitions will be identical to that of the previous algorithm. Hence, we will focus on the optimality of the centers wrt. the hierarchical k-median problem.

As a first step we modify Algorithm 1 to reduce its sequential dependencies. The downside of Algorithm 1 is that the selection to open the i-th point depends on the previously chosen points, i.e., S_{i-1}. The main idea is to remove this dependency and to make the decision dependent on other parameters that can be efficiently precomputed. Before describing our result we introduce some additional notation. We say that a node in the (2-RHST) tree is at level i if the subtree rooted at this node is of height i. We remark that with this convention a leaf is at level 0 and the root is at the maximum level h. Let S be a set of centers and x be a point in $P \setminus S$. We refer to the ancestor of x at level i by $a_i(x)$ and, with some abuse of notation, we denote with $p_i(x)$ the number of points in the subtree of $a_i(x)$ for $0 \leq i \leq h$. We also write $A_i(x) \subseteq P$ for the set of points in the subtree of node $a_i(x)$. Finally, we let $\ell(x) \in \{1, \ldots, h\}$ be the highest level ℓ for which $S \cap A_\ell(x) = \emptyset$. For clarity, when it is clear from the context, we drop the x and use a_i, A_i, p_i, ℓ.

Our first key observation is that the distance of a point x to its nearest center only depends on ℓ. This is true because i) the distance of x to the point $y \in S \cap A_{\ell+1}$ is exactly twice the distance of x to $a_{\ell+1}$, which is $2\sum_{i=0}^{\ell-1} 2^i = 2(2^\ell - 1) = 2^{\ell+1} - 2$. ii) The distance to any other point $w \in S$ is at least $2^{\ell+2} - 2$ since their common ancestor is at least at level $\ell + 1$. Therefore,

Observation 3.3 *The distance* $\textsc{Dist}(x, S)$ *of a point* x *to its nearest center in* S *only depends on* $\ell(x)$. *More precisely,* $\textsc{Dist}_T(x, S) = 2^{\ell(x)+1} - 2$.

In addition, we observe that the benefit of opening x (denoted by $\textsc{Benf}(x)$) only depends on the number of nodes in close subtrees, $p_1, \ldots, p_\ell$. In fact opening x only affects the cost of the points in A_ℓ. This is true because any point w that is in $A_{\ell+1} \cap S$ is at least as close to x as to any other point in $A_{\ell+1}$ and not in A_ℓ. Now, consider a point y in A_i for $0 \leq i \leq \ell$. The cost of y after opening x is $2^{i+1} - 2$, and before opening x its cost was $2^{\ell+1} - 2$. Furthermore, we can precisely compute the number of points in A_i and not in A_{i-1}, it equals $p_i - p_{i-1}$. So we get:

Lemma 3.4 *The benefit of opening point* x *only depends on* $p_1(x), \ldots, p_{\ell(x)}(x)$ *and is* $\textsc{Benf}(x) = \sum_{i=0}^{\ell(x)} (p_i(x) - p_{i-1}(x)) \cdot (2^{\ell(x)+1} - 2^{i+1})$, *where for simplicity we assume* $p_{-1}(x) = 0$.

These simple observations significantly reduces the dependency of computing the benefit of opening a point x from the set of centers S, which is the main step of Algorithm 1. In fact, the $\textsc{Benf}(x)$ value does not depend on the full structure of S but only on $\ell(x)$. Therefore, we can compute $\textsc{Benf}(x)$ for all ℓ before opening the centers and use only those values to make our selection of the centers. We denote this by $\textsc{Benf}(x, \ell)$, i.e., $\textsc{Benf}(x, \ell) = \sum_{i=0}^{\ell} (p_i(x) - p_{i-1}(x)) \cdot (2^{\ell+1} - 2^{i+1})$, for all points x and $0 \leq \ell \leq h$.

Now, to select the centers we can design an alternative algorithm based only on the $\textsc{Benf}(x, \ell)$ values. In fact we can sort all the $\textsc{Benf}(x, \ell)$ in an ordered list L and go over them from the highest value to the lowest. We show that it is sufficient to prune the list L by removing a pair (x, ℓ) in the following two cases:

– There is another point with larger benefit in A_ℓ. In this case we need to update our sorted list by removing (x, ℓ) from the ordering L. In general to deal with this case for each subtree rooted at level ℓ for all possible ℓ we preserve only one pair (x, ℓ) with highest benefit.

– For point x, collect all pairs (x, ℓ) that remained after the previous pruning step. Identify the pair with the maximum value of ℓ and prune all other collected pairs.

After these two steps, the ordering L can be used to obtain our final solution for Hierarchical k-median (resp. k-median) by returning the centers in the same order as in L (resp. the top k centers of L). The pseudo-code is presented in Algorithm 2.

Importantly, we show that the output sequence of centers and the implicitly defined partitions of Algorithm 2 are the same as the output of Algorithm 1. This is intuitively true because the pruning rules identify at which level x is a good candidate for the greedy algorithm and then we sort the x based on their score at that level. Note that in this algorithm we only output the partition implicitly because outputting them explicitly will take $\Omega(n^2)$ time. The full proof of this argument is presented in Appendix H.

Theorem 3.5 *Algorithm 2 finds an optimum solution for the hierarchical* k*-median problem on RHSTs in time* $O(n \log^2(\Delta + n))$.

4 Euclidean k-Median in the MPC Model

In this section we present our main results in the MPC model. The algorithm that we use is the same as Algorithm 2, but we implement each step in the MPC model. Our algorithm is based on basic operations: summation, computing the maximum and sorting a list of elements. All these operations can be done efficiently in the MPC model using [28]. We provide the details on the implementation of our algorithm along with its theoretical performance guarantees.

Theorem 4.1 *There is a distributed algorithm that uses memory $s \in \Omega(d\log(\Delta) + \log^2(\Delta + n))$ and computes a solution for the hierarchical Euclidean k-median problem such that for every fixed k the k-clustering of the hierarchy has an expected approximation factor of $O(d\log\Delta)$. Furthermore, the expected maximum approximation factor (over all k) is $O(d\log\Delta\log(\Delta dn))$. The algorithm uses $O(\log_s(n\log\Delta))$ parallel rounds and $O((n\log\Delta)/s)$ machines in the MPC model. Furthermore the total running time of the algorithm across all the machine is $O(nd\log(\Delta+n))$.*

Proof. Our distributed algorithm consists of a parallel implementation of Algorithm 2 that we run on the embedding described in Section 3.1. This gives us an expected approximation factor of $O(d\log\Delta)$, because from Theorem 3.5 we know that Algorithm 2 solves the problem optimally on 2-RHST and from Theorem 3.1 we know that we loose a factor $O(d\log\Delta)$ in expectation in our embedding step.

Algorithm 2 Quasi-linear alg. for hier. k-median on RHST

Input: Set of points P, cost function COST_T defined by a 2-RHST T

$\forall x \in P, 0 \le \ell \le h$, compute $\text{BENF}(x,\ell)$
for $0 \le \ell \le h$ **do**
  for subtree $T' \subseteq T$ rooted at level ℓ **do**
    Let $x \in T'$ be the point maximizing $\text{BENF}(x,\ell)$
    for $y \in T'$ **do**
      $C_{(x,\ell)} \leftarrow C_{(x,\ell)} \cup \{y\}$
      If $y \neq x$ delete $\text{BENF}(y,\ell)$
    end for
  end for
end for
For every x keep only the pair (x,ℓ) with maximum ℓ and its cluster $C_{(x,\ell)}$.
Sort all pairs (x,ℓ) according to $\text{BENF}(x,\ell)$ and let (c_i,ℓ_i) be the pair in position i.
Implicitly define the i-th partition $\mathcal{P}_i = \{P_1,\dots,P_i\}$ as follows: $P_j = C_{(c_j,\ell_j)} \setminus \cup_{z=j+1}^{i} C_{(c_z,\ell_z)}$ for $1 \le j \le i$.

Output: Return the ordering of the points $c_1,\dots,c_n$ as centers and the clusters $C_{(c_i,\ell_i)}$ $\forall i$ to implicitly define the partitions.

The details for the distributed implementation of this algorithm in the MPC model is as follows. In the implementation we need to be able to compute the embedding, compute $p_\ell(x)$ for all points x, compute BENF, discard the useless BENF pairs and sort the values. First, note that we can compute the embedding in parallel for each of the points because the embedding only depends on their coordinates and a uniform random shift. Then to compute $p_\ell(x)$ and BENF for all nodes we just need to be able to compute weighted sums efficiently. This can be done using memory $s \in \Omega(d\log\Delta + \log^2(\Delta+n))$ in $O(\log_s(n\log\Delta))$ MPC rounds with $O((n\log\Delta)/s)$ machine and total running time of $O(nd\log\Delta)$ using the algorithm in [28]. Then we need to sort the computed BENF values which can be done similar to be previous step using [28] with the same bounds as before. To apply the filter we just need to be able to find the maximum in a list of values and we can again do this using memory $s \in \Omega(\log^2(\Delta+n))$ in $O(\log_s(n\log\Delta))$ MPC rounds using the algorithm in [28]. Note that after computing the maximum we can also construct the sets $C_{(x,\ell)}$ by outputting as $C_{(x,\ell)}$ all nodes sharing the same ancestor of x at level ℓ, and this can be done in $O(1)$ rounds as well. Finally we can also do sorting with the same bound using the same techniques with total running time of $O(n\log n)$. □

We remark that we can speed up the algorithm for datasets of large dimension, by reducing the dimension from d to $O(\log n)$ by losing a small constant factor in the approximation ratio[21] and achieve the following result, the details are presented in the full version. We also remark that the guarantee on the expected maximum approximation ratio (over all k) from Theorem 3.1 carries over to this theorem and the following corollary.

Corollary 4.2 *There is a distributed algorithm that uses memory $s \in \Omega(\log^2(\Delta + n + d))$ and computes a solution for the hierarchical Euclidean k-median problem such that*

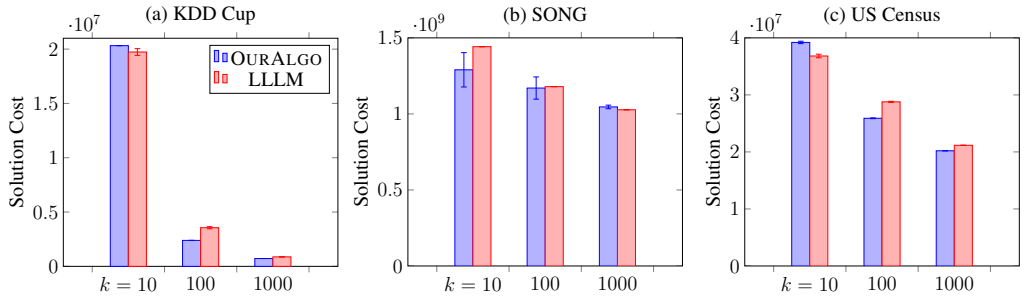

Figure 1: In this plot, we compare our algorithm (OURALGO), and LLLM. The results are presented in for KDD Cup (a), SONG (b), US Census (c). All the numbers are reported for 5 runs (average value and variance).

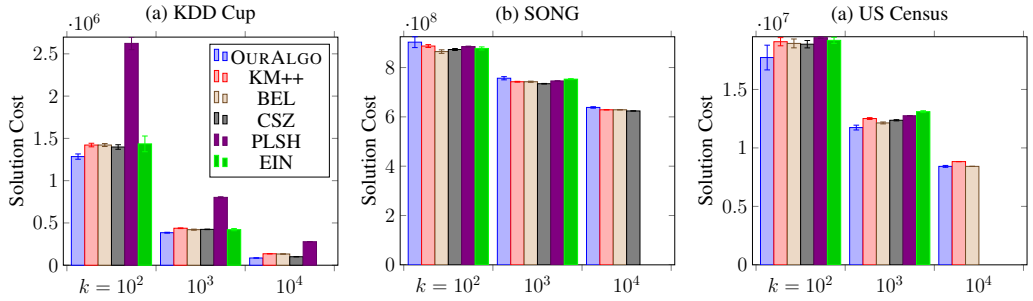

Figure 2: In this plot, we compare our algorithm (OURALGO), k-median++ seeding (KM++ Seed), BEL, PLSH, and EIM. The results are presented in for KDD Cup (a), SONG (b), Us Census (c). All the numbers are reported for 5 runs (average value and variance).

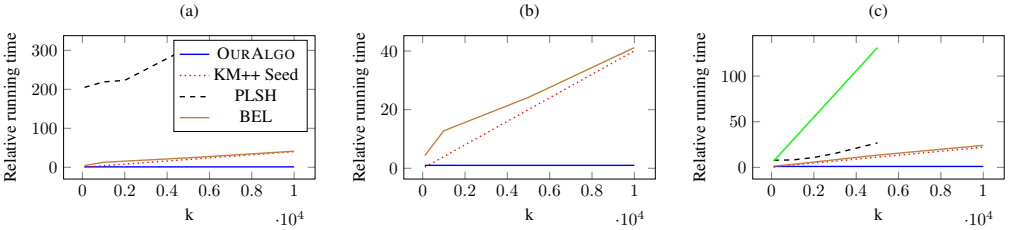

Figure 3: Figure (a), (c) compares the running time for our algorithm (OURALGO), k-median++ seeding (KM++ Seed), BEL, PLSH, and CSZ on HIGGS and US Census datasets, respectively. In (b) we focus on our algorithm and k-median++ seeding on HIGGS dataset to further highlight their differences.

for every fixed k the k-clustering of the hierarchy has an expected approximation factor of $O(\min(d, \log n) \log \Delta \log(nd\Delta))$. The algorithm uses $O(\log_s(nd \log(n + \Delta)))$ parallel rounds and $O(nd \log(n + \Delta)/s)$ machines in the MPC model. Furthermore the total running time of the algorithm across all the machine is $O(nd(\log(n + \Delta)))$. Finally the algorithm returns a solution also for the prefix k-median clustering and the hierarchical k-median clustering problems.

5 Empirical Evaluation

In this section we empirically evaluate our algorithm with both hierarchical and non-hierarchical algorithms. We report the expected value and the variance over 5 runs for all the randomized algorithms. We evaluate the algorithms on the following data sets from UCI Machine Learning Repository [23]; KDD Cup (a.k.a Intrusion, $n = 31,029; d = 35$),YearPredictionMSD (a.k.a SONG, $n = 515,345; d = 90$),US Census Data (1990) ($n = 2,458,285; d = 29$),and HIGGS ($n = 11,000,000; d = 28$).

5.1 Distributed Hierarchical Algorithms

We compare our algorithm with LLLM [41] - an algorithm for hierarchical k-median problem in distributed setting. In Fig. 1 we compare the quality of the produced solution. We observe that the algorithms are comparable and our algorithm out performs LLLM by $2-3\%$ on average. The running time of the the LLLM algorithm divided by the running time of our algorithm is presented in Table 1. Our algorithm is significantly faster (up to a factor $10-26$ depending on the dataset) and the difference increases with number of the machines used.

Dataset	m=1x	10x	100x
KDD Cup	9.58	13.5	26.9
SONG	2.32	5.26	19.3
US Census	1.72	3.10	6.71

Table 1: Distributed hierarchical algorithms running time comparison for various number of machines. The running time of the LLLM baseline divided by the running time of our algorithm with the same amount of machine used.

5.2 Distributed Algorithms

Our algorithm can be also used for the (non-hierarchical) k-median problem. In order to assess its quality, we compare it with sequential and distributed algorithms for the (non-hierarchical) k-median problem. i) We compare the quality of the computed set of centers and the sequential running time with both sequential and distributed algorithms for the k-median problem, ii) we compare the running time with distributed algorithms in a distributed setting, iii) we run our algorithm on massive datasets of with tens of billions of nodes and report the speedup gained by using more machines. Notice that the baselines do not scale to this size of data, therefore we cannot compare the performance.

Baselines. We compare our algorithm (without dimensionality reduction) with a well-known sequential algorithm: k-median++ seeding; and four distributed algorithms: PLSH [12], EIM [25], BEL [4], CSZ [15]. Description of these algorithms and the details of the setting used for running the algorithms are provided in Appendix B.

Quality and Sequential Running Time Comparison. Our algorithm is $40, 41, 300$ times faster than k-median++, BEL, and PLSH, respectively for census dataset for $k = 10,000$. Also it is significantly faster than CSZ [10]. In figure Fig. 3 we compare the running time of these four algorithms for different k[11] (recall that we compute the solution for all $1 \leq k \leq n$ at the same time). The plots are similar for other datasets and are presented in Appendix C. Let us focus on the quality of the solution. The results are presented in Fig. 2. Note that due to the slow running time and memory consumption of the PLSH, CSZ, and EIM we were not able to run it in some cases. The results suggest that our algorithm provides comparable results.

Distributed Running Time Comparison. Now we focus on the running time of OURALGO, BEL, EIM, and CSZ algorithms in distributed setting for HIGGS dataset. We compare the running time of the baselines with our algorithm with the same amount of machine used for k=10 to $10,000$. We present the running time of the baselines divided by the running time of our algorithm. The results are presented in Table 2 in Appendix. Our algorithm is significantly faster than all baselines. For small values of k, e.g: $k = 10$, we are faster by a factor $3.2-60.2$, $50-182$, and $65-1380$ compared to BEL, EIM, and CSZ, respectively. Moreover, for $k = 1,000$ we are $12.08-71.2$, $2082-3091$, $46-941$ times faster compared to BEL, EIM, and CSZ, respectively. Also for $k = 10,000$ we are $41.17-126$ and $171-708$ times faster than BEL and CSZ, respectively.

Scalability on Large Datasets. To provide an empirical scalability analysis of our algorithms over large datasets, we expanded the HIGGS dataset so as to experiment on datasets of size almost 100 millions, 1 billion, and 10 billions (see Appendix E for a more detailed description of the generation process). As depicted in Fig. 5 in Appendix K, the speed-ups achieved by our algorithm is significant when increasing the number of machines.

[10] The missing values are due to the slow running time of CSZ.

[11] EIM is slower that rest of the algorithms and its running time is presented in the Appendix.

6 Conclusions

We present the first distributed approximation algorithm for Euclidean Hierarchical k-Median problem in the MPC model. Interestingly our algorithm even works in the setting where each machine has very limited memory $s \in \Omega(\log^2(\Delta + n + d))$ and it is work efficient. In the future, it would be interesting to obtain similar results for other clustering problems and to improve the approximation factor of our algorithm. We believe that the core ideas used in this work can extend to the general k-clustering problem (e.g., k-means and higher powers). To be more precise, if we are given a RHST that increases the distances by at most a factor of $g(n)$, then we believe that our approach construct a $g(n)$ approximate solution, for any function g. The difficulty here is to construct such RHST.

Acknowledgments and Disclosure of Funding

This research was partially supported by the Swiss National Science Foundation projects 200021-184656 "Randomness in Problem Instances and Randomized Algorithms".

References

[1] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the 18th Symposium on Discrete Algorithms (SODA)*, pages 1027–1035, 2007.

[2] Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 405–413, 2019.

[3] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, 2012.

[4] Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 1995–2003, 2013.

[5] Maria-Florina F Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general topologies. In *Advances in Neural Information Processing Systems*, pages 1995–2003, 2013.

[6] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[7] M. Bateni, S. Behnezhad, M. Derakhshan, M. Hajiaghayi, S. Lattanzi, and V. Mirrokni. On distributed hierarchical clustering. In *NIPS 2017*, 2017.

[8] MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. Affinity clustering: Hierarchical clustering at scale. In *Advances in Neural Information Processing Systems*, pages 6864–6874, 2017.

[9] MohammadHossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab Mirrokni. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems*, pages 2591–2599, 2014.

[10] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 273–284. ACM, 2013.

[11] Robert Benkoczi, Binay K. Bhattacharya, Marek Chrobak, Lawrence L. Larmore, and Wojciech Rytter. Faster algorithms for *k*-medians in trees. In *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS*, pages 218–227, 2003.

[12] Aditya Bhaskara and Maheshakya Wijewardena. Distributed clustering via lsh based data partitioning. In *International Conference on Machine Learning*, pages 570–579, 2018.

[13] Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 576–585, 2017.

[14] Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 841–854, 2017.

[15] Jiecao Chen, Erfan Sadeqi Azer, and Qin Zhang. A practical algorithm for distributed clustering and outlier detection. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 2253–2262, 2018.

[16] Jiecao Chen, Erfan Sadeqi Azer, and Qin Zhang. A practical algorithm for distributed clustering and outlier detection. In *Advances in neural information processing systems*, pages 2248–2256, 2018.

[17] Vincent Cohen-Addad. A fast approximation scheme for low-dimensional k-means. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 430–440. SIAM, 2018.

[18] Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximations schemes for clustering in doubling metrics. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 540–559. IEEE Computer Society, 2019.

[19] Vincent Cohen-Addad, Varun Kanade, and Frederik Mallmann-Trenn. Clustering redemption-beyond the impossibility of kleinberg's axioms. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8526–8535, 2018.

[20] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 118–127, 2016.

[21] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003.

[22] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[23] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[24] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689, 2011.

[25] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 681–689. ACM, 2011.

[26] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[27] Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–217, 2005.

[28] Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011.

[29] J. C. Gower and G. J. S. Ross. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21(8):1313 – 1325, 1995.

[30] Sudipto Guha, Yi Li, and Qin Zhang. Distributed partial clustering. *ACM Transactions on Parallel Computing*, 6(3):11:1–11:20, 2019.

[31] Oktie Hassanzadeh, Fei Chiang, Renée J. Miller, and Hyun Chul Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.

[32] Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 373–380, 2004.

[33] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pages 59–72. ACM, 2007.

[34] Chen Jin, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok N. Choudhary. Incremental, distributed single-linkage hierarchical clustering algorithm using mapreduce. In *Proceedings of the Symposium on High Performance Computing, HPC 2015, part of the 2015 Spring Simulation Multiconference, SpringSim '15, Alexandria, VA, USA, April 12-15, 2015*, pages 83–92, 2015.

[35] Chen Jin, Ruoqian Liu, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok N. Choudhary. A scalable hierarchical clustering algorithm using spark. In *First IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2015, Redwood City, CA, USA, March 30 - April 2, 2015*, pages 418–426, 2015.

[36] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. SIAM, 2010.

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

[38] Jon M Kleinberg. An impossibility theorem for clustering. In *Advances in neural information processing systems*, pages 463–470, 2003.

[39] Stavros G Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. In *European Symposium on Algorithms*, pages 378–389. Springer, 1999.

[40] Silvio Lattanzi, Thomas Lavastida, Kefu Lu, and Benjamin Moseley. A framework for parallelizing hierarchical clustering methods. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 73–89. Springer, 2019.

[41] Silvio Lattanzi, Thomas Lavastida, Kefu Lu, and Benjamin Moseley. A framework for parallelizing hierarchical clustering methods. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part I*, volume 11906 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2019.

[42] Guolong Lin, Chandrashekhar Nagarajan, Rajmohan Rajaraman, and David P Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal on Computing*, 39(8):3633–3669, 2010.

[43] Q. Mao, W. Zheng, L. Wang, Y. Cai, V. Mai, and Y. Sun. Parallel hierarchical clustering in linearithmic time for large-scale sequence analysis. In *2015 IEEE International Conference on Data Mining*, pages 310–319, Nov 2015.

[44] Alessio Mazzetto, Andrea Pietracaprina, and Geppino Pucci. Accurate mapreduce algorithms for k-median and k-means in general metric spaces. In *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, pages 34:1–34:16, 2019.

[45] Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. *SIAM J. Comput.*, 32(3):816–832, 2003.

[46] Ramgopal R. Mettu and C. Greg Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1-3):35–60, 2004.

[47] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[48] Feng Qian, Abhinav Pathak, Yu Charlie Hu, Zhuoqing Morley Mao, and Yinglian Xie. A case for unsupervised-learning-based spam filtering. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 367–368, 2010.

[49] Sanguthevar Rajasekaran. Efficient parallel hierarchical clustering algorithms. *IEEE Trans. Parallel Distrib. Syst.*, 16(6):497–502, June 2005.

[50] M. Ali Rostami, Alieh Saeedi, Eric Peukert, and Erhard Rahm. Interactive visualization of large similarity graphs and entity resolution clusters. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018.*, pages 690–693, 2018.

[51] Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2316–2324, 2016.

[52] Mina Sheikhalishahi, Andrea Saracino, Mohamed Mejri, Nadia Tawbi, and Fabio Martinelli. Fast and effective clustering of spam emails based on structural similarity. In *International Symposium on Foundations and Practice of Security*, pages 195–211. Springer, 2015.

[53] Spark. https://spark.apache.org/docs/2.1.1/mllib-clustering.html, 2014.

[54] Arie Tamir. An $o(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Oper. Res. Lett.*, 19(2):59–64, 1996.

[55] Joshua Wang and Benjamin Moseley. Approximation bounds for hierarchical clustering: Average-linkage, bisecting k-means, and local search. In *NIPS*, 2017.

[56] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[57] Grigory Yaroslavtsev and Adithya Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under ℓ_p-distances. In *35th International Conference on Machine Learning (ICML'18)*, 2018.

[58] Grigory Yaroslavtsev and Adithya Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under lp distances. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 5596–5605, 2018.

[59] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

A Additional Related Works

Additional Related Work. The k-median problem has also been extensively studied for tree metrics (see e.g. [54, 11] and references therein) where exact polynomial-time algorithms exist. We remark that we focus on the special case of hierarchically separating trees, which we can exploit to get faster algorithms. Indeed previous work on tree metrics only give nearly linear running time if k is a constant.

Somewhat related to our work is the quadtree-based coreset construction in [27]. The resulting coreset can be computed in $O(n \log \Delta)$ time and may be interpreted as a bi-criteria approximation, i.e., it uses more than k centers to achieve an approximation. We also believe that the construction in [13], which is based on randomly shifted quadtrees, can be modified to efficiently get a bi-criteria approximation. A closely related problem is the k-medoid problem, that is as k-median with the constraint that the centers are restricted to be from the input points. For this problem, a number of algorithms have been developed (see, for example, [37, 47]), but unfortunately they are sequential and their running times do not scale in our setting.

We also remark that the question of obtaining an efficient distributed algorithm for hierarchical clustering has been studied in a series of recent paper [8, 40, 57]. Although no result was known for hierarchical Euclidean median before this work. Finally, we note that the HSTs have been exploited to design efficient algorithm for fair clustering [2], it is an interesting open question to extend our results in their settings.

Interesting results are also known for the robust variant of k-median with outliers [16] and for the k-means problem [3].

B Description of Baseline Algorithms and Experiments Setup

In this section for the sake of completeness we present a general overview of the baseline algorithms. Notice that this provides the reader with some of the ideas used in these algorithms. The implementation used is exactly the algorithm presented in the respective papers. Notice that we used the k-median++ seeding as the offline algorithm used for both EIM and BEL algorithms since its performs well in practice and its running time is of $O(nkd)$. Moreover, it suffices for the machines to have memory $1G$ for all the distributed algorithm for HIGGS dataset.

- Our implementation of k-median++ seeding - an adaptation of k-means++ seeding [1] for the k-median problem - works as follow. The algorithm first samples a point uniformly at random from the input points (i.e. P). Then in the next $k-1$ iterations, it picks the next center from P with probability proportional to the distance to the nearest of the current centers. It is known that this seeding produces an $O(\log k)$-approximate solution. In the experiments we use our own implementation of k-median++.
- PLSH [12] algorithm (code by authors of PLSH). This algorithm first finds a bicriteria solution based on LSH and then samples a set of k elements from it. In the implementation to obtain k centers from the bi-criteria approximation, the centers are chosen uniformly at random from it. We thank the authors for helping us with experiments using their code.
- Our implementation of EIM [25] algorithm. This algorithm has two phases. The first phase starts with two sets $S = \emptyset, H = \emptyset, R$, where R at the beginning is equal to the set of input points. In each iteration of the first phase two set of input elements are sampled and added to H, S. Then for each point in S the closets point in H is computed and based on that a threshold value is computed. Then for each point currently in R the closest point to any point in H is computed and the point is removed it the computed minimum distance is lower than the threshold. The iterations are repeated till size of R is below a certain value depending on k, n. Then a weighted coreset over $H \cup R$ is computed and the final solution is computed based on it using any offline algorithm.
- Our implementation of BEL [4] algorithm. This algorithm divides the input into m part, where m is the number of machines. Then runs any offline algorithm and creates an weighted coreset based on the offline algorithms solution.

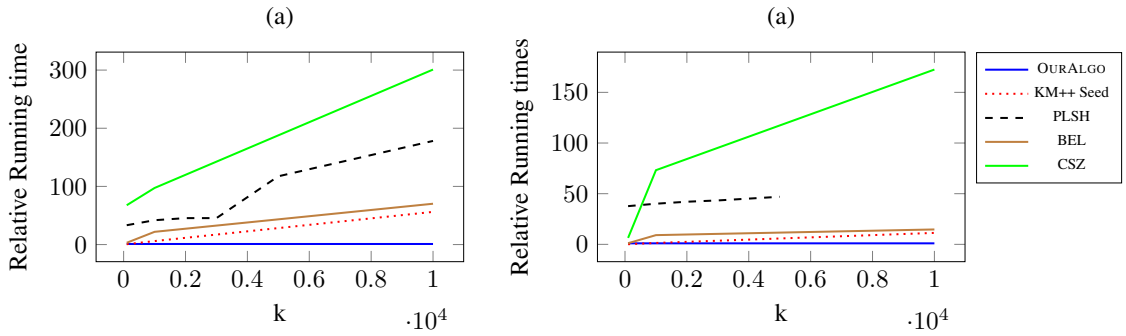

Figure 4: comparison between the running time of our algorithm with k-median++ seeding, BEL, CSZ, and PLSH for SONG (a) and KDDCUP (b) datasets.

comparison with BEL

k \ m	1x	10x	100x
10	3.2	12.45	60.2
25	3.75	12.61	61.3
50	4.01	14.7	62.0
100	4.36	15.3	62.4
1000	12.08	25.1	71.2
10000	41.17	87.6	126

comparison with EIM

k \ m	1x	10x	100x
10	50.4	60.5	182
25	90.6	72.2	192
50	144.4	110	262
100	258	330	452
1000	-	2082	3091
10000	-	-	-

comparison with CSZ

k \ m	1x	10x	100x
10	65.3	276.6	1380
25	59.33	278	1483
50	69.3	258	1375
100	41.67	176.67	1341
1000	46.67	145	941.67
10000	171.5	233	708.33

Table 2: Distributed running time comparison for various number of centers in the solution k and number of machines m. The running time of the BEL (first table), EIM (second table), and CSZ (third table) baselines divided by the running time of our algorithm with the same amount of machine used. The empty fields are due to slow running time of the baseline.

- Our implementation of LLLM [41] algorithm. This algorithm is distributed and hierarchical, it divides each cluster into two until the size of all the clusters are one. It guarantees that the number of parallel rounds is $O(\log n\Delta)$ but it does not provide a theoratical guarantee.

C Further Sequential Comparisons

Fig. 4 presents comparisons between the running time of our algorithm with k-median++ seeding. BEL, and PLSH for KDD Cup and SONG datasets. The running time of our algorithm is at least a factor 125, 437, and 3121 faster than EIM even for $k = 100$ for KDDCUP, SONG, and Census datasets, respectively. The gap increases by increasing k, therefore we do not add EIM results to the plots. These results are expected and is supported by the theoretical analysis of our algorithm.

D Coreset Comparisons

One of the ideas to improve the quality of the solution discussed in various works (e.g., [4]) is to construct a coreset, and then apply possibly slower algorithms that provide solutions with better

dataset / k	100	1000
KDD CUP	1136790	371302
Song	762199000	648547000
Census	15123100	10104100

k / dataset	100	1000
KDD CUP	1311390	436409
Song	805580000	668776000
Census	16761500	10679170

Table 3: In this table, we compare the quality of the solution of our algorithm (left table) and BEL(right table) after applying the greedy improvements. All the numbers are reported for 5 runs.

k / dataset	100	1000
KDD CUP	3.52	3.91
Song	1.11	4.53
Census	1.99	12.6

Table 4: In this table, we compare the running time if BEL divided by the running time of our algorithm after applying the greedy improvements. All the numbers are reported for 5 runs.

quality on the coreset. One such algorithm is k-median++ followed by greedy improvement steps likes Lloyds algorithm. In the BEL algorithm [4], authors after constructing the coreset use this technique to improve the quality of the solution. Notice that since the size of the coreset is smaller than the original input, the running time of these greedy steps are better than applying the same technique on the entire input. Each step of the Lloyds-style algorithm is as follows:

- Assign each point to the closest center.
- For each center, solve the 1-median problem on the points that are assigned to it.

We apply the same Idea to our approach as well. We first use our algorithm to create a solution of size $c \cdot k$ for some constant c. Then we let the weight of each point in this solution to be the number of points in the subtree in the tree embedding that this points is opened in. Therefore, we achieve a weighted coreset of size ck and then we run greedy improvement on the solution of size k produced by our algorithm.

Here we compare the quality and the running time of this approach with BEL algorithm. We run BEL algorithm on 100 machine, therefore the size of the produced coreset is $100k$. Similarly we let $c = 100$, so the size of our coreset is also $100k$. We run three rounds of greedy improvements for both algorithms. We present the running time of the BEL divided by the running time of our algorithm in Table 4. We also present the quality of the solution in Table 3. We observe that the quality of the solutions are very comparable while our algorithm is noticeably faster.

E Description of the Massive Datasets

The synthetic datasets are obtained from the HIGGS dataset. Let $n = 11$ million and $d = 28$ be respectively the number of points and the number of dimensions of the HIGGS dataset. The datasets are generated by copying each point c times. The i-th copy of a given point is defined as follows: the first d coordinates are the same as the original point. Then $d + 1$ coordinate is appended and set to i for the i-th copy. We create three datasets for $c = 10, 100, 1000$.

F Construction of the embedding and missing proofs of Section 3.1

In this section for completeness we present a (standard) random embedding of an input point set with coordinates from $\{0, \ldots, \Delta\}^d$ into a restricted 2-hierarchically separated T (see, for example [32]). We start by describing a deterministic embedding that uses a quadtree structure. We start with a d-dimensional axis-aligned cube of side length Δ. This cube corresponds to the root of the tree. We then split the box into 2^d subcubes of equal size. Each non-empty subcube corresponds to one node on the next layer of the tree and is connected by an edge of length $\Delta\sqrt{d}/2$, half the maximum

distance of two points inside the cube. Now we apply this procedure recursively: Non-empty cubes[12] of side length $\Delta/2^i$ correspond to nodes in the tree with hop distance i from the root. We subdivide each such cube into subcubes of side length $\Delta/2^{i+1}$ and connect each non-empty subcube to the node corresponding to its containing cube using an edge of length $\sqrt{d}\Delta/2^{i+1}$. The partition stops when the side length of the cube is 1 and so each cube contains a single point. This defines our deterministic embedding into a 2-RHST by mapping each input point to the corresponding leaf. Now we randomize the embedding. This is simply done by adding the same random shift r chosen uniformly at random from $[0, \Delta]^d$ to all input points. Then we employ the above process starting with an initial box of side length 2Δ.

Note that every point can compute its position in the embedding independently in fact it only need to have access to its own coordinate and the uniform random shift.

Recall that for two points $p, q \in \{1, \ldots, \Delta\}^d$ we use $\textsc{Dist}_T(p, q)$ to denote their (shortest-path) distance in the resulting tree T. We now show some interesting properties of our embedding.

Lemma F.1 *Let $p, q \in \{1, \ldots, \Delta\}^d$ be two points. Then*

$$\|p - q\|_2 \leq \textsc{Dist}_T(p, q) \quad \textit{and}$$
$$\mathbf{E}[\textsc{Dist}_T(p, q)] = O(d \cdot \log \Delta) \cdot \|p - q\|_2.$$

Proof. We observe that if $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ are in different subcells of the quadtree of side length $\Delta/2^{i+1}$ then their distance in the tree is at least $\sqrt{d}\Delta/2^i$. Next we observe that there exists a coordinate i such that $|p_i - q_i| \geq \|p - q\|_2/\sqrt{d}$. This implies that for every i with $\Delta/2^{i+1} < \|p - q\|_2/\sqrt{d}$ we have that p and q are in different cells, which implies the first inequality.

Next we prove the second item. Observe that we can view our random shift s as independently shifting each dimension by a random value from $[0, \Delta]$. This implies that the probability that two points are separated in dimension j in the quadtree cells of side length $\Delta/2^i$ is $\min\{1, \frac{|p_j - q_j| \cdot 2^i}{\Delta}\}$. By the union bound, the probability that they are separated in any dimension is at most $\frac{\|p-q\|_1 \cdot 2^i}{\Delta}$. If the points are separated in the quadtree cells of side length $\Delta/2^i$ then there are two edges of length $\sqrt{d}\Delta/2^{i-1}$ on their unique connecting path. We obtain for $h = \log \Delta + 1$ that

$$\begin{aligned}\mathbf{E}[\textsc{Dist}_T(p, q)] &\leq 2 \sum_{i=0}^{h} \frac{\sqrt{d} \cdot \Delta}{2^i} \cdot \frac{\|p - q\|_1 \cdot 2^i}{\Delta} \\ &\leq 2 \cdot (h + 1) \cdot \sqrt{d} \cdot \|p - q\|_1.\end{aligned}$$

Finally, the result follows since $\|p - q\|_1 \leq \sqrt{d}\|p - q\|_2$. □

Theorem 3.1 *Let $P \subseteq \{0, \ldots, \Delta\}^d$ be a point set. There exists a procedure that constructs a 2-RHST tree T in time $O(nd \log \Delta)$ such that for the its optimum solution $C^*_{T,k}$ using k centers, we have $\mathbf{E}[\textsc{Cost}(P, C^*_{T,k})] = O(d \cdot \log \Delta) \cdot \textsc{Opt}(P, k)$. Furthermore all the input points are mapped to leaves of the RHST. In addition, we have that $\mathbf{E}[\max_k \frac{\textsc{Cost}(P, C^*_{T,k})}{\textsc{Opt}(P,k)}] = O(d \cdot \log \Delta \log(dn\Delta))$.*

Proof. Let C^* be the optimal solution on the input data set. By the second item of the previous lemma and linearity of expectation we get $\mathbf{E}[\textsc{Cost}_T(P, C^*)] \leq O(d \log \Delta) \cdot \textsc{Opt}(P, k)$. By the first item of the previous lemma we then get $\mathbf{E}[\textsc{Cost}(P, C^*_{T,k})] \leq \mathbf{E}[\textsc{Cost}_T(P, C^*_{T,k})] \leq \mathbf{E}[\textsc{Cost}_T(P, C^*)] = O(d \cdot \log \Delta \cdot \textsc{Opt}(P, k))$.

In order to prove the second statement, observe that the cost of any non-trivial solution is at least 1 and at most $d\Delta n$. Let $k_1, k_2, \ldots, k_m$ be the sequence such that k_i is the largest number of centers such that the optimal k_i-median cost is at least 2^i. We observe that the expected maximum of the m non-negative random variables $\textsc{Cost}(P, C^*_{T,k_i})/\textsc{Opt}(P, k)$ is at

[12] We ignore here degenerate cases that may arise when a point lies on the boundary of the cell, since in the final embedding we will randomly shift the point set and this event will happen with probability 0.

most $m \cdot \max_k \mathbf{E}[\textsc{Cost}(P, C^*_{T,k_i})/\textsc{Opt}(P, k_i)]$. By the first item of the theorem this quantity is $O(md \log \Delta))$. Now observe that for any i and any k with $k_i > k > k_{i-1}$ we have $\textsc{Cost}(P, C_{T,k}) = O(\textsc{Cost}(P, C_{T,k_{i-1}}))$ since $\textsc{Cost}(P, C_{T,k})$ is non-increasing in k and we have $2\textsc{Opt}(P, k) \leq \textsc{Opt}(P, k_i)$ by the choice of the k_i. This implies that $\textsc{Cost}(P, C^*_{T,k})/\textsc{Opt}(P, k) = O(\textsc{Cost}(P, C^*_{T,k_i})/\textsc{Opt}(P, k_i))$. This implies the second part of the theorem.

Finally we note that the running time is $O(nd \log \Delta)$, since it consist of $\log \Delta + 1$ times of mapping all the points to the nodes of the tree. Also mapping each node is simply dividing all its coordinates by 2^i for layer i. We also note that this mapping is easily parallelizable because, in order to compute the mapping for a specific node, we only need to know its coordinates and the random shift r. □

G Missing proof of Section 3.2.1

Theorem 3.2 *For any set of points $P \subseteq \{0, \ldots, \Delta\}^d$ and distance function $\textsc{Dist}_T$ defined by an 2-RHST T, Algorithm 1 returns an optimum solution for the hierarchical k-median problem.*

Proof. The proof is by induction on the height of the tree T. For the base case, when T is of height 0, the statement is clear: in that case the instance consists of a single point so $\textsc{Cost}_T(P, S) = 0$ if $S \neq \emptyset$.

For the inductive step, suppose that the statement is true for all RHSTs of height less than h and consider an RHST T of height h. We use the following notation:

- Let C be the set of children of the root of T.
- For a child $y \in C$, let T_y denote the sub-tree rooted at y, let P_y denote the subset of leave-nodes in P, and let $p_y = |P_y|$.
- Finally, let D be the sum of edge-lengths from the root to a leaf in T.

A key observation is that, by the definition of RHSTs, we have $\textsc{Dist}_T(x, x') = 2D$ for points $x \in P_y$ and $x' \in P_{y'}$ from different sub-trees $y \neq y' \in C$. Therefore, when Algorithm 1 selects g_y centers in the sub-tree T_y for some child $y \in C$, it makes the same selection as if it was ran on the instance consisting of only the points in P_y. Now, fix a child $y \in C$ and consider running Algorithm 1 on the subinstance (corresponding to the sub-tree T_y) defined by the set of points P_y, cost function $\textsc{Cost}_{T_y}$, and the number of centers equal to p_y(Note that in this instance the number of centers is equal to the number of nodes in P_y). Let $x^{(y)}_1, \ldots, x^{(y)}_{p_y}$ denote the returned centers indexed in the order they were selected by the greedy algorithm. Then, if we let g_y be the number of centers that the greedy Algorithm 1 selects in P_y, the solution returned by Algorithm 1 on the whole instance equals $\bigcup_{y \in C: g_y > 0} \{x^{(y)}_1, \ldots, x^{(y)}_{g_y}\}$. Moreover, by the induction hypothesis (using that T_y has height $h-1$), we have that $\{x^{(y)}_1, \ldots, x^{(y)}_i\}$ is an optimal selection of $i \in \{1, 2, \ldots, p_y\}$ centers to the instance on points P_y and cost metric $\textsc{Cost}_{T_y}$. Now another important observation, that follows from the definition RHSTs, is that the cost of a solution S decomposes:

$$\textsc{Cost}_T(P, S) = \sum_{y \in C: S \cap P_y \neq \emptyset} \textsc{Cost}_{T_y}(P_y, S \cap P_y) + \\ +2D \cdot \sum_{y \in C: S \cap P_y = \emptyset} p_y \,.$$

Thus an optimal solution must optimally select the centers in each subinstance corresponding to a sub-tree. Therefore, since greedy is optimal on each subinstance, any solution that for $y \in C$ selects o_y centers[13] from sub-tree T_y has cost at least $\textsc{Cost}_T\left(P, \bigcup_{y \in C: o_y > 0} \{x^{(y)}_1, \ldots, x^{(y)}_{o_y}\}\right)$. To understand the cost of such a solution, for $y \in C$ and $i \in \{1, \ldots, p_y\}$, define the "cost decrease" achieved by adding the center $x^{(y)}_i$ to the set $\{x^{(y)}_1, \ldots, x^{(y)}_{i-1}\}$ of centers to be

$$d^{(y)}_i = \textsc{Cost}_{T_y}(P_y, \{x^{(y)}_1, \ldots, x^{(y)}_{i-1}\}) - \textsc{Cost}_{T_y}(P_y, \{x^{(y)}_1, \ldots, x^{(y)}_{i-1}, x^{(y)}_i\}) \,,$$

[13] Note that o_y can potentially be different than g_y because o_y is the number of centers in y selected by the optimal solution while g_y is the number of centers selected by the greedy algorithm.

where for notational convenience we let $\textsc{Cost}_{T_y}(P_y, \emptyset) = 2D \cdot p_y$. Then,

$$\textsc{Cost}_T\left(P, \bigcup_{y \in C: o_y > 0} \{x_1^{(y)}, \ldots, x_{o_y}^{(y)}\}\right) = \\ = 2D \cdot |P| - \sum_{y \in C} \sum_{i=1}^{o_y} d_i^{(y)} .$$

Now to conclude the inductive step, observe that the greedy selection criteria implies that $d_1^{(y)} \geq d_2^{(y)} \geq \ldots \geq d_{p_y}^{(y)}$ for $y \in C$. Hence, the greedy algorithm select in each subtree a solution of minimum cost. But now using the fact that the cost of solution can be expressed as cost of the solution in the subtrees it follows that Algorithm 1 returns a solution that contains the centers in an order that maximizes at every step the value $d_i^{(y)}$ which, by the above expression, minimize the the cost.

Finally, we note that the partitions induce a hierarchical clustering because every time we open a new center all the points in the sub-tree rooted at the new label node are assigned to it. In fact, suppose that this is false then there is another node in the sub-tree that is already label. But this is impossible by construction because we always label the highest unlabelled ancestor in the tree and so an already labelled node cannot be at a lower level of a newly labelled node. □

H Missing proof of Section 3.2

We start by showing the. following lemma:

Lemma H.1 *Algorithm 2 and Algorithm 1 output the same solution.*

Proof. Suppose by contradiction that the two algorithms return different ordering of the centers. In particular, let $\{v_1^1, v_2^1, \ldots\}$ the ordering in which points are selected by Algorithm 1 and $\{v_1^2, v_2^2, \ldots\}$ the ordering obtained after sorting by Algorithm 2 and let i be the first index for which $v_i^1 \neq v_i^2$. Recall that during the execution of the two algorithms we sort by breaking ties consistently(for example by looking at points ids). Our proof strategy is to show that if v_i^1 is selected by Algorithm 1 over v_i^2, this contradicts the fact that v_i^2 is the i-th element in the ordering Algorithm 2.

Now, let ℓ be the largest number such that $A_\ell(v_i^1)$ does not contain any node in $\{v_1^2, v_2^2, \ldots, v_{i-1}^2\}$ and let ℓ' be the largest number such that $A_{\ell'}(v_i^2)$ does not contain any node in $\{v_1^2, v_2^2, \ldots, v_{i-1}^2\}$. We note that there are two nodes $v_x \in A_\ell(v_i^1)$ and $v_y \in A_{\ell'}(v_i^2)$ for which $\textsc{Benf}(v_x, \ell')$ and $\textsc{Benf}(v_y, \ell')$ have not been discarded by Algorithm 2. This is true because in the first filtering Algorithm 2 does not discard the maximum benefit elements in the subtrees rooted $a_\ell(v_i^1)$ and $a_{\ell'}(v_i^2)$. And in addition, at any level larger than $\ell(\ell')$ no element in $A_\ell(v_i^1)$ ($A_{\ell'}(v_i^2)$) is the element of maximum benefit by definition of ℓ and by the fact that Algorithm 2 selected $\{v_1^2, v_2^2, \ldots, v_{i-1}^2\}$. So the maximum benefit elements in the subtrees rooted $a_\ell(v_i^1)$ and $a_{\ell'}(v_i^2)$ are not discarded even in this second phase.

Now, if $v_y \neq v_i^2$ Algorithm 2 would. have select v_y over v_i^2 because after filtering we know by the above observations that v_y is the element with maximum benefit in $A_{\ell'}(v_i^2)$. For the same reason we also have that $\textsc{Benf}(v_x, \ell) \geq \textsc{Benf}(v_i^1, \ell)$. In addition given that Algorithm 2 selects v_i^2 as i-th center we have $\textsc{Benf}(v_i^2, \ell') \geq \textsc{Benf}(v_x, \ell) \geq \textsc{Benf}(v_i^1, \ell)$. But now, by the definition of $\textsc{Benf}$ and by observation 3.3 and lemma 3.4 this imply that the cost of $\{v_1^1, v_2^1, \ldots, v_{i-1}^1, v_i^2\}$ is smaller than the cost $\{v_1^1, v_2^1, \ldots, v_{i-1}^1, v_i^1\}$ or they are the same but the ties are broken in favor of v_i^2. So Algorithm 1 will select v_i^2 over v_i^1 leading to a contradiction. So the two orderings and the two solutions are identical.

For the partition we note that the also in the new algorithm the points are assigned to the closest labelled ancestor and so the partitions are equivalent. □

We are now ready to show that Algorithm 2 has the desired properties.

Theorem 3.5 *Algorithm 2 finds an optimum solution for the hierarchical k-median problem on RHSTs in time $O(n \log^2(\Delta + n))$.*

Proof. The fact that Algorithm 2 finds the optimum solution follows from combining Lemma H.1 and Theorem 3.2. Furthermore, all the BENF (Line 1) can be compute in time $O(n \log^2 \Delta)$, since we have to compute $O(n \log \Delta)$ many and each is a sum of $O(\log \Delta)$ values. Now consider some level ℓ in the tree embedding. The total overall number of pairs $(\cdot, \ell)$ in each layer is exactly n. Therefore computing the maximum (Line 4), deleting the rest (Line 6) and construct the sets $C_{(x,\ell)}$ takes $O(n)$ time for each layer and $O(n \log \Delta)$ time in total. The second filtering step (Line 10) can also be done in time $O(n \log \Delta)$ since it finds the maximum elements in n lists (one for each element) and each list is of length $O(\log(\Delta))$. Finally, we have that after filtering at most one pair $(x, \cdot)$ for each point x remains. So we can sort the remaining pairs (at most n many) in time $O(n \log n)$, which concludes the lemma. □

I Results in the General Space

In this section we address the assumption over the space of the input points, $P \subseteq \{0, \ldots, \Delta\}^d$.

First observe that given a general instance $P \subseteq \mathbb{R}^d$ and a rough estimate of the optimum solution ($\mathrm{poly}(n)$), one can achieve an instance $P \subseteq \{0, \ldots, \Delta\}^d$ for some $\Delta \in \mathrm{poly}(n)$ by losing a factor $1 + 1/\mathrm{poly}(n)$ in the approximation guarantee in linear time. Unfortunately such estimate of the optimum solution might not be achievable in some cases. Therefore we explain an alternative idea.

The only part in our approach that is using the assumption of $P \subseteq \{0, \ldots, \Delta\}^d$ is the tree embedding step, i.e., 2-RHST. We present how to construct such embedding for $P \subseteq \mathbb{R}^d$. To this end, we first compute an upper bound MAXDIST on the maximum distance between two points within a factor of 2. This can be done by selecting any point and by computing the maximum distance between such point and any other point in the input. Then multiply this distance by 2. The root of the tree represents an axis-aligned cube of side length 2MAXDIST centered at x, we apply the same construction as before to construct each height of the 2-RHST. We continue until every cube contains at most a single point. This results in a tree where all leaves are at the same height, the height is at most $H = O(\log(d\Delta'))$ where Δ' is the ratio between the maximum and minimum distance of two points. Moreover to achieve that the length of the edges at height ℓ is 2^ℓ, we can divide the value of all the edges in the tree by the length of the edges at height zero. Recall that all the edges at the same height have the same value.

This ideas enables to get the results presented in the paper with the same bound but Δ is replaced by $d\Delta'$. Notice that since in all the bounds Δ only appears in $\log$ function, this change does not affect the performance significantly.

J Reducing Dimension in Distributed Setting

The dimensionality reduction is obtained by basically multiplying an n-by-d matrix (representing the input points) with a random d-by-$O(\log n)$ matrix. This can be done using memory $s \in \Omega(d \log n)$ in $O(\log_s(nd \log n))$ MPC rounds with $O((nd \log n)/s)$ machines and a total running time of $O(nd \log n)$ similar to algorithms in [28].

K Parallel Speed-up

In this section, we describe our empirical results on the speed-up obtained by increasing the number of machines. We consider artificial datasets of sizes 100 millions, 1 billion and 10 billions points. We give the relative running time as the number of machine increases in Table 5.

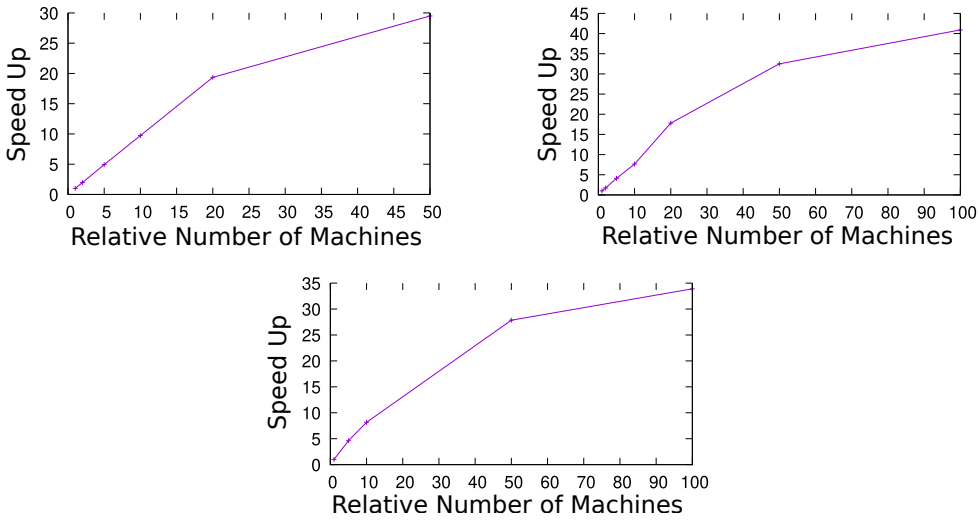

Figure 5: Speed-up for the massive datasets of size almost 100 millions, 1 billion, and 10 billion.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Please refer to the Introduction.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] This work is mostly a new clustering algorithm improving performance of previous work with no clear societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The links to the datasets are provided. We also submit the code with supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The number of machines used are reported.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]