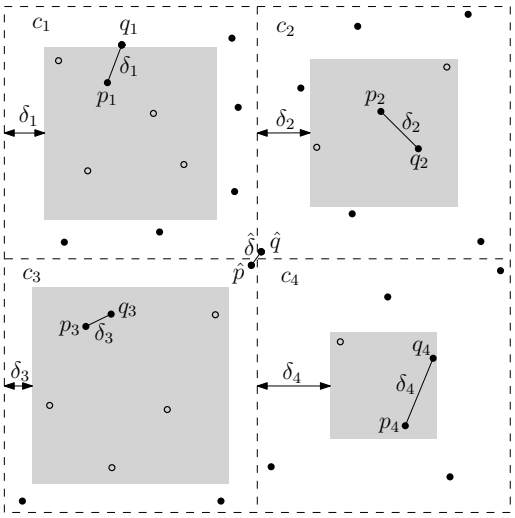

Figure 7: Closest Pair in SpatialHadoop

say that $\mathcal{C}_1$ dominates $\mathcal{C}_2$ if the minimum distance of $\mathcal{C}_1$ is greater than or equal to the maximum distance of $\mathcal{C}_2$. In this case, the pair $\mathcal{C}_2$ can be pruned as it can never contain the farthest pair in the dataset. This is depicted in Figure 6(b), where the farthest pair of $\mathcal{C}_1$ must have a distance greater than the farthest pair of $\mathcal{C}_2$. In this case, the pair of cells $\langle c_3, c_4 \rangle$ will never contribute to the final answer, and hence will not be considered further for any processing.

Once all dominated cell pairs are pruned, the algorithm computes the local farthest pair for each selected pair of cells by finding the *local* convex hull, then applying the rotating calipers algorithm on the result [33]. It is important to note that it is feasible here to use the in-memory algorithms for local convex hull as the size of each pair is bounded by twice the cell size. Finally, the algorithm computes the global farthest pair by collecting all local farthest pairs and selecting the one with largest distance. For interested readers, the pseudocode of the farthest pair algorithms is in Appendix A.4.

7. CLOSEST PAIR

The closest pair (Figure 1(e)) in any dataset can be found using a divide and conquer algorithm [25]. The idea is to sort all points by x coordinates, and then based on the median x coordinate, we partition the points into two subsets, P_1 and P_2, of roughly equal size and recursively compute the closest pair in each subset. Based on the two distances of the two closest pairs found, the algorithm then continues to compute the closest pair of points $p_1 \in P_1$ and $p_2 \in P_2$ such that the distance between them is better (smaller) than the two already found. Finally, the algorithm returns the best pair among the three pairs found. In this section, we introduce our closest pair algorithms for Hadoop and SpatialHadoop.

7.1 Closest Pair in Hadoop

Applying the divide and conquer algorithm described above in Hadoop as-is will be fairly expensive. First, it requires a presort for the whole dataset which requires, by itself, two rounds of MapReduce [29]. Furthermore, the merge step requires random access to the list of sorted points which is a well-known bottleneck in Hadoop file system [21]. On the other hand, using the default Hadoop loader to partition the data and compute the local closest pair in each partition (as in the farthest pair algorithm) may produce incorrect results. This is because data is partitioned randomly, which means that a point in one partition might form a closest pair

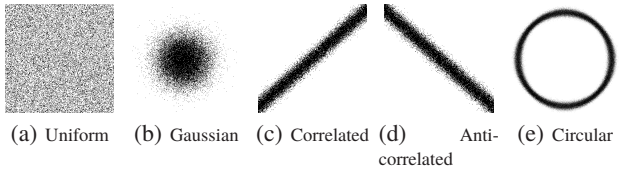

Figure 8: Synthetic data distributions

with a point in another partition. Finally, as we mentioned with the farthest pair problem in Section 5.1, the brute force approach would work but it requires too much computations for large files.

7.2 Closest Pair in SpatialHadoop

Our closest pair algorithm in SpatialHadoop is an adaptation of the traditional closest pair divide and conquer algorithm [25]. The algorithm works in three steps, *partitioning*, *local closest pair*, and *global closest pair*. In the *partitioning* step, the input dataset is loaded using SpatialHadoop loader which partitions the data into cells as shown in Figure 7. As the size of each partition is only 64MB, the algorithm computes the *local closest pair* in each cell using the traditional divide and conquer algorithm and returns the two points forming the pair. In addition, the algorithm must also return all candidate points that can possibly produce a closer pair when coupled with points from neighboring cells. Looking at Figure 7, let us assume that the closest pair found in c_1 has the distance δ_1. We draw an internal *buffer* with size δ_1 measured from the boundaries of c_1 and return all points inside this buffer (shown as solid points) as the candidate points while all other points are pruned. Notice that the two points forming the closest pair were returned earlier and are not affected by this pruning step. As shown in this example, each cell c_i may have a different buffer size δ_i based on the closest pair found inside this cell. While the minimum of all δ's would be a better and tighter value to compute all buffers, it cannot be used because the MapReduce framework enforces all map tasks to work in isolation which gives the framework more flexibility in scheduling the work. Finally, in the *global closest pair* step, all points returned from all cells are collected in a single machine which computes the global closest pair $\hat{p}, \hat{q}$ by applying the traditional divide and conquer algorithm to the set of all points returned by all machines.

For this algorithm to be correct, the cells must be non-overlapping, which is true for the cells induced by SpatialHadoop partitioning. This ensures that when a point p is pruned, there are no other points in the whole dataset P that are closer than the ones in its same cell. Otherwise, if cells are overlapping, a point p near the overlap area might be actually very close to another point q from another cell, thus none of them can be pruned. For readers familiar with the MapReduce programming, the pseudocode is in Appendix A.5.

8. EXPERIMENTS

In this section we give an experimental study to show the efficiency and scalability of CG_Hadoop. Both Hadoop and SpatialHadoop clusters are based on Apache Hadoop 1.2.0 and Java 1.6. All experiments were conducted on an internal university cluster of 25 nodes. The machines are heterogeneous with HDD sizes ranging from 50GB to 200GB, memory ranging from 2GB to 8GB and processor speeds ranging from 2.2GHz to 3GHz. Single machine experiments are conducted on a more powerful machine with 2TB HDD, 16GB RAM and an eight core 3.4GHz processor.

Experiments were run on three datasets: (1) OSM1: A real dataset extracted from OpenStreetMap [30] containing 164M poly-