What Makes Spatial Data Big? A Discussion on How to Partition Spatial Data

Alberto Belussi
Department of Computer Science, University of Verona, Italy
alberto.belussi@univr.it
https://orcid.org/0000-0003-3023-8020

Damiano Carra
Department of Computer Science, University of Verona, Italy
damiano.carra@univr.it
https://orcid.org/0000-0002-3467-1166

Sara Migliorini
Department of Computer Science, University of Verona, Italy
sara.migliorini@univr.it
https://orcid.org/0000-0003-3675-7243

Mauro Negri
Department of Electronics, Information and Bioengineering, Politecnico of Milan, Italy
mauro.negri@polimi.it

Giuseppe Pelagatti
Department of Electronics, Information and Bioengineering, Politecnico of Milan, Italy
giuseppe.pelagatti@polimi.it

Abstract

The amount of available spatial data has significantly increased in the last years so that traditional analysis tools have become inappropriate to effectively manage them. Therefore, many attempts have been made in order to define extensions of existing MapReduce tools, such as Hadoop or Spark, with spatial capabilities in terms of data types and algorithms. Such extensions are mainly based on the partitioning techniques implemented for textual data where the dimension is given in terms of the number of occupied bytes. However, spatial data are characterized by other features which describe their dimension, such as the number of vertices or the MBR size of geometries, which greatly affect the performance of operations, like the spatial join, during data analysis. The result is that the use of traditional partitioning techniques prevents to completely exploit the benefit of the parallel execution provided by a MapReduce environment. This paper extensively analyses the problem considering the spatial join operation as use case, performing both a theoretical and an experimental analysis for it. Moreover, it provides a solution based on a different partitioning technique, which splits complex or extensive geometries. Finally, we validate the proposed solution by means of some experiments on synthetic and real datasets.

2012 ACM Subject Classification Information systems → Geographic information systems

Keywords and phrases Spatial join, SpatialHadoop, MapReduce, partitioning, big data

Digital Object Identifier 10.4230/LIPIcs.GIScience.2018.2

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

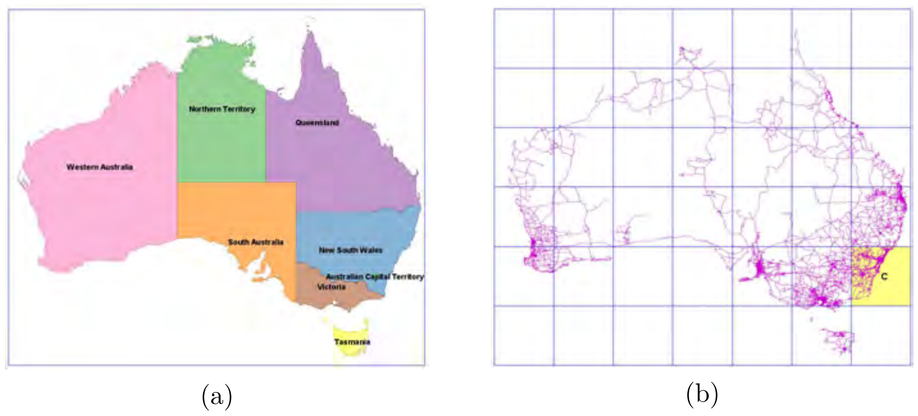

Figure 1 Example of unbalanced datasets between which a join has to be performed. (a) contains few geometries with a big extent described with a restricted number of vertices, while (b) contains many geometries with a small extent represented using several vertices.

1 Introduction

In recent years the amount of spatial data available to users have increased tremendously and the demand of resources for performing geo-spatial analysis on them cannot be satisfied any more by traditional GIS systems. As a consequence of this new scenario, in the last decade many efforts have been devoted to the extension of systems for big data processing based on the MapReduce paradigm, like Hadoop [15] or Spark [16], in order to make them able to deal with geo-spatial data. For instance, SpatialHadoop [7] is the result of one of these projects, it is an extension of Apache Hadoop which provides a native support for spatial data, in terms of spatial data types, operations and indexes. In particular, it provides various implementations of the spatial join, which is one of the most frequently used operation for analyzing spatial datasets and discovering connections between geo-spatial objects [2].

Various spatial join variants are available in literature [10] and some adaptations to the MapReduce context have been provided [8]. In particular, SpatialHadoop implements several spatial join algorithms which share the use of indexes for increasing their performance and avoiding a brute force approach that simply subdivides the Cartesian product of the two input datasets between tasks. As regards to the indexing techniques, all kind of indexes provided by SpatialHadoop are organized into two levels: (i) first data are physically partitioned in different blocks (usually called splits or partitions), producing a first level of index called *global index*, then (ii) in each block a specific index is built that works only on the data of the partition, producing a second level of index called *local index*. This indexing pattern directly derives from the way usually applied for organizing data inside the HDFS (Hadoop Distributed File System). In HDFS, a dataset is partitioned into splits whose size usually corresponds to the HDFS block size and each split represents the input for a single map task. This organization has been originally developed for processing large amount of mono-dimensional (textual) data where the execution time is directly affected by the number of bytes they occupy on the file system. This choice is justified by the observation that the amount of work to be performed on textual data usually depends on the data size (or number of records), thus partitioning data in blocks of the same size and assigning each block to a map task, produces a balanced work distribution among workers. This reasoning has been applied also to spatial data, since the physical partitioning induced by the global index uses again considerations based on the size in bytes of the dataset. However, geo-spatial objects are also embedded in a 2D or a 3D reference space and their extension in these spaces is another dimension that can have an impact on the workload of many spatial operations. Notice that the portion of space occupied by a geo-spatial object on the Earth surface can be completely independent from the size in bytes of its physical representation as a file record. On the contrary the number of bytes may partially represent the complexity of a shape, in terms of number of vertices, but not its extent. During the execution of a spatial join, the average extent of the geo-spatial objects in both datasets affects their mutual selectivity (i.e., the ratio between the effective pairs produced by the join and the total number of possible pairs given by the Cartesian product) and thus it has an impact on the workload of the tasks devoted to its computation. The average extent of the geometries in a dataset can be approximated by means of the average area of the MBR (Minimum Bounding Rectangle) containing them and this parameter can be easily computed during the index construction.

The impact of the average geometry extent on the spatial join becomes particularly relevant when the two datasets are very unbalanced in terms of extent and size. Let us consider the case in which one dataset contains few simple geo-spatial objects with a large extent (possibly covering almost the whole reference space), while the other one is instead huge and contains a large number of geo-spatial objects with a small extent. The first dataset may be possibly stored in a single split, since only few vertices are required for describing the shapes of all objects, while the second dataset requires more splits to accommodate the numerous objects it contains. As a first example, let us consider the case illustrated in Fig. 1, where dataset D_s contains only seven polygons representing the Australian States (Fig. 1.a) while dataset D_r contains several complex linestrings representing the main road elements of the Australian transportation network (Fig. 1.b). A generic partitioning of the two datasets which is based only on their size in bytes will produce only one partition for D_s, since the whole set of geometries can fit in one split, while several different partitions will be built for D_r. In this case, a spatial join operation on them ($D_s \bowtie_{int} D_r$) can be divided into several tasks, but each one of them will work on a split of D_r and on the single global split of D_s, thus the Cartesian product is computed and no pruning effect is obtained by using the index. This means for example that all geometries in the cell with label c in the Fig. 1.b will be tested for intersection with all the states of Australia. Clearly, an efficient use of a local index can improve the performance and avoid some useless tests, but it will not affect the number tasks to be instantiated. Moreover, the problem worsens as the extent of the geometries in D_s enlarges, covering at the end the whole space.

The aim of this paper is to formalize and evaluate the problem discussed above and further explained in Sect. 1.1, called here *parallel execution of unbalanced spatial join*, in order to identify the characteristics that really represents the complexity of spatial data, making them “big”. In particular, Sect. 2 provides a formalization of the problem and a discussion of the limits of the current spatial join algorithms available in SpatialHadoop when applied to unbalanced cases. Sect. 2.4 illustrates by means of some experiments the behavior of spatial join algorithms when applied on synthetic datasets with increasing unbalanced characteristics. Then, Sect. 3 proposes a new approach for dealing with unbalanced spatial join that requires the implementation of an alternative kind of repartition which is based on the geometry extent instead of on the file size. In Sect. 4 some additional experiments show the effectiveness of the proposed approach when applied both to synthetic and real datasets in the execution of unbalanced spatial joins. Finally, Sect. 6 summarizes the obtained results and proposed some future work.

Table 1 Some metadata about two real-world datasets representing the taxonomy of the soil usage (`cv_land`) inside the Basilicata region and its extent (`tot_reg`), respectively. The average number of vertices and the average extent area refer to each single geometry in the dataset.

Dataset	*size*	*#splits*	*#obj*	*#vert*^{avg}	*area*^{avg} (squared meters)
`cv_land`	1.5 (Gb)	12	913,428	70	10,550 (1e4)
`tot_reg`	263 (Kb)	1	1	8,000	10,589,998,917 (1e10)

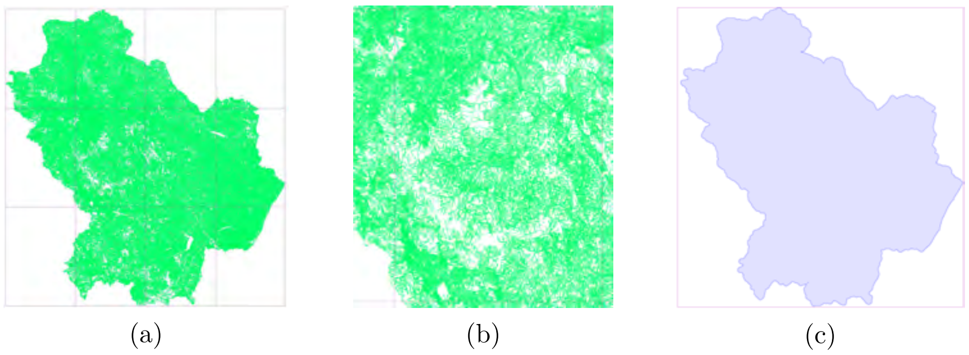

Figure 2 (a) Dataset `cv_land` with its grid. (b) A zoom on one cell of the `cv_land` grid. (c) Dataset `tot_reg` with its grid.

1.1 Motivating Example

The problem discussed in this paper originated form a real-wold case regarding a collection of datasets about a region in Southern Italy, called Basilicata. In particular, we consider two datasets: the first one, called `cv_land`, contains several geometries representing the taxonomy of land usage inside the region, while the second one, called `tot_reg`, contains one object representing the whole territory of Basilicata. Tab. 1 reports some metadata of the two datasets: they greatly differ on the number of objects, their complexity (average number of vertices in each geometry), and their extents (average area of each geometry). The aim of the original task was to perform a qualitative evaluation by verifying the satisfaction of some spatial integrity constraints. In particular, one test has to check if the set of geometries belonging to `cv_land` represents a geometric partition of the whole territory of Basilicata. As shown in [12], the execution of this check by means of a sequence of SQL queries takes several days when executed in a PostgreSQL+PostGIS environment. Therefore, the introduction of a parallel execution has become soon necessary. One of the required query in the above cited sequence coincides with the spatial join between the two datasets. Given two datasets D_1 and D_2, the spatial join determines the pairs $(d_1, d_2) \in D_1 \times D_2$ with an intersecting extent. This operation is usually performed exploiting a plane-sweep like algorithm, in order to reduce the number of required comparisons. Clearly, the case considered in this paper is particularly challenging, since as the extent of a geometry increases w.r.t. the other one, the number of comparisons increases. Similarly, the complexity of each comparison increases as the number of vertices describing each geometry becomes greater.

Fig. 2 illustrates the two datasets with the partitioning induced by the grid index of SpatialHadoop; the number of splits only depends on the dataset size in bytes, so `cv_land` is subdivided into 12 splits (Fig. 2.a), while `tot_reg` is contained into 1 split (Fig. 2.c). Moreover, datasets `cv_land` contains a great number of objects (see a zoom in Fig. 2.b), while the complexity of `tot_reg` in given by the average number of vertices in each geometry.

Table 2 Comparison between the different spatial join algorithms provided by SpatialHadoop. The number of produced pairs is 913,428. The last two algorithms make use of indexes, but their performance are not greatly increased w.r.t. the first algorithm which works on non-indexed data.

Algorithm	# maps	Effective time (min)	Heap usage (MB)	HDFS reads (MB)	HDFS writes (MB)
DJNI	12	92.57	21.87	1.57	243.64
DJGI	18	88.73	30.90	1.66	243.64
DJRE	18	80.06	24.77	1.66	243.64

Tab. 2 reports some data about the execution of the spatial join using the three main algorithms provided by SpatialHadoop, the distributed join with no index (DJNI), the distributed join with grid index (DJGI) and distributed join with repartition (DJRE), which will be briefly discussed in Sect. 2.3. Notice that the time required to perform the join is very high and the execution does not benefit so much from the use of index.

2 Problem Statement

This section formalizes the problem presented in Sect. 1 by discussing in details how data is traditionally partitioned in MapReduce environments (Sect. 2.1) and how such techniques are adapted in SpatialHadoop for implementing spatial indexes (Sect. 2.2). Finally, we introduce the problem of performing a spatial join and how this operation can be effected by the use of a spatial index (Sect. 2.3), anticipating some limits of a size-based partitioning technique that will be discussed in more details in Sect. 2.4.

2.1 Data Partitioning in MapReduce

Hadoop divides the input of a MapReduce job into fixed-size pieces called *splits* and creates one map task for each split. Each map task executes the user-defined (map) function on each record in its split. The main idea behind the MapReduce paradigm is that the time required to process each split individually is smaller than the time required to process the whole input. Therefore, the more such computation on each individual split can be performed in parallel, the more the process performance increases. The split size is generally set equal to the size of an HDFS (Hadoop Distributed File System) block, which is 128 Mbytes by default.

The partitioning of data into splits is a crucial operation for obtaining well balanced map tasks [3, 14, 13]. In particular, if the splits can be analyzed in parallel, the whole job is better balanced when the splits are small, since a faster machine will be able to process proportionally more splits during the map execution than a slower machine, while unbalanced tasks can frustrate the benefit of the parallelism, since a single heavy task can delay the end of the whole job. This observation tends to produce the conclusion that the smaller are the mappers the more the effective execution time of the job can be reduced; however, if the splits are too small, the overhead of managing the splits and creating map tasks begins to dominate the total job execution time. Thus, a tradeoff should be defined and the reference size of 128 Mbytes is the usual choice. Moreover, the partitioning of data is usually applied randomly and this might produce balanced tasks for uniformly distributed datasets, but not in general. In order to address this problem, when spatial data are analyzed, the introduction of auxiliary structures (indexes) is an option. Using a spatial index implies that a criteria based on spatial properties (i.e., closeness) will be used for grouping the records in the same split. The general structure of a spatial index is presented in the following subsection.

2.2 Spatial Indexes in SpatialHadoop

As discussed in the Sect. 1, SpatialHadoop has two level of indexes [5]: a global and a local one. The *global index* determines how data is partitioned among nodes, while the *local index* determines how data is stored inside each block. The construction of a global index g on a input dataset D causes that D is stored as a set of data files each one containing the records spatially belonging to one cell (or partition) of the grid g. More specifically, given a dataset representing the input data, a directory named `dataset.`⟨*index*⟩ will be created containing several files: `_master.`⟨*index*⟩, `part-00000`, `part-00001`, `part-00002`, and so on, where ⟨*index*⟩ denotes the kind of global index (e.g., grid, quadtree, rtree). File `_master.`⟨*index*⟩ represents the global index and it has one row for each partition containing the boundaries of the partition and the partition file name (e.g., -179.32, -54.93, 6.92, 71.28, `part-00000`). All the other files are data files containing the data records. For a grid index, each partition file is simply a text file containing one record for each row, conversely for a R-tree it has a more complex structure subdivided into two sections: the first one contains the tree structure in binary format, while the second one contains the data records.

As discussed in [5] despite the particular kind of index, the number n of desired partitions is computed considering only the file size and the HDFS block capacity which are both fixed for all partitioning techniques. Subsequently, the space is subdivided into n partitions and each record in the input dataset is assigned to one or more of them. Dependently on the fact that the index admits replication or not, geometries crossing partition boundaries can be assigned to more than one partition or to exactly one, respectively. The number of partitions n used for performing the subdivision is crucial in the identification of the number of mappers that will be executed in order to produce the result. As we will see in Sect. 2.4, if the determination of such number is computed considering only the file size and the HDFS capacity, we can obtain strange behaviours, as the one anticipated in Sect. 1.

2.3 Use of Spatial Indexes in Distributed Joins

SpatialHadoop provides five different alternatives of spatial join algorithm: distributed join with no index (DJNI), distributed join with grid-based index (DJGI), distributed join with repartition (DJRE), distributed join with direct repartition (DJDR), and the MapReduce implementation of the partition-based spatial merge join (SJMR). The main differences between them are: (i) the use of indexed or not-indexed data, (ii) the possibility to repartition one of the two datasets using the global index of the other, (iii) the execution of the intersection tests on the map or on the reduce side. All operators share the use of a plane-sweep like algorithm for checking the intersections between two list of geometries. The difference mainly resides in the way the two lists are built by the various operators. In particular, as regards to the map-side joins, the cardinalities of such lists and their composition directly depends on the content and size of each partition.

Since in this paper we are interested in analyzing the impact of data partitioning (i.e., global index) in spatial operators, such as the spatial join, we concentrate only on the map-side joins which exploit the use of indexes during the join computation. Therefore, in the following section we start by considering the behaviour of DJNI and DJGI in presence of unbalanced datasets, then we evaluate the possible positive effects of a repartition of the smaller dataset (in size) by evaluating the behaviour of DJRE. However, in all these algorithms, the extent of geometries and the geometry complexity (in terms of number of vertices) are not considered during the partition process. In DJNI the partition is performed considering only the size in bytes and the constraint of splits capacity, thus records are grouped randomly in the necessary number of splits. In case of the DJGI, the datasets are indexed (i.e., partitioned) considering again the split capacity constraint, so that partitions have a homogeneous size in terms of bytes, but records are grouped according to their spatial closeness, which is evaluated in the space the geometries are embedded in. In case of the DJRE, one dataset is indexed while the one (usually the smaller in size) is repartitioned using the grid of the bigger one. The effect is that geometries of the smaller dataset are partitioned using the spatial closeness principle, but producing splits with potentially less records (i.e., size less than 128 Mbytes) and consequently reducing the cost of the map tasks.

All these spatial join variants perform the join inside the map tasks: each map receives a combined split built by a special reader that matches a split of the first dataset with a split of the second one. Moreover, in DJGI and DJRE a combined split is built combining only pairs of input splits that intersect (through the use of a filter). Therefore, the number of map tasks which will be instantiated is equal for DJGI to the number of intersecting partitions of the two global indexes, while for DJRE it is equal to the number of partitions of the bigger datasets that intersect the smaller one. Given a combined split, each mapper initially split its content into two lists (one for each dataset) and then executes a plane-sweep like algorithm on them in order to identify the pairs of intersecting geometries.

As discussed in [1], the cost of this plane-sweep phase depends on three factors: (i) the cardinality of the two lists (which depends on the partition size), (ii) the mutual dataset selectivity (which depends on the average extent/MBR of the geometries in the two datasets), and (iii) the average number of vertices of the geometries in the two datasets.

▶ **Definition 1** (Plane-sweep cost). Given two lists of geometries $n_i \subseteq D_i$ and $n_j \subseteq D_j$ coming from two input datasets D_i and D_j, whose geometries have an average number of vertices equal to v_i and v_j, respectively, and a selectivity $\sigma(A)$ computed w.r.t. a certain reference space A, the complexity of the plain-sweep phase can be formulated as:

$$ps(n_i, n_j, v_i, v_j, A) = n_i \log(n_i) + n_j \log(n_j) + (v_i + v_j) \log(v_i + v_j) \cdot n_i \cdot n_j \cdot \sigma(A) \tag{1}$$

where the first two components represent the sorting the two input lists, while the last component is due to the intersection test between pairs of geometries with intersecting MBRs. The selectivity $\sigma(A)$ can be estimated by applying the following formula, proposed in [10]:

$$\sigma(A) = \frac{1}{A} \cdot \left(area^{avg}_{mbr}(D_i) + area^{avg}_{mbr}(D_j) + len^{avg}_x(D_i) len^{avg}_y(D_j) + len^{avg}_x(D_j) len^{avg}_y(D_i)\right) \tag{2}$$

Eq. 2 requires that some estimates about the datasets content are available, in particular: the average area of the MBR of the geometries belonging to D_* ($area^{avg}_{mbr}(D_*)$ and the average length on the x axis and y axis of the same MBRs ($len^{avg}_x(D_*)$, $len^{avg}_y(D_*)$).

Introducing the necessary coefficients of proportionality for each operation, Eq. 1 provides an estimate of the cost of each mapper involved in the spatial join computation. Let us analyze the case of a sequential execution ("one task" case) and compare it with the three map-side spatial join algorithms provided by SpatialHadoop, DJNI, DJGI and DJRE. We can conclude that the benefits induced by the application of one of the MapReduce spatial join derive not only from the parallel execution of different portions of the whole job, but also as a consequence of the non linear behavior of the plane-sweep algorithm.

▶ **Observation 1** (Benefits of parallel execution of spatial join with DJNI). Given two datasets D_i and D_j in the reference space of area A, with cardinality N_i and N_j and an average number of vertices equal to V_i and V_j, respectively. The cost of a "one task" execution of the spatial join can be estimated using Eq. 1. Conversely, by applying algorithm DJNI having s_i

and s_j number of splits for D_i and D_j respectively, we obtain from Eq. 1 the estimate of the cost of each map task as follows, where a_1 e a_2 are the coefficients that are necessary for taking into account the cost of comparing two MBRs during the ordering phase and the cost of testing a geometry intersection during the last phase, respectively:

$$ps_{\text{DJNI}}\left(\frac{N_i}{s_i}, \frac{N_j}{s_j}, V_i, V_j, A\right) = a_1\frac{N_i}{s_i}\log\left(\frac{N_i}{s_i}\right) + a_1\frac{N_j}{s_j}\log\left(\frac{N_j}{s_j}\right) + \\ a_2(V_i + V_j)\log(V_i + V_j)\cdot\frac{N_i}{s_i}\cdot\frac{N_j}{s_j}\cdot\sigma(A)$$

Notice that $\sigma(A)$ does not change w.r.t the "one task" case, since the geometries in each split are randomly chosen, thus they cover the whole reference space. The cost of a map task is obviously reduced compared to the single process, in particular: (i) the ordering phases are reduced proportionally w.r.t. the input reduction with an additional cut of: $a_1N_i\log(s_i)$ (or $a_1N_j\log(s_j)$), (ii) the intersection testing phase is reduced by a significant factor: $s_i \times s_j$, since the number of pairs considered each map task is a subset of the total amount of geometries. The total cost of DJNI is $ps_{\text{DJNI}} \cdot (s_i \times s_j)$, where $s_i \times s_j$ represents the number of mappers produced by DJNI. This is a greater cost compared to the "one task" case since the ordering phase of a split of D_i is replicated s_j times. However, under the hypothesis that we can execute in parallel all the map tasks, we can obtain a significant reduction of the effective time. Indeed, in this case the effective time will coincide with the execution time of the worst map task or to the average execution time of a map in a balanced situation.

▶ **Observation 2** (Benefits of parallel execution of spatial join with DJGI and DJRE)**.** By applying the DJGI algorithm on datasets D_i and D_j, having both a grid index with a number of cells s_i and s_j, respectively, we can obtain the estimate of the cost of each map task by computing $ps_{\text{DJGI}}(\frac{N_i}{s_i}, \frac{N_j}{s_j}, V_i, V_j, A_{cell})$ from Eq. 1. Notice that in this case the selectivity changes, since the geometries of a split are now spatially located only in a subset of the reference space, i.e. the space occupied by an index cell, namely A_{cell} (here the smallest cell of the two indexes is considered). A similar consideration holds for DJRE, even if in this case only a grid index is present, suppose the one of D_i, so s_j becomes the number of cells of s_i that intersect D_j.

The cost of each map task is reduced also for DJGI and DJRE. In particular, while the cost of the ordering phases is reduced as for DJNI, the intersection testing phase is more expensive since the selectivity is lower. This is the effect of the index that tends to balance the work among the map tasks and to reduce their number. The total cost can be obtained for DJGI by multiplying ps_{DJGI} by the factor $s_i \times \sigma(s_j)$, where $\sigma(s_j)$ is the number of cells of the index of D_j that are intersected by a cell of D_i, and for DJRE by the factor $\rho(s_i)$, where $\rho(s_i)$ is the number of cells of D_i that intersects D_j. In both cases it is in average less than the cost of the "one task" execution.

By applying the formulas in Def. 1 and Obs. 1-2 to the example in Sect. 1.1, we obtain as expected a lower cost for DJNI, DJGI and DJRE w.r.t. the "one task" case. However, we can also observe that with one big geometry the index does not have a significant impact on the execution time: both DJGI and DJRE do not reduce the cost of join w.r.t DJNI.

The following section shows the results of some experiments that have been performed with the aim to test how the number of geometries, the number of vertices and the selectivity can affect the effectiveness of the index partitioning in increasing the performance of a spatial join in MapReduce. As we will see, these factors can contribute in different ways, and their effect is not completely represented by the size in bytes of each split.

Table 3 Metadata about the two datasets used for the experiments on MBR size. Notice that **# VertPerGeom** is the number of vertices describing each of the **# Geometries** in the dataset.

Dataset	Size	# Splits	# VertPerGeom	# Geometries	MBR ext
D_i	1152 (MB)	9	1,000	2,156	1e-8
D_j	1 (MB)	1	1,000	40	1 → 1e-4

2.4 Experimental Analysis of the Problem

The previous section discusses the fact that a partitioning technique based only on the size in bytes of the input dataset does not properly capture the complexity of a spatial join operation, which instead depends on three factors (number of geometries, number of vertices and selectivity) that can be independent from the input size. More specifically, given a split s with a predefined size in Megabytes, its content can be very different: it can contain many simple geometries with a restricted number of vertices, or it can contain few complex geometries described by a huge number of vertices, again such geometries can have a very different extent which does not depend on the number of vertices. While the number of geometries contained in a split directly depends on the average number of vertices used to describe a shape, the extent is an independent aspect. Therefore, this section presents two kinds of experiments both performed by keeping constant the size in bytes of the two input datasets: (i) the average extent (MBR) of the second dataset is progressively augmented, producing a decrease of the selectivity (more join pairs), (ii) the number of vertices of the second dataset is progressively augmented, producing also a decreasing in the number of geometries contained inside a split.

2.4.1 Variation on the MBR Size

The first set of experiments tries to study the effect of the average geometry MBR size (namely the selectivity) on the join performance. In particular, we consider two synthetic datasets D_i and D_j with uniform distribution, D_i contains a huge number of geometries with a small extent, and D_j contains few geometries with a big extent. During the experiments the extent of geometries in D_j has been varied from 1 to 1e-4 w.r.t. to the overall dataset extent, namely initially the geometries occupy all the reference space, and this occupation is progressively decreased till a ratio of 1e-4. Tab. 3 reports some metadata about the two datasets, such as the number of geometries and the number of vertices in each geometry.

We compute the spatial join between these datasets by considering the three algorithm variants presented in Sect. 2.3. Fig. 3 reports the results of such experiments. As you can notice, the time required to perform the spatial join depends linearly on the selectivity (i.e., the average MBR size of the geometries) as predicted by the formulas presented in Obs. 1. Moreover, the performances of the three algorithms are very similar to each other and this proves that a partitioning technique that takes care only of the size in bytes does not capture the real complexity of the dataset. Indeed, given the same input size, in this experiment the performances of the spatial join considerably worsen passing from a couple of minute to more than one hour. While the time remains acceptable in the first cases (till an MBR area of D_j equal to 1e-1), it becomes incredibly bad when the geometries of D_j occupy the whole reference space. In this last case, none of the available partitioning techniques are able to completely exploit the parallelism and the benefit of a MapReduce framework.

Coming back to the real-world case illustrated in Sect. 1.1, in Fig. 4 we consider a join between the dataset `tot_reg` and a polygon with an increasing MBR and a constant number

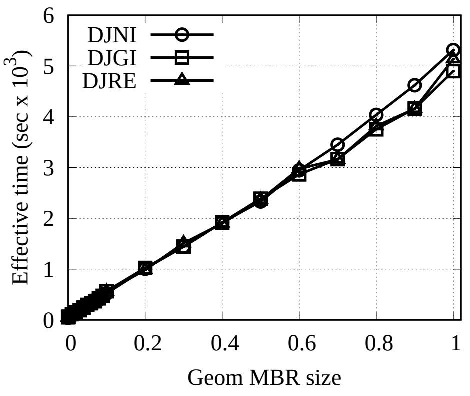

Figure 3 Effective time taken by the three considered spatial join algorithms by varying the MBR size of dataset D_j from an area of 1e-4 to 1 w.r.t. the area of the reference space.

`tot_reg` MBR	#Vert PerGeom	Join size	DJRE (sec)
5e-2	1,000	61,440	106
1e-1	1,000	104,934	145
1.5e-1	1,000	179,188	220
2.2e-1	1,000	262,311	329

Figure 4 Effective time taken by the DJRE algorithm applied to the real-world case by varying the MBR size of the dataset `tot_reg`.

Table 4 Metadata about the two datasets used for the experiments on the number of vertices. **# VertPerGeom** is the number of vertices describing each of the **# Geometries** in the dataset.

Dataset	Size	# Splits	# VertPerGeom	# Geometries	MBR ext
D_i	1152 (MB)	9	1,000	2,156	1e-8
D_j	1 → 75 (MB)	1	1,000 → 50,000	40	1e-2

of vertices (i.e., 1,000). In particular, the average MBR is changed from a radius of 15Km to a radius of 35Km, respectively. Again the time required for performing the join linerarly depends on the MBR size, namely the resulting selectivity.

2.4.2 Variation on the Number of Vertices

The second set of experiments evaluates the effect of the number of vertices on the complexity of the spatial join. In particular, we consider two datasets D_i and D_j with a fixed size in terms of occupied splits and containing geometries with a fixed extent. In order to check how the number vertices affects the time required for performing the spatial join, we vary the number of vertices of all geometries in D_j, while maintaining constant the number of occupied splits, the number of geometries and their average MBR area. Tab. 4 shows some metadata about the considered datasets. In particular, for D_j the number of vertices describing each geometry is varied from 1,000 to 5,000.

Fig. 5 presents the result of these experiments, again the use of the partitioning induced by the spatial index does not considerably increase the performance of the join: the difference between the three algorithms is no more than few minutes and in some cases the execution time is less for the join without index than for the other twos. In this case the number of map tasks to be executed is the same for all three algorithms: DJNI considers a number of combined splits equal to the Cartesian product, namely 1×9 splits, which is equal to the number of combined splits computed by DJGI, since the 1 split of D_j intersects all splits of D_i. Similarly, for DJRE the repartition does not discard any cell of D_i; moreover, the costs of performing a repartition phase is not compensated by its benefits, the number of repartitioned geometries is so very small. The trend of the curves in Fig. 5 completely adheres to the formulas presented in Obs. 1.

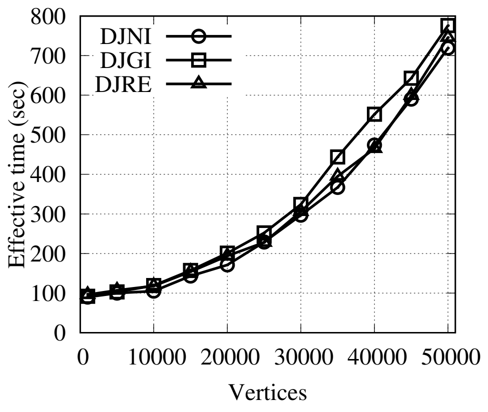

Figure 5 Effective time taken by the three considered spatial join algorithms by varying the number of vertices of each geometry in dataset D_j from 1,000 to 50,000.

`tot_reg` # vert	**Geom MBR**	**Join size**	DJRE (min)
1,000	1e-1	103,683	13
5,000	1e-1	103,704	60
10,000	1e-1	103,707	118
15,000	1e-1	103,710	174
20,000	1e-1	103,707	231
25,000	1e-1	103,708	288
30,000	1e-1	103,707	342
35,000	1e-1	103,707	408
40,000	1e-1	103,707	463
45,000	1e-1	103,707	519
50,000	1e-1	103,707	578

Figure 6 Effective time taken by the DJRE algorithms by varying the number of vertices of each geometry in dataset `tot_reg` from 1,000 to 50,000.

Referring to the real-wold case introduced in Sect. 1.1, we evaluate the performance of DJRE by varying the number of vertices in dataset `tot_reg` while maintaining constant the characteristics of dataset `cv_land`. The results are reported in Fig. 6, again the time by the spatial join increases with the number of vertices in each geometry.

3 Proposed Solution and Discussion

Considering the experimental results presented in the previous section, we can conclude that the effective execution time of three spatial join algorithms, DJNI, DJGI and DJRE, are affected by both the selectivity of the datasets, which directly depends on the MBR area of the geometries they contain, and the average number of vertices of the same geometries. Indeed, in the experiments the size in bytes of the input file remained unchanged, while the selectivity and the number of vertices are varied, obtaining very different execution times. However, the partitioning techniques provided by SpatialHadoop are only based on the size in bytes of the input files, thus they cannot react to the variations of these parameters.

In order to avoid this problem we propose an alternative partitioning technique to be applied during the existing index building phase. Given the grid to be used for grouping the geometries of a dataset, the existing indexing phase scans the whole dataset and for each geometry g detects the subset of cells $S(g)$ that it intersects, then g will be inserted in the split of each cell in $S(g)$. This means that sometimes a geometry g can be replicated in more than one split. If g is relatively small w.r.t. the index cells, then the replication is not frequent, but when geometries are bigger, the repetition occurs more frequently. The replication rate has not been considered in the cost estimation (see Eq. 1), since it can be neglected in the considered experiments.

The proposed technique enriches the indexing phase with a splitting operation which should affect the partitioning result in particular when the MBR area increases or the number of vertices increases. In the first case, i.e. lower selectivity, we can split the geometries that cross two or more index cells, so that the average area of their MBR is reduced and thus their contribution to the join selectivity is reduced (see Eq. 2). Notice that in this case the number of tested geometries does not change (in the original approach the whole geometry will be replicated in all combined splits), so the cost of a map task is reduced according to Eq. 1. In the second case, i.e. higher number of vertices, we can split the geometries when the number of their vertices exceed a threshold. In this way, a big geometry g can be substituted by a set of smaller geometries $\{g_1, ..., g_n\}$, that represent a partition of g and have a number of vertices smaller than the number of vertices of g.

In order to combine the two cases, we consider a new partitioning technique where the dataset with the bigger (in terms of average area of their MBR) geometries are splitted by considering the grid of the other dataset. This *splitting phase* reduces both parameters discussed above, thus reducing the cost of the map tasks. The following proposition shows the effective cost reduction that the splitting phase introduces.

▶ **Observation 3** (Benefits of the splitting phase)**.** Consider two datasets D_i and D_j in a reference space of area A, with cardinality N_i and N_j and an average number of vertices equal to V_i and V_j, respectively, and such that D_j is the dataset having the bigger geometries in terms of occupied area. The cost of the "one task" execution of the spatial join can be estimated by Eq. 1. Conversely, if we consider the application of DJGI on D_i and D_j in presence of grid indexes having respectively a number of cells s_i and s_j, and assuming that the geometries in D_j have been splitted so that they are spatially contained in one cell of D_i, the cost of each map task can be estimated from Eq. 1 as follows:

$$ps_{\text{DJGI}}\left(\frac{N_i}{s_i}, \frac{\alpha N_j}{s_j}, V_i, \frac{V_j}{\alpha}, A_{cell}\right) = a_1\frac{N_i}{s_i}\log\left(\frac{N_i}{s_i}\right) + a_1\frac{\alpha N_j}{s_j}\log\left(\frac{\alpha N_j}{s_j}\right) + a_2\left(V_i + \frac{V_j}{\alpha}\right)\log\left(V_i + \frac{V_j}{\alpha}\right)\cdot\frac{N_i}{s_i}\cdot\frac{\alpha N_j}{s_j}\cdot\sigma(A_{cell}) \quad (3)$$

where α represents the average number of cells of D_i that are intersected by a geometry of D_j, namely the average number of small geometries obtained from each big geometry in D_j after the splitting phase. Accordingly to [1], it can be estimated as follows: $\alpha = \lceil len_x^{avg}(D_j)/len_x^{cel}(D_i)\rceil \cdot \lceil len_y^{avg}(D_j)/len_y^{cel}(D_i)\rceil + \beta$, where, considering the MBR of the geometries belonging to D_j, $len_x^{avg}(D_j)$ ($len_x^{avg}(D_j)$) is the average length on the x (y) axis of these MBRs, while $len_x^{cel}(D_i)$ ($len_y^{cel}(D_i)$) represents the average length on the x (y) axis of the index cells of D_i. β is an additional factor taking into account the displacement between MBRs and cells, namely it is a function of the probability that the MBR of a geometry of D_j crosses the boundaries of the cells of D_i.

Notice that, as shown in Obs. 1, the cost of a map task is obviously reduced compared to the "one task" case, in particular: (i) the ordering phases are reduced proportionally w.r.t. the input reduction with an additional cut quantifiable in: $a_1 N_i \log(s_i)$ for D_i (or $a_1\alpha N_j(\log(\alpha) - \log(s_j))$ for D_j), (ii) the intersection testing phase is also reduced in two ways: by the reduction of the pairs of geometries to be considered, with a factor $(\alpha \cdot \sigma(D_j)/s_j)$, and also by the reduction of the cost for testing the intersection between two geometries, since the number of vertices is decreased by a factor α.

4 Validation of the Solution

This section presents some additional experiments that verify the theoretical behavior of the algorithms when the new splitting technique described in Sect. 3 is applied. In particular, we first consider the experiments related to the variation of the average MBR size (i.e., selectivity) and check the effect of splitting the geometries of D_j using the grid of the constant dataset D_i of size 9 splits. These results are reported in Tab. 5 where the first three columns

Table 5 Comparison between the execution time of the three distributed join algorithms when performed considering the original and the modified synthetic datasets with a variable MBR size.

D_j **Avg MBR size**			DJNI		DJGI		DJRE	
Orig	Part	% Decr.	(sec)	% Improv.	(sec)	% Improv.	(sec)	% Improv.
1e-1	2.27e-2	77.25%	491	5.66%	527	7.17%	527	5.66%
2e-1	3.08e-2	84.58%	941	6.71%	934	8.72%	945	3.94%
3e-1	3.66e-2	87.79%	1,223	15.20%	1,285	10.95%	1,280	15.31%
4e-1	4.35e-2	89.12%	1,752	8.58%	1,633	14.91%	1,705	10.33%
5e-1	4.40e-2	91.19%	2,008	13.86%	1,814	23.99%	1,747	26.17%
6e-1	4.93e-2	91.79%	2,553	13.35%	2,586	9.81%	2,259	24.11%
7e-1	5.75e-2	91.78%	3,009	12.78%	2,650	16.28%	2,665	15.67%
8e-1	6.58e-2	91.77%	3,573	11.49%	3,570	4.86%	3,311	13.21%
9e-1	7.41e-2	91.76%	4,352	5.83%	4,021	3.37%	3,629	12.71%
1e+0	8.21e-2	91.79%	4,558	14.22%	4,473	8.73%	4,746	7.60%
Average				10.77%		10.88%		13.47%

Table 6 Comparison between the execution times of the three distributed join algorithms when performed on the original and the modified synthetic datasets with a variable number of vertices.

D_j # **VertPerGeom**			DJRE	
Original	Splitted	% Reduction	(sec)	% Improvement
1,000	149	85%	303	5%
10,000	1,545	84%	397	20%
20,000	2,924	85%	753	57%
30,000	4,512	85%	1,239	72%
40,000	6,020	85%	2,123	78%
50,000	8,272	83%	3,194	84%
Average		85%		52%

contain: (a) the original MBR size, (b) the MBR size after the splitting and (c) the average percentage of decrease in the MBR size. The area of the used grid cells is 8.21e-2, while the average percentage of decrease in the MBR size increases as the average MBR area of the original geometries increases. For each algorithm the table reports the execution time on the splitted geometries and the percentage of improvements w.r.t. the original situation. All three versions of the distributed join benefit from the partitioning with an average improvement of around 10-13% with respect to the previous executions.

As second set of validation experiments we consider the case in which the geometry MBR remains unchanged but we vary the number of vertices describing each geometry. In particular, we consider only the case of DJRE since the execution time of the various algorithms are not much different from each other and DJRE is on average the most efficient one. These results are reported in Tab. 6 where the first three columns contain: (a) the original number of vertices in each geometry, (b) the number of vertices after the splitting (c) the average percentage of decrease in the number of vertices. The other two columns contain the execution time of DJRE on the splitted geometries and the percentage of improvement w.r.t. the original situation. The results of these experiments confirms what verified in Sect. 2.4, namely the not negligible effect of the number of vertices on the spatial join execution time. Indeed, this time greatly decreased by decreasing the complexity of the geometries in terms of the average number of vertices in each geometry.

5 Related Work

A common strategy to reduce the cost of a spatial join is the filter-and-refine approach which consists on a filter phase that traditionally works on the MBR (minimum bounding rectangle) of the involved geometries, and a refining step which performs the actual test on the filtered pairs. The filter phase can usually benefit from the use of a spatial index while the identification of both the overlapping MBRs or geometries is performed using the plane-sweep algorithm [10]. In [17] the authors analysed the problem of how to partition spatial data in order to perform parallel spatial join. They promoted the use of spatial locality in task decomposition in order to speed-up the join computation. This partitioning reflects the way data is partitioned by SpatialHadoop during the construction of a global index. However, it is not effective in the case considered in this paper, since we assume the presence of some big and complex geometries which occupy the whole reference space. In the context of parallel spatial join execution, some research has been done in order to define partitioning techniques which produce balanced partitions even in presence of skewed data [4, 9, 11]. This paper does not consider the effect of the data distribution (skewed or uniform), but concentrates on the presence of big and complex geometries that do not allow to completely exploit the parallelism induced by the MapReduce approach.

In [5] the author analysed the various partitioning techniques available in SpatialHadoop and they experimentally studied the effect of such indexes on some operations, such as the range query and the spatial join. The work mainly evaluates such partitioning techniques based on four quality measures, but it assumed that the considered objects occupy a small space w.r.t. the reference space, so it did not consider the problem treated in this paper. The problem of processing big complex geometries together with small ones has been investigated for the first time in [12], where the author detected a difference in the performance of some Pigeon operations when performed on spatially equivalent datasets with different configurations for what regards the extent and complexity of the involved geometries. Pigeon [6] is an extension of Pig Latin for dealing with spatial data in SpatialHadoop.

6 Conclusion

This paper deals with the problem of identifying the characteristics that really represents the complexity of spatial data, making them “big” w.r.t. the most common operations. These characteristics have to play a central role in the definition of an effective partitioning technique able to exploits all potentiality of a MapReduce environment, like Hadoop. Traditionally, in such environments the partitioning of data is performed by subdividing the records in the original datasets so that each obtained split has an upper bound size given in terms of the number of occupied bytes. This kind of partitioning is used also in spatial-aware MapReduce systems, like SpatialHadoop, where the data partitioning, even the one induced by the construction of spatial indexes, is driven only by the data size in bytes. However, spatial data is characterized by other kinds of dimensions, such as the number of vertices (complexity) used to described a single geometry, or the average area of the MBR of the geometries (extent). These characteristics usually affect the cost of spatial analysis operations, such as the spatial join. Therefore, we can assume that what makes spatial data big is not only their size in bytes, but also their complexity and their extent. In order to validate such hypothesis, in this paper we analyse the behaviour of some distributed spatial join algorithms provided by SpatialHadoop when varying the average MBR size and the number of vertices, showing how such characteristics affect the performance of the spatial join and that they are not correctly captured by a partitioning technique based only on the size in bytes of the input datasets. We propose the idea of a new partitioning technique which takes care of such characteristics by also performing a splitting of the original geometries in order to reduce their complexity and better exploit the parallelism induced by a MapReduce environment. Further improvements will regard the identification of the grid which is more appropriate on the base of the average MBR size of geometries and the average number of vertices.

References

1 A. Belussi, S. Migliorini, and A. Eldawy. A Cost Model for Spatial Join Operations in SpatialHadoop. Technical Report RR108/2018, Dept. of Computer Science, University of Verona, 2018. URL: `https://iris.univr.it/handle/11562/981957`.

2 Alberto Belussi, Sara Migliorini, Mauro Negri, and Giuseppe Pelagatti. Validation of spatial integrity constraints in city models. In *4th ACM SIGSPATIAL Int. Workshop on Mobile Geographic Information Systems*, pages 70–79, 2015. `doi:10.1145/2834126.2834137`.

3 Matteo Dell'Amico, Damiano Carra, and Pietro Michiardi. PSBS: Practical size-based scheduling. *IEEE Transactions on Computers*, 65(7):2199–2212, 2016.

4 D. J. DeWitt, J. F. Naughton, D. A. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *18th Int. Conf. on Very Large Data Bases*, pages 27–40, 1992.

5 A. Eldawy, L. Alarabi, and M. F. Mokbel. Spatial partitioning techniques in SpatialHadoop. *Proc. VLDB Endow.*, 8(12):1602–1605, 2015.

6 A. Eldawy and M. F. Mokbel. Pigeon: A spatial MapReduce language. In *IEEE 30th Int. Conf. on Data Engineering*, pages 1242–1245, 2014. `doi:10.1109/ICDE.2014.6816751`.

7 A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce framework for spatial data. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1352–1363, 2015.

8 A. Eldawy and M. F. Mokbel. *Spatial Join with Hadoop*, pages 2032–2036. Springer International Publishing, Cham, 2017. `doi:10.1007/978-3-319-17885-1_1570`.

9 K. A. Hua and C. Lee. Handling data skew in multiprocessor database computers using partition tuning. In *17th Int. Conf. on Very Large Data Bases*, pages 525–535, 1991.

10 Edwin H. Jacox and Hanan Samet. Spatial Join Techniques. *ACM Trans. Database Syst.*, 32(1), 2007. `doi:10.1145/1206049.1206056`.

11 Masaru Kitsuregawa and Yasushi Ogawa. Bucket spreading parallel hash: A new, robust, parallel hash join method for data skew in the super database computer (SDC). In *16th Int. Conf. on Very Large Data Bases*, pages 210–221, 1990.

12 S. Migliorini, A. Belussi, M. Negri, and G. Pelagatti. Towards massive spatial data validation with spatialhadoop. In *5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, pages 18–27, 2016. `doi:10.1145/3006386.3006392`.

13 Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. Access-time-aware cache algorithms. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(4):21, 2017.

14 Mario Pastorelli, Damiano Carra, Matteo Dell'Amico, and Pietro Michiardi. HFSP: bringing size-based scheduling to hadoop. *IEEE Trans. on Cloud Computing*, 5(1):43–56, 2017.

15 Tom White. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 4th edition, 2015.

16 Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache Spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016. `doi:10.1145/2934664`.

17 Xiaofang Zhou, David J. Abel, and David Truffet. Data partitioning for parallel spatial join processing. *GeoInformatica*, 2(2):175–204, 1998.