Supplementary Material

Distributed Clustering via LSH Based Data Partitioning

1 Proof of Lemma 3

In this section, we prove the lemma concerning the number of distinct tuples that a cluster hashes to (Lemma 3 of the main text).

Proof. Let us start by considering a single LSH. Let $n(C)$ be the number of distinct hash values for the points of C ($n(C)$ is thus a random variable). The first step is to condition on the diameter of $C' = \{Ax : x \in C\}$. We saw in the proof of Lemma 1 that with probability $1 - 1/n^2$, the diameter of C' is $\leq r'$, where $r' = 4\sigma\sqrt{t + \log n}$. Thus,

$$\mathbb{E}[n(C)] \leq \mathbb{E}[n(C)|\text{diam}(C') < r'] + \frac{1}{n^2}|C|.$$

The second term on the RHS is at most $1/n$. Thus we only need to bound the first. Let us now consider a ball B^* of radius r' as before (the ball contains C' and has radius $< \text{diam}(C')$). As our choice of w satisfies $w > 2r'$, we cannot have two points in C' being close to two different lattice points of a grid G_i^t. Thus, under our conditioning, two points in C' cannot have the same first coordinate for the LSH (this is the coordinate that gives the index of the shift), and different values among the next t coordinates.

Now, let us understand the quantity $n(C)$. Recall Figure 1 and let us denote by B_{outer} and B_{inner} the outer and inner balls containing B^* respectively. Consider the shifts $s_1, s_2, \dots$ in order. For each s_i, either (a) $s_i \in B_{\text{inner}}$, or (b) $s_i \in (B_{\text{outer}} \setminus B_{\text{inner}})$, or (c) $s_i \notin B_{\text{outer}}$. Let us consider s_1. If case (a) occurs for s_1, then all the points in C' are w-close to a point in G_1^t, and they all have the same hash (so $n(C) = 1$). If case (c) occurs for s_1, then none of the points have a w-close point in G_1^t, and thus we move on to s_2. If case (b) occurs, then we may have some points of C hashing with first coordinate equal to 1, and the rest will move on to s_2. Thus $n(C)$ can be upper bounded by the number of times we encounter case (b) before encountering case (a) for the first time (starting from $i = 1, 2, \dots$).

This is thus equivalent to a coin tossing process in which we have three outcomes – heads, tails and "don't-care", where there is a given probability for each outcome (say q_1, q_2, q_3 resp.), the draws are i.i.d., and we wish to compute the expected number of trials to see the first heads, while not counting the don't-cares. This is precisely $(q_2 + q_1)/q_1$. Applying this to our situation,

$$\mathbb{E}[n(C)] = \frac{\text{vol}(B_{\text{outer}})}{\text{vol}(B_{\text{inner}})} = \frac{(w + r')^t}{(w - r')^t} = \left(1 + \frac{2r'}{w - r'}\right)^t.$$

This completes the analysis for one LSH. Since the different hashes are independent and the overall hash is simply a concatenation, the expected number of different hash values is $\mathbb{E}[n(C)]^\ell = O(1)$, by our choice of parameters. □

2 Proof of Lemma 4

Let us thus prove Lemma 4. We introduce one piece of notation. For a cluster C, let $C^{(r)}$ be the set of all points in C that are at a distance $\leq r$ from the centroid of C.

Proof of Lemma 4. Let us consider the radius scale that contains the quantity $2\overline{\rho}$. By Lemma 1, for the hashing at this scale, all the points in $C^{(2\overline{\rho})}$ are hashed to the same integer tuple, with probability $\geq 3/4$. Let us condition on this event, and let j be the bin into which this tuple is hashed in the 2-level hashing scheme. Recall that U_j is the set of *all* points assigned to the jth bin.

Intuitively, we wish to show that points in $C^{(2\overline{\rho})}$ form a good fraction of the points in U_j. We show something roughly to this effect. Let us examine the points in $U_j \setminus C^{(2\overline{\rho})}$. First we have points that are not in $C^{(2\overline{\rho})}$ but hash to the PLSH tuple. Let us call this set of vertices V_0. By Lemma 2, all of these points are at a distance $\leq \alpha \cdot \overline{\rho}$ from the center of C, with probability $\geq 1 - 1/n^2$.

Next, we have points that hash to different PLSH tuples, but are hashed to U_j in the second step of the hashing. We divide these points into two sets: first, we have the points $V_1 = \cup_i C_i^{(2\overline{\rho})}$, i.e., the points that are at distance $\leq 2\overline{\rho}$ from their respective cluster centers. Then, we have $V_2 = \cup_i (C_i \setminus C_i^{(2\overline{\rho})})$.

Claim 1. Points in V_1 hash to $O(k)$ distinct PLSH tuples, w.p. $\geq 9/10$.

Using Lemma 3, the expected number of PLSH tuples for each $C_i^{(2\overline{\rho})}$ is Δ, for some constant Δ; thus by the linearity of expectation, the number of PLSH tuples for V_1 is Δk. By Markov's inequality, the probability of having more than $10\Delta k$ tuples for V_1 is $\leq 1/10$, which proves the claim.

Claim 2. $|V_2| \leq k|C^{(2\overline{\rho})}|$.

Note that any point in V_2 contributes at least $4\overline{\rho}^2$ to the k-means objective. Thus the number of such points is $\leq n\theta^2/4\overline{\rho}^2 \leq k|C|/4 \leq k|C^{(2\rho)}|$, from the definition of the adjusted radius.

Now, Claim 1 allows us to compute the probability that *none* of the $10\Delta k$ PLSH tuples corresponding to V_1 hash to the index j. This is precisely $\left(1 - \frac{1}{Lk}\right)^{10\Delta k}$, which is $\geq 9/10$, for $L = 100\Delta$. (We have used the inequality $(1-x)^n \geq 1 - nx$.) Let us call this event E_1.

Finally, Claim 2 allows us to say that

$$\mathbb{E}[V_2 \cap U_j] = k|C^{(2\overline{\rho})}|/Lk < |C^{(2\overline{\rho})}|/L,$$

where the expectation is over the second step of hashing. Once again, Markov's inequality gives that $\Pr[|V_2 \cap U_j| \leq |C^{(2\overline{\rho})}|] \geq 1 - 1/L > 9/10$. Let us call this event E_2.

Now, consider sampling 4 random points from U_j. If we sample at least one point from $V_0 \cup C^{(2\overline{\rho})}$, our desired conclusion follows. If E_1 and E_2 both occur, the probability of not sampling any such point can be bounded by

$$\left(\frac{|C^{(2\overline{\rho})}|}{|V_0| + 2|C^{(2\overline{\rho})}|}\right)^4.$$

This is clearly at most $1/16$, and thus the lemma follows. □

3 Proof of Theorem 3

In this section, we prove Theorem 3 of the main text, thus giving a parallel implementation of the algorithm from Section 4 of the paper.

We first show a weaker version of the theorem, with the number of machines being $O(nk \log^3 n/s)$.

Proof of weaker version. Note that most parts of the algorithm from Theorem 2 can be immediately parallelized: the $O(\log k)$ runs of the algorithm can be made parallel on different sets of machines. The same is true of trying different radius ranges. Furthermore, assuming that all the machines use the same random seed, the two step hashing can be done entirely in parallel.

The main step that requires work is that of aggregating points that hash to a given bin (recall that this set was denoted U_j, and $j \in [Lk]$) and choosing a random sample from U_j. Aggregating points is a standard "reduce" operation, *provided* the number of points going into a bin is $\leq s$ (in this case all these points will fit on one machine). However, we have two issues: first, the average size $|U_j|$ is n/Lk, which could be larger than s. Second, the numbers $|U_j|$ can vary widely depending on j.

For the weaker version, we can do the following: for each bin j, we associate a group of $n \log n/s$ machines (called G_j), and when a point is hashed to bin j, we randomly assign it to a machine in G_j. Since $|U_j|$ is at most n, each machine will receive $O(s)$ points w.h.p. Now, unless $|U_j| > n \log n/s$, each machine will receive much fewer than s points (and some machines could receive no points). Thus to output a random sample, the machines in G_j need to determine the size of U_j. This is simply an aggregation problem, and can be done in $\lceil \log_s n \rceil$ rounds of MapReduce computation using $\leq s$ memory on each machine: the first round consists of the machines G_j. Now, the first s machines send the number of points they were assigned to machine 1, the next s machines send the number they were assigned to machine $s+1$, and so on. In the next round, only these $|G_j|/s$ machines are active, and they send their counts to a smaller set of machines (this time $|G_j|/s^2$), and so on. This results in the desired bound.

Finally, once $|U_j|$ is computed, the machines independently output each point they have w.p. $O(1)/|U_j|$.[1] □

The main drawback above is the number of machines needed — while storing n points requires n/s machines of memory s, the number above is larger by a k polylog(n) factor. To show Theorem 3, we need an additional "pre-sampling" trick.

Pre-sampling. Let Q be a random sample of the dataset formed by including each vertex w.p. q/n independent of the rest (thus $\mathbb{E}[|Q|] = q$). Now, suppose all the machines know Q, and at radius scale r, suppose that they only hash points $u \in U$ such that $d(u, Q) \geq \alpha \cdot r$. Here $\alpha = O(\log n \log\log n)$, more specifically, the term from Lemma 2. We claim that this results in much smaller bins, w.h.p. First, we observe the following:

Lemma 1. *Let us fix a radius scale, and let h be an $\ell(t+1)$ tuple of integers for which there exists some $u \in U$ that hashes to h. Then in the procedure described above (machines hash only points that are far from the pre-sample), the number of points hashing to h is at most $4n \log n/q$, w.p. at least $1 - 1/n^2$.*[2]

Proof. Let us denote the PLSH hash by H, for convenience. Now, for any $h = H(u)$, we consider $H^{-1}(h) := \{v \in U : H(v) = h\}$. The first observation is that if $|H^{-1}(h)| \geq 4n \log n/q$, then w.p.

[1] This gives only $O(1)$ points in expectation. To output precisely a certain number of points, the machines first output points independently w.p. $O(\log n)/|U_j|$, collect the outcome on one machine, and then randomly pick a subset of the desired size. This results in one extra round.

[2] The probability is over the choice of the pre-sample.

$1 - 1/n^4$, the pre-sample has a non-empty intersection with it; i.e., $H^{-1}(h) \cap Q \neq \emptyset$. This is easy to see: the probability that none of the elements of $H^{-1}(h)$ is picked in Q is at most

$$\left(1 - \frac{q}{n}\right)^{4n \log n/q} \leq \frac{1}{n^4}.$$

We can take a union bound over all the h (as there are at most n of them), and thus w.p. $1 - 1/n^3$, the pre-sample has a non-empty intersection with all $H^{-1}(h)$ that have size $\geq 4n \log n/q$. Now, by Lemma 2, we have that w.h.p., $\text{diam}(H^{-1}(h)) \leq \alpha \cdot r$. Thus if we only hash points that are far from the pre-sample, then *none* of the points in such an $H^{-1}(h)$ get hashed! This means that the number of points actually hashing to any given h is at most $4n \log n/q$, which gives the lemma. □

We now see how the lemma lets us improve the number of machines.

Proof of Theorem 3. We choose a pre-sample of size $q = \min\{Lk, s\}$. Thus by Lemma 1, for any hash value h, the number of points hashed to it in the pre-sample based algorithm is at most $4n \log n/q$. Suppose there are g different tuples that have a non-zero number of points hashed to them, and let these number of points be denoted $a_1, a_2, \ldots, a_g$. Thus $\max_i a_i \leq 4n \log n/q$.
Claim. In the second step of hashing, none of the bins get more than $O(n \log^2 n)/\min\{k, s\}$ points, with probability $> 1 - 1/n$.
Proof of claim. Note that in the second step of hashing, the tuples are mapped independently and u.a.r. to a bin. Thus the number of points in bin j is distributed as $\sum_i a_i Y_i$, where Y_i is a Bernoulli random variable that is 1 w.p. $1/Lk$ and 0 otherwise. Thus, using a Chernoff bound (e.g., Theorem 3.3 from [CL06]), we have

$$\Pr\left[\left|\sum_i a_i Y_i - \frac{1}{Lk}\sum_i a_i\right| > t\right] \leq \exp\left(-\frac{t^2}{\sum_i a_i^2 \cdot \frac{1}{Lk} + \frac{2t}{3}\max_i a_i}\right). \quad (1)$$

Now, using $\sum_i a_i^2 \leq (\max_i a_i)(\sum_i a_i) \leq (n \log n/q)(n)$, the RHS above can be simplified to $\exp\left(-\frac{t^2}{n^2 \log n/Lkq + 2tn \log n/3q}\right)$. Plugging in $t = (4n \log^2 n)/q$, this simplifies to $1/n^2$. Thus we can take a union bound over all the bins, and the claim follows.

Now, if $(4n \log^2 n)/q < s$, then we have one machine handle $sq/(4n \log^2 n)$ bins. If it is $> s$, then for each bin, we assign $(n \log^2 n)/qs$ machines. They can perform at most $\log_s n$ rounds of computation (as in the proof of the simpler case above) and return the desired sample. In both cases, the number of machines is bounded by $O(n \text{ polylog}(n))k/qs$ and the number of points assigned to each machine is s, as we desired. □

References

[CL06] Fan Chung and Linyuan Lu. Concentration inequalities and martingale inequalities: a survey. *Internet Math.*, 3(1):79–127, 2006.