

Distributed Execution Plans for Multiway Spatial Join Queries using Multidimensional Histograms

Thiago Borges de Oliveira, Fábio Moreira Costa, Vagner José do Sacramento Rodrigues

Universidade Federal de Goiás, Brazil
thborges@ufg.br, {fmc, vsacramento}@inf.ufg.br

Abstract. Multiway spatial join is a common and heavyweight query in spatial data processing. This article presents a cost-based optimizer for multiway spatial join queries, based on a novel multidimensional histogram. The multidimensional histogram has three metrics: cardinality, size of spatial objects and locality of partitions in a cluster. A new method for histogram construction is proposed, together with a formulae to estimate the cost of multiway spatial join queries and select good executions plans. An improvement to Clone Join algorithm that improves its execution time and adapts it to process multiway spatial join queries is also proposed. The evaluations demonstrated that the new method for histogram construction is up to four times better than a previous one in estimating the network communication and plan cost. The complete methodology selected the best plan for all queries in experiments, and provided a nearly exact ordering of the query plans, comparing the real execution time with the estimated cost.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications—*Spatial databases and GIS*; H.2.4 [Database Management]: Systems—*Distributed databases; Query processing*

Keywords: Distributed Processing, GIS, Multiway Spatial Join, Spatial Data

1. INTRODUCTION

The amount of spatial data had greatly increased recently due to popularization of computer devices equipped with Global Positioning System (GPS) and data communication networks, like smart-phones and sensors. Spatial data like geotagged images, open data and census data are continuously collected, organized on thematic datasets and released.

This datasets are stored in Geographic Information Systems (GIS), e.g., spatial databases, and processed using spatial queries. An important query is the spatial join. A spatial join is a query that finds correlated objects in two or more datasets, obeying a spatial predicate like intersection or proximity [Brinkhoff et al. 1996]. A spatial join can be simple, when it processes only two datasets, and complex or multiway, when it process more than two datasets in the same query [Mamoulis and Papadias 2001b]. The former can be seen as a simplified version of the latter, for two inputs.

Multiway spatial join queries are important in several application fields, including geography (e.g., to find animal species living in preservation areas that were damaged by fire), VLSI (e.g., to find circuits that formulate a specific topological configuration) and digital pathology imaging (e.g., to verify on topological images of a brain if a cancer is increasing in size).

Due to complexity of computational geometry algorithms – which evaluate the predicates on spatial data – even queries on small datasets have a high processing cost. Spatial join is also a good candidate for parallelization as spatial data can be partitioned and processed on many computing nodes, and the partial results concatenated to compose the final result. This subject has been extensively researched, and distributed algorithms have been proposed to process spatial join [Patel and DeWitt 2000; Chung

This work was partially supported by CNPq, process number 473.939/2012-6.

Copyright©2014 Permission to copy without fee all or part of the material printed in JIDM is granted provided that the copies are not made or distributed for commercial advantage, and that notice is given that copying is by permission of the Sociedade Brasileira de Computação.

et al. 2005; de Oliveira et al. 2013]. However, the main focus was the processing of simple spatial join query.

Multiway spatial join processing is more complex than simple spatial join because a query can be executed in many different ways, called execution plans. An execution plan is a definition of the order in which datasets will be combined and the algorithms that will be used to find the final result of the query. The number of execution plans for a query is non-linear with respect to the number of datasets, as discussed in [Mamoulis and Papadias 2001b]. The execution plan has a huge impact on the processing cost of a query, and thus needs to be carefully determined.

In a distributed system, besides considering local CPU and I/O costs, the selection of execution plans must take into account the approach used to do the data partitioning and the communication needed among machines. The proper balance of query has to be considered, choosing plans that divide the work evenly among nodes, considering both the bandwidth limit in network interface and load on CPU.

This article proposes a set of methods that together composes an optimizer to process multiway spatial join queries in distributed systems. The major contributions are:

- Identification of characteristics of datasets and data distribution, which are relevant for efficient processing of multiway spatial join queries in distributed systems;
- Definition of a multidimensional histogram to organize these characteristics, observing data skewness of real spatial datasets;
- A new method to construct the multidimensional histogram, that improves estimate of query processing cost;
- An algorithm that estimate the cost of execution plans in distributed systems, using multidimensional histograms; and
- A greedy algorithm to schedule data copy between cluster nodes, which reduce bandwidth usage while maintaining load balance in the cluster.

The remaining of this article is organized in the following way: in Section 2 we present a survey of multiway spatial join processing, describing key concepts, estimation techniques for plan selection, algorithms for processing of multiway join queries, as well as a discussion in each subsection about how to adapt these methods to distributed systems. In Section 3, we describe the new multidimensional histogram and the methods for histogram construction, the algorithm for estimating network I/O, the data copy scheduling algorithm, and a formulae for selection of execution plans. Section 4 presents the evaluation of methods proposed and discuss the results. Section 5 presents related work, and finally, Section 6 presents the conclusion and future work.

2. MULTIWAY SPATIAL JOIN PROCESSING

Multiway spatial join is a technique of combining successive join algorithms to process more complex spatial join queries with more than two input datasets [Papadias et al. 1999]. A multiway spatial join can be represented as a graph $G(V, E)$, where each node represents a dataset and the edges represent the join predicates [Papadias et al. 1999; Mamoulis and Papadias 2001b].

Many distinct execution plans can be used to process a multiway spatial join query. The number of ways a query can be processed was investigated in [Mamoulis and Papadias 2001b] for serial processing (non-parallel, non-distributed). It can be determined by the query type, the number of input datasets, and how many different join algorithms can be used at each step.

Figure 1 illustrates some alternative plans to process a chain query graph (1a). In 1b the datasets are pairwise joined in a first step, producing two intermediate results to be joined in a second step.

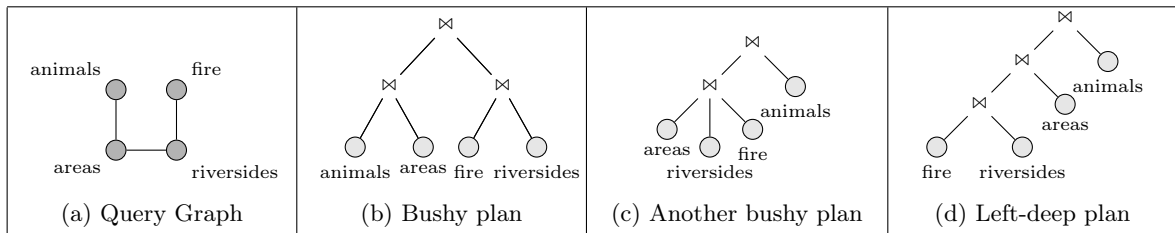


Fig. 1: Alternate execution plans to process a multiway spatial join query.

In 1c three datasets are joined in a single step and the intermediate result is later joined with the animals dataset. In 1d fire and riversides are joined together and in subsequent steps the intermediate results are processed with the other two datasets, one at a time.

All the execution plans for a query maintain its semantic, but can take different amounts of time to produce the final result. The graph that represents a query is used as an input for a plan definition algorithm. The plan definition algorithm is also called optimizer. The optimizer considers some aspects of the datasets and the associated costs of the algorithms to select an execution plan that determines: (i) how the datasets will be combined, (ii) what is the processing order and (iii) what algorithms will be used in each step. Despite the name, an optimizer, in general, is not an exact algorithm. The next section will discuss how the computational cost could be determined.

2.1 Estimating the Cost of Spatial Join for Plan Selection

Despite the complexity to estimate the cost of execution plans, some studies have proposed formulae that estimate the cost of simple spatial join queries [Acharya et al. 1999; Sivasubramaniam 2001; Fornari et al. 2006], and how to combine the formulae to estimate the cost of complex spatial join queries [Mamoulis and Papadias 2001a]. These formulae assume that the spatial extent is filled uniformly on the dataset and takes the I/O and CPU cost of the join algorithm to estimate the cost of an execution plan. The difficulty of using these formulae on real datasets was latter studied by the same authors [Mamoulis and Papadias 2001b], which concludes that, when used on real spatial datasets, the formulae can conduct to bad execution plans, specially in the presence of dataset skewness.

The alternative proposed by Mamoulis and Papadias [2001a] was the use of this formulae in small regions of the dataset. They proposed the use of a 2D spatial histogram that represents the density of small regions of the dataset, as illustrated in Figure 2b. In the Figure, the spatial dataset of fire warnings for the Brazilian cerrado biome has been divided with a grid of 2500 cells (50x50).

The cardinality value of each cell, $C(x)$, is defined based on the cardinality of the set of spatial objects that have their MBR center within the limits of cell x . The cardinality of the output result C_o of a spatial join can be estimated for two histograms (H_r and H_s), constructed from the R and S spatial datasets, through the formula $C_o = \sum C(c_i) * C(c_j)$, $\forall c_i \in H_r, c_j \in H_s \mid c_i \cap c_j \neq \emptyset$ [Mamoulis and Papadias 2001b], considering that spatial objects are uniformly distributed in each cell.

In our experiments, we concluded that the use of the MBR center, as proposed in [Mamoulis and Papadias 2001b], generates bad cost estimations compared with the real execution of the algorithms. We proposed and evaluated a new method for histogram building, called Proportional Overlap Method, that is described in Section 3.1.

Besides the cardinality of the datasets, the size of spatial objects (how many points) and the place (server or processor) where the object is located is also relevant for cost estimation in distributed systems. We improved the 2D histogram to represent this two new values and developed an algorithm to estimate the cost of a spatial join, based on the three metrics: cardinality, points, and place. The details is discussed in Section 3.

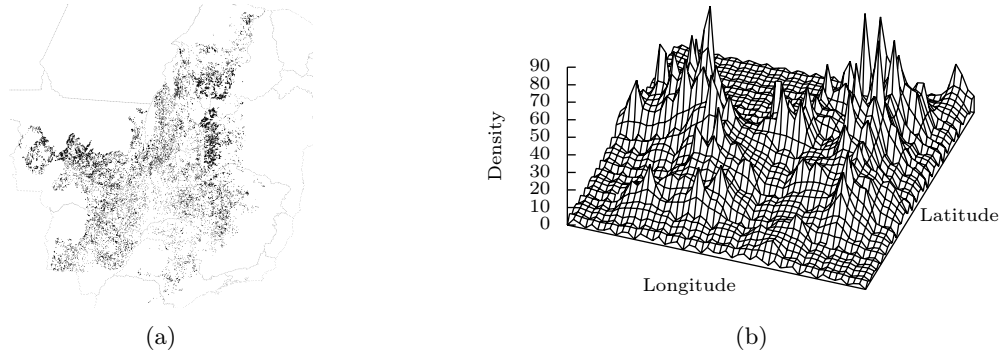


Fig. 2: Fire warnings for the Brazilian cerrado biome (a) and the 50x50 histogram generated for the dataset (b).

There are also other types of histogram proposed for spatial data. One such type is the Min-Skew histogram, with variable cells size, proposed in [Acharya et al. 1999]. This histogram can be used to improve the estimation on very skewed datasets. However, the complexity of building and estimating the cost with this histogram is greater than with the histograms with fixed cells, and this has an impact on the total execution time of a query. We carried out experiments with histograms with fixed cells and demonstrated promising results even with real and skewed datasets, as is shown in the evaluation presented in Section 4.

2.2 Algorithms for Distributed Processing of Multiway Spatial Join

To process a spatial join in distributed or parallel systems a data partitioning method needs to be used. A data partitioning method divides the dataset objects into groups, called data partitions or cells, that are assigned to servers in the filtering or refinement step. The methods for data partitioning can be classified in two main types:

- (1) Disjoint space partitioning, that uses a grid to split the space extent. Each cell of the grid is used to group the spatial objects according to their intersection with the cells. The cells do not intersect each other. Objects that intersect more than one cell are replicated, and
- (2) Non-disjoint space partitioning, in which partitions can overlap each other, to accommodate the extent of the objects that intersects it. The spatial objects are not replicated. An example of this type of partitioning is the set of MBR's on a level of an R-Tree index.

We found distributed algorithms for these two types of data partitioning in the literature. For non-disjoint space partitioning there are: *Replicated Semi-packed Parallel R-Tree* (RSPR) [Mutenda and Kitsuregawa 1999], *Proximity Area Spatial Join* (PASJ) [de Oliveira et al. 2013] and *Distributed Synchronous Traversal* (DST) [Cunha et al. 2015], all of them using distributed R-Tree indexes to process the spatial join. For disjoint space partitioning there are *Clone Join* (CJ) [Patel and DeWitt 2000], *Shadow Join* (SJ) [Patel and DeWitt 2000] and *Non-blocking Parallel Spatial Join* (NPSJ) [Naughton and Ellmann 2002]. CJ is the simplest of the three algorithms. It uses a fixed grid to partition the datasets and replicate spatial objects that intersect more than one grid cell. The difference between CJ and SJ is that the latter uses a more sophisticated technique to reduce object replication. The NPSJ algorithm also uses a fixed grid and replicate objects as CJ. However, it is designed as a non-blocking algorithm, that produces results early on the execution. Another difference between NPSJ and the others is that it creates local R-Trees on each data partition and produces the results using an *R-Tree Join* (RJ) [Brinkhoff et al. 1993], while CJ and SJ use a *Partition-based Spatial Merge Join* (PBSM) [Patel and DeWitt 1996].

An issue in distributed spatial join processing is the distribution of the data partitions. The simpler and most used distribution method is round-robin. It distributes the partitions evenly between the servers, favoring load balance in the cluster. Other methods that favor the colocation of the spatial data have also been proposed, such as Proximity Area [de Oliveira et al. 2013], for non-disjoint partitioning. The Proximity Area method distributes the partitions based on their location, favoring reduction of network bandwidth usage, during spatial join execution.

In a round-robin distribution, data partitions located in the same geographic region, but from different datasets, will often be distributed to different servers. The join algorithm thus needs to copy the intersecting partitions from different servers to execute the join, but the predicate checking will be more balanced among the nodes of the cluster. As discussed in [de Oliveira et al. 2013], distributing the partitions based on their location will reduce network bandwidth usage, but the load balancing of spatial join execution will be compromised. Due to spatial join being a CPU-bound algorithm, we implemented the round-robin method and we carried out experiments to measure the impact of communication on the total execution time of the queries. Despite this choice, the methods we propose in Section 3 were designed for use with any data distribution, as is demonstrated in Section 3.3.

Following the results reported in [Patel and DeWitt 2000], we used a disjoint space partitioning. Each grid cell was used to group and assign the spatial objects to the servers. For join execution, we used the CJ algorithm with the duplication avoidance technique, based on the Reference Point method. The CJ algorithm is simpler than NPSJ and duplication avoidance also transforms it into a non-blocking algorithm. The next section cover the CJ algorithm and the Reference Point Method in more detail.

2.3 Clone Join and Reference Point Method

Clone Join (CJ) [Patel and DeWitt 2000] is a distributed spatial join algorithm for joining two non-indexed datasets. The data partitioning used by the algorithm is a grid of disjoint cells, each cell representing a small part of spatial extent. The spatial objects are assigned to each cell based on the intersection between their MBR and the cell boundaries. Objects that intersects more than one cell are replicated. Each cell, and the corresponding spatial objects, form a data partition and is assigned to one server using a round-robin distribution or a similar hash function applied on the cell number. The join is performed in parallel on each server, using a local *Partition-based Spatial Merge Join* (PBSM) [Patel and DeWitt 1996], which is a parallel spatial join algorithm for shared memory parallel systems.

Patel and DeWitt [2000] assume that the two datasets are partitioned with the same grid, at the beginning of the execution of the join. This can be a problem on a database system, as distribution will occur every time a join is executed using the dataset. The datasets can also be inserted at distinct times or have different geographical space extents.

Because of object replication, one additional step is necessary to eliminate duplicate results reported by the refinement. A duplicate result is reported whenever two spatial objects that intersect each other are replicated on two or more cells, and the cells are distributed to different servers. Each server will report the intersection between the objects individually.

Patel and DeWitt [2000] propose a distinct operator, to be executed at the end of the join to eliminate the duplicate results. This operator needs to check each result reported against all the others, and has a significant cost. It also needs to maintain an intermediate result to do this verification.

A solution for the result duplication problem, when replicating spatial join objects, was been proposed in [Dittrich and Seeger 2000]: the Reference Point method. The method consists in identifying the possible cells that will report duplicate results and allowing only one of them to report the result. The method was originally proposed for the PBSM algorithm and later adapted in NPSJ algorithm

[Naughton and Ellmann 2002], a spatial join algorithm for distributed systems. However, it could be applied to CJ as well, as is proposed in Section 3.2.

2.4 Selecting an Execution Plan for Multiway Spatial Join

The size of the space of possible plans to select from is non-linear with respect to the number of datasets. In [Mamoulis and Papadias 2001b], there is a study of the number of possible plans, considering three different algorithms to process a spatial join in a non-distributed system. The recurrence $P(n) = 1 + 2P(n-1) + \sum_{2 \leq k < n-1} P(k)P(n-k)$, with $P(2) = 1$, gives the amount of plans, $P(n)$, for a chain query with n datasets [Mamoulis and Papadias 2001b]. In asymptotic terms, $P(n) = \Omega(2^n)$. Cycle queries and clique queries has even more possible plans. That means that, if a query has enough datasets, an optimizer can take more time planning than executing the query if it enumerates all possible plans and computes the cost of each one.

In distributed systems, besides the combination order of datasets, there are different strategies to copy data partitions during join execution. Some options are *i*) copy smaller data partitions to server where largest partitions are located; *ii*) copy partitions in such a way that network contention does not make processors sleep, and *iii*) copy partitions to maintain cluster balance. These choices can also be considered to select the execution plan, increasing even more the amount of possible plans. We propose a greedy algorithm for this problem in Section 3.3.

To quickly identify good execution plans while searching only a small fraction of the space, Mamoulis and Papadias [2001b] proposed an heuristic that randomly transforms an execution plan (seed) using a set of pre-defined rules, like associativity and commutativity, together with methods such as iterative improvement and simulated annealing. In the experiments, the heuristic method found plans only slightly more expensive than the optimal execution plans found by the exhaustive method. The proposed algorithm is a good strategy for queries with a large amount of datasets. In our work, we are using only the exhaustive method and left for future work the implementation of this randomized algorithm.

3. MULTIDIMENSIONAL HISTOGRAM AND DISTRIBUTED OPTIMIZER

In this section we describe how to combine the parameters of datasets and data distribution in the data structure and access method denominated multidimensional histogram.

A multidimensional histogram is a data structure that divides the spatial extent of a dataset using a grid with cells of fixed size, and records for each cell: *i*) the amount of spatial objects within the boundaries of the cell (cardinality), *ii*) the size of objects (amount of points of all geometries), and *iii*) the place (server on cluster) where the cell is located. The first metric is used to estimate the cardinality of the join output, as already done for non-distributed spatial join execution. The second metric is used to estimate intermediate histograms used in selection of execution plan for a spatial join query. The three metrics together can be used to estimate network bandwidth usage, as is detailed in Section 3.3.

Figure 3 shows a visual representation of these three metrics of a multidimensional histogram generated for political limits of Brazilian counties. There are important aspects of this real dataset represented in the figure: in top left corner of Figure 3a, cardinality is low because on this region there are counties of Amazon forest. They are large in size, but small in quantity. In contrast, Figure 3b shows the points metric, for the same counties. The amount of points of these counties is much higher than in others regions, exactly because their area (extent) is larger and more points are needed to represent their contour. The Figure 3c is a map for the third metric: locality. The cells of this histogram has been distributed in a cluster with 8 servers, using a round-robin algorithm. The legend indicates the server where the cell was located (1 to 8). The zero (black) represents a cell that not contains any object and do not need to be sent to a server.

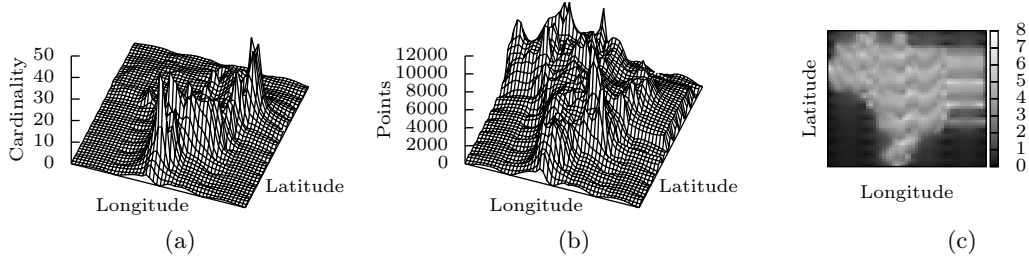


Fig. 3: Multidimensional histogram for a two dimensions dataset of political limits of Brazilian counties. Representation of cardinality metric (a), points metric (b), and the location of partitions in the cluster (c) in gray tones, as show in the right legend.

3.1 Proportional Overlap Method

To build the histogram, each object in a dataset is hashed in the histogram grid, to find the cell or the set of cells that it intersects. The set of objects hashed to a cell forms a group or data partition. We use the term cell or data partition interchangeably in the remaining text.

We propose a new method to build the multidimensional histogram, called Proportional Overlap method. Instead of using the center of the MBR to hash a spatial object into histogram, as proposed in [Mamoulis and Papadias 2001b], we used the proportional covering area of each MBR, as described in the following.

For the cardinality metric, the method consists in incrementing the value stored in each cell based on the proportion of two areas (a/b): a is the area of intersection between the spatial object MBR and the histogram cell boundaries, and b is the area of the spatial object MBR. Formally, let H_r be the histogram of the dataset R , and $o \in R$ an object of the dataset. The cardinality $C(x)$ of a cell $x \in H_r$ is given by Formula 1.

$$C(x) = \sum_{o \in R} \text{area}(\text{MBR}(o) \cap \text{MBR}(x)) / \text{area}(\text{MBR}(o)) \quad (1)$$

Figure 4 shows a spatial object being hashed into an histogram. In the MBR center method (left), the MBR center of the object is used to identify the histogram cell. The cardinality metric of the cell identified is increased by one. The Proportional Overlap Method (right in the Figure 4) increments the proportion computed with the Formula 1 in each cell that the MBR of the object intersects.

For the points metric, the amount of points is added to each cell that the object intersects. This reflects the behaviour of CJ algorithm, that replicate an object in each cell it intersects. The quantity of points $P(x)$ in a cell x is given by Formula 2.

$$P(x) = \sum_{o \in R \mid \text{MBR}(o) \cap x \neq \emptyset} \text{Points}(o) \quad (2)$$

After the hashing process, the locality metric is defined based on the distribution of data partitions. We used a round-robin algorithm to distribute data partitions. For each cell of the histogram, a server is taken from a circular list to be the container of the cell. The data partitions are copied to servers, according to distribution, at the end of the histogram construction.

3.2 Clone Join with Reference Point Method

As described in Section 2.3, *Clone Join* (CJ) [Patel and DeWitt 2000] is a simple and efficient algorithm to process a distributed spatial join. In this section we describe how to adapt it to process multiway spatial join queries efficiently.

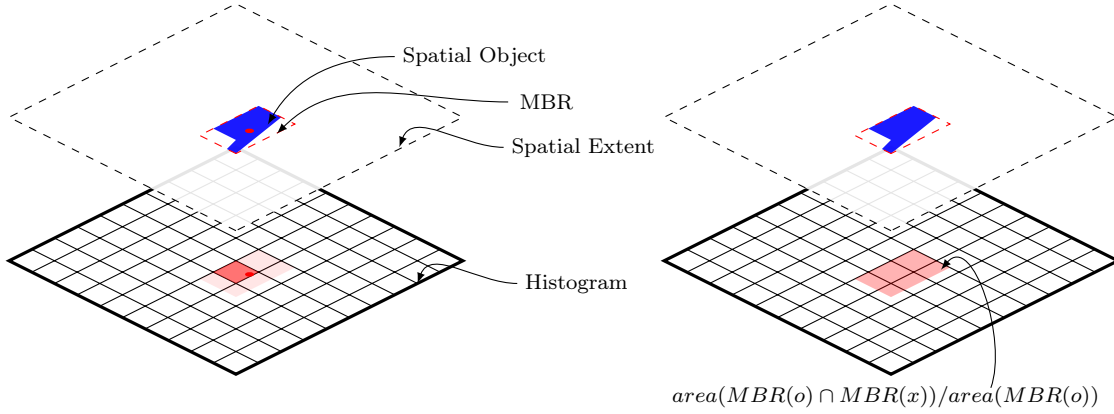


Fig. 4: An spatial object being hashed using MBR Center (left) and Proportional Overlap Method (right). In the formula, o is a spatial object and x is an histogram cell.

The CJ algorithm has two drawbacks when dealing with multiway spatial join query execution: *i*) the distinct operator is a step with high communication cost, and *ii*) the distinct operator prevents multiway from being processed in a pipeline format, i.e., the distinct operator acts like a barrier, preventing that the next step of a plan starts processing until its end, causing system contention.

As discussed in [Dittrich and Seeger 2000], the Reference Point method can be used on any algorithm for spatial join, since it uses disjoint partitioning with object replication. As CJ has this characteristics, we propose the following changes to implement the Reference Point method on CJ algorithm:

- (1) Distribute and maintain the cells boundaries as a metadata along with the spatial objects in the data partitions;
- (2) Remove the distinct operator at the end of the algorithm;
- (3) Change the predicate check algorithm:
 - (a) Whenever a pair of objects need to be verified by the predicate check algorithm, compute the Reference Point with the MBR of the two objects and check if it is within the boundaries of the cell being processed;
 - (b) If the Reference Point is not within the cell boundaries, ignore the object pair, as it will be reported in another server. Otherwise, continue with the predicate check algorithm in the pair.

As the distinct operator is deleted, the modified algorithm do not have the two drawbacks discussed above. Furthermore, the modified algorithm also has the non-blocking characteristic proposed in more recent algorithms, such as *Non-blocking Parallel Spatial Join* (NPSJ) [Naughton and Ellmann 2002], that improves further the multiway spatial join query execution due to the early reporting of results to the next steps of a multiway spatial join query.

3.3 Intermediate Histogram, Data Copy and Plan Cost

To process a multiway spatial join in a distributed system, we need to move the partitions of the datasets that are distributed in the cluster following the semantics of the query predicate. In this section, we describe how to estimate the cost of the data movement and the cost of a plan, through the creation and use of intermediate histograms.

The data movement cost of a plan, or communication cost, can be estimated using the three metrics of the multidimensional histogram of datasets in the following way: Let H_a and H_b be two histograms,

for a and b datasets, respectively, $C(c_x)$ the cardinality of cell c_x , $P(c_x)$ the quantity of points in cell c_x , and $L(c_x)$ the set of servers where cell c_x is stored. A intermediate histogram and the communication cost could be obtained using Algorithm 1.

The Algorithm 1 is used to perform three tasks: *i*) to determine a data copy strategy based on the pre-distribution of datasets, *ii*) to estimate the communication cost of joining a and b , and *iii*) to estimate an intermediated histogram, H_r , for the result of the spatial join between datasets a and b . This intermediated histogram will be used in the estimation of posterior join steps of multiway spatial join query. To construct the H_r histogram, each pair of cells from H_a and H_b , intersecting each other, are evaluated. The cardinality and points metrics of this histogram are estimated observing the proportional overlap, described in Section 3.1, and the F function, on lines 5 to 8 of the algorithm.

The algorithm uses a greedy strategy to determine data copy and to define the locality metric of H_r histogram (Lines 14 to 26). The procedure CHOOSEMINCOMMSERVER returns the server that needs minimum communication to put the cell pair ca and cb in the same place: If $L(ca) \cap L(cb) \neq \emptyset$, the cells has a server in common and it is returned, otherwise a server of $L(ca)$ is returned if $P(ca) < P(cb)$ or vice versa. The estimation of the communication is based on the amount of points copied and the size of each point in the system (two double floating points)¹, and is computed separately for each server in the vector IO (Line 2). The amount of points to be compared in each server is also computed in the vector $Pnts$ (Line 3). The standard deviation of values in the $Pnts$ vector gives an estimation of the plan load balance. If the difference between the server with minimum and maximum amount of points reaches a certain *tradeoff*, instead of choosing the server with minimum IO, the server with minimum amount of points is chosen. This procedure attempts to maintain low communication, but if the balance in the cluster drops, the communication is sacrificed at the expense of maintain the load balance. The return of the algorithm is used in plan selection, described in the Section 3.4.

3.4 Distributed Execution Plan Selection

To select an execution plan for a query, the Algorithm 1 is called in each step of the multiway join. In the first step, the histogram of two datasets is used. In intermediate steps, a intermediate histogram created by the algorithm is used together with a histogram of other dataset in the query. The vector $Pnts$ returned by the Algorithm 1 is aggregated for all steps in a final vector for the plan, named $PlanPnts$.

As a multiway spatial join query has many possible plans, each plan is estimated using this procedure and generates a $PlanPnts$ vector. Let $AllPlans$ be the set of vectors $PlanPnts$, one for each plan $p \in P$, we propose that the best distributed execution plan for a query is the plan with cost O :

$$O = \text{Min}(\bigcup_{p \in P} \text{Max}(AllPlans[p])) \quad (3)$$

The first part of Formula 3, $\text{Max}(AllPlans[p])$, selects the maximum value of $PlanPnts$. This value refers to the server with the maximum estimated load for plan p . This server probably will be a CPU bottleneck in execution, because the execution of the predicate check algorithm will last longer due to the amount of points in the server. To conclude the selection, the latter part of formula, $\text{Min}(\dots)$, selects in the set of maximum values, the plan with the minor amount of points.

¹We designed the system transfer protocol to send the spatial object data in binary format. Its overhead is a constant on the number of cells and objects transferred.

BUILDHRANDIO(H_a, H_b)

```

1   $H_r$  = new empty histogram, based on the cells limits of  $H_a$  and  $H_b$ 
2   $IO[servers] = 0$ 
3   $Pnts[servers] = 0$ 
4  for each  $ca \in H_a, cb \in H_b$ , such that  $ca \cap cb \neq \emptyset$ 
5       $d_a = area(ca \cap cb) / area(ca) * C(ca)$ 
6       $d_b = area(ca \cap cb) / area(cb) * C(cb)$ 
7       $p_a = P(ca) / C(ca) * d_a$ 
8       $p_b = P(cb) / C(cb) * d_b$ 
9       $cr$  = the cell of  $H_r$  to be filled, based on the boundaries of  $ca, cb$ 
10      $C(cr) = d_a * d_b$ 
11      $P(cr) = p_a$  or/and  $p_b$ , based on the next step predicate
12
13     // Determine on what server the cell pair will be processed
14      $s = \text{CHOOSEMINCOMMSERVER}(ca, cb)$ 
15      $balance = (\text{MAX}(Pnts) - \text{MIN}(Pnts)) / \text{MAX}(Pnts)$ 
16     if  $balance > tradeoff$ 
17          $s = \text{MINIDX}(Pnts)$ 
18      $Pnts[s] = P(ca) + P(cb)$ 
19      $IO[s] = \text{COPYAMMOUNT}(ca, cb, s)$ 
20      $L(ca) = L(ca) \cup s$ 
21      $L(cb) = L(cb) \cup s$ 
22     for  $s = 1$  to  $servers$ 
23         if  $s \notin L(ca)$ 
24              $IO[s] += P(ca)$ 
25         if  $s \notin L(cb)$ 
26              $IO[s] += P(cb)$ 
27     return  $IO, Pnts, H_r$ 

```

Algorithm 1: Algorithm to estimate the communication cost and to create H_r for a spatial join between datasets a and b .

4. EVALUATION

To evaluate our proposal, we choose a set of real spatial datasets, obtained from Brazilian Institute of Geography and Statistics² (IBGE), from LAPIG laboratory³ of UFG Institute of Social and Environmental Studies (IESA) and from the new version of Digital Chart of the World⁴(DCW). The selected datasets and its characteristics are detailed in Table I. Datasets with small cardinality are selected to make experiments with selective execution plans, in which one of the datasets restricts almost totally the query output. The datasets with more complex spatial objects, like rivers and roads contours, were selected due to the difficulty and inaccuracy of their representation on histograms.

Queries used in experiments are listed in Table II. All queries are multiway spatial join of practical application, and they involve all datasets. The queries are of type chain and the spatial predicate of each join is intersection of the first with the second dataset. The queries with three datasets has two execution plans each, maintaining its semantics. The queries with four datasets has six execution plan each. The execution plans are presented in the format $Q1..8 P_{1..6}$, for example, $Q1 P_1, Q1 P_2$.

²www.ibge.gov.br

³www.lapig.iesa.ufg.br/lapig/

⁴<http://gis-lab.info/qa/vmap0-eng.html>

Table I: Datasets used in experiments.

Name	Abrev.	Type	Cardinality	SHP File Size (MB)
<i>Brazilian datasets (IBGE and LAPIG)</i>				
Seaport	<i>P</i>	Points	120	<1,0
Vegetation	<i>V</i>	Polygons	2.140	4,7
Counties	<i>M</i>	Polygons	5.564	38,8
Fire alerts	<i>A</i>	Polygons	32.578	11,2
Roads	<i>R</i>	Lines	51.646	15,2
Rivers	<i>H</i>	Lines	226.963	64,5
<i>Word wide datasets (DCW)</i>				
Crops	<i>CU</i>	Polygons	123.746	69,3
Rails	<i>FM</i>	Lines	194.261	28,7
Inland Water	<i>RA</i>	Polygons	338.860	136,7
Elevation Contour	<i>CR</i>	Lines	703.574	572,5
Rivers	<i>HM</i>	Lines	943.638	243,2

Table II: Multiway spatial queries used in experiments.

Abrev.	Query	Main characteristic	Join Cardin.
Q1	$FM \bowtie HM \bowtie CU$	Join with polygons and lines	2313
Q2	$A \bowtie HM \bowtie FM \bowtie CU$	Small set output	168
Q3	$HM \bowtie FM \bowtie CR$	Lines bigger datasets	2659
Q4	$HM \bowtie CR \bowtie RA$	Bigger datasets	771
Q5	$V \bowtie A \bowtie M$	Small datasets, colocated	36.469
Q6	$A \bowtie P \bowtie M \bowtie V$	Null output set	0
Q7	$HM \bowtie CR \bowtie FM \bowtie R$	All lines datasets	3.139
Q8	$RA \bowtie CU \bowtie A \bowtie M$	All polygons datasets	26.128

The experiments presented in the next sections were performed in Azure Platform, using eight A2 virtual machines, with 2 vCPUs and 3.5 GB of RAM each. The machines were allocated in the same data center and are interconnected by a virtual network. To the best of our knowledge, there are no public document from the provider that informs network interface bandwidth. However, we experienced a limit in network interface near 200 Mbps.

In our experiments we used $tradeoff = 0.2$ in the Algorithm 1, since this value provided a good trade off between load balance and communication. Further reducing this value resulted in a small improvement in load balance with a huge increase in the communication. This occurs because the metric for balance is an approximation, and using it to force a better estimated balance not always resulted in a better balance in the execution of the query.

4.1 Proportional Overlap Method Evaluation

This section presents the experiment designed to evaluate the Proportional Overlap Method, used to construct the cardinality and points metrics of multidimensional histogram, described in Section 3.1. The experiment consists in creating a multidimensional histogram for all datasets present in the queries in Table II, using both Proportional Overlap Method and MBR center method [Mamoulis and Papadias 2001b], and executing Algorithm 1 in each query plan to obtain the estimated total of bytes to be transfered in each method. The estimated values are compared with the real communication value, collected from a real execution of the plan in the cluster. To compare the values we computed how distant estimated values are from real ones, through formula $error = abs(real - estimated)/real * 100$.

The result of the experiment is presented in the Figure 5. The chart shows the error for each plan and method. The last two bar groups has the average and maximum error between all plans. The error of MBR center method is $\approx 50.7\%$ in average for all plans, and in the worst case ($Q6 P_5$) it is 90.19%. Proportional Overlap method has an average error of $\approx 28.7\%$ and a maximum of 78.80% (also for $Q6 P_3$). These values shows the effectiveness of Proportional Overlap method proposed, in

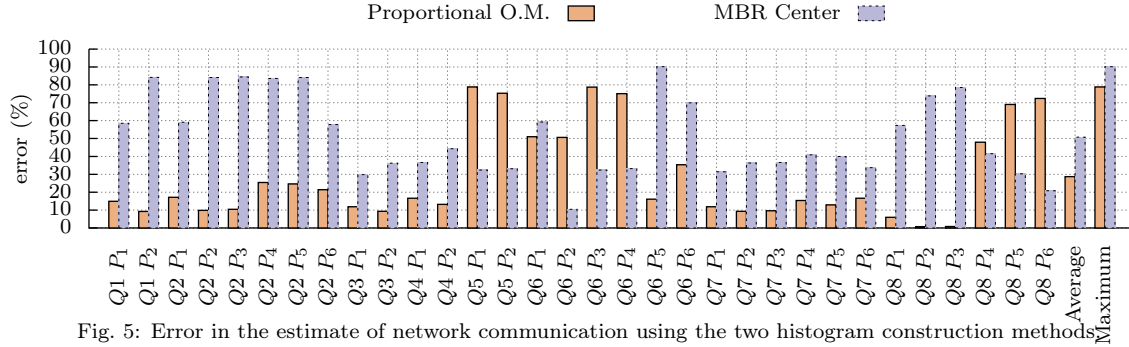


Fig. 5: Error in the estimate of network communication using the two histogram construction methods

constructing a better histogram to estimate the communication cost of plans. In the next experiments, we decided to use only Proportional Overlap method, since it is more precise.

4.2 Estimated Communication Evaluation

In this section we describe the experiment designed to evaluate Algorithm 1, proposed to generate intermediate histograms and to aggregate the number of spatial object points and network I/O per server. Algorithm 1 is an important part of the method for plan selection, since it estimates the number of points and communication, as well as the load balance in the cluster. The experiment consists in obtain the estimated communication for each plan and server, as well as the server with maximum and minimum estimated communication, and compare the obtained values with real ones measured in the execution of plans in the cluster. As the estimated communication is based on the number of points transfered, and the number of points is obtained from histograms, a good estimation indicates that the methodology could be used to get plan costs that will be used in plan selection.

The results of the experiment is shown in Figure 6. To better visibility and comparison of values, the chart has two sections, each with a separately y axis. Right section shows four plans ($Q3 P_2$, $Q7 P_2$, $Q7 P_3$ and $Q8 P_2$) with larger y values. Each bar in chart represents the average of estimated and real network communication, between all servers. The total amount of communication for each plan can be calculated multiplying the value of the bar by eight (the quantity of servers in the test), and ranges from $20.13MB$ ($Q6 P_6$) to $1.47GB$ ($Q7 P_3$). Each bar also has a superior and inferior limit, which represent, respectively, the server with max and minimum amount of network communication. A huge variation on this limits indicates that the network load will be unbalanced in the cluster.

As can be seen in Figure 6, the estimated values is different from the real ones, besides it is visually very approximated. This behaviour also repeats for error bars, comparing the estimated maximum and minimum error with the maximum and minimum real values. This confirms that the algorithm provides a good estimate of the total network communication, as well as the network communication for each server. There are a set of plans that present a distinct behaviour: $Q5 P_{1..2}$, $Q6 P_{2..4}$, and $Q8 P_{4..6}$. While in all others plans the estimated value is below the real one, these plans shows the inverse. This behaviour occurs due to the propagation of the error in the estimation of the size of the objects, in the line 11 of Algorithm 1. Because an average of the object size is propagated to the intermediate histogram, big objects that could be filtered in the real join execution remain in the estimation, causing this error.

The result reflects the simplification of datasets made in histogram construction. The method can not precisely estimate due to this simplification. However, the data shows the proximity of the estimation with the real values, in all plans tested. We conclude that Algorithm 1 and the associated methodology can provide a good estimate of communication of each plan, based in data distribution and the three metrics of the multidimensional histogram.

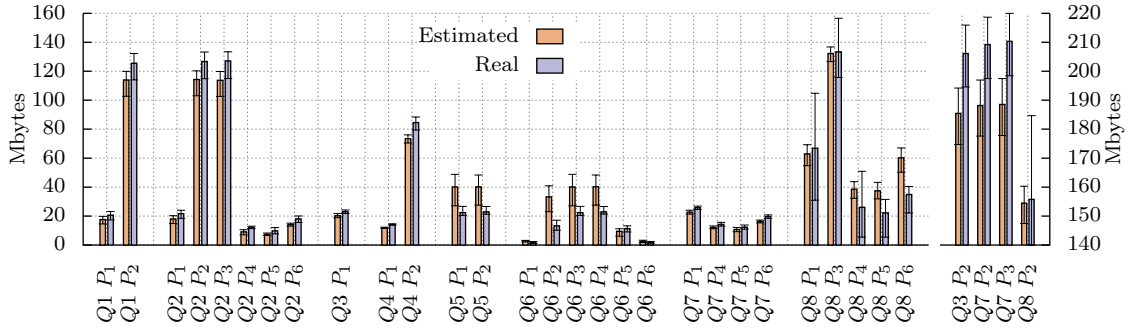


Fig. 6: Average server communication: estimated and real for each query. The error bars represent the minimum and maximum network communication between all servers.

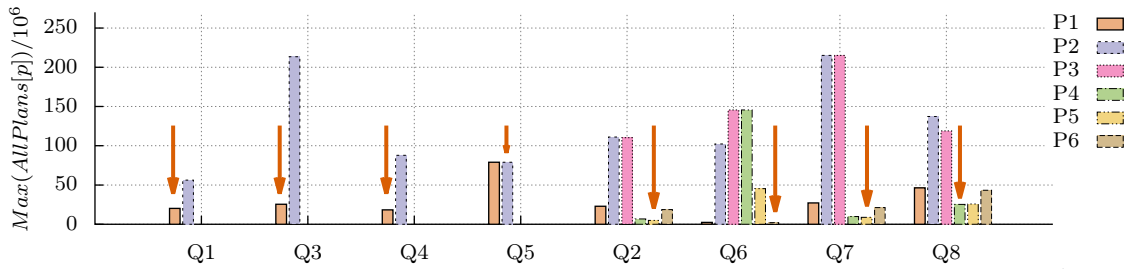


Fig. 7: Estimated maximum number of spatial points in a server for each plan, after query scheduling by Algorithm 1.

4.3 Distributed Execution Plan Selection

In this section we describe the evaluation of plan selection methodology, described in Section 3.4. The experiment consists in selecting an execution plan using the estimate provided by Algorithm 1 and Formula 3, and, thereafter, get the execution time of all plans in the cluster to check if the plan selected with the estimate is the plan with minor execution time in the cluster.

Figure 7 shows the estimated cost of all query plans, based on the number of points of the spatial objects, and an arrow to indicate the plan selected by the method. Each bar in the chart indicates the estimated amount of points to be compared (in millions) by the server with the highest amount of points (The $Max(AllPlans[p])$ part of Formula 3). The lowest bar in each group is the selected plan.

The execution time of each of the 32 plans is presented in the Figure 8. The chart uses a logarithmic scale due the difference between execution time of the best (seconds) and worst plan (hundreds of seconds). The chart shows that the selected plan is the best for all queries in the experiment. Furthermore, the method could establish a relative ordering of all plans for almost all queries. The only two differences in the ordering are $Q8 P_5$ and $Q8 P_6$, inverted between estimate and real execution (0.8 seconds of difference between the plans), and $Q6 P_{2/5,4/3}$, also different from estimate, but only by 0.2 seconds, in average. This error in ordering could lead to wrong choice of the best plan in a scenario where the error involves the best plan. However, as can be observed in the chart, the error always occurred with very similar plans with respect of their execution time. A wrong choice in this scenario still will be a good plan, with a slight worst execution time. This confirms that the number of points of the spatial objects is a good metric to determine the cost of the execution plans, and confirms that the Formula 3 selects a good plan based on the estimated cost of plans.

5. RELATED WORK

Despite the importance of multiway spatial join queries and the widespread use of spatial datasets, limited work has been proposed to efficiently process multiway spatial join queries in distributed systems. In this section, we describe the most relevant work and some studies that propose isolated

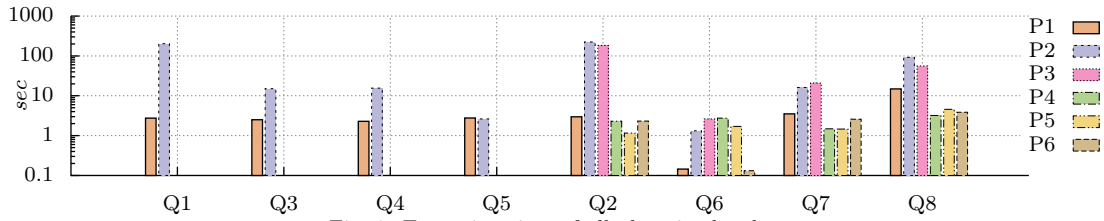


Fig. 8: Execution time of all plans in the cluster.

contributions that can be combined or adapted to form a complete solution for distributed spatial join query processing.

For non-distributed processing of multiway spatial join, and to the best of our knowledge, the study in [Mamoulis and Papadias 2001b] is the most relevant in the literature. The authors make an in-depth evaluation of algorithms, data partitioning techniques, plan cost estimation, plan selection and query execution. Many of the algorithms and methods they proposed was adapted in our work, in order to build a solution for distributed systems. The work reported in [Fornari et al. 2006] also applies to non-distributed systems and proposes an optimizer for spatial join queries, although limited to simple spatial join with only two datasets. A later paper from the same authors [Fornari et al. 2007] also proposes a rule based optimizer for joining two spatial datasets.

The distributed processing of multiway spatial join was studied by Aji et al. [2012] using a MapReduce framework. The authors propose a data partitioning method specific for MapReduce, and a distributed algorithm that processes the multiway query in a single step, combining all the datasets at once. The processing of all datasets in one step is due a limitation imposed by the framework, which needs that intermediate results are persisted, incurring in high I/O costs. Others studies that uses MapReduce framework also do not evaluate the alternate plans for executing a query, as in [Zhang et al. 2009; Zhong et al. 2012; Gupta et al. 2013]. Gupta et al. [2013] discuss the execution of multiway spatial join on MapReduce. The main focus is on data partitioning and replication, in order to reduce the communication between nodes of the cluster. Zhang et al. [2009] and Zhong et al. [2012] propose MapReduce algorithms and data partitioning techniques to process simple spatial join with two datasets. Cunha et al. [2015] propose an algorithm to process multiway spatial join queries in distributed systems, also combining all datasets at once, but with an independent framework for spatial data processing, proposed in [de Oliveira et al. 2011] and [de Oliveira et al. 2013].

While the most popular strategy to process multiway spatial join in distributed systems is the combination of all datasets in a single step, following the *Synchronous Traversal* (ST) [Papadias et al. 2001] algorithm, originally proposed for non-distributed processing, most of the studies were limited by the use of the MapReduce framework. To the best of our knowledge, there are no studies that compare this strategy against the selection of the plan on the entire space of possible plans. In non-distributed systems, Mamoulis and Papadias [2001b] made this comparison and concluded that the best plan on real datasets is a hybrid plan, with some parts being processed with ST and others with join algorithm for two datasets at a time. Mamoulis and Papadias [2001b] also concludes that the main reason that pure ST plan is not the best is the degradation of this type of plan when a query has many datasets.

The work in [Patel and DeWitt 2000; Naughton and Ellmann 2002; Chung et al. 2005; de Oliveira et al. 2013] propose distributed algorithms for spatial join using two datasets at a time. Despite they are not algorithms for multiway spatial join, these algorithms can be adapted to process multiway spatial join queries in successive steps, as has been done in [Mamoulis and Papadias 2001b] for non-distributed systems. We choose and adapted the Clone Join algorithm to multiway spatial join query processing due to its simplicity and efficiency. However, others algorithm can be adapted and integrated in the optimizer as well.

As these studies focus in distributed processing of multiway spatial join in only one step or are specific for two datasets, we have not found comparable methods for distributed plan cost estimation and distributed plan selection. We propose in this article a new method for cost estimation in Section 3, based on a new multidimensional histogram for distributed systems, and a new method for plan selection in Section 4.3. To the best of our knowledge, this is the first attempt to propose and evaluate such methods for processing multiway spatial join queries in distributed systems, evaluating the cost of alternate execution plans.

6. CONCLUSION

We have identified dataset and data distribution metadata that are relevant to distributed processing of multiway spatial join queries and proposed a new multidimensional histogram: a data structure to organize this metadata, which is also used as a distributed data access method in distributed systems. A new method for histogram construction was proposed based on proportional overlap of spatial objects and cell boundaries. This method improves plan cost estimation and lead the optimizer to choose the best plan for all queries tested, using real spatial datasets. Furthermore, the methodology also provides a good ordering of query plans, which can select good plans even for more complex queries.

We also proposed an improved version of Clone Join. The new algorithm was adapted with the Reference Point method to prevent that duplicate results was reported in the join. The modified algorithm is also a non-blocking algorithm that can be used to efficiently process multiway spatial join queries in distributed systems.

We plan to investigate new metrics to map in the histogram, such as the existing work load of the execution of concurrently multiway spatial join queries. This is a more realistic scenario for a spatial database, and present interesting challenges in the scheduling of distributed queries in a cluster. We also plan to improve the greedy algorithm to use more elaborated combinatorial methods, to achieve a better query scheduling in the cluster.

REFERENCES

- ACHARYA, S., POOSALA, V., AND RAMASWAMY, S. Selectivity Estimation in Spatial Databases. *SIGMOD Record* 28 (2): 13–24, 1999.
- AJI, A., WANG, F., AND SALTZ, J. H. Towards Building a High Performance Spatial Query System for Large Scale Medical Imaging Data. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*. Redondo Beach, CA, USA, pp. 309–318, 2012.
- BRINKHOFF, T., KRIEDEL, H. P., AND SEEGER, B. Efficient Processing of Spatial Joins Using R-trees. *SIGMOD Record* 22 (2): 237–246, 1993.
- BRINKHOFF, T., KRIEDEL, H.-P., AND SEEGER, B. Parallel Processing of Spatial Joins Using R-trees. In *Proceedings of the IEEE International Conference on Data Engineering*. New Orleans, LA, USA, pp. 258–265, 1996.
- CHUNG, W., PARK, S.-Y., AND BAE, H.-Y. Efficient Parallel Spatial Join Processing Method in a Shared-Nothing Database Cluster System. In Z. Wu, C. Chen, M. Guo, and J. Bu (Eds.), *Embedded Software and Systems*. Lecture Notes in Computer Science, vol. 3605. Springer, pp. 81–87, 2005.
- CUNHA, A. R., DE OLIVEIRA, S. S. T., ALEIXO, E. L., DE C. CARDOSO, M., DE OLIVEIRA, T. B., AND DO SACRAMENTO RODRIGUES, V. J. Processamento Distribuído da Junção Espacial de Múltiplas Bases de Dados - Multi-way Spatial Join. In *Anais do XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Vitória, ES, Brazil, pp. 165–178, 2015.
- DE OLIVEIRA, S. S. T., DO SACRAMENTO RODRIGUES, V. J., CUNHA, A. R., ALEIXO, E. L., DE OLIVEIRA, T. B., DE C. CARDOSO, M., AND JUNIOR, R. R. Processamento Distribuído de Operações de Junção Espacial com Bases de Dados Dinâmicas para Análise de Informações Geográficas. In *Anais do XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Brasília, Brazil, pp. 1009–1022, 2013.
- DE OLIVEIRA, T. B., DO SACRAMENTO RODRIGUES, V. J., DE OLIVEIRA, S. S. T., DE ALBUQUERQUE LIMA, P. I., AND DE C. CARDOSO, M. Processamento Distribuído de Operações de Junção Espacial com Bases de Dados Dinâmicas para Análise de Informações Geográficas. In *Anais do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Campo Grande, MS, Brazil, pp. 719–732, 2011.

- DITTRICH, J.-P. AND SEEGER, B. Data Redundancy and Duplicate Detection in Spatial Join Processing. In *Proceedings of the IEEE International Conference on Data Engineering*. San Diego, CA, USA, pp. 535–546, 2000.
- FORNARI, M., COMBA, J. L. D., AND IOCHPE, C. A Rule-Based Optimizer for Spatial Join Algorithms. In C. A. D. Jr. and A. M. V. Monteiro (Eds.), *Advances in Geoinformatics*. Springer, pp. 73–90, 2007.
- FORNARI, M. R., COMBA, J. L. D., AND IOCHPE, C. Query Optimizer for Spatial Join Operations. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*. Arlington, Va, USA, pp. 219–226, 2006.
- GUPTA, H., CHAWDA, B., NEGI, S., FARUQUIE, T. A., SUBRAMANIAM, L. V., AND MOHANIA, M. Processing Multi-way Spatial Joins on Map-reduce. In *Proceedings of the International Conference on Extending Database Technology*. Genoa, Italy, pp. 113–124, 2013.
- MAMOULIS, N. AND PAPADIAS, D. Selectivity Estimation of Complex Spatial Queries. In C. S. Jensen, M. Schneider, B. Seeger, and V. J. Tsotras (Eds.), *Advances in Spatial and Temporal Databases*. Lecture Notes in Computer Science, vol. 2121. Springer, pp. 155–174, 2001a.
- MAMOULIS, N. AND PAPADIAS, D. Multiway Spatial Joins. *ACM Transactions on Database Systems* 26 (4): 424–475, 2001b.
- MUTENDA, L. AND KITSUREGAWA, M. Parallel R-tree Spatial Join for a Shared-Nothing Architecture. In *Proceedings of the International Symposium on Database Applications in Non-Traditional Environments*. Kyoto, Japan, pp. 423–430, 1999.
- NAUGHTON, J. AND ELLMANN, C. A non-blocking parallel spatial join algorithm. In *Proceedings of the IEEE International Conference on Data Engineering*. San Jose, CA, USA, pp. 697–705, 2002.
- PAPADIAS, D., MAMOULIS, N., AND THEODORIDIS, Y. Processing and Optimization of Multiway Spatial Joins Using R-trees. In *Proceedings of the ACM Symposium on Principles of Database Systems*. Philadelphia, PA, USA, pp. 44–55, 1999.
- PAPADIAS, D., MAMOULIS, N., AND THEODORIDIS, Y. Constraint-Based Processing of Multiway Spatial Joins. *Algorithmica* 30 (2): 188–215, 2001.
- PATEL, J. M. AND DEWITT, D. J. Partition Based Spatial-Merge Join. *SIGMOD Record* 25 (2): 259–270, 1996.
- PATEL, J. M. AND DEWITT, D. J. Clone Join and Shadow Join: two parallel spatial join algorithms. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*. McLean, VA, USA, pp. 56–61, 2000.
- SIVASUBRAMANIAM, A. Selectivity Estimation for Spatial Joins. In *Proceedings of the IEEE International Conference on Data Engineering*. Berlin, Heidelberg, Germany, pp. 368–375, 2001.
- ZHANG, S., HAN, J., LIU, Z., WANG, K., AND XU, Z. SJMR: parallelizing spatial join with mapreduce on clusters. In *Proceedings of the IEEE International Conference on Cluster Computing and Workshops*. New Orleans, LA, USA, pp. 1–8, 2009.
- ZHONG, Y., HAN, J., ZHANG, T., LI, Z., FANG, J., AND CHEN, G. Towards Parallel Spatial Query Processing for Big Spatial Data. In *Proceedings of the International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. Shanghai, China, pp. 2085–2094, 2012.