Dipartimento di Informatica
Università degli Studi di Verona

Rapporto di ricerca
Research report
May 2018

RR 108/2018

A cost model for spatial join operations in SpatialHadoop

Alberto Belussi
Sara Migliorini
Ahmed Eldawy

Questo rapporto è disponibile su Web all'indirizzo:
This report is available on the web at the address:
http://www.di.univr.it/report

Abstract

Spatial join is an important operation in geo-spatial applications, since it is frequently used for performing data analysis involving geographical information. Many efforts have been done in the past decades in order to provide efficient algorithms for spatial join and this is particularly important as the amount of spatial data to be processed increases. In recent years, the MapReduce approach has become a de-facto standard for processing large amount of data (big-data) and some attempts has been made for extending existing frameworks for the processing of spatial data. In this context, SpatialHadoop [1] is an extension of Apache Hadoop [2] which includes a native support for spatial data, in terms of spatial data types, operations and indexes. In particular, its provides five different variants of spatial join which mainly differ in the use of a spatial index and in the way this index is built and used. In general, none of these algorithm can be considered better than the others, but the choice might depend on the characteristics of the involved datasets. The aim of this work is to deeply analyse the characteristics of these algorithms and to define a cost model for them which is based on some dataset characteristics (i.e., selectivity or spatial properties). The main goal of the proposed cost model is to rank the spatial join implementations by defining a partial order among them using a dominance relation. This cost model has been extensively tested w.r.t. a set of synthetic datasets in order to prove its effectiveness.

Keywords: Spatial join, cost model, map-reduce, SpatialHadoop

1 Introduction

In the last few years a large amount of efforts has been devoted by researchers to provide a MapReduce implementation of several operations that are usually required for performing big data analysis. In particular, the join operation has attracted much attention since it is frequently used in data processing, for instance a join is necessary for linking log data to user records.

This effort has produced a set of different MapReduce implementations of the join operations [3, 4], each one applicable to a particular situation. Therefore, there was a set of follow up works that study some sort of heuristics that allow the system to decide which implementation to apply, given some parameters that characterize the specific case. These studies produce some proposals of cost model for MapReduce implementations of the join operator [3, 4].

This paper considers a particular kind of join, called *spatial join*, namely a multi-dimensional join specifically defined for spatial data [5]. In particular, it proposes a new cost model for spatial join that can be used to model five different variants of spatial join available in SpatialHadoop [1]: 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). For each of these algorithms, we provide in Sect. 6 a cost model that takes into account three data metrics, namely size of the geometries, geometry selectivity, and degree of spatial overlapping.

The cost model produces three separate costs for the three phases of a MapReduce job – map, shuffle and reduce, and for each phase, it produces three different metrics – CPU processing, local disk I/O and network I/O. In this way, the cost model provides a 9×9 cost matrix that characterizes each MapReduce implementation. In addition the cost model gives an estimate of the required number of map and reduce tasks (see Sect. 3.3).

To our knowledge, previous work only deals with the cost estimation for implementations in MapReduce of the traditional join operation [3, 4] and do not consider spatial or multi-dimensional datasets. Additional details about related work will be discussed in Sect. 2.

For each one of the five variants of spatial join considered in this paper, we propose two different scenarios: in the first one we suppose that they work on indexed input datasets, and in the second one on datasets without indices. On the data side, we consider different input sets of geometries with increasing cardinality and containing polygons with different characteristics w.r.t. the size and selectivity of the geometries (in terms of average MBR area), and the degree of spatial overlapping of the datasets. Thus, all these

features will be considered as parameters of the cost model (see Sect. 3.1). The main goal of the cost model will be to rank the spatial join implementations according to the matrix of cost estimates, and define a partial order among them introducing a dominance relation.

The experiments in Sect. 7 confirmed the effectiveness of the proposed cost model in ranking the alternative solutions for computing the spatial join in SpatialHadoop. Indeed, in average the cost model chooses the best algorithm in the 89% of the cases.

2 Related Work

Spatial join algorithms. Many algorithms have been defined in literature with the aim to efficiently perform a spatial join between two datasets, considering the cases in which none, one or both inputs have been previously indexed. A comprehensive survey about all these variants can be found in [6]. At the same time, to coupe with all these variants, several benchmarking studies have been performed to evaluate their performance on the basis of the given input [7–10]. Even if these studies cannot be directly applied to a MapReduce context, they represent a starting point for the definition of the cost model presented in this paper.

Cost model for map-reduce join algorithms. The MapReduce framework has become a popular execution environment for the analysis of large amount of textual data. Among all possible processing activities, the join between two inputs is one operation that require particular attention and has been studied. In [3] the authors performs an analysis of a number of well-known join strategies in MapReduce and provide an experimental comparison between them. They also explored how the join algorithms can benefit from certain types of preprocessing techniques, such as a repartition phase. Another comparison between join algorithms in Hadoop can be found in [4]. The authors identified the various parts of a join operation and further subdivide them into mappers, shufflers and reducers. An attempt to define an accurate performance model for a generic MapReduce operation has been done in [11]. The authors analyzed the composition of MapReduce tasks and relationships among them, they decomposed the major cost items, and presented a vector style cost model which inspired the cost model presented in this paper.

Cost model for map-reduce spatial-join algorithms. A first attempt to define a cost model for spatial-join algorithms in MapReduce can be found in [12] where the authors proposed a cost-based and a rule-based optimizer for a generic spatial join. In particular, the authors abstracted from any specific implementation and considered a generic spatial join composed of two

phases: the partitioning (performed by mappers) and the joining (performed by reducers). The work mostly evaluates the convenience to perform a preliminary partitioning on only one or both datasets. Conversely, by concentrating on the specific SpatialHadoop implementations, the model proposed in this paper is able to provide a more precise and detailed cost analysis, also evaluating the advantages of performing a map-side or a reduce-side join in various situations. Moreover, it also based on some dataset metadata, such as the selectivity, the dataset cardinalities, and the number of vertices, in place of the cluster characteristics.

3 A General Cost Model Framework

This section lays the basis for the definition of a cost model for the various MapReduce spatial join operators mentioned in Sect. 1. In particular, it starts by defining a set of parameters that characterize the execution environment and the input datasets, and then it provides a general notion of cost for a given operator.

3.1 Characterization of the MapReduce Environment

The cost model proposed in this paper has been defined by considering some parameters that characterize the environment in which the spatial join operator is executed. This set of parameters will be called *Hadoop configuration* and is defined as follows.

Definition 1 (Hadoop configuration)**.** *A Hadoop execution environment is characterized by:*

- *#nodes: number of nodes in the cluster.*
- *#containers: number of execution containers in the cluster. The number of containers is typically equal to the number of cores.*
- *#parMaps: maximum number of mappers that can be executed in parallel. This number can be at most equal to the number of containers.*
- *#parReds: maximum number of reducers that can be executed in parallel. We can safely assume that all reducers can be executed in parallel, namely only one reduce step will be performed in the cluster.* [1]

[1] From the official Hadoop documentation, the maximum number of parallel reducers could be set equal to the number of available containers multiplied by a factor of 0.95.

- *splitSz: default size of a split provided to a mapper. It usually corresponds to the block size in the Hadoop distributed file system (HDFS) which can be for instance 128 MBytes,*
- *#rep: number of file replications (3 by default).*

Moreover, the cost model requires some additional statistical parameters concerning the input datasets which will be called *dataset statistics* and are defined as follows.

Definition 2 (Dataset statistics)**.** *Given a dataset D_*, the following parameters regarding the dataset content can be defined (the abbreviation MBR is used to denote the Minimum Bounding Rectangle of a geometry):*

- *size(D_*): size of the dataset D_* in bytes,*
- *#geo(D_*): number of geometries in the dataset D_*,*
- *mbr(D_*): MBR covering all geometries in D_*,*
- *$mbrArea^{avg}(D_*)$: given the MBR of all geometries in D_*, it represents the average area of such MBRs,*
- *$len_x^{avg}(D_*)$ and $len_y^{avg}(D_*)$: given the MBR of all geometries in D_*, they represent the average length on the X and Y axis of such MBRs,*
- *$\#vert^{avg}(D_*)$: average number of vertices of the geometries in D_*,*

Parameters $size(D_*)$ and $\#geo(D_*)$ can be obtained by querying the HDFS. An estimate for parameters $mbr(D_*)$, $len_x^{avg}(D_*)$ and $len_y^{avg}(D_*)$ can be obtained by sampling the input datasets, or we can suppose that they were computed during the scan of the geometries in a previous access, and that the system collects these statistics for refining the quality of the cost model predictions.

For the datasets that has an indexed structure, the following additional parameters are assumed to be known. Notice that this paper concentrates for simplicity on grid-based indexes [13, 14], however the extension to other kind of indexes (e.g., R-Trees) is straightforward.

- $\#cells(\mathcal{I}_*)$: number of cells in a grid index $\mathcal{I}_*$. Sometimes the abbreviation $\#cells(D_*)$ is used for denoting the number of index cells for the dataset D_*.
- $len_x^{cel}(\mathcal{I}_*)$ and $len_y^{cel}(\mathcal{I}_*)$: length on X and Y axis of the cells in the grid index $\mathcal{I}_*$. Sometimes the parameter D_* can be used in place of $\mathcal{I}_*$ to denote its index.

For obtaining more accurate estimations, we define the following two data-related metrics regarding the mutual selectivity between two dataset D_i and D_j.

- $\sigma(A)$: selectivity of the spatial join between two datasets D_i and D_j w.r.t. a reference space A. The selectivity of the spatial join is a real number between 0 and 1 representing the probability that given a pair of geometries (g_i, g_j), such that $g_i \in D_i$ and $g_j \in D_j$, this pair belong to the spatial join result.

- $\sigma^{mbr}(A)$: selectivity of the spatial join between the MBR of the geometries in the datasets D_i and D_j. This selectivity is similar to the previous one, but here the MBRs are considered instead of the real geometries.

These metrics are assumed to be known, indeed they can be estimated in some way as explained in Sect. 3.2 or in some contexts system administrators can provide an educated guess based on their experience with the data.

Finally, some additional parameters characterize the size in bytes for storing a vertex of a geometry, an MBR and a record of a dataset: (i) *vertSz* denotes the number of bytes that are needed for the representation of a single vertex; (ii) *mbrSz* indicates the number of bytes required to represent a generic MBR and $recSz(D_*)$ denotes the bytes needed to store a record of the dataset D_*. More specifically, $mbrSz = 4 \cdot vertSz$ and $recSz(D*) = \#vert^{avg}(D_*) \cdot vertSz$.

3.2 Selectivity Estimation

The parameters $\sigma(A)$ and $\sigma^{mbr}(A)$ represent the basis for the proposed cost functions, some formulas for their estimation in specific cases have been proposed in [15, 16]. Some possible estimations can be obtained using different levels of information about the input datasets. In particular, without any knowledge about the dataset characteristics, we can only assume that each geometry might intersect any other geometry in the other dataset. Conversely, by knowing some statistics about the two datasets and assuming a uniform distribution for them, we can obtain a more precise estimation by generalizing the formula proposed in [16] as discussed below.

Definition 3. *Given two datasets D_i and D_j, the selectivity $\sigma(A)$ between them w.r.t. to a reference space A can be obtained by generalizing the formula in [16] as follows:*

$$\sigma(A) \simeq \frac{1}{A} \cdot \Big(mbrArea^{avg}(D_i) + mbrArea^{avg}(D_j) + (len_x^{avg}(D_i) \cdot len_y^{avg}(D_j) + len_x^{avg}(D_j) \cdot len_y^{avg}(D_i)) \Big) \quad (1)$$

Proof. Let us consider two random rectangles r_i and r_j representing the MBR of a geometry in D_i and D_j, respectively. If A is the area of the entire given spatial context, i.e. $A = area(mbr(D_i) \cap mbr(D_j))$, the probability p that the two rectangles intersect is given by:

$$p(r_i, r_j) = \frac{(len_x(r_i) + len_x(r_j))(len_y(r_i) + len_y(r_j))}{A}$$

Fig. 1 illustrates the rationale of the formula: the two rectangles r_i and r_j intersect if and only if, choosing the upper-left corner of r_1 (point p), it is contained inside the rectangle whose sides are $len_x(r_i) + len_x(r_j)$ and $len_y(r_i) + len_y(r_j)$.

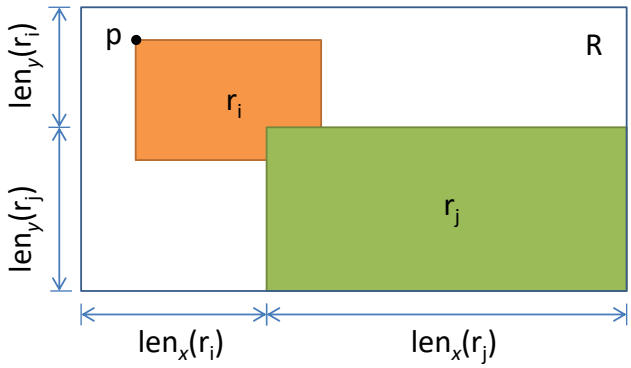

Figure 1: The two rectangles r_i and r_j intersects if and only if the upper-left point p is contained in the rectangle R.

The selectivity σ between two datasets D_i and D_j can be obtained by generalizing the previous formula and averaging the probability computed for all pairs of objects[2].

$$\sigma(A) = \frac{1}{\#geo(D_i) \cdot \#geo(D_j)} \cdot \sum_{a \in D_i} \sum_{b \in D_j} \frac{(len_x(a) + len_x(b))(len_y(a) + len_y(b))}{A}$$

[2]Notice that the formula assumes that D_i and D_j occupy the same reference space, namely $mbr(D_i) = mbr(D_j)$.

$$\frac{1}{A}\left(\sum_{a\in D_i}\#geo(D_j)\cdot\frac{(len_x(a)len_y(a))}{\#geo(D_i)\cdot\#geo(D_j)}+\right.$$
$$\sum_{b\in D_j}\#geo(D_i)\cdot\frac{(len_x(b)len_y(b))}{\#geo(D_i)\cdot\#geo(D_j)}+$$
$$\left.\sum_{a\in D_i}\sum_{b\in D_j}\frac{(len_x(a)len_y(b)+len_x(b)len_y(a))}{\#geo(D_i)\cdot\#geo(D_j)}\right)$$
$$\frac{1}{A}\left(\overbrace{\sum_{a\in D_i}\frac{(len_x(a)len_y(a))}{\#geo(D_i)}}^{coverage(D_i)}+\overbrace{\sum_{b\in D_j}\frac{(len_x(b)len_y(b))}{\#geo(D_j)}}^{coverage(D_j)}+\right.$$
$$\left.\sum_{a\in D_i}\sum_{b\in D_j}\frac{(len_x(a)len_y(b)+len_x(b)len_y(a))}{\#geo(D_i)\cdot\#geo(D_j)}\right) \tag{2}$$

Notice that the first expanded term depends only on D_i, so the second summation produces the number of geometries in D_j, and this similarly holds for the second term. The term $1/A \cdot \sum_{g\in D_*}(len_x(g)\cdot len_y(g))$ represents the ratio between the sum of the areas of each data item in D_* w.r.t. A. It is called $coverage(D_*)$ and can be estimated as: $coverage(D_*) \simeq mbrArea^{avg}(D_*)\cdot\#geo(D_*)/A$, while the summation can be simplified by considering the average length of MBR sides. Therefore, the formula can be simplified as follows:

$$\sigma(A) \simeq \frac{1}{A}\cdot\Big(mbrArea^{avg}(D_i)+mbrArea^{avg}(D_j)+ \tag{3}$$
$$(len_x^{avg}(D_i)\cdot len_y^{avg}(D_j)+len_x^{avg}(D_j)\cdot len_y^{avg}(D_i))\Big)$$

□

In the experiments, the cost model has been applied by considering the selectivity estimation produced by this formula. Notice that the original formula has been defined for rectangles while the proposed cost model is intended for any kind of geometry. Therefore, the use of σ and σ^{mbr} can produce an overestimation of the real selectivity.

3.3 Cost of a MapReduce Operator

This section provides a general definition for the cost of a MapReduce operator *op*. The cost of each operator is divided in three components: (i) the

cost of the mappers; (ii) the cost of the shufflers and (iii) the cost of the reducers. The shuffle phase is the process through which data is sorted by key and transferred from the mappers to the reducers. Clearly, this phase is performed only if there are some reducers in the considered job.

The cost of each phase is further subdivided in three parts: CPU, local I/O and network I/O. The measurement of these cost components is based on a set of hypothesis:

- The unit of measure for the CPU cost is the time μ required to compares the x (or y) component of two coordinates, namely to compare two double values: $\mu = \text{time}(\leq (d_1, d_2))$. From this, the time required to test the intersection between two MBRs can be defined as $4 \cdot \mu$, since it corresponds to the comparison of 4 double values.

- The measure of a disk I/O or network I/O operation is given by the number of bytes read or written.

Given such hypothesis, the cost of a MapReduce operator can be defined as follows:

Definition 4 (Cost of an operator)**.** *Given a MapReduce operator op and a Hadoop configuration, the cost of op can be defined by the following tuple:*

$$\mathcal{C}(op) = \langle \#map_{op}, \#red_{op}, \mathcal{M}_{op} \rangle \tag{4}$$

where $\#map_{op}$ *(*$\#red_{op}$ *) is an estimate of the number of mappers (reducers) and* $\mathcal{M}_{op}$ *is a matrix describing the different cost components (cpu, disk, net) for the different phases (*`M`*: map,* `S`*: shuffle,* `R`*: reduce) of a job and has the following structure:*

$$\mathcal{M}_{op} = \begin{bmatrix} M^{op}_{cpu} & S^{op}_{cpu} & R^{op}_{cpu} \\ M^{op}_{disk} & S^{op}_{disk} & R^{op}_{disk} \\ M^{op}_{net} & S^{op}_{net} & R^{op}_{net} \end{bmatrix}$$

Each elements of $\mathcal{M}_{op}$ *refers to the cost of an single mapper, reducer or shuffler.*

The matrix can be used to obtain the estimated total cost or the estimated effective/parallel cost of a job.

Definition 5 (Estimated total and effective cost)**.** *Given a MapReduce implementation op of an operator, an Hadoop configuration and the tuple* $\mathcal{C}(op)$

defining its cost. The estimated total cost of op can be obtained by multiplying $\mathcal{M}_{op}$ by a vector containing the estimation of the number of mappers and reducers as below:

$$cv_{tot} = \mathcal{M}_{op} \times \begin{bmatrix} \#map_{op} \\ \#red_{op} \\ \#red_{op} \end{bmatrix} = \begin{bmatrix} cpu_{tot} \\ disk_{tot} \\ net_{tot} \end{bmatrix}$$

In a similar way, the estimated effective cost of op can be obtained by multiplying $\mathcal{M}$ by a vector containing the estimation of the number of map and reduce runs:

$$cv_{par} = \mathcal{M}_{op} \times \begin{bmatrix} \#mapRuns \\ \#redRuns \\ \#redRuns \end{bmatrix} = \begin{bmatrix} cpu_{par} \\ disk_{par} \\ net_{par} \end{bmatrix}$$

Notice that $\#mapRuns$ has a lower bound equal to $\lceil \#map_{op}/\#parMaps \rceil$, where $\#parMaps$ denotes the maximum number of mappers that can be executed in parallel using the current Hadoop configuration, while $\#redRuns$ is bound by $\lceil \#red_{op}/\#parReds \rceil$, which usually produce only one phase for the reducers.

Sect. 5-6 present the cost model for each spatial join operator *op* available in SpatialHadoop. The cost model focuses on the estimation of $\mathcal{C}(op)$, from which we can obtain both cv_{par} and cv_{tot}.

4 Processing two Inputs in MapReduce

The join is an operation that requires particular attention when performed in MapReduce, since it needs to process two datasets (files) at time, while Hadoop traditionally processes only one argument. This section describes the details of the reader used by SpatialHadoop to process two input files at time generating compound splits.

Many works are available in literature that discuss how to perform a generic join in map-reduce. Solutions in [2,3] use a strategy that combines the two inputs into a unique file by keeping a reference to the source file. A similar strategy is applied only by the SJMR algorithm in Section 6.5. Conversely, the algorithms in Sections 6.1-6.3 are based on the definition of an input format and a corresponding software module, called reader, which is able to scan two files at time, with the aim to produce a set of records each one containing a pair of geometries coming from the two input datasets. In particular, the `DjInputFormatArray` extends the class `BinarySpatialInputFormat` which is able to read a pair of files simultaneously. The produced records have as

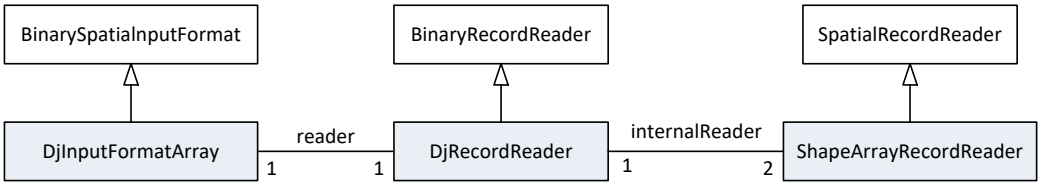

Figure 2: Classes which implements the input format and the input reader for the distributed join algorithms.

key the pair $\langle k_1, k_2 \rangle$, built from the input keys, and as value the pair $\langle v_1, v_2 \rangle$ where each component has type `ArrayWritable`, namely an array of shapes. In particular, the `DjRecordReader`, provided by the `DjInputFormatArray` class, makes a Cartesian product of the values produced by two internal record readers of type `ShapeArrayRecordReader`. This last reader reads all shapes in a split and produces a record whose value is an array containing all shapes in the block and whose key can be either a rectangle indicating the MBR of the index cell, if the input file is indexed, or an invalid rectangle, in case of non-indexed files. Fig. 2 illustrates the main relations between these classes. Algorithms in Sect. 6.1-6.3 can use different filters in order to refine the Cartesian product before the map tasks are instantiated.

The use of this reader induces another issue to solve which regards the fact that it is not guaranteed that both splits, composing the compound split and containing the geometries to be joined, reside in the same node. In order to minimize the network I/O cost, the reader tries to put in the same compound split data residing on the same node; in this way a mapper can be allocated to that node and read the split locally. However, when this is not possible, a mapper is assigned by Hadoop to a node where at least one of the two splits resides. In the cost model we need to estimate the local and the network I/O costs, thus given a node n chosen for the execution of a mapper which contains a replica of a D_i split, it is necessary to introduce the estimation $\mathcal{P}_{loc}$ of the probability that also the split of D_j is located in n. It may be computed as:

$$\mathcal{P}_{loc} = 1 - \mathcal{P}_{net} \tag{5}$$

$$\mathcal{P}_{net} = \frac{\binom{\#nodes - \#rep}{\#rep}}{\binom{\#nodes}{\#rep}} \tag{6}$$

where $\mathcal{P}_{net}$ is the ratio between the combinations corresponding to an allocation of the replicas of the second split on nodes that do not contain replicas of the first split and all the possible combinations in which the replicas of the second split can be allocated on the nodes.

5 Spatial Index in SpatialHadoop

Before proceeding with the analysis of the various spatial join operators, we first analyze the concept of spatial index in such environmen and how it is built. This section is useful to completely understand and compare the spatial join operators, since some of them make direct use of indexes, while others work without them.

SpatialHadoop has two level of indexes [17]: 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. In particular, the construction of a grid global index on a input dataset D, determines that D is stored as a set of data files each one containing the records belonging to one cell (or partition). Some spatial join operators are able to exploit the use of a global index in order to efficiently retrieve the data to be processed. SpatialHadoop provides different kinds of global and local indexes, this paper concentrates only on grid index, the extension to other kinds of indexes is straightforward.

The construction of an index involves two MapReduce jobs, the first one determines the grid to be used for the dataset partitioning (see Sect. 5.1), while the second one partitions the data using the computed grid (see Sect. 5.2).

5.1 Grid Construction

The construction of the grid implies the identification of a global MBR for the entire dataset, this is done by the MapReduce job called MBR. During the map phase, the algorithm computes the MBR of each geometry inside the splits. Thanks to an intermediate combiner, the reducer receives only one MBR from each mapper and generates the final MBR covering the whole dataset.

The cost for operator MBR can be defined as $\mathcal{C}(\text{MBR}) = \langle \#map_{\text{MBR}}, 1, \mathcal{M}_{\text{MBR}} \rangle$ where $\#map_{\text{MBR}} = \lceil size(D_*)/splitSz \rceil$, since the data contained in D_* are partitioned among mappers according to the split size, while only one reducer is used to obtain the final result. The estimation of $\mathcal{M}_{\text{MBR}}$ is discussed below.

Estimate 1 (Estimate for $\mathcal{M}_{\text{MBR}}$)**.** *The components of $\mathcal{M}_{\text{MBR}}$ are reported in Table 1 and their rationale is discussed in the following paragraphs.*
cpu rationale. *(i) The CPU cost of each mapper corresponds to the computation of the MBR for all geometries in a split, thus it linearly depends on their average number of vertices. The total number of vertices in a split can be estimated by dividing the split size by the dimension of a vertex in bytes ($\lceil splitSz/vertSz \rceil$). (ii) The CPU cost of each shuffler is dominated by the*

Table 1: Estimation of the components of the matrix $\mathcal{M}$ for the grid index construction. Column MBR regards the corresponding operator which computes the dataset MBR, while column PART regards the corresponding operator which partitions the dataset across the grid.

	Cost	MBR	PART
Map (M)	#*map*	$\left\lceil \frac{size(D_*)}{splitSz} \right\rceil$	$\left\lceil \frac{size(D_*)}{splitSz} \right\rceil$
	cpu	$\overbrace{\left\lceil \frac{splitSz}{vertSz} \right\rceil \cdot 2\mu}^{\text{MBR computation}}$	$\overbrace{\#geo_{\texttt{sp}}(D_*) \cdot \#cells(\mathcal{I}_*) \cdot 4\mu}^{\text{intersection check}}$
	disk	$\overbrace{splitSz}^{\text{reading}} + \overbrace{\#geo_{\texttt{sp}}(D_*) \cdot mbrSz}^{\text{writing}}$	$\overbrace{splitSz}^{\text{reading}} + \overbrace{(\#pairs_{\texttt{mp}}(D_*, \mathcal{I}_*) \cdot recSz(D_*))}^{\text{writing}}$
	net	0	0
Shuffle (S)	*cpu*	$\overbrace{\#map_{\text{MBR}} \cdot \log(\#map_{\text{MBR}}) \cdot 2\mu}^{\text{MBR ordering}}$	$\left(\overbrace{\frac{\#pairs(D_*, \mathcal{I}_*)}{\#red_{\text{PART}}}}^{\text{list construction}} + \overbrace{\frac{\#cells(\mathcal{I}_*)}{\#red_{\text{PART}}} \log \left(\frac{\#cells(\mathcal{I}_*)}{\#red_{\text{PART}}} \right)}^{\text{ordering}} \right) \cdot 2\mu$
	disk	$\overbrace{\#map_{\text{MBR}} \cdot mbrSz}^{\text{writing}}$	$\overbrace{\frac{\#pairs(D_*, \mathcal{I}_*)}{\#red_{\text{PART}}} \cdot recSz(D_*)}^{\text{writing}}$
	net	$\overbrace{\#map_{\text{MBR}} \cdot mbrSz}^{\text{reading}}$	$\overbrace{\frac{\#pairs(D_*, \mathcal{I}_*)}{\#red_{\text{PART}}} \cdot recSz(D_*)}^{\text{reading}}$
Reduce (R)	#*red*	1	$\max(1, \min(\#cells(\mathcal{I}_*), \#parReds)$
	cpu	$\overbrace{\#map_{\text{MBR}} \cdot 4\mu}^{\text{MBR enlarge}}$	$\frac{\#cells(\mathcal{I}_*)}{\#red_{\text{PART}}} \simeq 0$
	disk	$\overbrace{\#map_{\text{MBR}} \cdot mbrSz}^{\text{reading}} + \overbrace{mbrSz}^{\text{writing}}$	$\overbrace{2 \cdot \left(\frac{\#pairs(D_*, \mathcal{I}_*)}{\#red_{\text{PART}}} \cdot recSz(D_*) \right)}^{\text{reading and writing}}$
	net	$\overbrace{mbrSz \cdot (\#rep - 1)}^{\text{writing}}$	$\overbrace{\frac{\#pairs(D_*, \mathcal{I}_*)}{\#red_{\text{PART}}} \cdot recSz(D_*) \cdot (\#rep - 1)}^{\text{writing}}$

ordering procedure it applies on the MBRs produced by the mappers. Thanks to the use of a combiner, the number of MBRs to be ordered is equal to the number of mappers ($\#map_{\text{MBR}}$). (iii) The reducer only computes the global MBR by scanning the MBRs received from the shuffler.

disk i/o rationale. *(i) Each mapper reads locally one split of size splitSz and writes locally one MBR of size mbrSz for each processed geometry. In particular, $\#geo_{sp}(D_*)$ is an estimates of the number of geometries of D_* contained in a split and it can be computed as follows:*

$$\#geo_{sp}(D_*) = \frac{splitSz}{recSz(D_*)} \tag{7}$$

(ii) The shuffler only writes locally one MBR of size mbrSz for each mapper. (iii) The reducer reads locally what the shuffler has produced and writes locally one copy of the global MBR of size mbrSz.

network i/o rationale. *(i) The mappers do not read/write remotely, (ii)*

the shuffler reads remotely one MBR of size mbrSz for each mapper, and finally (iii) the reducer writes remotely $(\#rep - 1)$ *copies of the result.*

Given a global MBR for the entire dataset D_*, the number of grid cells (or partitions) is determined by considering the size of D_* so that the content of each cell can fit inside a split. Since $\mathcal{I}_*$ is an index with replication, namely a geometry can be stored several times if it intersects multiple cells, the dataset size is multiplied by a replication factor α in order to consider such situation.

$$\#cells(\mathcal{I}_*) = \max\left(1, \left\lceil \sqrt{\frac{size(D_*) \cdot \alpha}{splitSz}} \right\rceil^2\right) \tag{8}$$

Notice that the number of required cells is also enlarged in order to obtain a squared grid.

5.2 Data Partitioning

The grid built by the previous job is used during the following phase which performs the actual data partition. In particular, each mapper receives a split containing a set of geometries of D_* and all cells of $\mathcal{I}_*$, and it produces as output the pairs $\langle c, g \rangle$ where the geometry g intersects the cell c. In order, to evaluate the result produced by the mappers, it is necessary to estimate the average number of cells of $\mathcal{I}_*$ that are intersected by a geometry $g \in D_*$.

$$\#cell^{\cap geo}(D_*, \mathcal{I}_*) = \left\lceil \frac{len_x^{avg}(D_*)}{len_x^{cel}(\mathcal{I}_*)} \right\rceil \cdot \left\lceil \frac{len_y^{avg}(D_*)}{len_y^{cel}(\mathcal{I}_*)} \right\rceil + \beta \tag{9}$$

The formula takes care of both the fact that a geometry can span between multiple cells because its extent on the X or Y axis is greater than the corresponding extent of a cell (first two terms) and/or it crosses a cell boundary (see factor β).

Moreover, to account for the case when some geometries in D_* are completely outside the grid (as we will see in Sect. 6.3), we estimate the number of geometries intersecting the index grid by multiplying the number of geometries (i.e., $\#geo(D_*)$) by the following factor r_{int}:

$$r_{int}(D_*, \mathcal{I}_*) = \frac{area(mbr(D_*) \cap mbr(\mathcal{I}_*))}{area(mbr(D_*))} \tag{10}$$

This factor considers the size of geometries negligible w.r.t. the size of the reference space. Moreover, when the dataset D_* completely overlaps the grid of $\mathcal{I}_*$, r_{int} is equal to 1.

The cost of PART can be defined as $\mathcal{C}(\text{PART}) = \langle \#map_{\text{PART}}, \#red_{\text{PART}}, \mathcal{M}_{\text{PART}} \rangle$, where $\#map_{\text{PART}} = \lceil size(D_*)/splitSz \rceil$, since the number of mappers only depends on the input size, while $\#red_{\text{PART}} = \max(1, \min(\#cells(\mathcal{I}_*), \#parReds))$, since the number of reducers can be greater than one only if $\#parReds$ is greater that one with a maximum that is equal to the number of cells in the index $\mathcal{I}_*$. The estimation of $\mathcal{M}_{\text{PART}}$ is discussed below.

Estimate 2 (Estimate for $\mathcal{M}_{\text{PART}}$)**.** *The components of the $\mathcal{M}_{\text{PART}}$ are reported in third column of Table 1.*
cpu rationale. *(i) The CPU cost of each mapper is given by the cost of checking the MBR intersection between the geometries in a split and all the cells of its index $\#cells(\mathcal{I}_*)$ (this check costs 4μ), where $\#geo_{sp}(D_*)$ is estimated using Eq. 7 and $\#cells(\mathcal{I}_*)$ using Eq. 8. (ii) The shuffler combines the results produced by the mappers obtaining a list for each cell and orders such lists based on their key (i.e., the cell geometry). The parameter $\#pairs(D_*, \mathcal{I}_*)$ is an estimate of the number of pairs $\langle c, g \rangle$ produced by all mappers. It can be computed in different ways according to the available statistics, some possible estimates are shown in Table 2. In the a priori case a geometry overlaps all grid cells, while using Eq. 9-10 we can obtain a more precise estimate. The number of cells to be ordered by each shuffler is computed by dividing the total number of cells ($\#cells(\mathcal{I}_*)$) by $\#red_{\text{PART}}$. Insertion in the list and the test for ordering cells cost both 2μ. (iii) Finally, the reducer simply writes to the HDFS the result, so that a separate file split is generated for each partition. Therefore, its CPU cost can be considered negligible.*
disk i/o rationale. *(i) Each mapper reads locally its split of size splitSz and writes locally the resulting pairs $\langle c, g \rangle$ whose number is estimated by parameter $\#pairs_{mp}(D_*, \mathcal{I}_*)$ (see Table 2). (ii) Each shuffler writes locally the total number of produced pairs, estimated by $\#pairs(D_*, \mathcal{I}_*)$, divided by the number of reducers ($\#red_{\text{PART}}$) and (iii) finally, the reducers read and write the pairs produced by the shufflers.*
network i/o rationale. *These estimations regard only the reading phase of the shufflers and the writing phase of the reducers, for which the same considerations done in the previous paragraph apply.*

6 Spatial Join Algorithms

SpatialHadoop provides five different operators for performing the spatial join. 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 index of the other, (iii) the execution of the intersection tests on the

Table 2: Estimates for parameters $\#pairs(D_*, \mathcal{I}_*)$ and $\#pairs_{\texttt{mp}}(D_*, \mathcal{I}_*)$.

Par	Estimate
$\#pairs(D_*, \mathcal{I}_*)$	***a priori***
	$\#geo(D_*) \cdot r_{int}(D_*, \mathcal{I}_*) \cdot \#cells(\mathcal{I}_*)$
	with complete statistics
	$\#geo(D_*) \cdot r_{int}(D_*, \mathcal{I}_*) \cdot \#cell^{\cap geo}(D_*, \mathcal{I}_*)$
$\#pairs_{\texttt{mp}}(D_*, \mathcal{I}_*)$	***a priori***
	$\#geo_{\texttt{sp}}(D_*) \cdot r_{int}(D_*, \mathcal{I}_*) \cdot \#cells(\mathcal{I}_*)$
	with complete statistics
	$\#geo_{\texttt{sp}}(D_*) \cdot r_{int}(D_*, \mathcal{I}_*) \cdot \#cell^{\cap geo}(D_*, \mathcal{I}_*)$

map or on the reduce side. All operators share a plane-sweep like algorithm (PS_{algo}) for checking the intersections between two list of geometries. The difference mainly resides in the way they build the two lists.

PS_{algo} firstly orders the geometries in the two lists based on the minimum X coordinate of their MBR. Then given the two ordered lists, it scans them switching from one list to the other one according to the MBR distribution along the X axis. Finally, for each pair of intersecting MBRs, the actual intersection between the underlying geometries is checked. The estimate of the PS_{algo} cost is presented below based the study in [16].

Estimate 3 (CPU cost of PS_{algo})**.** *Given two datasets D_i, D_j and two subsets of their geometries $l_i \subseteq D_i$ and $l_j \subseteq D_j$, with cardinality n_i and n_j, respectively, the CPU cost for executing PS_{algo} on them is estimated as:*

$$ps(n_i, n_j, A) = \overbrace{n_i \log(n_i) \cdot \mu}^{ordering\ l_i} + \overbrace{n_j \log(n_j) \cdot \mu}^{ordering\ l_j} + \overbrace{n_i \cdot n_j \cdot \sigma^{mbr}(A) \cdot 4\mu}^{MBR\ intersection} + \overbrace{n_i \cdot n_j \cdot \sigma^{mbr}(A) \cdot T^{\cap}_{geo}}^{geometry\ intersection}$$

where A is the area of the reference space used for computing the selectivity using Eq. 2.

***Rationale**: (i) The cost of the ordering phases depends on the cardinality of the lists and is classically estimated as $n \log(n)$. (ii) The number of comparisons between MBRs can be estimated by means of the MBR selectivity between the two datasets (i.e., $\sigma^{mbr}(A)$), for which an estimate has been proposed in Eq. 2. (iii)The number of intersection test between geometries can be estimated using the same selectivity parameter. (iii) $T^{\cap}_{geo}$ is the cost of testing the intersection between two geometries using a plane-sweep algorithm applied to their vertices. Therefore, given $v = \#vert^{avg}(D_i) + \#vert^{avg}(D_j)$, $T^{\cap}_{geo} = v \log(v) \cdot 2\mu$.*

Table 3: Summary of the various spatial join operators.

Op	Reader	Index	Join-side	Rep.	Sect.
DJNI	✓	0	map	✗	6.1
DJGI	✓	2	map	✗	6.2
DJRE	✓	1	map	✓	6.3
DJDR	✗	1	reduce	✓	6.4
SJMR	✗	0	reduce	✗	6.5

Table 4: Estimation for the spatial join operators (map-side). Notice that $\mathtt{M}_*^{\text{PART}}$, $\mathtt{S}_*^{\text{PART}}$ and $\mathtt{R}_*^{\text{PART}}$ are obtained from column 3 of Table 1 by properly instantiating the input dataset and grid index.

Cost		DJNI	DJGI	DJRE (REP)	DJRE (`join`)
Map (M)	*#map*	$\left\lceil \frac{size(D_i)}{splitSz} \right\rceil \left\lceil \frac{size(D_j)}{splitSz} \right\rceil$	$\#cells(\mathcal{I}_i) \cdot \#cells(\mathcal{I}_j)\mathcal{P}^{grid}_{\cap cells}$	$\#map_{\text{PART}}(D_i, \mathcal{I}_j)$	$\#cells(\mathcal{I}_i) \cdot \#cells(\mathcal{I}_j)\mathcal{P}^{rep}_{\cap cells}$
	cpu	plane-sweep alg.: $ps(\#geo_{\mathtt{sp}}(D_i), \#geo_{\mathtt{sp}}(D_j), A)$	filtering phase: $(\#geo_{cl}(D_i,\mathcal{I}_i) + \#geo_{cl}(D_j,\mathcal{I}_j)) \cdot 4\mu +$ plane-sweep algorithm: $ps(\#geo^{sel}_{cl}(D_i,\mathcal{I}_i), \#geo^{sel}_{cl}(D_j,\mathcal{I}_j), A_{\mathtt{mmbr}})$	$\mathtt{M}^{\text{PART}}_{cpu}(D_i, \mathcal{I}_j)$	$\mathtt{M}^{\text{DJGI}}_{cpu}$ $A_{\mathtt{mmbr}} = A_{\mathtt{cl}}$
	disk	reading D_i: $splitSz$ + reading D_j: $splitSz \cdot \mathcal{P}_{loc}$ + writing: $joinSz_{\mathtt{map}}(A)$	reading D_i: $cellSz(D_i)$ + reading D_j: $cellSz(D_j) \cdot \mathcal{P}_{loc}$ + writing: $joinSz_{\mathtt{map}}(A_{\mathtt{mmbr}})$	$\mathtt{M}^{\text{PART}}_{disk}(D_i, \mathcal{I}_j)$	$\mathtt{M}^{\text{DJGI}}_{disk}$ $A_{\mathtt{mmbr}} = A_{\mathtt{cl}}$
	net	reading D_j: $splitSz \cdot \mathcal{P}_{net}$ + writing: $joinSz_{\mathtt{map}}(A) \cdot (\#rep - 1)$	reading D_j: $cellSz(D_j) \cdot \mathcal{P}_{net}$ + writing: $joinSz_{\mathtt{map}}(A_{\mathtt{mmbr}}) \cdot (\#rep - 1)$	$\mathtt{M}^{\text{PART}}_{net}(D_i, \mathcal{I}_j)$	$\mathtt{M}^{\text{DJGI}}_{net}$ $A_{\mathtt{mmbr}} = A_{\mathtt{cl}}$
Shuffle (S)		0	0	$\mathtt{S}^{\text{PART}}(D_i, \mathcal{I}_j)$	0
Reduce (R)		0	0	$\mathtt{R}^{\text{PART}}(D_i, \mathcal{I}_j)$	0

The following sections provide a brief description of each algorithm and an analysis of its costs. Table 3 summarizes the main differences between them and provides a reference to the corresponding section. In the table, column **Reader** indicates the use by the mappers of a `reader` module, which accesses two files at time; column **Index** indicates the number of datasets that require an index, column **Join-side** indicates if the join task is performed by the mappers or the reducers, column **Rep.** indicates if a repartition is applied before the join, and finally the column **Sect.** reports the subsection describing the algorithm.

6.1 Distributed Join with No Index

The first considered spatial join operator works on two input datasets that are not indexed, it is the MapReduce implementation of the Block Nested Loop Join (BNLJ) and it will be called DJNI in the following. DJNI is a map-only job, namely it has no reducers, and clearly it can also be classified as a map-side join.

Given two input files F_i, F_j, the map input is prepared by the reader, which generates one pair of splits for each mapper; overall all the pairs of splits belonging to the Cartesian products $F_i \times F_j$ will be considered. Fig. 3 illustrates the behaviour of a mapper of DJNI when it works on a "combined" split $s = (split_i, split_j) \in F_i \times F_j$. It initially loads the content of such splits into two lists, then it applies PS_{algo} for checking the intersection between the geometries in the two lists.

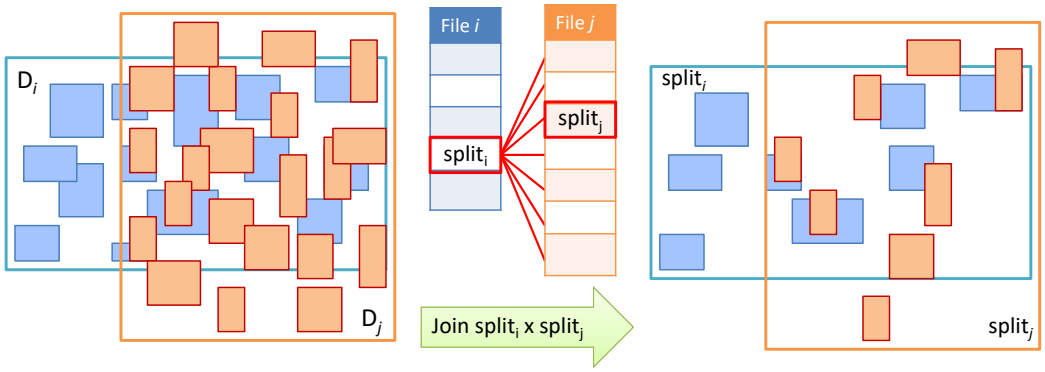

Figure 3: Example of execution of the DJNI algorithm. The two input datasets D_i and D_j are stored respectively into two files. Each split $split_i$ of D_i will be compared with every split of D_j. Moreover, a generic split can contain geometries belonging to any area of the reference space, since data is not indexed.

The cost for operator DJNI(D_i, D_j) can be defined as:

$$\mathcal{C}(\text{DJNI}) = \langle \#map_{\text{DJNI}}, 0, \mathcal{M}_{\text{DJNI}} \rangle$$

where: $\#map_{\text{DJNI}} = \lceil size(D_i)/splitSz \rceil \cdot \lceil size(D_j)/splitSz \rceil$, since all the possible pairs of splits are generated and each pair is processed by one mapper, while the cost estimates of $\mathcal{M}_{\text{DJNI}}$ are reported in of Table 4 (column 2) and are justified by the following considerations.

Estimate 4 (Estimate for $\mathcal{M}_{\text{DJNI}}$)**.** DJNI *is a map-only job, thus Table 4 reports only the cost of the mappers.*

cpu rationale. *Notice that this cost is influenced only by the application of PS_{algo} to the input lists and considering as reference area A the area of the entire reference space $A = area(mbr(D_i) \cup mbr(D_j))$. Thus, the cost only depends on the number of geometries contained in the lists $\#geo_{sp}(D_*)$, which derives directly from the split size (see Eq. 7). The total CPU cost for* DJNI *is dominated by the number of mappers ($\#map_{\text{DJNI}}$).*
disk i/o rationale. *Each mapper certainly reads locally one split of size splitSz, and with probability $\mathcal{P}_{loc}$ (see Eq. 5) also the second one. Moreover, it writes locally the result of the join between the two lists of geometries. The size in bytes of such result (i.e., $joinSz_{map}(A)$) depends on the selectivity between the input datasets and can be estimated as:*

$$joinSz_{map}(A) = \#geo_{sp}(D_i) \cdot \#geo_{sp}(D_j) \cdot \sigma(A) \cdot (recSz(D_i) + recSz(D_j)) \quad (11)$$

where $A = area(mbr(D_i) \cup mbr(D_j))$.
network i/o rationale. *Each mapper reads the second split of size splitSz from the network with a probability $\mathcal{P}_{net}$ (see Eq. 6) and it writes remotely $(\#rep - 1)$ copies of the join result ($joinSz_{map}(A)$).*

6.2 Distributed Join with Grid Index

The second spatial join operator considered in this paper works on two indexed datasets, it will be called DJGI and is a MapReduce adaptation of the Grid File Spatial Join algorithm [18]. This operator is similar to the previous one, it is again a map only job (and consequently a map-side join) However, in this case the reader work on indexed data, namely the key of each record represents an index cell, and a filter is used for preparing the input splits, so that only the pairs of splits regarding intersecting cells are generated. Therefore, the number of generated combined splits, and consequently the number of mappers, is equal to the number of pairs of intersecting cells. With reference to Fig. 4, given a cell $c_i \in \mathcal{I}_i$, it is combined only with the cells of $\mathcal{I}_j$ for which the intersection is not empty (i.e., k_h, k_i, k_j, k_k).

In order to estimate the number of mappers, we need to introduce a formula to compute the probability $\mathcal{P}^{grid}_{\cap cells}$ that given two index cells, $c_i \in \mathcal{I}_i$ and $c_j \in \mathcal{I}_j$, their intersection is not empty. Suppose that the cells of $\mathcal{I}_j$ are smaller than the cells of $\mathcal{I}_i$, this probability can be defined as:

$$\mathcal{P}^{grid}_{\cap cells} = r_{int}(D_j, \mathcal{I}_i) \cdot r_{in} D_i, \mathcal{I}_j) \cdot \frac{\left\lceil \frac{len^{cel}_x(D_i)}{len^{cel}_x(D_j)} \right\rceil \cdot \left\lceil \frac{len^{cel}_y(D_i)}{len^{cel}_y(D_j)} \right\rceil \cdot \cancel{\#cells^{\cap}(\mathcal{I}_i, mbr(D_j))}}{\#cells^{\cap}(\mathcal{I}_j, mbr(D_i)) \cdot \cancel{\#cells^{\cap}(\mathcal{I}_i, mbr(D_j))}} \quad (12)$$

where $r_{int}(D_*, \mathcal{I}_*)$ has been defined in Eq. 10 and is the percentage of cells of $\mathcal{I}_*$ that falls inside the MBR of D_*, while $\#cells^{\cap}(\mathcal{I}_*, mbr(D_*))$ is the number of cells of $\mathcal{I}_*$ that intersect the MBR of D_* and it can be explicitly computed from the available statistics.

The formula is obtained by considering the conjunction of the event corresponding to the choice of a cell of $\mathcal{I}_i$ that falls in the intersection $mbr(D_i) \cap mbr(D_j)$ (namely, $r_{int}(D_j, \mathcal{I}_i)$) with the event corresponding to the choice of a cell of $\mathcal{I}_j$ that falls in the same intersection (namely, $r_{int}(D_i, \mathcal{I}_j)$); then among all the possible pairs of cells that fall in the intersection (denominator) we count the number of intersecting cells (numerator), producing the fraction that appears in the formula.

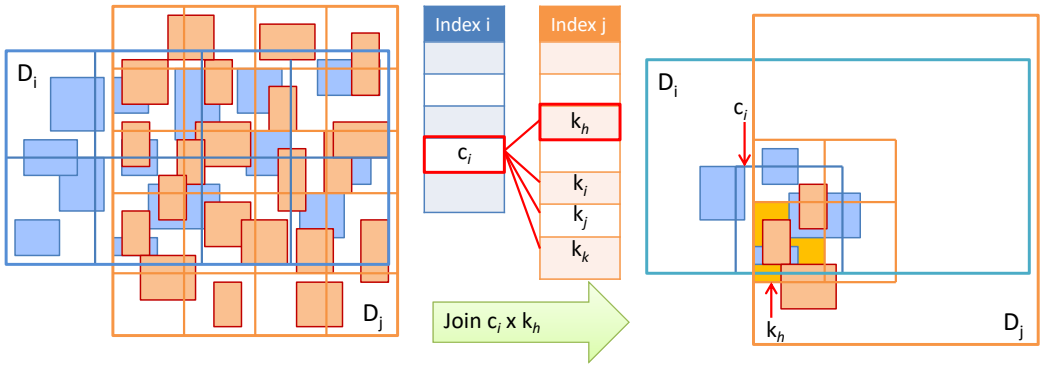

Figure 4: Example of execution of the DJGI algorithm. The two input datasets D_i and D_j have been indexed using a grid. Each cell c_i of $\mathcal{I}_i$ will be compared only with any other cell of $\mathcal{I}_j$ for which the intersection is not empty (i.e., k_h, k_i, k_j and k_k). Notice that in this case only geometries that reside in a nearby space will be compared.

The cost for the operator DJGI can be defined as:

$$\mathcal{C}(\text{DJGI}) = \langle \#map_{\text{DJGI}}, 0, \mathcal{M}_{\text{DJGI}} \rangle$$

where $\#map_{\text{DJGI}} = \#cells(\mathcal{I}_i) \cdot \#cells(\mathcal{I}_j) \cdot \mathcal{P}^{grid}_{\cap cells}$, while the components of $\mathcal{M}_{\text{DJGI}}$ are discussed below. Notice that if the two datasets occupy the same region, namely their MBRs completely overlap, the terms $r_{int}(D_*, \mathcal{I}_*)$ are equal to 1 and $\#cells^{\cap}(\mathcal{I}_*, mbr(D_*)) = \#cells(\mathcal{I}_*)$, so the estimation of the number of mappers becomes: $\#map_{\text{DJGI}} = \lceil len^{cel}_x(D_i)/len^{cel}_x(D_j) \rceil \cdot \lceil len^{cel}_y(D_i)/len^{cel}_y(D_j) \rceil \cdot \#cells(\mathcal{I}_i)$.

Estimate 5 (Estimate for $\mathcal{M}_{\text{DJGI}}$)**.** DJGI *is a map-only job, thus column three of Table 4 contains only the mapper costs.*
cpu rationale. *The CPU cost of a* DJGI *mapper is very similar to the*

cost of a DJNI *mapper; the only difference is the presence of the preliminary filter phase that reduces the number of geometries that are contained in the lists received by* PS_{algo}*. This filter is applied at the beginning of each map iteration: the intersection between two cells is computed (called* `mmbr`*) and only the geometries that intersect* `mmbr` *are considered during the plane-sweep. The filter cost is dominated by the number of geometries of* D_i *and* D_j *contained in each of cell of their corresponding indexes* $\mathcal{I}_i$ *and* $\mathcal{I}_j$*. The number of geometries in each cell of* $\mathcal{I}_*$*, namely* $\#geo_{cl}(D_*, \mathcal{I}_*)$*, can be estimated, considering a uniform distribution, by dividing the number of geometries in* D_* *by the number of cells of* $\mathcal{I}_*$*. In the general case, the overlap between a grid* $\mathcal{I}_*$ *and the MBR of* D_* *can be only partial (as we will see for* DJRE *in Sect. 6.3), thus in the following formula we use the number of intersecting cells (i.e.,* $\#cells^{\cap}(\mathcal{I}_*, mbr(D_*))$*) instead of the total number of cells, even if in the* DJGI *case these numbers coincide:*

$$\#geo_{cl}(D_*, \mathcal{I}_*) = \frac{\#geo(D_*) \cdot \alpha}{\#cells^{\cap}(\mathcal{I}_*, mbr(D_*))} \tag{13}$$

The replication factor $\alpha > 1$ *introduced in Tab. 2 is also used here to determine the geometries per cell.*

The filter phase reduces the number of geometries to be considered by PS_{algo}*. In particular, parameter* $\#geo^{sel}_{cl}(D_*, \mathcal{I}_*)$ *is an estimate of the average number of geometries of* D_* *that survive after the filter phase. It is obtained by multiplying* $\#geo_{cl}(D_*, \mathcal{I}_*)$ *by a filter factor, denoted as* $\psi(D_*)$*, that can be estimated by considering the cells dimensions of the indexes. Let us assume that* $\mathcal{I}_j$ *has cells are smaller than the cells of* $\mathcal{I}_i$*, then* $\psi(D_j) = 1$ *while* $\psi(D_i) = area(\text{cells of } \mathcal{I}_j)/area\ (\text{cells of } \mathcal{I}_i)$*. Clearly, this can be an overestimation of the selectivity, since it considers only the cell dimensions and not their displacement. Notice also that in this case the cost estimation of* PS_{algo} *considers a selectivity computed on an area equal to the average area of the not empty* `mmbr` *(called* $A_{\texttt{mmbr}}$*).*

disk i/o rationale. *As in the case of the operator* DJNI*, each mapper reads locally the first dataset and with a probability* $\mathcal{P}_{loc}$ *the second one, the only difference is that the split size is not fixed but it depends on the number of geometries in each index cell, namely* $cellSz = \#geo_{cl}(D_*, \mathcal{I}) \cdot recSz(D_*)$*). Moreover, it writes locally one copy of the join result (i.e.,* $joinSz_{\texttt{map}}(A_{\texttt{mmbr}})$*) whose dimension depends on the selectivity computed using* $A_{\texttt{mmbr}}$*.*

network i/o rationale. *As in the case of the operator* DJNI*, each mapper reads remotely the second dataset with a probability* $\mathcal{P}_{net}$ *and the split size is the same of the one used for the local I/O. Moreover, it writes remotely* $(\#rep - 1)$ *copies of the join result with the same size estimated for the local I/O.*

6.3 Distributed Join with Repartition

A variant of DJGI is the operator denoted as DJRE, which additionally performs a repartition of one of the two datasets w.r.t. the index of the other. It is a MapReduce adaptation of the Bulk-Index Join [19]. In particular, if both input datasets are indexed, the smaller one is repartitioned using the index of the bigger one; conversely, if only dataset D_j (D_i) is indexed, then D_i (D_j) is repartitioned using the index of D_j (D_i). This operator consists of two map-reduce jobs, the first one performs the repartition and the second one is similar to DJGI. The repartition phase can be particularly useful when the two datasets have a partial overlapping in this case the geometries of the repartitioned dataset that do not intersect the MBR of the other are filtered out, since they certainly will not participate to the join result. Moreover, only pairs of fully overlapping cells will be considered, thus leading to a reduced number of balanced mappers.

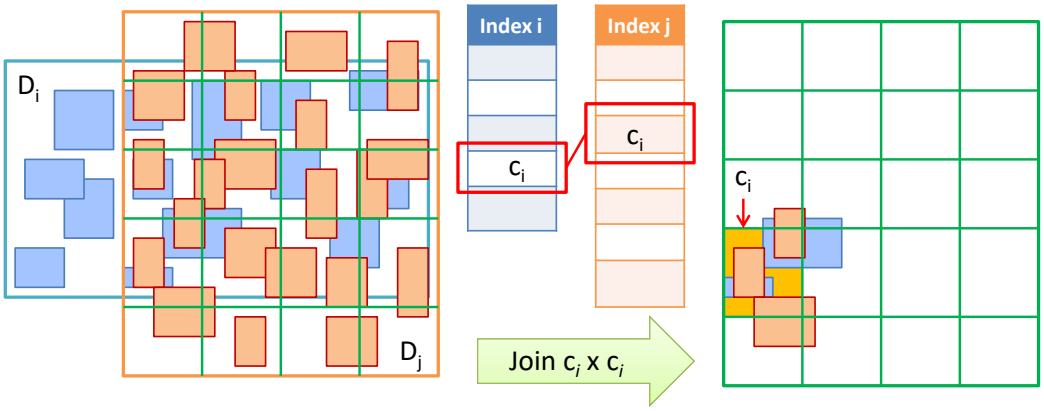

Figure 5: Example of execution of the DJRE algorithm. In this case dataset D_i is repartitioned using the grid index of D_j. Therefore, after such repartition any cell $c_i \in \mathcal{I}_i$ will be compared only with the corresponding cell $c_i \in \mathcal{I}_j$.

6.3.1 Repartition Phase

Without loss of generality we consider that D_i has to be repartitioned w.r.t. the index of D_j. The repartition is performed by a job, called REP, composed of a map and a reduce phase. The mappers scan each geometry of D_i and build a pair $\langle c, g \rangle$ for each cell $c \in \mathcal{I}_j$ that intersects a geometry $g \in D_i$. This job is similar to the PART job performed during the index construction and illustrated in Sect. 5.2. However, in this case the dataset to be partitioned is

D_i while the considered index is $\mathcal{I}_j$, namely the index of D_j. Therefore, with reference to the formulas defined in Eq. 10 the factor r_{int} could be different from 1.

6.3.2 Join Phase

The cost of the join job is the same as the cost of the DJGI operator except for the formula which computes the $\mathcal{P}_{\cap cells}$ since in this case the two datasets share the same index grid. In other words, the number of combined splits that will be generated is equal to the number of cells that intersect the MBR of both datasets. $\mathcal{P}_{\cap cells}$ can be obtained by simplifying Eq. 12 as follows:

$$\mathcal{P}^{rep}_{\cap cells} = \frac{\#cells^{\cap}(\mathcal{I}_j, mbr(D_i))}{\#cells^{\cap}(\mathcal{I}_j, mbr(D_j)) \cdot \#cells^{\cap}(\mathcal{I}_j, mbr(D_i))} \tag{14}$$

Moreover, as regards to the selectivity used for computing the result dimension, namely $joinSz_{\texttt{map}}(A_{\texttt{mmbr}})$, since the cells of both indexes are the same, for each mapper it always occurs that: $A_{\texttt{mmbr}} = A_{\texttt{cl}}$, where $A_{\texttt{cl}}$ is the area of a cell.

6.4 Distributed Join with Direct Repartition

A variant of DJRE consists in performing the repartition and the join inside the same job. In this case, the mappers are responsible for performing the repartition, while the reducers perform the join. This operator can be classified as a reduce-side join and will be denoted as DJDR. In particular, the mappers perform the same job described for the REP job in Section 6.3.1, while the reducers receive from the shuffler a set of cells together with their intersecting geometries coming from the repartitioned dataset D_i. For each received cell, a reducer loads the geometries of D_j that resides in the same cell, through the use of the index, then it executes PS_{algo} on these two lists.

The cost for operator DJDR can be defined as follows:

$$\mathcal{C}(\text{DJDR}) = \langle \#map_{\text{DJDR}}, \#red_{\text{DJDR}}, \mathcal{M}_{\text{DJDR}} \rangle$$

where $\#map_{\text{DJDR}} = \lceil size(D_i)/splitSz \rceil$ and $\#red_{\text{DJDR}} = \max(1, \min(\ \#cells^{\cap}(\mathcal{I}_j, mbr(D_i)), \#parReds))$, which are equal to the number of mappers and reducers for the REP job presented in Sect. 6.3.1. The estimates of $\mathcal{M}_{\text{DJDR}}$ are discussed below.

Estimate 6 (Estimate for $\mathcal{M}_{\text{DJDR}}$)**.** *The cost estimates of $\mathcal{M}_{\text{DJDR}}$ are reported in Table 5 (column 2) and are justified by the following reasoning.*
cpu rationale. *(i) The CPU cost of each mapper and shuffler is equal to*

Table 5: Estimation for the spatial join operators (reduce-side).

Cost		DJDR	SJMR (D_i grid)	SJMR (D_j grid)	SJMR (join)
Map (M)	$\#map$	$\#map_{\text{REP}}(D_i)$	$\#map_{\text{MBR}}(D_i)$	$\#map_{\text{MBR}}(D_j)$	$\left\lceil \frac{size(D_i)+size(D_j)}{splitSz} \right\rceil$
	cpu	$\mathtt{M}^{\text{REP}}_{cpu}(D_i, D_j)$	$\mathtt{M}^{\text{MBR}}_{cpu}(D_i)$	$\mathtt{M}^{\text{MBR}}_{cpu}(D_j)$	$\overbrace{\#pairs_{\mathtt{mp}}(D_\cup, \mathcal{I}_\cup) \cdot 4\mu}^{\text{cell-geom pairs}}$
	disk	$\mathtt{M}^{\text{REP}}_{disk}(D_i, D_j)$	$\mathtt{M}^{\text{MBR}}_{disk}(D_i)$	$\mathtt{M}^{\text{MBR}}_{disk}(D_j)$	$\overbrace{splitSz}^{\text{reading}} + \overbrace{\#pairs_{\mathtt{mp}}(D_\cup, \mathcal{I}_\cup) \cdot recSz(D_\cup)}^{\text{writing}}$
	net	$\mathtt{M}^{\text{REP}}_{net}(D_i, D_j)$	$\mathtt{M}^{\text{MBR}}_{net}(D_i)$	$\mathtt{M}^{\text{MBR}}_{net}(D_j)$	0
Shuffle (S)	*cpu*	$\mathtt{S}^{\text{REP}}_{cpu}(D_i, D_j)$	$\mathtt{S}^{\text{MBR}}_{cpu}(D_i)$	$\mathtt{S}^{\text{MBR}}_{cpu}(D_j)$	$(\overbrace{\#pairs_\cup}^{\text{list constr.}} + \overbrace{\#cells^{red}_\cup \log \#cells^{red}_\cup}^{\text{ordering}})2\mu$
	disk	$\mathtt{S}^{\text{REP}}_{disk}(D_i, D_j)$	$\mathtt{S}^{\text{MBR}}_{disk}(D_i)$	$\mathtt{S}^{\text{MBR}}_{disk}(D_j)$	$\overbrace{\#pairs_\cup \cdot recSz(D_\cup)}^{\text{writing}}$
	net	$\mathtt{S}^{\text{REP}}_{net}(D_i, D_j)$	$\mathtt{S}^{\text{MBR}}_{net}(D_i)$	$\mathtt{S}^{\text{MBR}}_{net}(D_j)$	$\overbrace{\#pairs_\cup \cdot recSz(D_\cup)}^{\text{reading}}$
Reduce (R)	$\#red$	$\#red_{\text{REP}}$	$\#red_{\text{MBR}}(D_i)$	$\#red_{\text{MBR}}(D_j)$	$\max(1, \min(\#cells(\mathcal{I}_\cup), \#parReds))$
	cpu	$\#cells^{red}(D_i) \cdot \overbrace{ps(\#geo_{cl}(D_i, \mathcal{I}_j), \#geo_{cl}(D_j, \mathcal{I}_j, A_{\mathtt{cl}}))}^{\text{intersection test}}$	$\mathtt{R}^{\text{MBR}}_{cpu}(D_i)$	$\mathtt{R}^{\text{MBR}}_{cpu}(D_j)$	$\#cells^{red}_\cup \cdot \overbrace{ps(\#geo_{cl}(D_i, \mathcal{I}_\cup), \#geo_{cl}(D_j, \mathcal{I}_\cup), A_{\mathtt{cl}})}^{\text{intersection test}}$
	disk	$\#cells^{red}(D_i) \cdot (\overbrace{(cellSz(D_i) + cellSz(D_j) \cdot \mathcal{P}_{loc})}^{\text{reading}} + \overbrace{joinSz_{cl}(A_{\mathtt{cl}}, \mathcal{I}_j)}^{\text{writing}})$	$\mathtt{R}^{\text{MBR}}_{disk}(D_i)$	$\mathtt{R}^{\text{MBR}}_{disk}(D_j)$	$\overbrace{\#pairs_\cup \cdot recSz(D_\cup)}^{\text{reading}} + \overbrace{\#cells^{red}_\cup \cdot joinSz_{cl}(A_{\mathtt{cl}}, \mathcal{I}_\cup)}^{\text{writing}}$
	net	$\#cells^{red}(D_i) \cdot (\overbrace{cellSz(D_j) \cdot \mathcal{P}_{net}}^{\text{reading}} + \overbrace{joinSz_{cl}(A_{\mathtt{cl}}, \mathcal{I}_j) \cdot (\#rep - 1)}^{\text{writing}})$	$\mathtt{R}^{\text{MBR}}_{net}(D_i)$	$\mathtt{R}^{\text{MBR}}_{net}(D_j)$	$\overbrace{\#cells^{red}_\cup \cdot joinSz_{cl}(A_{\mathtt{cl}}, \mathcal{I}_\cup) \cdot (\#rep - 1)}^{\text{writing}}$

the corresponding costs of the REP *job (Sect. 6.3.1). (ii) For each cell c passed to a reducer, it applies PS_{algo} to the list of geometries coming from the two datasets and belonging to c. The number of geometries belonging to a cell of $\mathcal{I}_j$, namely $\#geo_{cl}(D_*, \mathcal{I}_j)$, is estimated as done in Eq. 13. Parameter $\#cells^{red}(D_i)$ is an estimate of the number of cells processed by each reducer and it can be computed as: $\#cells^{\cap}(\mathcal{I}_j, mbr(D_i))/\#red_{\text{DJDR}}$.*

disk i/o rationale. *The local I/O cost for each mapper and shuffler is the same as the corresponding cost of the* REP *job (Sect. 6.3.1). As regards to the reducers, each task reads the records of D_i locally, while the records of D_j can be read locally only with probability $\mathcal{P}_{loc}$. Finally, $joinSz_{cl}(A_{\mathtt{cl}}, \mathcal{I}_j)$ is the estimated size in bytes of the result produced by each reducer for each cell*

and it can be computed as:

$$joinSz_{cl}(A_{cl}, \mathcal{I}_j) = \#geo_{cl}(D_i, \mathcal{I}_j)\#geo_{cl}(D_j, \mathcal{I}_j)\sigma(A_{cl}) \cdot (recSz(D_i) + recSz(D_j)) \quad (15)$$

which is similar to the output produced by each mapper in the Rep *job. However, since a reducer could process more than one cell, this parameter is multiplied for the number of cells assigned to each reducer (*$\#cells^{red}(D_i)$*).*
network i/o rationale. *As for the previous costs, also the cost of the network I/O for the mappers and shufflers is equal to the corresponding costs of the* Rep *job (Sect. 6.3.1). As regards to the reducers, for each cell c, they read remotely, with a probability* $\mathcal{P}_{net}$*, the portion of* D_j *overlapping c of size* $cellSz(D_j)$*; while they write remotely* $(\#rep - 1)$ *copies of the result of size* $joinSz_{cl}(A_{cl}, \mathcal{I}_j)$*.*

6.5 Spatial Join Map Reduce

The last operator considered in this paper is called SJMR (Spatial Join Map Reduce) and has been designed to perform spatial join efficiently for non-indexed datasets. It is the map-reduce implementation of the Partition Based Spatial Merge Join [20] and it will be denoted as Sjmr in the following. It uses a uniform grid for performing the spatial join which is computed from the union of the MBR of the two datasets, while the cell dimension is automatically determined based on the input files size.

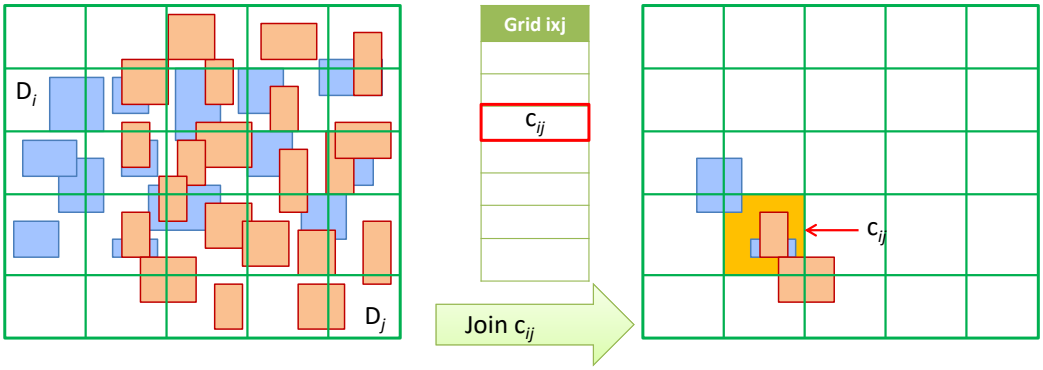

Figure 6: Example of execution of the Sjmr algorithm. In this case a global index $\mathcal{I}_\cup$ is build which includes the union of the MBRs of the two datasets. Each cell c_{ij} is separately processed considering the geometries coming from both datasets.

6.5.1 Grid Computation

The uniform grid is built by using two map reduce jobs, each one is responsible for determining the MBR of a dataset. The cost of the each MBR job can be estimated as shown in Sect. 6.2. The two MBRs are then merged into a global one and the number of required cells is determined so that the content of each cell can fit into a split. In particular, $\#cells(\mathcal{I}_{\cup})$ denotes the number of cells for this uniform grid and it is computed using Eq. 8 where $D_* = D_i \cup D_j$.

6.5.2 Join Phase

As regards to the join phase, each mapper receives in input a set of geometries coming from both datasets. Notice that SJMR does not use the binary reader introduced in Sect. 4 for combining the input files. Conversely, the input files are merged into a single one by concatenating them. For each geometry g in input, a mapper directly computes the set of cells that intersect its MBR. For each match cell-geometry found in this step, it writes in output the pair $\langle c, \langle f, g \rangle \rangle$, where the key is the grid cell c, and the value is again a pair containing the identifier f of the file from which the geometry comes from and the geometry g itself. Each reducer can work on one or more cells. For each cell, it builds two lists by dividing the geometries in a cell based on the file from which they come from. Given the two lists, PS_{algo} is performed on them for producing the final result. Figure 6 illustrates the behaviour of SJMR when processes a grid c_{ij} belonging to the global grid built considering the union of the two datasets.

The cost of operator SJMR can be defined as follows:

$$\mathcal{C}(\text{SJMR}) = \langle \#map_{\text{SJMR}}, \#red_{\text{SJMR}}, \mathcal{M}_{\text{SJMR}} \rangle$$

where $\#map_{\text{SJMR}} = \lceil (size(D_i) + size(D_j))/splitSz \rceil$, since it works on the union of the two input datasets, while $\#red_{\text{SJMR}} = \max(1, \min(\#cells(D_{\cup}), \#parReds))$, since the number of reducers can be greater than one only if the Hadoop configuration allows more than one reducers, with a maximum equal to the number of cells. The components of the $\mathcal{M}_{\text{SJMR}}$ are discussed below.

Estimate 7 (Estimate for $\mathcal{M}_{\text{SJMR}}$)**.** *The cost estimates of $\mathcal{M}_{\text{SJMR}}$ are reported in Table 5 (column 5) and are justified by the following rationale.*
cpu rationale. *(i) Each mapper works on a split coming from the union of the two input datasets: $D_{\cup} = D_i \cup D_j$. The operation performed on each geometry takes a constant time (i.e., 4μ) to determine the intersecting cells, since it only uses some comparisons between the MBR coordinates and the cell lengths. The average number of geometries contained in a*

split is represented by the parameter $\#geo_{sp}(D_\cup)$*, which can be estimated as* $\#geo_{sp}(D_\cup) = splitSz/recSz(D_\cup)$*, where* $recSz(D_\cup)$ *is the average record size computed considering the records of both datasets. Moreover, the estimated number of matches cell-geometry (*$\#pairs_{mp}(D_\cup, \mathcal{I}_\cup)$*) is computed as in Table 2. (ii) The shufflers collect the pairs produced by the mappers, combines the record related to the same cell into lists and order such lists based on the key (i.e., the cell). The number of records to be combined by the shuffler is estimated by the parameter* $\#pairs_\cup$*:*

$$\#pairs_\cup = \frac{\#pairs(D_i, \mathcal{I}_\cup) + \#pairs(D_j, \mathcal{I}_\cup)}{\#red_{\text{SJMR}}}$$

The number of cells ordered by each shuffler is estimated by the parameter $\#cells_\cup^{red}$*:*

$$\#cells_\cup^{red} = \frac{1}{\#red_{\text{SJMR}}} \cdot (\\ \#cells^\cap(\mathcal{I}_\cup, mbr(D_i)) + \#cells^\cap(\mathcal{I}_\cup, mbr(D_j)) - \\ \#cells^\cap(\mathcal{I}_\cup, mbr(D_i \cap D_j)) \tag{16}$$

where the term $\#cells^\cap(\mathcal{I}_\cup, D_*)$ *is the number of cells of* $\mathcal{I}_\cup$ *that intersect* D_**. Notice that in the formula* $\#cells^\cap(\mathcal{I}_\cup, mbr(D_i \cap D_j))$ *has been subtracted for not counting twice the same cell. (iii) Finally, the reducers perform the spatial join using* PS_{algo} *inside each cell. Each reducer works on a number of cells* $\#cells_\cup^{red}$ *(see Eq. 16).*

disk i/o rationale. *(i) Each mapper reads locally a union split of size splitSz and writes locally a record for each intersecting pair of geometry-cell. The average number of intersecting pairs has been estimated above by the parameter* $\#pairs_{mp}(D_\cup, \mathcal{I}_\cup)$*, while the size of each record is estimated as* $recSz(D_\cup) = \#vert^{avg}(D_\cup) \cdot vertSz$*, where* $\#vert^{avg}(D_\cup)$ *is the average number of vertices of the geometries contained in the union of the two datasets* $D_i \cup D_j$*. (ii) Given the output produced by the mappers, each shuffler writes locally its combined records whose number is estimated by* $\#pairs_\cup$ *and whose size by* $recSz(D_\cup)$*. (iii) Each reducer reads locally the input produced by its corresponding shuffler and writes a portion of the join result whose size is obtained by multiplying the number of its cells (*$\#cells_\cup^{red}$*) by the parameter* $joinSz_{cl}(A_{cl}, \mathcal{I}_\cup)$ *(see Eq. 15).*

network i/o rationale. *Only the shufflers and the reducers performs network I/O. In particular, each shuffler reads a portion of the data produced by all mappers (of size* $\#pairs_\cup \cdot recSz(D_\cup)$*) and the reducers remotely write* $(\#rep - 1)$ *copies of the results.*

7 Validation and Experiments

The cost model presented in the previous sections has been validated using a set of experiments on synthetic datasets. We first evaluate the quality of the cost model by comparing the estimated costs and the actual measured costs, considering a representative case where geometries in each dataset have an MBR area equal to 1e−8 w.r.t. the area of the reference space and five vertices, while the size of a dataset is 128 MBytes and the size of the other one varies from 128 MBytes to 5120 MBytes. In this comparison we try to keep the various cost components as much separated as possible. However, as regards to the I/O, we compare the sum of the local and network I/O estimates with the total number of bytes read and written as reported in the Hadoop logs (`HDFS/File: Number of bytes read/written`); indeed, the statistics that we can find in the logs do not distinguish between readings and writings. The actual comparison regarding this overall I/O cost can be done directly, since the estimates and measured values are both in bytes, while for the CPU cost we can only compare the trend of the estimates with the trend produced by the log value `CPU time spent`. Fig. 7 reports the estimated cost for the overall I/O, while Fig. 8 reports the number of bytes read and written by each algorithm as reported in the Hadoop logs. The average difference between the two is about 9%. Similarly, Fig. 9-10 report the estimated and actual CPU costs; as we can notice the two trends are very similar. However, the estimated CPU costs are lower than the measured one, since the CPU cost taken from the logs includes the time required by Hadoop to instantiate the map-reduce jobs, while we omit it in the proposed cost model formulation.

In order to compare the results produced by the cost model with the ones produced by the experiments, we introduce a relation of dominance. This relation can be applied to the cost estimations in order to produce a partial order among the operators, with the aim to choose the best candidate in a given situation.

Definition 6 (Dominance)**.** *Given two cost vectors $cv_*(op_1)$, $cv_*(op_2)$ defined as in Def. 5, representing the estimates of the cost for operators op_1 and op_2 respectively, where * stands for tot or par. We say that op_1 dominates op_2 according to the cost model ($op_1 \prec op_2$), if the following conditions hold:*

- $\forall i \in \{1, 2, 3\} : cv_*(op_1)[i] \leq cv_*(op_2)[i]$
- $\exists j \in \{1, 2, 3\} : cv_*(op_1)[j] < cv_*(op_2)[j]$

where $cv_*(op)[i]$ denotes the i-th elements of vector $cv_*(op)$.

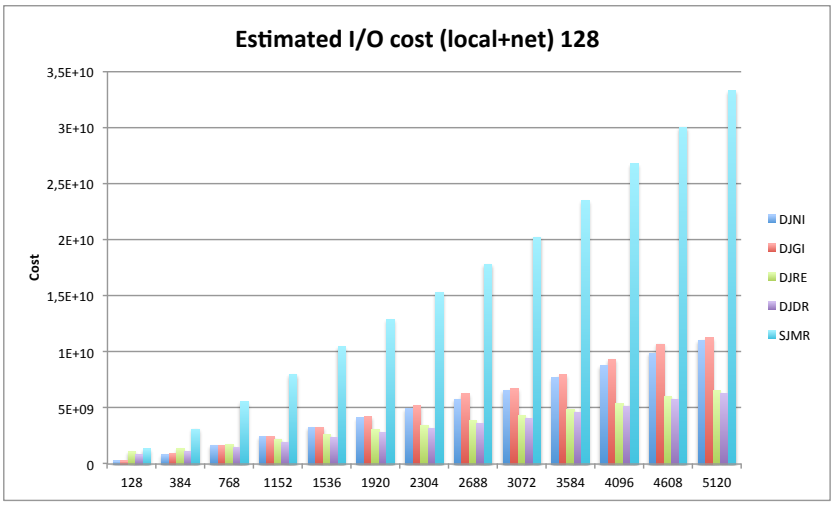

Figure 7: Estimated trend for the local and network I/O in MBytes for the cases $|D_i| = 128$ MBytes and $|D_j|$ from 128 MBytes to 5120 MBytes, considering the various spatial-join algorithms. SJMR has the greater I/O since it initially repartitions the two datasets, while DJRE and/or DJDR has the lower I/O since they repartition only the smaller dataset.

The equality among the estimated costs is evaluated considering a threshold of 1%. Moreover, since the partial order cannot always produce one best candidate, we apply the following procedure to choose the best algorithm:

1. given the characteristics of the input datasets, we compute the dominance relations among the operators by considering their cost estimates and we insert in the set F the non-dominated operators.

2. if $F = \{op\}$, then op is the best choice.

3. if F contains more then one operator, we can apply some heuristics in order to reduce its size. Some possible heuristics are described below. Finally, we randomly choose one of the operators from F.

In the experiments we apply the following heuristic. It is based on a constant δ computed with a set of experiments on the cluster performance for comparing the CPU and I/O costs: the aim is to determine in which cases an advantage on the CPU can balance a disadvantage on the I/O. This behaviour has been observed in particular for the SJMR.

Definition 7 (Cost model heuristic)**.** *Given two operators op_1 and op_2 belonging to the set F of non-dominated ones and the parameter δ. If op_1*

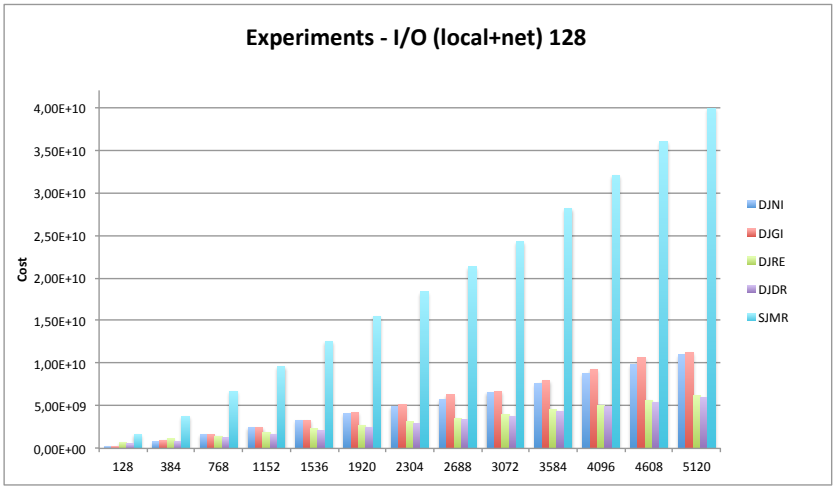

Figure 8: Trend for the local and network I/O in MBytes obtained from the experiments performed with $|D_i| = 128$MBytes and $|D_j|$ from 128 MBytes to 5120 MBytes, considering the various spatial-join algorithms.

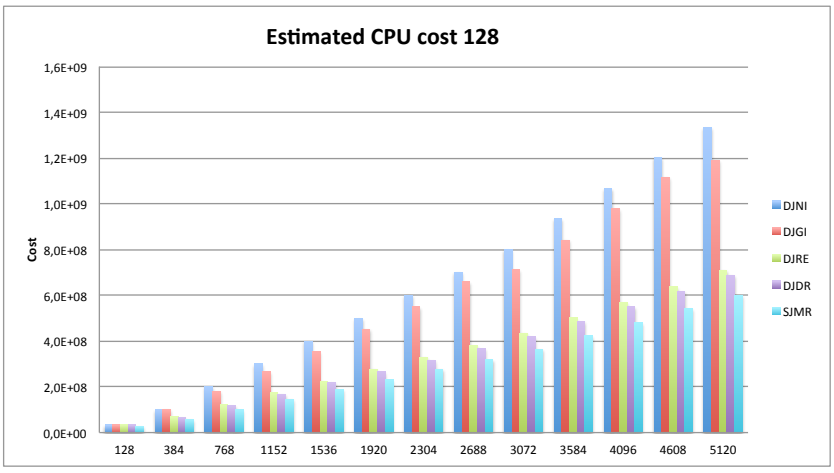

Figure 9: Estimated trend for the CPU cost for the cases $|D_i| = 128$ MBytes and $|D_j|$ from 128 MBytes to 5120 MBytes, considering the various spatial-join algorithms. SJMR is the algorithm with the least number of estimated comparisons to perform, while DJNI is the one with the greatest number of comparisons to perform.

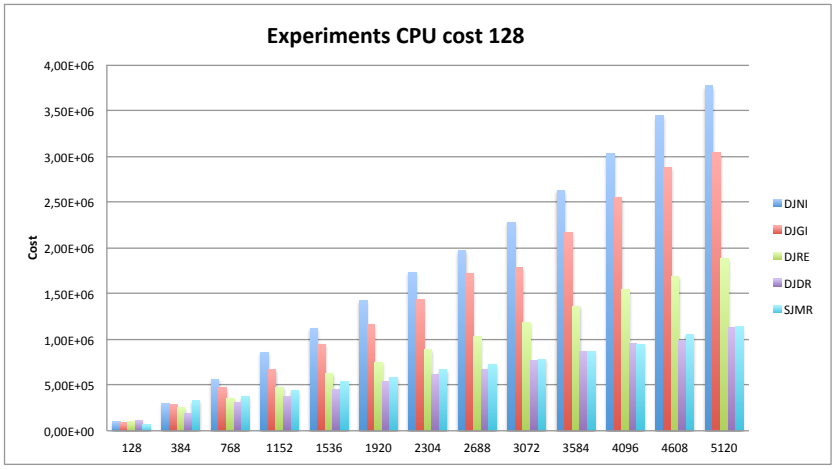

Figure 10: Trend for the CPU cost obtained from the experiments performed for the cases $|D_i| = 128$MBytes and $|D_j|$ from 128 MBytes to 5120 MBytes, considering the various spatial-join algorithms.

is better than op_2 on the CPU cost and op_2 is better than op_1 on the total I/O cost, then op_1 is discarded from F if the difference on CPU cost does not balance the difference on the total I/O costs, evaluated considering the parameter δ.

The cost model has been tested in various scenarios by varying different characteristics of the involved datasets such as: the dataset cardinalities, the dimension of the geometry MBRs and the rate of dataset overlapping. The experiments reveal that the cost model is able to detect the best algorithm in 88% of cases and, considering the effective execution time, it always gains from a minimum of 21% to a maximum of 67% w.r.t. a random choice.

Experiment 1 (Cardinality – ExpCard)**.** *Given two datasets with the same reference space (i.e., percentage of overlapping equals to 100%) and whose elements are polygons with the same number of vertices (i.e., 5) and an MBR of size* 1e−8 *w.r.t. the reference space,* ***ExpCard*** *changed the cardinality of the two datasets starting from 128 MBytes (1 split) to 5120 MBytes (40 splits).*

Tables 6-15 reports the results of **ExpCard** for some of the considered pairs of cardinalities. Each table is relative to a specific cardinality for dataset D_i (reported in the caption) while column # contains the number of splits for the dataset D_j, the other four columns reports the ordering obtained from the experiments as a number and with a color the recommendation of the cost model, column **DL** contains the percentage of delay in choosing

Table 6: Results of **ExpCard** without consider the index cost for the case $|D_i| = 1$ split, while $|D_j|$ is equal to the number of splits in column #.

#	DJNI	DJGI	DJRE	DJDR	SJMR	DL	GW	GR
1	2	1	5	4	3	5%	32%	10%
3	4	1	3	2	5	36%	62%	22%
6	4	2	3	1	5	6%	59%	25%
9	4	2	1	3	5	4%	47%	21%
12	4	3	2	1	5	1%	52%	13%
15	4	3	1	2	5	2%	49%	27%
18	4	3	1	2	5	1%	54%	30%
21	4	3	1	2	5	4%	53%	29%
24	4	3	1	2	5	5%	50%	28%
28	4	3	1	2	5	4%	47%	28%
32	5	3	1	2	4	1%	48%	28%
36	5	3	1	2	4	3%	50%	30%
40	4	3	1	2	5	3%	47%	29%
Average						6%	50%	25%

one of the solutions recommended by the cost model in place of the actual best algorithm, while **GW** is the percentage of gain in choosing one of the recommended algorithms in place of the actual worst algorithm and **GR** is the same as **GW** but w.r.t. performing a random choice.

We consider the two cases in which the cost of the index construction is or is not considered (+2in and +1in means that an algorithm requires the preliminary presence of two or one index). A green square denotes the algorithms recommended by the cost model, namely the non-dominated ones, while a red square denotes the worst algorithm, namely the one that is dominated by all the other algorithms. Notice that since the dominance relation induces a partial order, more than one green square can be present in a row. Thus, the more the recommendation is focused the more the delay decreases and the gain increases.

Experiment 2 (MBR size – ExpMBR)**.** *Given the datasets considered in Exp. 1,* ***ExpMBR*** *changes the MBR size of each geometry to* 1e−7 *w.r.t. the reference space.*

Table 16 reports the results of **ExpMBR** without and with considering the cost of the index construction (column **IC**). In particular, column **G** denotes the cardinality in MBytes of the first dataset D_i, for each of these groups the cardinality of the second dataset D_j is changed from 128 MBytes

Table 7: Results of **ExpCard** without consider the index cost for the case $|D_i| = 9$ splits, while $|D_j|$ is equal to the number of splits in column #.

#	DJNI	DJGI	DJRE	DJDR	SJMR	DL	GW	GR
1	4	1	2	3	5	6%	55%	22%
3	2	1	3	5	4	90%	7%	-13%
6	5	1	3	4	2	63%	42%	12%
9	5	1	3	4	2	43%	57%	23%
12	5	1	3	4	2	32%	63%	25%
15	5	1	3	4	2	50%	66%	24%
18	5	1	3	4	2	38%	69%	28%
21	5	1	3	4	2	27%	74%	37%
24	5	1	3	4	2	26%	76%	39%
28	5	1	3	4	2	21%	77%	41%
32	5	1	3	4	2	14%	79%	44%
36	5	1	3	4	2	15%	80%	44%
40	5	1	3	4	2	11%	81%	46%
Average						34%	63%	28%

(1 split) to 5120 MBytes (40 splits) and the averages are computed. Column $|\mathbf{F}|$ denotes the average number of non-dominated solutions produced by the cost model, while column $\mathbf{b} \in \mathbf{F}$ reports the percentage of cases in which the operator b, which is the best in the experiments, is contained in F. Column **%DL** is the average percentage of delay w.r.t b obtained by selecting one of the non-dominated solutions in F. Columns **%GW** and **%GR** are the average percentage of gain in choosing one of the non-dominated solutions in F w.r.t. the experimental worst operator and a random operator, respectively. Finally, column $\mathbf{w} \in \mathbf{L}$ reports the percentage of cases in which the operator w, which is the worst in the experiments, is contained in the set of dominated solutions L produced by the cost model.

Experiment 3 (Overlapping – ExpOver)**.** *Given the datasets considered in Exp. 1,* ***ExpOver*** *changes the percentage of overlapping of the two datasets to 50% and 25%.*

Table 17 reports the results of **ExpOver** without and with considering the cost of the index construction and applying an overlap of 50% and 25% (column **OV**), the meaning of other columns has been described for **ExpMBR**.

Table 8: Results of **ExpCard** without consider the index cost for the case $|D_i| = 18$ splits, while $|D_j|$ is equal to the number of splits in column #.

#	Djni	Djgi	Djre	Djdr	Sjmr	DL	GW	GR
1	4	3	1	2	5	1%	52%	29%
3	5	1	2	3	4	16%	40%	14%
6	5	1	4	3	2	38%	63%	26%
9	5	1	3	4	2	53%	67%	23%
12	5	1	3	4	2	45%	71%	28%
15	5	2	3	4	1	20%	77%	41%
18	5	1	3	4	2	0%	82%	50%
21	5	1	3	4	2	21%	82%	48%
24	5	2	3	4	1	16%	84%	51%
28	5	2	3	4	1	20%	85%	52%
32	5	2	3	4	1	18%	86%	54%
36	5	3	2	4	1	15%	86%	56%
40	5	2	3	4	1	19%	86%	56%
Average						22%	74%	40%

Table 9: Results of **ExpCard** without consider the index cost for the case $|D_i| = 28$ splits, while $|D_j|$ is equal to the number of splits in column #.

#	Djni	Djgi	Djre	Djdr	Sjmr	DL	GW	GR
1	5	3	1	2	4	3%	44%	25%
3	5	1	3	4	2	6%	48%	14%
6	5	1	3	4	2	23%	729%	34%
9	5	1	2	4	3	20%	78%	42%
12	5	1	3	4	2	19%	80%	43%
15	5	1	3	4	2	33%	80%	42%
18	5	2	3	4	1	20%	84%	51%
21	5	2	3	4	1	25%	85%	53%
24	5	2	3	4	1	28%	85%	54%
28	5	2	3	4	1	37%	86%	56%
32	5	2	3	4	1	27%	88%	60%
36	5	2	3	4	1	29%	88%	60%
40	5	2	3	4	1	30%	89%	60%
Average						23%	78%	46%

Table 10: Results of **ExpCard** without consider the index cost for the case $|D_i| = 40$ splits, while $|D_j|$ is equal to the number of splits in column #.

#	DJNI	DJGI	DJRE	DJDR	SJMR	DL	GW	GR
1	5	3	1	2	4	1%	47%	28%
3	5	4	1	2	3	8%	60%	26%
6	5	1	3	4	2	17%	37%	
9	5	1	3	4	2	7%	75%	47%
12	5	2	1	4	3	7%	82%	51%
15	5	2	3	4	1	12%	84%	54%
18	5	2	3	4	1	19%	86%	54%
21	5	2	3	4	1	21%	86%	58%
24	5	2	3	4	1	24%	88%	60%
28	5	2	3	4	1	28%	88%	61%
32	5	3	2	4	1	40%	89%	62%
36	5	3	2	4	1	51%	89%	61%
40	5	2	3	4	1	42%	89%	62%
Average						21%	81%	51%

Table 11: Results of **ExpCard** consider also the index cost for the case $|D_i|$ = 1 split, while $|D_j|$ is equal to the number of splits in column #.

#	DJNI	DJGI +2in	DJRE +1in	DJDR +1in	SJMR	DL	GW	GR
1	1	5	4	3	2	1%	58%	40%
3	1	5	4	3	2	68%	64%	50%
6	1	5	4	3	2	51%	64%	49%
9	1	5	3	4	2	17%	61%	46%
12	1	5	4	3	2	25%	58%	43%
15	1	5	3	4	2	6%	57%	41%
18	1	5	3	4	2	12%	53%	37%
21	1	5	3	4	2	12%	54%	38%
24	1	5	3	4	2	7%	52%	37%
28	1	5	3	4	2	3%	53%	37%
32	2	5	3	4	1	5%	50%	33%
36	2	5	3	4	1	5%	49%	32%
40	1	5	3	4	2	1%	50%	33%
Average						16%	56%	40%

Table 12: Results of **ExpCard** consider also the index cost for the case $|D_i|$ = 9 splits, while $|D_j|$ is equal to the number of splits in column #.

#	Djni	Djgi +2in	Djre +1in	Djdr +1in	Sjmr	DL	GW	GR
1	1	5	3	4	2	0%	72%	62%
3	1	5	3	4	2	45%	39%	17%
6	2	5	3	4	1	0%	56%	46%
9	5	4	2	3	1	0%	58%	49%
12	5	4	2	3	1	0%	68%	54%
15	5	4	2	3	1	0%	70%	51%
18	5	4	2	3	1	0%	72%	52%
21	5	4	2	3	1	0%	73%	52%
24	5	4	2	3	1	0%	77%	56%
28	5	4	2	3	1	0%	79%	58%
32	5	4	2	3	1	0%	79%	56%
36	5	4	2	3	1	0%	81%	59%
40	5	4	2	3	1	0%	81%	59%
Average						3%	70%	52%

Table 13: Results of **ExpCard** consider also the index cost for the case $|D_i|$ = 18 splits, while $|D_j|$ is equal to the number of splits in column #.

#	Djni	Djgi +2in	Djre +1in	Djdr +1in	Sjmr	DL	GW	GR
1	1	5	3	4	2	0%	57%	43%
3	2	5	3	4	1	43%	37%	14%
6	5	4	3	2	1	46%	45%	25%
9	5	4	2	3	1	53%	59%	31%
12	5	4	2	3	1	50%	66%	34%
15	5	4	2	3	1	0%	81%	61%
18	5	4	2	3	1	0%	82%	60%
21	5	4	2	3	1	0%	84%	62%
24	5	4	2	3	1	0%	86%	65%
28	5	4	2	3	1	0%	87%	66%
32	5	4	2	3	1	0%	88%	67%
36	5	4	2	3	1	0%	88%	67%
40	5	4	2	3	1	0%	89%	68%
Average						15%	73%	51%

Table 14: Results of **ExpCard** consider also the index cost for the case $|D_i|$ = 28 splits, while $|D_j|$ is equal to the number of splits in column #.

#	Djni	Djgi +2in	Djre +1in	Djdr +1in	Sjmr	DL	GW	GR
1	2	5	3	4	1	5%	52%	36%
3	2	5	3	4	1	66%	33%	12%
6	5	4	2	3	1	45%	57%	29%
9	5	4	2	3	1	47%	68%	35%
12	5	4	2	3	1	47%	75%	42%
15	5	4	2	3	1	51%	77%	45%
18	5	4	2	3	1	0%	87%	66%
21	5	4	2	3	1	0%	88%	68%
24	5	4	2	3	1	0%	89%	69%
28	5	4	2	3	1	0%	90%	72%
32	5	4	2	3	1	0%	91%	72%
36	5	4	2	3	1	0%	91%	73%
40	5	4	2	3	1	0%	91%	73%
Average						20%	76%	53%

Table 15: Results of **ExpCard** consider also the index cost for the case $|D_i|$ = 40 splits, while $|D_j|$ is equal to the number of splits in column #.

#	Djni	Djgi +2in	Djre +1in	Djdr +1in	Sjmr	DL	GW	GR
1	2	5	3	4	1	4%	48%	32%
3	4	5	2	3	1	77%	31%	9%
6	5	4	2	3	1	47%	65%	35%
9	5	4	2	3	1	50%	74%	41%
12	5	4	2	3	1	42%	78%	45%
15	5	4	2	3	1	47%	81%	50%
18	5	4	2	3	1	45%	83%	52%
21	5	4	2	3	1	0%	90%	70%
24	5	4	2	3	1	0%	90%	72%
28	5	4	2	3	1	0%	91%	73%
32	5	4	2	3	1	0%	92%	75%
36	5	4	2	3	1	0%	93%	77%
40	5	4	2	3	1	0%	92%	76%
Average						24%	78%	54%

Table 16: Results of **ExpMBR**. Column "G" is the cardinality in MBytes of D_i, for each of them the cardinality of D_j is changed from 1 to 40 splits.

G	IC	$\|F\|$	$b \in F$	%DL	%GW	%GR	$w \in L$
128	✗	2.1	85%	9.9%	47.6%	21.8%	100%
1152	✗	3.2	77%	32.6%	61.5%	26.2%	100%
2304	✗	2.8	62%	40.0%	69.3%	31.9%	100%
3584	✗	2.7	46%	51.2%	77.3%	33.8%	100%
5120	✗	2.2	15%	52.6%	75.5%	35.8%	100%
128	✓	2.0	100%	20.3%	46.0%	31.4%	100%
1152	✓	1.2	100%	8.8%	67.9%	48.6%	100%
2304	✓	1.4	92%	20.4%	70.2%	47.0%	100%
3584	✓	1.5	92%	34.1%	71.6%	45.3%	92%
5120	✓	1.8	92%	45.1%	71.3%	41.5%	92%

Table 17: Results of **ExpOver**. Column "G" is the cardinality in MBytes of D_i, for each of them the cardinality of D_j is changed from 1 to 40 splits.

G	IC	OV	$\|F\|$	$b \in F$	%DL	%GW	%GR	$w \in L$
128	✗	0.50	2.1	92%	5%	61%	34%	100%
1152	✗	0.50	2.2	85%	44%	73%	43%	100%
2304	✗	0.50	3.0	100%	3%	80%	50%	100%
3584	✗	0.50	2.8	92%	25%	83%	55%	100%
5120	✗	0.50	2.6	92%	20%	86%	60%	100%
128	✓	0.50	2.0	100%	26%	55%	39%	100%
1152	✓	0.50	1.9	100%	30%	57%	34%	100%
2304	✓	0.50	2.2	100%	32%	68%	40%	100%
3584	✓	0.50	2.4	100%	34%	76%	53%	100%
5120	✓	0.50	2.0	100%	21%	76%	53%	100%
128	✗	0.25	2.3	85%	9%	72%	45%	92%
1152	✗	0.25	2.2	92%	82%	81%	57%	100%
2304	✗	0.25	2.2	100%	46%	86%	63%	100%
3584	✗	0.25	2.2	100%	47%	88%	67%	100%
5120	✗	0.25	2.2	92%	42%	84%	57%	100%
128	✓	0.25	2.0	100%	40%	53%	38%	100%
1152	✓	0.25	3.1	100%	53%	48%	21%	100%
2304	✓	0.25	3.1	100%	38%	64%	34%	100%
3584	✓	0.25	2.9	100%	35%	70%	41%	100%
5120	✓	0.25	2.8	100%	31%	73%	46%	92%

8 Conclusion

In this paper we present a cost model for ranking the five spatial join algorithms available in SpatialHadoop 2.4. The cost model proposes for each algorithm some formulas for estimating the cost of the map, shuffle and reduce tasks distinguishing three components: CPU, Local I/O and Network I/O cost. The estimates vary according to the properties of the input datasets in terms of: cardinality, extent and number of vertices of the geometries, and spatial overlapping of the datasets.

Exhaustive experiments using synthetic datasets with variable characteristics allow us to confirm that the proposed cost model is able to detect the best algorithm in 88% of cases and it always gains from a minimum of 21% to a maximum of 67% with respect to the random choice of the algorithm to be executed. In particular, the cost model was able to detect the point at which the best algorithm changes. Future work will regard the extension of the cost model to other index types, currently only grid-index is considered, and to non-uniformly distributed datasets.

References

[1] A. Eldawy and M. F. Mokbel, "SpatialHadoop: A MapReduce framework for spatial data," in *2015 IEEE 31st International Conference on Data Engineering*, April 2015, pp. 1352–1363.

[2] T. White, *Hadoop: The Definitive Guide*, 4th ed. O'Reilly Media, Inc., 2015.

[3] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, "A Comparison of Join Algorithms for Log Processing in MapReduce," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 975–986. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807273

[4] J. Gu, S. Peng, X. S. Wang, W. Rao, M. Yang, and Y. Cao, "Cost-Based Join Algorithm Selection in Hadoop," in *Proceedings of the 15th International Conference on Web Information Systems Engineering*, ser. WISE 2014, 2014, pp. 246–261. [Online]. Available: https://doi.org/10.1007/978-3-319-11746-1_18

[5] A. Eldawy and M. F. Mokbel, "The era of big spatial data," in *31st IEEE International Conference on Data Engineering Workshops*, April 2015, pp. 42–49.

[6] E. H. Jacox and H. Samet, "Spatial Join Techniques," *ACM Trans. Database Syst.*, vol. 32, no. 1, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1206049.1206056

[7] E. G. Hoel and H. Samet, "Benchmarking spatial join operations with spatial output," in *Proceedings of the 21th International Conference on Very Large Data Bases*, ser. VLDB '95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 606–618.

[8] A. Papadopoulos, P. Rigaux, and M. Scholl, "A performance evaluation of spatial join processing strategies," in *Proceedings of 6th International Symposium on Advances in Spatial Databases*, ser. SSD'99. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 286–307.

[9] D. Šidlauskas and C. S. Jensen, "Spatial joins in main memory: Implementation matters!" *Proc. VLDB Endow.*, vol. 8, no. 1, pp. 97–100, 2014.

[10] B. Sowell, M. V. Salles, T. Cao, A. Demers, and J. Gehrke, "An experimental analysis of iterated spatial joins in main memory," *Proc. VLDB Endow.*, vol. 6, no. 14, pp. 1882–1893, 2013.

[11] X. Lin, Z. Meng, C. Xu, and M. Wang, "A practical performance model for hadoop mapreduce," in *2012 IEEE International Conference on Cluster Computing Workshops*, Sept 2012, pp. 231–239.

[12] I. Sabek and M. F. Mokbel, "On spatial joins in mapreduce," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL'17. New York, NY, USA: ACM, 2017, pp. 21:1–21:10.

[13] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The grid file: An adaptable, symmetric multi-key file structure," in *Proceedings of 3rd Conference of the European Cooperation in Informatics – Trends in Information Processing Systems*, A. Duijvestijn and P. C. Lockemann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 236–251.

[14] *Spatial Databases with Application to GIS*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[15] N. An, Z.-Y. Yang, and A. Sivasubramaniam, "Selectivity estimation for spatial joins," in *Proceedings 17th International Conference on Data Engineering*, 2001, pp. 368–375.

[16] W. Aref and H. Samet, "A cost model for query optimization using r-trees." in *Proceedings of ACM GIS*.

[17] A. Eldawy, L. Alarabi, and M. F. Mokbel, "Spatial partitioning techniques in spatialhadoop," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1602–1605, Aug. 2015.

[18] L. Harada, M. Nakano, M. Kitsuregawa, and M. Takagi, "Query processing for multi-attribute clustered records," in *Proceedings of 16th International Conference on Very Large Data Bases*. Morgan Kaufmann, 1990, pp. 59–70.

[19] J. van den Bercken, B. Seeger, and P. Widmayer, "The bulk index join: a generic approach to processing non-equijoins," in *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, 1999, pp. 257–.

[20] J. M. Patel and D. J. DeWitt, "Partition based spatial-merge join," *SIGMOD Rec.*, vol. 25, no. 2, pp. 259–270, Jun. 1996. [Online]. Available: http://doi.acm.org/10.1145/235968.233338

University of Verona
Department of Computer Science
Strada Le Grazie, 15
I-37134 Verona
Italy

http://www.di.univr.it