Optimizing Spatial Queries in MapReduce

Ibrahim Sabek
PhD Student, University of Minnesota, USA
sabek@cs.umn.edu

1. PROBLEM AND MOTIVATION

Spatial query processing has become a fundamental part of many important applications, e.g. human brains modeling [31], medical images processing [44], and urban planning [17,42]. Due to its importance, there have been numerous research efforts in the last three decades to support many spatial operations (e.g., range query [34], spatial join [24], kNN joins [45] and computational geometry [8]). Most recently, and coupled with the recent explosion of big spatial data [13], recent research efforts are dedicated to take advantage of the widely-used MapReduce platform [9] to enable spatial query processing for big spatial data (e.g. [11, 28, 36, 46–48]).

Unlike the case of traditional algorithms of spatial operations where there are extensive studies on benchmarking and query optimization issues (e.g., see [2, 14, 16, 18, 22, 24, 35, 39–41]), there are no similar efforts in MapReduce-based algorithms. This work shows our efforts in optimizing the spatial join operator as a first step towards building a full-fledged MapReduce-based spatial query optimizer. Spatial join is an expensive and crucial operation that joins two spatial datasets based on a spatial predicate, e.g., overlap, cover, touch. Our objective in optimizing the spatial join operation is not to compare between existing MapReduce-based spatial join techniques, and find which is best at what settings. Instead, we come up with a very thorough taxonomy that includes the spectrum of all possible ways of executing a spatial join in MapReduce. All existing algorithms for MapReduce-based spatial join (e.g., [1,12,19,38,43,47–49]) and non-explored yet techniques can be mapped to our taxonomy as special cases. In addition, our spatial join optimizer is equipped with a cost model that estimates the costs of different spatial join options in this taxonomy while taking its decisions. Unlike traditional cost models that focus on the cost of I/O accesses and processing times, the cost model in our optimizer estimates the costs of MapReduce phases.

2. BACKGROUND AND RELATED WORK

Traditional spatial join algorithms. There have been numerous efforts to spatially join two input relations, e.g., see [24] for a comprehensive survey. In general, these algorithms are categorized into three categories based on whether the two input relations are indexed or not, as follows: (a) None of the input relations is indexed [23, 29, 36, 37, 50], (b) Only one of the input relations is indexed [3, 10, 20, 27, 30], and (c) Both input relations are indexed [5–7, 21, 25]. With such numerous algorithms, many benchmarking studies were performed to show the strengths and weaknesses of each algorithm using many query workloads [18, 22, 35, 39, 40].

MapReduce-based spatial join algorithms. Research efforts in MapReduce-based spatial join algorithms come in two flavors, either as stand alone algorithms (e.g., [19,38,43,48,49]), or as part of big spatial data engine (e.g., HadoopGIS [1], SpatialHadoop [12]). In either case, the proposed algorithms are usually tailored towards one specific scenario, e.g., spatial join in SpatialHadoop assumes that both input data sets are spatially partitioned, while most of the algorithms assume that none of the two input data sets is spatially partitioned [1, 19, 38, 43, 48, 49].

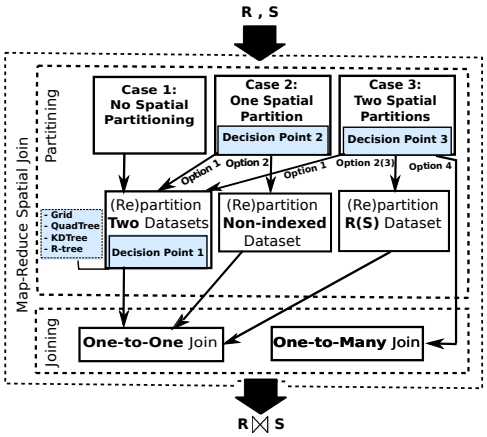


Figure 1: Taxonomy of Spatial Joins in MapReduce.

3. SPATIAL JOIN TAXONOMY

Figure 1 gives our taxonomy that covers the spectrum of almost all possible ways of executing a spatial join in MapReduce. In general, we abstract any spatial join algorithm in MapReduce into two phases, namely, *partitioning* and *joining*, which are executed in the *map*, and *reduce* functions, respectively. We study the possible decisions that cover the three possible cases of input R and S:

Case 1: No Spatial Partitioning. Both R and S are partitioned using the default Hadoop File System (HDFS) partitioning, i.e., no spatial partitioning. In this case, we have only one option which is partitioning both R and S based on a common spatial partitioning scheme in a way that each partition in R will be joined with only one corresponding partition in S. As our query optimizer is built inside SpatialHadoop [12], we have five partitioning choices to select from; STR [26], STR+ [26], QuadTree [15], KDTree [4] and Grid [32], which is depicted as *Decision Point 1* in Figure 1.

Case 2: One File is Spatially Partitioned. Only one input dataset, say S, has the default HDFS partitioning, while the other data set R is spatially partitioned. In this case, we have one of two options: (1) Ignore the partitioning of R, and repartition both input files with one of the five partitioning schemes as in Case 1, or (2) Only repartition S using the exact partitioning scheme of R. In both cases, R and S will be partitioned using the same scheme, and hence each partition from R will be joined with exactly one partition from S. Using these two options, we have six choices in total to select from, which is depicted as *Decision Point 2* in Figure 1. In the first option,

we will pay the cost of partitioning R and S, while in the second option, we will pay the cost of partitioning only S.

Case 3: Two Files are Spatially Partitioned. Both input data sets R and S are already spatially partitioned on two different schemes. In this case, we have one of four options: (1) Ignore the partitioning of R and S, and completely repartition them with one of the five partitioning schemes as in Case 1, (2) Ignore the partitioning of one of the two input files, say S, and repartition it using R partitioning scheme as in Case 2, (3) Similar to Option 2, ignore the partitioning of R and repartition it using S partitioning scheme as in Case 2, or (4) Keep R and S partitioning intact, and go directly to *Joining* phase where each partition in R could be joined with many partitions from S that overlap with it. Using these four options, we have eight choices to select from, which is depicted as *Decision Point 3* in Figure 1. In the fourth option, there is no partitioning overhead, yet, it encounters more overhead in the *joining* phase as more partition pairs need to be joined (one-to-many join).

4. SPATIAL JOIN COST ESTIMATION

In general, the spatial join cost comes from; estimating the partitioning boundaries, actual data partitioning, transferring the overlapping partitions to the computing machines, and actual joining of overlapping partitions. Based on these cost factors, we define a cost model to estimate the performance of different options in the three input cases discussed earlier (derivations are omitted).

Case 1: No Spatial Partitioning. Assume that R and S have N_R and N_S input partitions, respectively. The spatial join cost, C_{tot_1}, of any of the five alternatives in Case 1 can be estimated as:

$$C_{tot_1} = C_{est} + [N_R + N_S]C_p + M[C_{shuf} + C_j] \quad (1)$$

where C_{est} is the cost of estimating partitioning boundaries, C_p is the cost of partitioning one partition in any dataset, M is the number of generated partitions from the partitioning phase, C_{shuf} is the cost of moving a pair of partitions to the same computing machine, and C_j is the cost of in-memory joining this pair of partitions. In Equation 1, the estimation C_{est} and partitioning $[N_R + N_S]C_p$ costs are almost constant across the different partitioning techniques because they are based on the same input data R and S. In contrast, the costs of moving MC_{shuf} and joining overlapping partitions MC_j depend on the value of M.

Case 2: One File is Spatially Partitioned. Let us assume that R is spatially partitioned, while S is not. The spatial join cost, C_{tot_1}, of any of the five alternatives of the first option in Case 2 can be estimated as in Equation 1. In addition, the spatial join cost, C_{tot_2}, of the second option can be estimated as follows:

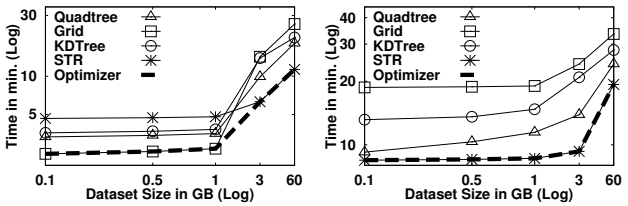
$$C_{tot_2} = N_S C_p + N_R[C_{shuf} + C_j] \quad (2)$$

As shown in Equation 2, the second option saves the boundaries estimation cost C_{est} because the partitioning boundaries are already specified. In addition, the second option significantly reduces the partitioning cost by partitioning one dataset, i.e. S dataset, instead of two datasets. Therefore, the partitioning cost will be $N_S C_p$ instead of $[N_R + N_S]C_p$ as in the first option. On the other hand, the partitioning boundaries of the N_R partitions are not estimated to partition both R and S together. By contrasting Equations 1 and 2, we find that the second option will be preferred over the first option only if the saving of the partitioning phase $[C_{est} + N_R C_p]$ in the second option is larger than the saving of the joining phase in the first option, $[N_R - M][C_{shuf} + C_j]$.

Case 3: Two Files are Spatially Partitioned. Similar to Case 2, the spatial join cost, C_{tot_1}, of any of the five alternatives of the first option in Case 3 can be estimated as in Equation 1. Analogously, the spatial join costs of the second, C_{tot_2}, and third C_{tot_3} options can be estimated as in Equation 2. In addition, the spatial join cost of the fourth option, C_{tot_4} can be estimated as follows:

$$C_{tot_4} = [\sum_{i=1}^{N_R}\sum_{j=1}^{N_S} I(i,j)][C_{shuf} + C_{join}] \quad (3)$$

where $I(i,j)$ is a function that returns 1 if partition i from R overlaps with partition j from S, and 0 otherwise. As shown in Equation 3, the cost of partitioning phase, including the cost of boundaries estimation C_{est} and actual partitioning C_{part}, is eliminated, because there is no partitioning overhead. However, the number of overlapping partitions still affects the total cost.



(a) Uniform Datasets (b) Skewed Datasets

Figure 2: Case 1 - Effect of data size and skewness.

5. SPATIAL JOIN OPTIMIZER LOGIC

Having R and S in Case 1, we apply Equation 1 on the only option in Case 1, but, with the five partitioning techniques supported in SpatialHadoop [12], and then choose the partitioning technique that has the minimum of C_{tot_1}. Experimentally, we find that it is always best to use a data partitioning technique (e.g. STR [26]) when inputs are large or skewed. Having R and S in Case 2, we apply Equation 1 on the first option with five partitioning techniques (similar to Case 1), apply Equation 2 on the second option, and then choose the option that has the minimum of C_{tot_1} and C_{tot_2}. Experimentally, we find that if R is already partitioned with a data partitioning technique or if both inputs are small, then, it is better to use the second option. Having R and S in Case 3, we follow the route of Case 2 in all options, except in the fourth option (direct joining) where we apply Equation 3. Experimentally, we find that the fourth option is chosen only if both R and S are small.

6. RESULTS AND CONTRIBUTIONS

We show the results of our query optimizer in Case 1. We use the Lakes and Parks datasets from OpenStreetMap [33] as inputs. The experiment runs on a Hadoop cluster of 30 machines, each of 32 cores, 64GB RAM, and 4TB disk. In Figure 2(a), the size of Parks is varied from 0.1GB to 60GB, while the size of Lakes is fixed at 3GB. The two datasets are completely uniform. We omit the STR+ results as they are similar to the STR ones. In this case, our optimizer selects the Grid option at small datasets, and the STR option at large datasets which are the best options at these data sizes. Figure 2(b) shows the performance of our optimizer with skewed datasets. Our optimizer is still able to predict the best performance in case of skewed datasets by consistently selecting the STR option that usually generates the minimum number of balanced partitions M. The high accuracy in our experiments makes our optimizer a very appealing solution for big spatial data engines.

7. REFERENCES

[1] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop GIS: A High Performance Spatial Data Warehousing System over Mapreduce. *VLDB*, 6(11), Aug. 2013.

[2] N. An, Z.-Y. Yang, and A. Sivasubramaniam. Selectivity Estimation for Spatial Joins. In *ICDE*, 2001.

[3] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, J. Vahrenhold, and J. S. Vitter. A Unified Approach for Indexed and Non-indexed Spatial Joins. In *EDBT*, 2000.

[4] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *CACM*, 1975.

[5] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger. Multi-step Processing of Spatial Joins. *SIGMOD Record*, 23(2):197–208, may 1994.

[6] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-trees. In *SIGMOD*, 1993.

[7] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Parallel Processing of Spatial Joins using R-trees. In *ICDE*, pages 258–265, 1996.

[8] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *CACM*, 51(1):107–113, jan. 2008.

[10] J. V. den Bercken, B. Seeger, and P. Widmayer. The Bulk Index Join: A Generic Approach to Processing Non-Equijoins. In *ICDE*, 1999.

[11] A. Eldawy, Y. Li, M. F. Mokbel, and R. Janardan. CGHadoop: Computational Geometry in MapReduce. In *SIGSPATIAL*, 2013.

[12] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce Framework for Spatial Data. In *ICDE*, pages 1352–1363, Seoul, Korea, apr. 2015.

[13] A. Eldawy and M. F. Mokbel. The Era of Big Spatial Data (Tutorial). In *ICDE*, 2016.

[14] C. Faloutsos, B. Seeger, A. Traina, and C. T. Jr. Spatial Join Selectivity Using Power Laws. In *SIGMOD*, 2000.

[15] R. Finkel and J. Bentley. Quad Trees a Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 1974.

[16] M. R. Fornari, J. L. D. Comba, and C. Iochpe. Query Optimizer for Spatial Join Operations. In *GIS*, 2006.

[17] H. Gao, H. Zhang, D. Hu, R. Tian, and D. Guo. Multi-scale Features of Urban Planning Spatial Data. In *Geoinformatics*, 2010.

[18] O. Gunther, V. Oria, P. Picouet, J.-M. Saglio, and M. Scholl. Benchmarking Spatial Joins A La Carte. In *SSDM*, 1998.

[19] H. Gupta, B. Chawda, S. Negi, T. A. Faruquie, L. V. Subramaniam, and M. Mohania. Processing Multi-way Spatial Joins on Map-reduce. In *EDBT*, 2013.

[20] C. Gurret and P. Rigaux. The Sort/Sweep Algorithm: A New Method for R-tree based Spatial Joins. In *SSDM*, 2000.

[21] L. Harada, M. Nakano, M. Kitsuregawa, and M. Takagi. Query Processing for Multi-Attribute Clustered Records. VLDB, pages 59–70, 1990.

[22] E. G. Hoel and H. Samet. Benchmarking Spatial Join Operations with Spatial Output. In *VLDB*, pages 606–618, 1995.

[23] E. H. Jacox and H. Samet. Iterative Spatial Join. *TODS*, 28(3):230–256, sep. 2003.

[24] E. H. Jacox and H. Samet. Spatial Join Techniques. *TODS*, 32(1), mar. 2007.

[25] J.-D. Kim and B.-H. Hong. Parallel Spatial Join Algorithms using Grid Files. In *DANTE*, pages 226–234, 1999.

[26] S. T. Leutenegger, M. A. Lopez, and J. Edgington. STR: A Simple and Efficient Algorithm for R-tree Packing. In *ICDE*, pages 497–506, 1997.

[27] M.-L. Lo and C. V. Ravishankar. Spatial Joins Using Seeded Trees. In *SIGMOD*, 1994.

[28] W. Lu, Y. Shen, S. Chen, and B. C. Ooi. Efficient Processing of K Nearest Neighbor Joins Using MapReduce. *PVLDB*, 5(10), jun. 2012.

[29] G. Luo, J. F. Naughton, and C. J. Ellmann. A Non-blocking Parallel Spatial Join Algorithm. In *ICDE*, pages 697–705, 2002.

[30] N. Mamoulis, P. Kalnis, S. Bakiras, and X. Li. Optimization of Spatial Joins on Mobile Devices. In *SSTD*, 2003.

[31] H. Markram, K. Meier, T. Lippert, S. Grillner, R. Frackowiak, S. Dehaene, A. Knoll, H. Sompolinsky, K. Verstreken, J. DeFelipe, S. Grant, J.-P. Changeux, and A. Saria. Introducing the human brain project. *Procedia Computer Science*, 2011.

[32] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *TODS*, 9(1):38–71, mar. 1984.

[33] OpenStreetMap. https://www.openstreetmap.org/.

[34] A. Papadopoulos and Y. Manolopoulos. Multiple Range Query Optimization in Spatial Databases. In *ADBIS*, 1998.

[35] A. Papadopoulos, P. Rigaux, and M. Scholl. A Performance Evaluation of Spatial Join Processing Strategies. *Advances in Spatial Databases*, 1999.

[36] J. M. Patel and D. J. DeWitt. Partition Based Spatial-merge Join. In *SIGMOD*, 1996.

[37] J. M. Patel and D. J. DeWitt. Clone Join and Shadow Join: Two Parallel Spatial Join Algorithms. In *GIS*, pages 54–61, 2000.

[38] S. Puri, D. Agarwal, X. He, and S. K. Prasad. MapReduce Algorithms for GIS Polygonal Overlay Processing. In *IPDPSW*, 2013.

[39] D. Sidlauskas and C. S. Jensen. Spatial Joins in Main Memory: Implementation Matters! *PVLDB*, 8(1):97–100, sep. 2014.

[40] B. Sowell, M. V. Salles, T. Cao, A. Demers, and J. Gehrke. An Experimental Analysis of Iterated Spatial Joins in Main Memory. *VLDB*, 6(14), sep. 2013.

[41] C. Sun, D. Agrawal, and A. E. Abbadi. Selectivity Estimation for Spatial Joins with Geometric Selections. In *EDBT*, 2002.

[42] M. Ubell. The Montage Extensible DataBlade Architecture. In *SIGMOD*, 1994.

[43] K. Wang, J. Han, B. Tu, J. Dai, W. Zhou, and X. Song. Accelerating Spatial Data Processing with MapReduce. In *ICPADS*, 2010.

[44] K. Wang, Y. Huai, R. Lee, F. Wang, X. Zhang, and J. H. Saltz. Accelerating Pathology Image Data Cross-comparison on CPU-GPU Hybrid Systems. *PVLDB*, 2012.

[45] C. Xia, H. Lu, B. C. Ooi, and J. Hu. Gorder: An Efficient Method for KNN Join Processing. In *VLDB*, 2004.

[46] C. Zhang, F. Li, and J. Jestes. Efficient Parallel kNN Joins for Large Data in MapReduce. In *EDBT*, 2012.

[47] S. Zhang, J. Han, Z. Liu, K. Wang, and S. Feng. Spatial Queries Evaluation with MapReduce. In *GCC*, pages 287–292, Aug. 2009.

[48] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu. SJMR: Parallelizing spatial join with MapReduce on clusters. In *CLUSTER*, pages 1–8, Aug. 2009.

[49] Y. Zhong, J. Han, T. Zhang, Z. Li, J. Fang, and G. Chen. Towards Parallel Spatial Query Processing for Big Spatial Data. In *IPDPSW*, 2012.

[50] X. Zhou, D. J. Abel, and D. Truffet. Data Partitioning for Parallel Spatial Join Processing. *Geoinformatica*, 1998.