Detecting Skewness of Big Spatial Data in SpatialHadoop

Alberto Belussi
Dept. of Computer Science,
University of Verona
Italy
alberto.belussi@univr.it

Sara Migliorini
Dept. of Computer Science,
University of Verona
Italy
sara.migliorini@univr.it

Ahmed Eldawy
Dept. of Computer Science,
University of California Riverside
USA
eldawy@ucr.edu

ABSTRACT

In recent years several extensions of Hadoop system have been proposed for dealing with spatial data and SpatialHadoop belongs to this group. In the MapReduce paradigm a task can be parallelized by partitioning data into chunks and performing the same operation on them, eventually combining the partial results at the end. Thus, the applied partitioning technique can tremendously affect the performance of a parallel execution, since it is the key point for obtaining balanced map tasks. However, when skewed distributed datasets are considered, using a regular grid might not be the right choice and other techniques have to be applied, which in turn are more expensive to build. This paper illustrates an approach for detecting the degree of skewness of a spatial dataset, based on the box counting function. Moreover, given the degree of skewness and some experimental observations, a heuristic is sketched in order to decide which partitioning technique to apply in order to improve as much as possible the performance of subsequent operations.

CCS CONCEPTS

• **Information systems** → *Geographic information systems*;

KEYWORDS

SpatialHadoop, Skewed data, Partitioning, MapReduce, BigData

ACM Reference Format:
Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2018. Detecting Skewness of Big Spatial Data in SpatialHadoop. In *26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18), November 6–9, 2018, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3274895.3274923

1 INTRODUCTION

In recent years many efforts have been devoted to the implementation in MapReduce of many spatial operations. For instance, SpatialHadoop [7] implements the well-known range query and spatial join with several variants that in addition can be combined with different strategies for data partitioning (i.e., global indexing). The MapReduce paradigm is based on the fundamental principle that the operation to be performed on a dataset can be parallelized by partitioning the data into chunks and performing the same sub-operation on them (map phase), eventually combining the partial results at the end (reduce phase). Thus, the data partitioning is fundamental and has an impact on the effectiveness of the parallel execution. A good partitioning strategy has to produce balanced executions of the map tasks, i.e. they should require more or less the same amount of resources.

For traditional (textual) datasets, the partitioning techniques applied by Hadoop are based only on the data size, namely the goal of the partitioning is to produce resulting chunks (called splits) having almost the same size in bytes. This aseptic partitioning technique may not be the best way to partition spatial data, since while it provides a perfect load balance, it will result in a poor performance as spatially nearby records will end up in two splits [3]. For this reason, several spatial indexes have been implemented in SpatialHadoop that take into account also spatial properties during partitioning. For instance, indexes based on a regular grid, Quadtrees and R-trees are available in SpatialHadoop [6] and can be built on a dataset before applying a spatial join or a range query.

However, not all spatial indexes behave in the same way and in different cases the best index to use can change according to the spatial characteristics of the dataset at hand and the operations that have to be applied afterwards. For example, the spatial index based on a regular grid can be the optimal choice with uniformly distributed datasets, while other indexes (e.g., Quadtrees and R-trees) require more time for their creation and this cost may be justified only for dataset with skewed distribution. Without knowing any details about the distribution of the geometries in the space covered by the dataset, it is difficult to choose the right index (i.e., the right partitioning technique). Moreover, the construction of the wrong kind of index can affect the benefit of the MapReduce paradigm. For instance, building a regular grid index on a very skewed dataset can lead to very unbalanced map tasks.

Tab. 1 shows the result of the execution in SpatialHadoop of the Distributed Join (DJ) [8] applied to two datasets, where the first one is uniformly distributed and indexed using a regular grid, while the second one vary from a uniform to a skewed distribution and has been indexed using different techniques, namely regular grid, Quadtree and R-tree. As expected, when both datasets are uniformly distributed the response time of the DJ is similar regardless of the used index, while, when a skewed distributed dataset is considered, then the differences are significant and in this particular case are in favor of the R-tree. This is mainly due to the fact that when the distribution is skewed, the partitioning of the geometries based on a regular grid does not produce balanced splits, while the Quadtree and the R-tree indexes perform better and produce more balanced splits. This is evident from columns 4, 5 and 6 of Tab. 1, which report the characteristics of the map tasks in the different cases. It is clear that balancing the cost of the single map tasks is crucial for the total cost of the MapReduce job.

SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5889-7/18/11.
https://doi.org/10.1145/3274895.3274923

Table 1: Execution of the DJ in SpatialHadoop with different kind of indexes (i.e., Gr = regular grid, Qt = Quadtree, Rt = R-tree) and different distribution of the datasets (i.e., Uni = uniform distribution, Skw = skewed distribution).

Dataset distribution	Dataset index	Tot. time (mills)	Map tasks # tasks	Map tasks AVG time (mills)	Map tasks %RSD time
Uni/Uni	Gr/Gr	145,307	37	15,833	4%
Uni/Uni	Gr/Qt	150,458	51	18,902	9%
Uni/Uni	Gr/Rt	147,646	54	16,231	7%
Uni/Skw	Gr/Gr	125,327	33	22,710	90%
Uni/Skw	Gr/Qt	96,001	52	11,209	50%
Uni/Skw	Gr/Rt	40,205	21	18,087	28%

The aim of this paper is to provide a way to easily detect some hints about the dataset distribution, so that the more effective partitioning technique (index) can be applied to it. We will also empirically study the effects of the various types of indexes on datasets with different distributions.

The proposed technique for determining the characteristics of the dataset distribution is based on the concept of box-counting that was first proposed in [5] for computing the fractal dimension of a dataset of points. The behavior of the box-counting function measured in a restricted range of values (representing the cell side of the grid) can be described by a power law and it was used in [5] for estimating the selectivity of self-join and range query, then extended in [9] to the spatial join on distinct datasets. In general this technique can be applied to any real dataset of multidimensional data. We propose its application in the context of big spatial data in particular for the following reasons: (i) it is an efficient technique for detecting information about the dataset distribution; (ii) it produces just one number characterizing the data distribution and does not require to store auxiliary structures, like histograms; (iii) the box-counting function can be computed in parallel, since it calculates a uniform histogram storing the counts and this can be easily implemented in MapReduce. Moreover, according to the degree of skewness, some heuristics are proposed that allow one to decide which kind of index can be more effective to balance the cost of the MapReduce execution of the subsequent operations.

Related works mainly regards the sampling-based methods, like SATO [11], SpatialHadoop [6, 7], ScalaGiST [10], and Simba [12], or histogram-based methods, like AQWA [1].

2 EVALUATION OF DATASET SKEWNESS

This section presents the definition of the box-counting function $BC_D^q(r)$ for a given dataset D containing 2D geometries of type point, line or polygon embedded in the Euclidean plane. This is an extension of the box-counting function proposed in [5] which applies only to finite set of points. We will see later that this function can provide some hints about the skewness of the dataset.

Definition 2.1 (Box-counting function). Given a dataset D, containing 2D geometries (i.e, points, lines or polygons), and a scale r, representing the cell size of a grid covering the reference space of D (i.e., the MBR of the whole dataset D), the function Box-counting $BC_D^q(r)$ is defined as follows:

$$BC_D^q(r) = \sum_i p_i(D)^q \qquad with\ q \neq 1 \tag{1}$$

where $p_i(D) = count$(geometries of D intersecting the i cell).[1]

In [5] the author shows that the box-counting function is useful for computing the generalized fractal dimension of a finite set of points, where q represents the exponent in Eq. 1 and r is the considered scale (i.e. the cell side of the grid). Intuitively, given a grid with cells of side r, the box-counting function with $q = 0$ counts the number of cells that are intersected by at least one point of D, similarly here the function $BC_D^0(r)$ counts the number of cells that are intersected by at least one geometry of D. In this way, when exponents q is greater than 1, the box-counting becomes the sum of the number of geometries intersecting a cell raised to q.

As we will show shortly, this function can be used to detect the skewness of a dataset by computing it for $q = 0$ and $q = 2$ while varying the value of r. More specifically, the level of skewness of a dataset depends on how this value changes while increasing r.

Notice that with respect to the definition given in [5], which applies only to set of points and counts the number of points contained in a cell, here we count the number of geometries that intersect a cell. This extension of the box-counting function from set of points to generic geometric datasets could also be obtained in other ways. (i) A first option could be to choose a representative point for each geometry of the dataset (e.g., its centroid) and then apply the classical box-counting function. This can be the simplest solution, but it does not ensure to always detect the real behavior of the dataset. For instance, considering a set of big polygons covering almost the whole reference space, their centroids would be clustered in a small region, thus producing a point set that does not describe at all the original dataset layout. (ii) Another solution could be to substitute the geometries with their vertices; again, there could be regions covered by geometries that are not covered by their vertices, with the same effect described in the previous case.

The adopted solution, namely to count the number of geometries intersecting a cell, is equivalent to suppose that each geometry g is converted to a set of points $\mathcal{P}(g)$ covering the same space with a granularity that satisfies the following hypothesis: *if g intersects a cell i, then there exists at least one point $p \in \mathcal{P}(g)$ such that p intersects the cell i*. With this hypothesis, we can extend the box-counting function from point sets to sets of geometries and apply to our case the results of [5]. Notice that if this solution can produce an over-estimation for the selectivity, it does not have negative effects for the problem considered in this paper, namely the estimation of the dataset distribution, conversely it leads to a more precise result.

Definition 2.2. Given a dataset D, containing 2D geometries (i.e, points, lines or polygons), and a fixed exponent q the *Box-counting plot* is the plot of $BC_D^q(r)$ versus r in logarithmic scale. Now, we can consider such plot and exploit the following observation of [5]: for real datasets the box-counting plot reveals a trend of the box-counting function that, in a large interval of scale values r, behaves as a power law:

$$BC_D^q(r) = \alpha \cdot r^{E_q} \tag{2}$$

[1] The case $q = 1$ is excluded, since it always produces a fixed number which is equal to the total number of geometries in the input.

where α is a constant of proportionality and E_q is a fixed exponent that characterizes the power law.

The Box-counting plot is vital for the computation of the exponent E_q for a given dataset D, since this exponent becomes the slope of the straight line that approximates $BC_D^q(r)$ in a range of scales (r_1, r_2), thus it can be computed by a linear regression procedure. E_q characterizes the dataset distribution as explained below.

(i) For $q = 0$, E_0 is negative and the power law, given the length of the cell side r, computes the number of cells that are intersected by the dataset D. Notice that, if D is uniformly distributed in the reference space (the Euclidean plane in our case), then the number of cells intersecting D coincides with the total number of cells of the grid, thus the more r increases the more this number decreases. As a consequence, in case of an uniform distribution, E_0 is equal to minus the dimension of the embedding space, in our case $E_0 = -2$.

(ii) For the datasets F representing fractals (like the Sierpinski's triangle), it is known from the theory that E_q coincides with the fractal dimension of F for each q (it is a consequence of the self similarity property), thus for the Sierpinski's triangle $E_0 = -1.585$.

(iii) Finally, we can observe that E_0 and E_2 could be chosen as descriptors for the distribution of a dataset D. Indeed, E_0 can be an indicator of the cases where the dataset leaves empty some areas of the reference space, while E_2 can also be affected by the concentration of the datasets in some areas with respect to other ones, i.e. the situations where there are no empty areas, but different concentrations in different areas.

In order to practically use E_0 and E_2 as indicators of distribution for real datasets, it is necessary to find and easy and efficient way for computing them given a dataset D. Due to space constraints the presentation of the MapReduce implementation of an algorithm for computing both E_0 and E_2 is skipped.

3 PARTITIONING TECHNIQUES

This section briefly describes the partitioning techniques that characterize the indexes available in SpatialHadoop and we show their effects on skewed distributed datasets.

Hadoop divides the input of a MapReduce job into fixed-size blocks, called *splits*, and instantiates and executes one map task for each split, which applies the map function on each record in its split. The split size is generally set equal to the size of an HDFS (Hadoop Distributed File System) block, that is 128 Mbytes by default. The main idea behind the MapReduce paradigm is that, since the time required to process one split is smaller than the time required to process the whole input, it is convenient to execute the map tasks in parallel so that the total execution time will be reduced. If all map tasks can be executed in parallel, then such cost depends on the map task that takes longer. Therefore, the faster parallel executions can be obtained when the map task are well balanced.

The partitioning of data into splits is a crucial operation for obtaining well balanced map tasks. Moreover, the data partitioning is usually applied randomly and this might produce balanced tasks for uniformly distributed datasets, but not in general. Indeed, as we have shown in Tab. 1, when skewed distributed datasets are considered the cost of the map tasks is often unbalanced, causing a performance degradation.

When a global index is built on an input file of a MapReduce job, this means that you subdivide such file into splits by using a criterion that is different from the random choice. For spatial data the partitioning criterion is based on the locality property, i.e. geometries that are closed in space will be placed in the same split. Therefore, in order to guarantee that splits contains more or less the same number of geometries, we should use different types of grids according to the dataset distribution. In SpatialHadoop we have three types of grid for data partitioning at global level: *Regular grid*: based on space partitioning, it identifies the cells by dividing the 2D space in both axes by a constraint measure; it is suitable for uniform distributions. *Quadtree-based grid*: based on space partitioning, it identifies the cells by recursively subdividing a cell (starting from the whole space) in 4 equal cells until the number of geometries per cell reaches a threshold; it is suitable for skewed distributions. *Rtree-based grid*: based on partitioning of geometries, it identifies the cells by recursively aggregating the geometries of the dataset until the number of geometries per cell reaches a threshold; again this grid is more effective for skewed datasets.

Now, in order to choose the right index type, we need to introduce some criteria for assessing the effectiveness of the data partitioning techniques with respect to the operations that we want to perform on the indexed data. Considering the range query and the spatial join, we define two descriptors of the partitioning produced by the index. These descriptors have to be minimized in order to improve the effectiveness of an index on a dataset D: (i) the %RDS (relative standard deviation with respect to the mean) of the split cardinality (i.e. the number of geometries), denoted as $d_1(D)$; (ii) the percentage of the reference space covered by the grid cells that represents dead space, i.e. space containing no data ($d_2(D)$).

Notice that, d_1 affects the cost of a single map task, while d_2 has an impact on the total number of map tasks to be instantiated by the range query or the spatial join. Considering some synthetic and real datasets, we apply the three partitioning techniques and we obtain the results shown in Tab. 2 which confirm the theoretical behavior of E_0 and E_2. Indeed, for the uniform distribution the obtained partitions are very similar for all indexing techniques; in this case the regular grid can be the best choice, since its creation cost is less. For the diagonal with buffer we can see that, the Quadtree-based and the Rtree-based grids adapt best to the dataset distribution. However, the partitioning produced by the Rtree-based grid has more balanced cells w.r.t. the Quadtree-based partition, in terms of number of geometries per cells. Finally, when a clustered dataset is considered, we obtain the best partitioning with the Quadtree-based grid, while the Rtree-based grid produces a partition with lots of dead space.

At this point the idea is to exploit the exponent E_0 and E_2 for choosing the right grid for data partitioning, without building all kinds of indexes. Notice that d_1 and d_2 are only known after an index has been constructed, while E_0 and E_2 are index-independent. Our goal is to use E_0 and E_2 to choose a good index that will have good (small) values for d_1 and d_2.

Property 1 (Dataset diffusion). *Given a dataset D, when its exponent E_0 is close to -2.0, then the descriptor d_2 for any index is close to zero, i.e. no dead space exists.*

Table 2: Datasets used in the experiments: UD = uniform distribution, DL = diagonal line, DL_B = diagonal line with buffer, DC_B = double cluster with buffer, PR_{USA} = USA primary roads, PR_{AUS} = Australian primary roads, WA_{USA} = USA water areas and ST_{AUS} = Australian states. In the 6th column: RT = Rtree, QT = Quadtree, and RG = regular grid.

Dataset	Size GB	E_0	E_2	d_1/d_2 RG	d_1/d_2 QT	d_1/d_2 RT	Index
UD	1.0	-2.00	2.00	0.1%/0.0%	0.2%/0.0%	0.5%/0.0%	RG
DL	1.0	-1.13	1.07	123%/94%	122%/70%	2%/93%	RT
DL_B	1.0	-1.76	1.18	119%/49%	98%/28%	1%/36%	RT
DC_B	1.0	-1.60	0.04	155%/33%	97%/25%	2%/44%	QT
PR_{USA}	1.0	-1.27	1.03	128%/36%	53%/49%	7%/57%	RT
WA_{USA}	2.2	-1.84	1.60	101%/13%	62%/10%	11%/15%	RG
PR_{AUS}	1.2	-1.55	0.67	156%/27%	137%/27%	21%/22%	QT
ST_{AUS}	0.3	-1.73	1.69	39%/32%	27%/26%	54%/20%	RG

Property 2 (Dataset distribution). *Given a dataset D, when its exponent E_2 is close to 2.0, then the descriptor d_1 of a regular grid is close to zero, i.e. every cell of the grid contains the same number of geometries belonging to D.*

Now we add to the above presented formal properties some experimental observations. Given a dataset D: (1) when the computed E_0 is around 1.0, then the dataset is skewed, has some dead space and is located around a curve, thus it is usually connected. In this case, the regular grid will have high values for both d_1 and d_2, since the dataset cannot occupy all cells and be uniformly partitioned into regular cells, while the Rtree-based grid will have the lowest value for d_1, because it is the technique that starts from geometries and not from the space for clustering data, but also a good value for d_2, since data cover a connected region. Finally, the Quadtree-based grid will have a good value for d_2, but a higher value for d_1 w.r.t. Rtree-based grid. (2) When the computed E_0 is around 0.0, then the dataset is skewed, has lots of dead space and is located around two or more points, thus it is usually not connected. In this case, the regular grid will have high values of d_1 and d_2, as before; the Rtree-based grid will have the lowest value for d_1 but a higher value for d_2, since the connectivity is lost, while the Quadtree-based grid will have better values for both d_1 and d_2, since it adapts better to the clustered datasets. Similar considerations are valid for the values of E_2, where instead of dead space it detects regions of lower/higher concentration, thus affecting more deeply the descriptor d_1.

By applying Prop. 1 and 2 and the above listed observations, we can derive a heuristic, based on the decision tree in Fig. 1, for choosing the more effective index suitable for a given dataset D. Notice that in the tree we use threshold values equal to -1.5, -0.5 for the choices regarding E_0, while we use threshold values equal to 1.0, 1.5 for E_2, since we consider E_2 only when the dataset is spread throughout the reference space or when E_0 is around 1.0, thus in this case the values near to 0.0 cannot be reached by E_2.

Experiments performed on the datasets in Tab. 2 confirm that the heuristic produces the best index suggestion for both the spatial join and the range query operations. For instance, when a join is performed between PR_{USA} and WA_{USA} using the suggested indexes, the time required is about 316 seconds versus about 450/550 seconds required using all the other index combinations.

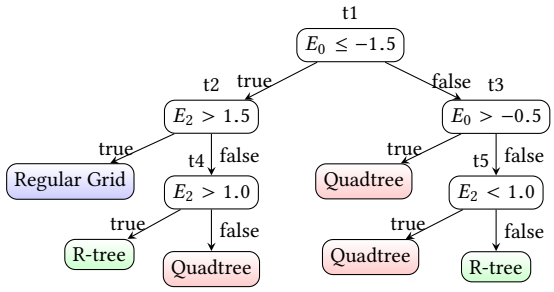

Figure 1: Decision tree for data indexing in SpatialHadoop.

4 CONCLUSION

This paper considers the impact of a skewed distribution on the performances of range query and spatial join, considering as reference framework SpatialHadoop and its partitioning techniques: regular grid, Quadtree-based grid and R-tree based grid. We proposed a new technique based on the Box-counting function [5] for efficiently estimating a dataset distribution and accordingly choose the more suitable partitioning technique. Future work regards the application of the knowledge about dataset distribution to reduce-side joins, the application of the box-counting function for estimating the skewness of multidimensional and spatio-temporal datasets [2, 4].

ACKNOWLEDGMENTS

This work was partially supported by the Italian National Group for Scientific Computation (GNCS-INDAM) and by "Progetto di Eccellenza" of the Computer Science Dept., Univ. of Verona, Italy.

REFERENCES

[1] A. M. Aly, A. R. Mahmood, M. S. Hassan, W. G. Aref, M. Ouzzani, H. Elmeleegy, and T. Qadah. 2015. AQWA: Adaptive Query Workload Aware Partitioning of Big Spatial Data. *Proc. VLDB Endow.* 8, 13 (2015), 2062–2073.
[2] A. Belussi, O. Boucelma, B. Catania, Y. Lassoued, and P. Podestà. 2006. Towards similarity-based topological query languages. In *10th International Conference on Extending Database Technology, EDBT 2006*. Springer, Berlin, 675–686.
[3] A. Belussi, D. Carra, S. Migliorini, M. Negri, and G. Pelagatti. 2018. What Makes Spatial Data Big? A Discussion on How to Partition Spatial Data. In *10th Int. Conf. on Geographic Information Science*. LIPIcs, Dagstuhl, Germany, 1–15.
[4] A. Belussi, C. Combi, and G. Pozzani. 2008. Towards a formal framework for spatio-temporal granularities. In *Proceedings of the 15th Int. Workshop on Temporal Representation and Reasoning*. 49–53.
[5] A. Belussi and C. Faloutsos. 1998. Self-spacial Join Selectivity Estimation Using Fractal Concepts. *ACM Trans. Inf. Syst.* 16, 2 (1998), 161–201.
[6] A. Eldawy, L. Alarabi, and M. F. Mokbel. 2015. Spatial Partitioning Techniques in SpatialHadoop. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1602–1605.
[7] A. Eldawy and M. F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *2015 IEEE 31st Int. Conf. on Data Engineering*. 1352–1363.
[8] A. Eldawy and M. F. Mokbel. 2017. *Spatial Join with Hadoop*. Springer International Publishing, 2032–2036.
[9] C. Faloutsos, B. Seeger, A. Traina, and C. Traina, Jr. 2000. Spatial Join Selectivity Using Power Laws. *SIGMOD Rec.* 29, 2 (2000), 177–188.
[10] P. Lu, G. Chen, B. C. Ooi, H. T. Vo, and S. Wu. 2014. ScalaGiST: Scalable Generalized Search Trees for Mapreduce Systems. *Proc. VLDB Endow.* 7, 14 (2014), 1797–1808.
[11] H. Vo, A. Aji, and F. Wang. 2014. SATO: A Spatial Data Partitioning Framework for Scalable Query Processing. In *Proc. of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 545–548.
[12] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. 2016. Simba: Efficient In-Memory Spatial Analytics. In *Proc. of the 2016 Int. Conf. on Management of Data*. 1071–1085.