Distributed Execution Plans for Multiway Spatial Join Queries using Multidimensional Histograms

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

Instituto de Informática – Universidade Federal de Goiás (UFG), Brazil
thborges@ufg.br, {fmc, vsacramento}@inf.ufg.br

Abstract. Multiway spatial join is a common and heavyweight type of query for spatial data processing on relational database management systems. This article presents a complete solution to process this type of query in distributed systems. We proposed a cost-based optimizer for multiway spatial join queries, based on a novel use of multidimensional histograms, which are used to represent two metrics that describe a dataset: cardinality and size of the spatial objects, together with one feature that represents the location of the dataset partitions in the cluster. A new method for histogram construction is proposed, together with a formula to estimate the cost of multiway spatial join queries and to select good executions plans. A greedy algorithm is proposed to schedule query execution while minimizing both the makespan and the communication cost. We adapted the Clone Join algorithm to improve its execution time and to process multiway spatial join queries. The evaluation demonstrated that the new method for histogram construction performs better than a well-know method, and provides a better estimate for the cost of execution plans in 75% of the queries evaluated. The cost estimated by the proposed method is shown to be sufficiently close to real execution times, and our complete methodology was able to use the estimated costs to select the best execution plan for all queries used in the experiments. It also provided a nearly exact ordering of the query execution plans with respect to execution time.

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 has greatly increased with the popularization of GPS-enabled devices. Spatial data, such as geotagged images, the Internet of Things (IoT) and "Smart Cities" sensors data, open data and census data, are continuously collected and organized in thematic datasets to support decision-making, in order to improve business intelligence and logistics efficiency in both companies and government. Spatial data is a complex, multidimensional data type that is handled by a relational database management system (RDBMS) through the use of queries with spatial predicates [Carvalho et al. 2014].

An important type of query is the spatial join query, which finds correlated objects in two or more datasets by applying some spatial predicate like intersection or proximity [Brinkhoff et al. 1996]. Spatial joins can be simple, when processing only two datasets, or complex or multiway, when processing more than two datasets in the same query [Mamoulis and Papadias 2001b]. 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) [Mamoulis and Papadias 2001b] and digital pathology imaging [Aji et al. 2012] (e.g., to analyze topological images of the brain in order to check whether a certain cancer is progressing).

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

The processing of a multiway spatial join is more complex than the processing of a 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 execution plan has a huge impact on the processing cost of a query, and thus, in industrial instances of practical size, a significant amount of effort has to be dedicated to determine the best plan for a query on relational databases [Mishra and Eich 1992]. However, this effort is mainly dedicated to non-distributed databases or to distributed databases with scalar columns. Spatial data impose particular challenges to join processing and, in general, algorithms designed for scalar data in relational databases do not apply to spatial data due to the absence of total ordering in multidimensional data [Jacox and Samet 2007].

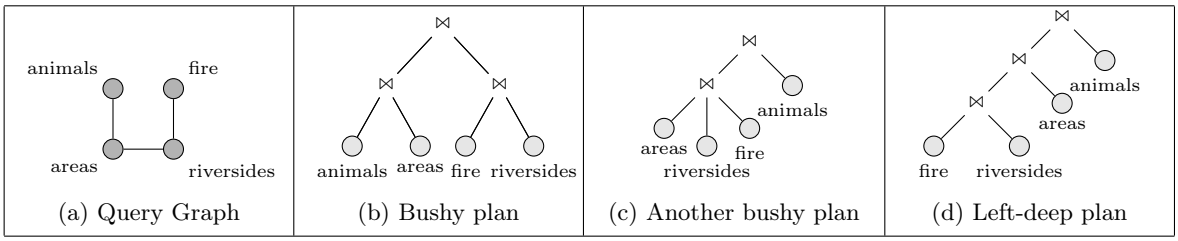
Another particular challenge of spatial data is the complexity of computational geometry algorithms – which evaluate the predicates over spatial data. Even queries on small datasets have a high processing cost due to the complexity and the different types of spatial objects (e.g., points, lines, polygons). One way of reducing query execution time is to process spatial queries in distributed systems by partitioning the datasets using spatial columns [Jacox and Samet 2007]. The partitioning of datasets using spatial columns imposes significant challenges to distributed systems due to the skewed nature of spatial datasets. The selection of execution plans in distributed systems, besides considering local CPU and I/O costs, must take into account the proper load balance of both data and queries. The balance has to respect the communication cost among machines and the even distribution of the workload among processing nodes, with regard to both the bandwidth limit on the network interfaces and the CPU load.

A significant effort in the literature is dedicated to the distributed processing of spatial join queries [Patel and DeWitt 2000; Chung et al. 2005; Jacox and Samet 2007; de Oliveira et al. 2013]. However, the main focus has been on processing simple spatial joins. To the best of our knowledge, there is no work reported in the open literature that proposes the selection of efficient execution plans for multiway spatial join query execution in distributed systems. It appears that any such selection should be based on an evaluation of network cost, CPU cost, and load balance among the machines, considering the partitioning of datasets using spatial columns.

Choosing good execution plans and efficiently processing multiway spatial join queries in distributed systems is a step towards moving spatial data analysis to scalable platforms, as has already happened to relational and unstructured data. However, new methods and algorithms to estimate the cost of an execution plan need to be specified, taking into consideration the specifics of spatial data and the characteristics of distributed systems. Processing spatial data analysis in such environments can greatly improve the capabilities of spatial data processing, especially in today's scenario of cloud computing platforms, taking advantage of scalability, elasticity, and pay-as-you-go offers.

This article builds on our previous work [de Oliveira et al. 2015], with improvements on the methodology for cost estimation of query execution plans, and a new algorithm for query scheduling and load balancing in distributed systems. The entire set of proposed methods constitutes an optimizer to process multiway spatial join queries in distributed systems. The main contributions of the present article are:

—The identification of the characteristics of datasets and data distribution which are relevant to the efficient processing of multiway spatial join queries in distributed systems;
—The definition of a multidimensional histogram data structure to organize these characteristics, observing data skewness in real spatial datasets;
—A new method to construct a multidimensional histogram that improves the estimate of query processing cost;
—An algorithm that estimates the cost of execution plans in distributed systems using multidimensional histograms; and



(a) Query Graph (b) Bushy plan (c) Another bushy plan (d) Left-deep plan

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

—A greedy algorithm to schedule data copying between cluster nodes, which reduces bandwidth usage while maintaining load balance in the cluster.

The remainder of this article is organized as follows. In Section 2 we present a survey of multiway spatial join processing, describing key concepts, existing estimation techniques for plan selection and algorithms to process multiway join queries, as well as a discussion of how to adapt these methods for distributed systems and related work. In Section 3, we propose a multidimensional histogram data structure, methods for constructing such histograms, an algorithm for estimating network I/O, a data copying scheduling algorithm and a basis for the selection of execution plans. Section 4 presents an evaluation of the proposed methods and discusses the results. Section 5 presents conclusions and suggested future work.

2. CONCEPTS AND RELATED WORK

In this section we cover some concepts and existing related work. We start with the fundamentals of multiway spatial join and, then, describe its processing steps: plan cost estimation using multidimensional histograms, plan enumeration and selection, and existing algorithms to process join queries. Finally, we present related work.

2.1 Multiway Spatial Join

Multiway spatial join is a technique for combining successive join algorithms to process (complex) spatial join queries with more than two input datasets [Papadias et al. 1999]. A multiway spatial join can be represented by 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].

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

Figures 1b, 1c, and 1d illustrate the alternative plans to process the chain query graph presented in Figure 1a. In a chain query, the graph $G(V, E)$ is a tree, with any two vertices connected by exactly one path. In Figure 1b, the datasets are pairwise joined in a first step, producing two intermediate results to be joined in a second step. In Figure 1c, three datasets are joined in a single step and the intermediate result is then joined with the remaining data set "animals". In Figure 1d, the datasets "fire" and "riversides" are joined and the intermediate results are processed in subsequent steps with the other two datasets, one at a time.

All the execution plans for a given query preserve the same query semantics, but may take different amounts of time to produce the final result. The graph that represents a query is used as an input for a plan selection algorithm, also called an *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 the processing order is, and (*iii*) which algorithms will be used in each step. Despite the name, an optimizer, in general, is not an exact algorithm. In the next section, we discuss how the computational cost of an execution plan can be determined.

2.2 Estimating the Cost of Spatial Join for Plan Selection

Despite the complexity of estimating the cost of execution plans, some studies have proposed formulae that estimate the cost of simple spatial join queries [Roh et al. 2010; Sivasubramaniam 2001; Fornari et al. 2006], as well as methods to combine these formulae to estimate the cost of complex spatial join queries [Mamoulis and Papadias 2001a]. These formulae assume that spatial objects fill the spatial extent uniformly and take the I/O and CPU costs 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], who conclude that, when used on real spatial datasets, the formulae can induce bad execution plans, especially in the presence of dataset skewness.

Mamoulis and Papadias [2001a] studied the applicability of the formulae in small regions of the dataset. The authors proposed the use of an uniform histogram that divides the spatial extent of the dataset in a map with fixed-size cells, each of them mapping the density of small regions. In this histogram, the cardinality value of each cell x, denoted by $C(x)$, is defined based on the number of spatial objects that have the center of their Minimum Bounding Rectangle (MBR) within the limits of x.

The main advantages of uniform histograms are *i*) simplified construction, *ii*) time efficiency to estimate queries, and *iii*) incremental maintenance [Cormode et al. 2012]. In distributed systems, this type of histogram also provides a simple way to partition the datasets, a step needed for distributed processing. A deficiency of uniform histograms is the estimation error when data is skewed. The error can be reduced by increasing the number of cells, although this increases the quantity of memory needed to store the histogram structure [Cormode et al. 2012]. There are other types of histograms for spatial data [Roh et al. 2010; Ray et al. 2014] that can be used to improve the estimation of skewed datasets. However, the complexity of estimating query costs with these histograms is greater, and their recursive nature creates important drawbacks in incremental maintenance and data partitioning.

An important challenge in the construction of multidimensional histograms for spatial data is how to hash the objects into cells. Although the MBR center method, proposed by Mamoulis and Papadias [2001b], is accurate for point data, other types of spatial objects, such as lines and polygons, bring additional challenges to histogram construction due to their spatial extent. A single line or polygon object can overlap more than one cell in the histogram and hashing it through the MBR center can lead to large errors in estimates.

In distributed systems, besides the cardinality of the datasets, the size of spatial objects (given by the number of points) and the place (server) where the partitions are located are also relevant to estimate the communication cost. These variables need to be stored as part of the histogram statistics.

2.3 Selecting an Execution Plan for Multiway Spatial Join

The size of the space of possible plans to select from is nonlinear with respect to the number of datasets. Mamoulis and Papadias [2001b] studied the number of possible plans, considering three different algorithms to process spatial join queries 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 number 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 have an even larger number of possible plans. As a result, if a query involves a sufficiently large number of datasets, an optimizer can take more time planning (through having to enumerate all possible plans and to compute their costs) than actually executing the query.

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 the servers where the 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 for the selection of the execution plan, further increasing the number of possible plans.

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, such as associativity and commutativity, together with methods such as iterative improvement and simulated annealing. In their experiments, the heuristic method was able to find 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 number of datasets.

2.4 Algorithms for the Distributed Processing of Multiway Spatial Join

Processing spatial join queries in distributed or parallel systems requires some data partitioning methods that divides the dataset objects into groups called data partitions or cells, which are assigned to servers during the filtering or refinement step. The methods for data partitioning can be classified into two main categories:

(1) Disjoint space partitioning: uses a grid to split the space extent. Each cell of the grid groups 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.
(2) Non-disjoint space partitioning: partitions can overlap each other to accommodate the extent of the objects that intersect them. Spatial objects are not replicated. An example of this type of partitioning is the set of MBR's on a given level of an R-Tree index.

For non-disjoint space partitioning, the relevant distributed algorithms are *Replicated Semi-packed Parallel R-Tree* (RSPR) [Mutenda and Kitsuregawa 1999], *Distributed Synchronous Traversal* (DST) [Cunha et al. 2015], and *Proximity Area Spatial Join* (PASJ) [de Oliveira et al. 2013]. All of them use distributed R-Tree indexes to process the spatial join. For disjoint space partitioning, the relevant algorithms 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 them. 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 replicates 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 *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 simplest 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 co-location of spatial data have also been proposed, such as the Proximity Area method [de Oliveira et al. 2013] for non-disjoint partitioning, that distributes the partitions based on their location, favoring reduction of network bandwidth usage during spatial join execution.

In a round-robin distribution, data partitions that are 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 by de Oliveira et al. [2013],

distributing the partitions based on their location will reduce network bandwidth usage at the expense of compromising the load balancing of spatial join execution.

2.5 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 the spatial extent. The spatial objects are assigned to each cell based on the intersection between their MBRs and the cell boundaries. Objects that intersect more than one cell are replicated. Each cell, and the corresponding spatial objects, form a data partition and are assigned to one server using a round-robin distribution or a similar hash function applied to the cell number. The join operation is performed in parallel in 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, at the beginning of the execution of the join, that the two datasets are partitioned with the same grid. This can be a problem with 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, an additional step is necessary to eliminate duplicated results reported by the refinement. A duplicated 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 separately report the intersection between the objects. Patel and DeWitt [2000] used a *distinct* verification at the end of the algorithm to eliminate the duplicated results. This is a deficiency of the algorithm because the distinct verification is a costly operation in distributed systems due to its network-bound behavior. Objects with large spatial extents (such as lines and polygons) naturally increase the number of replicated objects, and, consequently, also increase the usage of resources (notably, bandwidth and execution time).

Dittrich and Seeger [2000] proposed a solution for the result duplication problem called the reference point method. The method consists of identifying the possible cells that will report duplicate results and allowing only one of them to do so. The method was originally proposed for the PBSM algorithm and later adapted to NPSJ [Naughton and Ellmann 2002], a spatial join algorithm for distributed systems. However, it can be applied to CJ as well, as is detailed in Section 3.2.

2.6 Related Work

Despite the importance of multiway spatial join queries and the widespread use of spatial datasets, little work has been proposed to efficiently process multiway spatial join queries in distributed systems. In this section, we discuss the most relevant research efforts in the field. Overall, they propose isolated contributions that can be combined to form a complete solution for distributed multiway spatial join query processing.

To the best of our knowledge, Mamoulis and Papadias [2001b] performed the most relevant study in the literature for the non-distributed processing of multiway spatial join. The authors make an in-depth evaluation of algorithms, data partitioning techniques, plan cost estimation, plan selection and query execution. In order to build a solution for distributed systems, several of the algorithms and methods they propose were adapted in our work. For simple spatial joins with only two datasets Fornari et al. [2006] proposed an optimizer for spatial join queries in non-distributed systems and also a rule-based optimizer [Fornari et al. 2007].

A method for the distributed processing of multiway spatial join queries is presented 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 to a limitation imposed by the framework, which requires that intermediate results are stored, incurring in high I/O costs. Several studies that use the MapReduce framework do not evaluate the alternative plans for executing a query [Zhang et al. 2009; Zhong et al. 2012; Gupta et al. 2013]. Gupta et al. [2013] discuss the spatial data partitioning and replication for the MapReduce framework. In order to reduce communication between nodes in 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 the independent framework for spatial data processing proposed by 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 (which was 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 with the selection of the execution plan by considering the entire space of possible plans. For non-distributed systems, Mamoulis and Papadias [2001b] made this comparison and concluded that the best plan for real datasets is a hybrid plan, with some parts being processed with ST and others with a join algorithm for two datasets at a time.

A number of research efforts [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 the fact that they are not algorithms for multiway spatial join, these algorithms can be adapted to process multiway spatial join queries in successive steps, as it has been proposed by Mamoulis and Papadias [2001b] for non-distributed systems.

As these studies focus on distributed processing of multiway spatial join in only one step or are specific to two datasets, we have not found comparable methods for distributed plan cost estimation and distributed plan selection. In this article, we propose a new method for cost estimation (see Section 3) based on a new multidimensional histogram for distributed systems, together with a new method for plan selection. 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.

3. MULTIDIMENSIONAL HISTOGRAM AND OPTIMIZER

In this section we describe the proposed data structure, called a multidimensional histogram, along with its access method. This data structure combines parameters that describe the datasets and their allocation in the distributed system.

A multidimensional histogram divides the spatial extent of a dataset using a grid with cells of fixed size. For each cell, it records: *i*) the number of spatial objects within the boundaries of the cell (the cardinality of the cell), *ii*) the combined size of the objects in the cell (in terms of their total number of points), and *iii*) the place (server in the cluster) where the cell is located. The first proposed metric is used to estimate the cardinality of the join output, in a similar way to the non-distributed execution of spatial join queries. The second proposed metric is used to estimate the intermediate histograms used to select an execution plan for a spatial join query. We use the three values of *i*), *ii*) and *iii*) together to estimate network bandwidth usage, as it is detailed in Section 3.3.

Figure 2 shows a visualization of a multidimensional histogram generated for a dataset that represents the political limits of Brazilian municipalities. The histogram captures important aspects of the dataset. At the top left corner of Figure 2a, the cardinality is low because this region corresponds to the Amazon Forest, where there are few large municipalities, resulting in lower cardinalities for

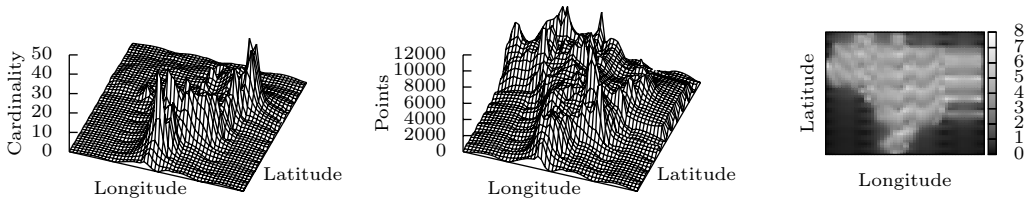


Fig. 2: Multidimensional histogram for a two-dimensional dataset that represents the political limits of Brazilian municipalities. The graphs represent: the cardinality (a) and the points (b) metrics; and the location of partitions (c).

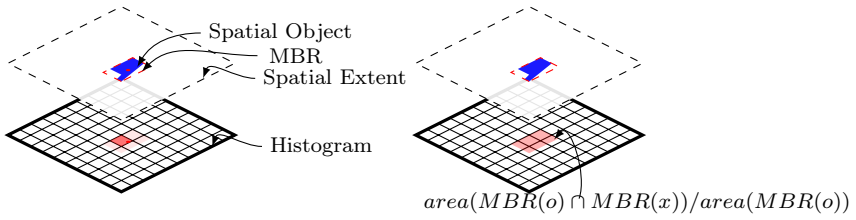


Fig. 3: A spatial object being hashed using the MBR Center Method (left) and the Proportional Overlap Method (right). In the equation, o is a spatial object and x is a histogram cell.

the cells. In contrast, Figure 2b shows the points metric for the same municipalities. The number of points is much higher than in other regions because the area(extent) of the objects is larger and more points are needed to represent their contour. Figure 2c is a map of the location of spatial data in the cluster. For this sample histogram, the legend indicates the location of each cell (in servers 1 to 8). A value of zero (black) represents a cell that contains no objects and does not need to be sent to a server.

3.1 Proportional Overlap Method

To build a histogram, each object in a dataset needs to be hashed into the histogram grid in order to find the cell or the set of cells that it intersects. The set of objects hashed to a cell forms a data partition. We use the terms *cell* and *data partition* interchangeably in the remaining text.

We propose a new method to build the multidimensional histogram, called the Proportional Overlap method. Instead of using the center of the MBR to hash a spatial object into the histogram, as proposed by Mamoulis and Papadias [2001b], we use the proportional covering area of each MBR. To calculate the cardinality metric, we increment the value stored in each cell based on the proportion of two areas (a/b), where a is the area of intersection between the MBR of the spatial object and the histogram cell boundaries and b is the area of the MBR. Formally, let H_r be the histogram of the dataset R, and let $o \in R$ be an object of the dataset. The cardinality $C(x)$ of a cell $x \in H_r$ is given in (1). Figure 3 illustrates the hash process of a spatial object in a histogram. In the MBR center method (left), the center of the object's MBR identifies the histogram cell. The cardinality metric of the identified cell is thus increased by unity. In contrast, the proportional overlap method (right) increments the cardinality of each cell that intersects the object's MBR by adding the proportion obtained in (1).

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

Calculation of the points metric, in turn, is performed by summing the number of points of each of the objects that intersects a cell. This reflects the behavior of the CJ algorithm, which replicates an object in each of its intersecting cells. The number of points $P(x)$ in a cell x is thus given in (2).

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

After the hashing process, we determine the location of the data partitions – their physical distribution in the cluster. The location feature reflects the initial distribution of the partitions in the cluster. As such, it is updated whenever the partitions are replicated as part of the join processing.

3.2 Clone Join with the Reference Point Method

As described in Section 2.5, *Clone Join* (CJ) [Patel and DeWitt 2000] is designed to process simple spatial join queries, which involve only two datasets. In this section we describe an adaptation of CJ to efficiently process multiway spatial join queries.

The CJ algorithm has two drawbacks when dealing with multiway spatial join query execution: *i*) the *distinct* operator used to prevent duplicated results has a high communication cost, and *ii*) the *distinct* operator prevents pipeline execution of multiway joins, i.e., it acts as a barrier, delaying the start of the next step of a plan until the end of the previous step, thus causing slow performance.

To efficiently process a multiway spatial join query using CJ, we can use the reference point method [Dittrich and Seeger 2000] to identify duplicated results during the predicate checking, rather than as a separate step at the end. The predicate checking part of CJ consists of a simple loop that iterates over each pair of candidate spatial objects [Patel and DeWitt 2000]. The reference point method consists of defining a reference point for each candidate pair of spatial objects and verifying whether or not it is within the boundaries of the cell being processed [Dittrich and Seeger 2000]. As the cell boundaries are disjoint in CJ, only the cell containing the reference point will report the candidate pair. The required changes in the CJ algorithm are:

(1) Distribute and maintain cell boundaries as metadata on each server. Such metadata will be used to check if the reference point is inside a cell belonging to the server that is checking the predicate;
(2) Change the CJ predicate checking algorithm:
    (a) Whenever a pair of objects has to be verified, compute the reference point using the MBRs of the two objects, r and s: $rp = (max(r.xl, s.xl), min(r.yh, s.yh))$, where xl is the lowest x value, and yh is the highest y value for each of the two MBRs;
    (b) Check if the reference point is within the boundaries of the cell that is being processed: if the reference point is not within the cell boundaries, ignore it (as it will be reported by another server). Otherwise, perform the predicate checking for the pair.
(3) Remove the distinct operator at the end of the algorithm.

The main advantage of these changes are that no communication is necessary to define the reference point. The only increase in communication is in data distribution, when the cell boundaries are transferred. This is a constant cost increase, however, as data partitions need to be transferred anyway. This cost is also compensated, as no distinct verification is needed at the end of the algorithm. The previous complexity of this verification is a function of the number of replicated spatial objects. Furthermore, the modified algorithm also has the non-blocking characteristic proposed in more recent algorithms, such as the *Non-blocking Parallel Spatial Join* (NPSJ) [Naughton and Ellmann 2002], which further improves multiway spatial join query execution due to early reporting of results in subsequent steps.

3.3 Query Scheduling and Plan Cost Estimation using Multidimensional Histogram

To process multiway spatial join queries in a distributed system, we need to move the partitions of the involved datasets, which are already distributed in the cluster. This movement should be performed according to the semantics of the query predicate. In this section, we describe a method to estimate the cost of data movement and the overall cost of an execution plan based on the creation and use of intermediate histograms.

The data movement (communication) cost of a plan can be estimated using the metrics and features of the multidimensional histograms derived from the involved datasets. This is performed as follows. Let H_a and H_b be two histograms for datasets a and b, respectively; let $C(c_x)$ be the cardinality of cell c_x; let $P(c_x)$ be the total number of points in cell c_x; and let $L(c_x)$ be the set of servers where cell c_x is stored. An intermediate histogram, along with the communication cost, can be obtained using Algorithm 1, whose code is given later.

Algorithm 1 performs three tasks: *i*) it determines a data copying strategy based on the pre-distribution of the datasets; *ii*) it estimates the communication cost of joining a and b; and *iii*) estimate an intermediate histogram, H_r, for the result of the spatial join between datasets a and b. This intermediate histogram will be used to estimate the next join steps of the multiway spatial join query. To construct the H_r histogram, the algorithm evaluates each pair of cells from H_a and H_b that intersect each other. The algorithm estimates the metrics of this histogram using the proportional overlap method (lines 5-8), and fills the resulting histogram with the estimated values (lines 9 to 11).

The algorithm uses a greedy strategy to determine how data copying will be performed and to define the location feature of the H_r histogram (lines 12 to 23). The ChooseMinCommServer procedure (line 12) returns the server that requires the lowest communication cost to locate the cell pair ca and cb in the same place: If $L(ca) \cap L(cb) \neq \emptyset$, the two cells have a server in common and this server is returned; otherwise, a server of $L(ca)$ is returned if $P(ca) < P(cb)$, or vice-versa. The standard deviation of values in the $Pnts$ vector gives an estimate of the load balance for the execution plan. If the difference between the servers with the smallest and the largest number of points reaches a certain *tradeoff* (line 14), instead of choosing the server with smallest amount of IO, the server with the smallest number of points is chosen (line 15). This strategy attempts to maintain low communication cost, but if the balance in the cluster drops, communication is sacrificed at the expense of maintaining load balance. The number of points to be compared is then updated in the vector $Pnts$ (line 16), and the location of ca and cb cells are updated based on the chosen server (lines 17 and 18). The communication cost is updated based on the number of points to be copied and the size of each point in the system (two double floating points)[1]. This is performed for each server in the cluster, and stored in the IO vector (lines 19 to 23). The return of the algorithm (line 24) is used for plan selection (in Section 3.4).

3.4 Selection of Distributed Execution Plans

Execution of the spatial predicate of a multiway spatial join query uses CPU intensive algorithms from computational geometry as a part of the computation of the query results. In a distributed system, this CPU cost should be evenly shared among the computing nodes in order to reduce the total execution time of the query (i.e., the makespan). Thus, we need to consider cluster balancing while selecting a plan to execute a query.

We use an exhaustive method to enumerate all the possible execution plans for a query. Let P be the set of plans. For each execution plan $p \in P$, Algorithm 2, given later, is invoked to compute the cost of all steps in the plan. For the first step, the algorithm uses the histograms of two datasets,

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

```
DMSJ_StepCostsAndHr(H_a, H_b)
 1  H_r = new empty histogram, based on the cell limits of H_a and H_b
 2  IO[] = 0
 3  Pnts[] = 0
 4  for each ca ∈ H_a, cb ∈ H_b, such that ca ∩ cb ≠ ∅
 5       d_a = area(ca ∩ cb)/area(ca) * C(ca)
 6       d_b = area(ca ∩ cb)/area(cb) * C(cb)
 7       p_a = P(ca)/C(ca) * d_a
 8       p_b = P(cb)/C(ca) * 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 predicate for the next step
12       s = ChooseMinCommServer(IO, ca, cb)
13       balance = (Max(Pnts) − Min(Pnts))/Max(Pnts)
14       if balance > tradeoff
15            s = MinIdx(Pnts)
16       Pnts[s] += P(ca) + P(cb)
17       L(ca) = L(ca) ∪ s
18       L(cb) = L(cb) ∪ s
19       for s = 1 to servers
20            if s ∉ L(ca)
21                 IO[s] += P(ca)
22            if s ∉ L(cb)
23                 IO[s] += P(cb)
24       return IO, Pnts, H_r
```

Algorithm 1: This algorithm determines a schedule to execute a spatial join query between datasets a and b, returning both the CPU and network costs, and creates an intermediate histogram, H_r.

$p.steps[0].H_a$ and $p.steps[i].H_b$ (lines 1 and 4), as input parameters to invoke Algorithm 1 (line 5). The returned $Pnts$ vector is aggregated in the $PlanPnts$ vector (lines 6 and 7), and the intermediate histogram, H_r, is considered as the first histogram in the next step (line 8). Finally, in line 9, the procedure stores the $PlanPnts$ vector in $AllPlans[p]$, which is used to select the best plan for the query.

We suggest that the best distributed execution plan for a query is the plan with cost O defined in (3). The expression $Max(AllPlans[p])$ in (3) selects the largest value of $PlanPnts$ for each execution plan p. This value refers to the server with the largest estimated load. This server will most likely be the bottleneck because the execution of the predicate checking algorithm will last longer in it. To conclude the selection, the expression $Min(...)$ in (3) selects from among the set of maximum values, the plan with the least number of points. In other words, we select the execution plan for which the bottleneck server has the lowest estimated load compared to the bottleneck servers from all the other plans. Thus, according to the estimate, the selected plan will be the one with the shortest makespan.

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

The execution time need to select a plan using Algorithm 2 and (3) is just a small fraction of the total execution time of the sample queries used in the evaluation (see Section 4). However, this execution time increases substantially for queries that involve a large number of datasets, as the number of plans grows exponentially. For such queries, the use of pruning methods for relational queries, as proposed by Ioannidis and Kang [1990] and latter adapted for spatial queries by Mamoulis and Papadias [2001b],

```
DMSJ_PlanCost(p)
1  H_left = p.steps[0].H_a
2  PlanPnts[] = 0
3  for i = 0 to length(p.steps)
4        H_right = p.steps[i].H_b
5        _, Pnts, H_r = DMSJ_StepCostsAndHr(H_left, H_right)
6        for s = 1 to servers
7              PlanPnts[s] += Pnts[s]
8        H_left = H_r
9  AllPlans[p] = PlanPnts
```

Algorithm 2: Algorithm to compute the cost of a query execution plan p.

alleviates the problem. Such methods work by pruning the set of plans that need to be evaluated. Thus, our methodology can still be applied to the restricted set of plans that results from pruning.

4. EVALUATION

To evaluate our proposed methodology, we chose a set of real spatial datasets obtained from the Brazilian Institute of Geography and Statistics[2] (IBGE), from the LAPIG Laboratory[3] of the Institute of Social and Environmental Studies (IESA) at UFG, and from the current version of the Digital Chart of the World [4](DCW). The selected datasets and their characteristics are detailed in Table I. Datasets with small cardinality were selected to conduct experiments with selective execution plans, in which one of the datasets strongly restricts the query output. On the other hand, datasets comprising complex spatial objects, such as rivers and roads, were selected due to the difficulty and inaccuracy of their representation in histograms.

The queries used in the experiments are listed in Table II. This set of queries involves all of the datasets and represents multiway spatial joins that are common in practical applications. All queries are of the chain type and the spatial predicate used in each individual join operation corresponds to the intersection between two datasets. The queries with three and four datasets have two and six execution plans, respectively. The execution plans for a query are presented in the form $Q1..8\ P_{1..6}$, for example, $Q1\ P_1$, $Q1\ P_2$. We argue that this is a representative set of queries as it uses real datasets and involves all types of complex spatial objects and their combinations. However, queries involving more than four datasets should be evaluated in the future, as well as other less common types of queries, such as clique and cycle queries.

The experiments presented in the next sections were performed on the Azure Platform, using eight A2 virtual machines, with 2 vCPUs and 3.5 GB of RAM each. The machines were allocated at the same data center and were interconnected by a virtual network. To the best of our knowledge, there is no public specification provided by the provider that defines the bandwidth of such a network. However, we experienced a network bandwidth limit of approximately 200Mbps.

4.1 Trade-off Between Load Balance and Communication

This section presents an experiment designed to choose a value for the parameter *tradeoff* in Algorithm 1. As discussed in Section 3, the algorithm performs a greedy search to find a schedule that is

[2] `www.ibge.gov.br`
[3] Image Processing and Geoprocessing Laboratory – `www.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)
Seaport	P	Points	120	<1,0
Vegetation	V	Polygons	2.140	4,7
Counties	M	Polygons	5.564	38,8
Fire alerts	A	Polygons	32.578	11,2
Roads	R	Lines	51.646	15,2
Rivers	H	Lines	226.963	64,5
Crops	CU	Polygons	123.746	69,3
Rails	FM	Lines	194.261	28,7
Inland Water	RA	Polygons	338.860	136,7
Elevation Contour	CR	Lines	703.574	572,5
Rivers	HM	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

well-balanced and has a small communication cost. We executed the algorithm for each query plan, varying the *tradeoff* from 0.02 to 0.9, and measured the total estimated communication cost and the standard deviation of the CPU cost – the sum of the IO vector and the standard deviation of $Pnts$ vector, respectively.

The results are presented in Figure 4. The y axis indicates the CPU and the Network costs, which have been normalized. Due to space constraints, we present the results of only the four queries: $Q1$, $Q3$, $Q5$, and $Q8$. The other queries produced very similar results. For all execution plans, the point where communication cost matched load balance was approximately at the *tradeoff* $= 0.2$. Within this point (*tradeoff* ≤ 0.2), load balance continued improving but communication cost grew quickly. Beyond this point, the inverse occurred.

Since *tradeoff* values smaller than 0.2 resulted in an improvement in load balance with a disproportional increase in communication cost, we used *tradeoff* $= 0.2$. However, as the graphs show, in systems with better network bandwidth, this value could be significantly reduced to improve query balance.

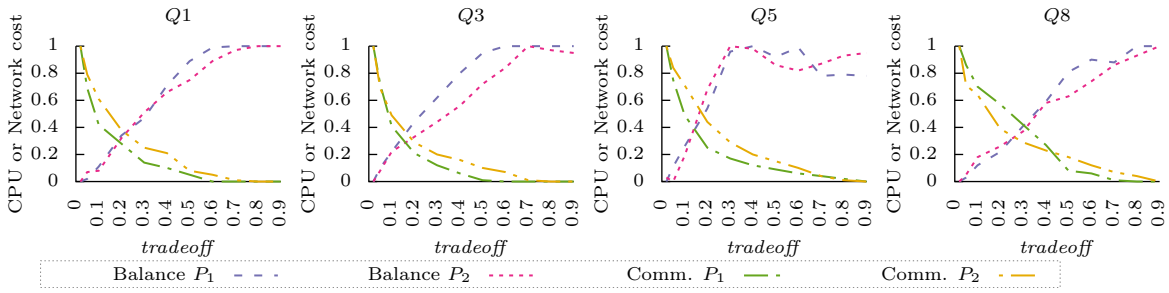


Fig. 4: Trade-off between communication cost and load balancing in the cluster for queries $Q1$, $Q3$, $Q5$, and $Q8$. The y axis is normalized and corresponds to the CPU or Network cost, conforming to the legend.

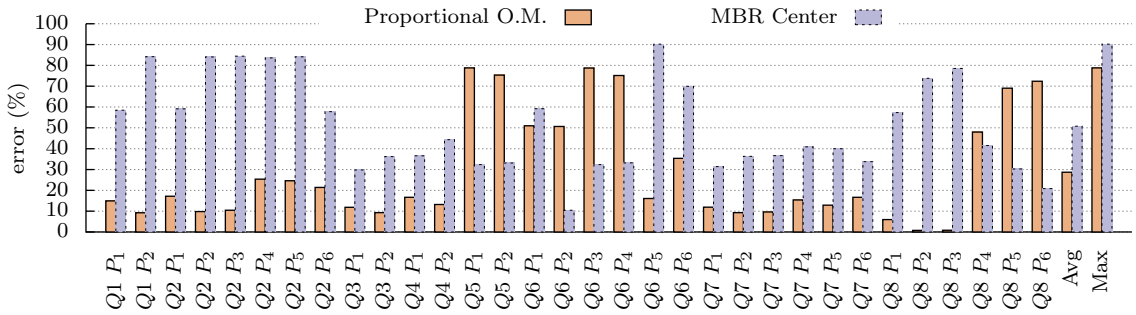


Fig. 5: Error in the estimation of network communication cost using the two histogram construction methods.

4.2 Evaluation of the Proportional Overlap Method

This section presents an experiment designed to evaluate the Proportional Overlap Method, used to compute the cardinality and points metrics of multidimensional histograms, as described in Section 3.1. The experiment consisted of creating a multidimensional histogram for all the datasets involved in the queries of Table II, using both the Proportional Overlap Method and the MBR center method [Mamoulis and Papadias 2001b]. Algorithm 1 was executed for each execution plan to obtain an estimate of the total number of data (in bytes) to be transferred using each method. The estimates were compared with the real communication cost of each plan, obtained from real executions of the plans in the cluster. For this comparison, we used equation $error = abs(real - estimated)/real * 100$ to compute the distance between the estimated and the real communication costs.

The result of the experiment is presented in Figure 5. The chart shows the error for different execution plans using each of the two methods. The last two groups of bars, in turn, show the average and the maximum error considering all the execution plans. The error incurred with the MBR center method is $\approx$ 50.7% on average for all plans, and, in the worst case, ($Q6$ P_5) it was 90.19%. The Proportional Overlap method had an average error of $\approx$ 28.7%, and a maximum error of 78.80% (also for $Q6$ P_3). The error of proportional overlap method is lower in 24 of the 32 queries evaluated (75%). These values show the effectiveness of the Proportional Overlap method to build better histograms to estimate the communication cost of execution plans. In the following experiments, we used only the Proportional Overlap method, since it provides more precision for the datasets used.

4.3 Evaluation of Communication Cost Estimation

In this section we describe an experiment designed to evaluate Algorithm 1, by generating intermediate histograms and computing the overall number of spatial object points and network I/O per server. Algorithm 1 is an important part of the plan selection method. It estimates the overall number of points and the communication cost, as well as the load balance in the cluster. The experiment consisted of obtaining an estimate of the communication cost for each plan and server pair, as well as the server with maximum and minimum estimated communication cost. The method then compared the estimated values with real values measured in the current execution of the plans in the cluster. As the estimated communication cost was based on the number of points transferred, and the number of points was obtained from the histograms, a good estimate indicates that the methodology can effectively be used to determine the cost of execution plans for plan selection.

The results of the experiment are shown in Figure 6. To facilitate the comparison of values, the charts are organized in two sections, each with a separated y axis. The section on the right-hand side shows the four plans ($Q3$ P_2, $Q7$ P_2, $Q7$ P_3 and $Q8$ P_2) with larger y values. Each bar in the chart represents the average of estimated and real network communication cost, among all servers. The total number of communication for each plan can be calculated by multiplying the value of the corresponding bar by 8 (the number of servers used in the experiment). It ranges from 20.13MB ($Q6$ P_6) to 1.47GB

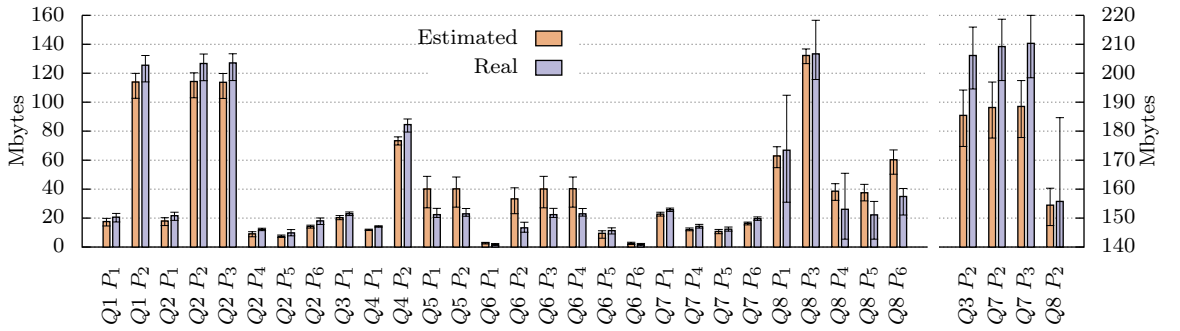


Fig. 6: Average server communication: estimated and real for each query. The error bars represent the minimum and maximum network communication between all servers.

($Q7\ P_3$). Each bar also has a superior and inferior limit, which represent, respectively, the server with the maximum and minimum number of network communication. A large variation of these limits indicates that the network load is unbalanced in the cluster.

As can be seen in Figure 6, the estimated values are different from the real ones, although they are very close. This behavior also holds for the error bars, which compare the estimated maximum and minimum errors with the maximum and minimum real values. This confirms that the algorithm provides a good estimate of the total network communication cost, as well as the network communication cost for each server. There is a set of plans that shows a distinct behavior: $Q5\ P_{1..2}$, $Q6\ P_{2..4}$, and $Q8\ P_{4..6}$. While, for all other plans, the estimated value is below the real value, these plans show the opposite. This behavior is due to the propagation of the error in the estimation of the size of the objects (see line 11 of Algorithm 1). Because an average of the object sizes is propagated to the intermediate histogram, big objects that could be filtered in a real execution of the join remain in the estimation, causing this error.

The results reflect the fact that histograms are a simplified representation of the datasets. The method cannot produce precise estimates due to this simplification. However, the data show reasonable proximity between the estimates and the real values for all plans used in the experiment. We concluded that Algorithm 1 and the associated methodology can provide a good estimate of the communication cost for each execution plan based on the distribution of the data and on the multidimensional histograms.

4.4 Evaluation of the Selection of Distributed Execution Plans

In this section we describe an evaluation of the plan selection methodology described in Section 3.4. The experiment consisted of selecting an execution plan using the estimates provided by Algorithm 1 and (3). It then measured the actual execution time of all plans in the cluster to check whether or not the selected plan (based on the estimates) was indeed the plan with the shortest execution time.

Figure 7 shows the estimated cost of all query plans, based on the number of points of the spatial objects. An arrow indicates the plan that was selected by the method – the lowest bar in each group. Each bar in the chart indicates the estimated number of points to be compared (in millions) by the server with the highest number of points ($Max(AllPlans[p])$ according to (3)).

The execution time for each of the 32 plans is presented in Figure 8. The chart uses a logarithmic scale due to the difference between the execution times of the best plan (in seconds) and the worst plan (in hundreds of seconds). The chart shows that the selected plan was indeed the best one for all the queries in the experiment. Furthermore, the method was able to establish a consistent relative ordering of all plans for almost all queries. The only two differences in the ordering were $Q8\ P_5$ and $Q8\ P_6$, which appeared in inverse order between estimate and real execution (0.8 seconds of difference between the plans), and $Q6\ P_{2/5,4/3}$, which also differed from the estimates, but only by 0.2 seconds

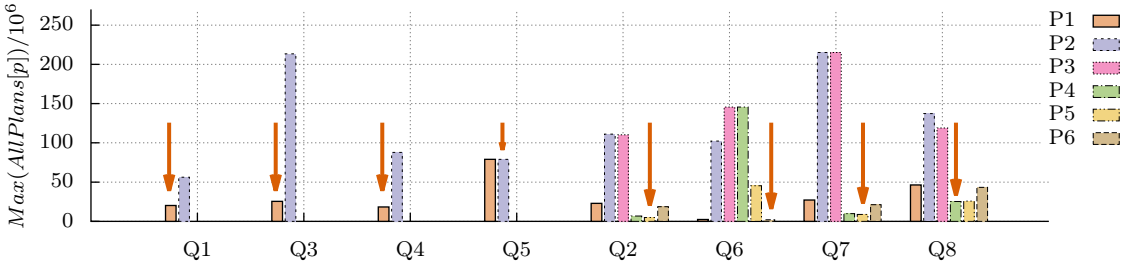


Fig. 7: Estimated maximum number of spatial points in a server for each plan, after query scheduling by Algorithm 1. Red arrows indicate the best plan selected for each query by (3).

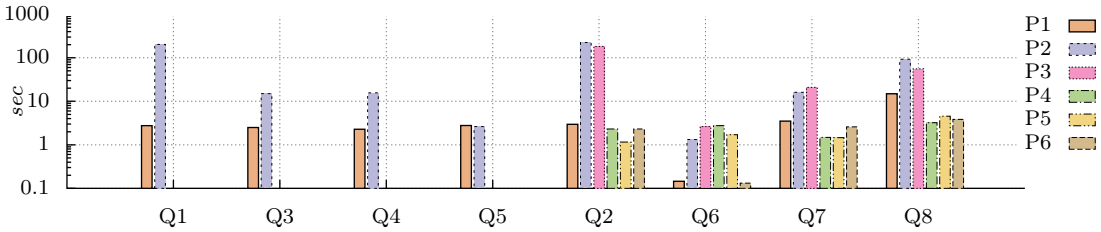


Fig. 8: Execution time of all plans in the cluster.

on average. The error in the ordering could lead to the wrong choice of an execution. However, as can be observed in the chart, the error always occurred with very similar plans with respect to their execution times. A wrong choice in this scenario will still lead to a good plan, with a slightly worse execution time. This confirms that the overall number of points of the spatial objects is a good metric to determine the cost of execution plans. It also confirms that (3) selects a relatively good plan, based on the estimated cost of plans.

5. CONCLUSION

In this article we proposed a comprehensive solution methodology for multiway spatial join query processing in distributed systems. We identified metadata related to the datasets and the data distribution, which are relevant for the distributed processing of multiway spatial join queries. We proposed a novel use of multidimensional histogram as a data structure to organize such metadata, and which is also used as the basis for a distributed data access method in computing clusters. A new method for histogram construction was proposed based on the proportional overlap of spatial objects and cell boundaries. The proposed method performs better than a well-know method, and provides a better estimate for the cost of execution plans in 75% of the queries evaluated. The complete methodology for cost estimation was used to implement an optimizer that was able to choose the best execution plan for all the queries considered in the experiments (which use real spatial datasets). Furthermore, the methodology also provided a good ordering of query plans (only two errors, but producing very similar plans), which indicates that it can be used to select good plans 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 duplicated results from being reported by the join operation. The modified version of CJ is also a non-blocking algorithm, meaning that it can be used to efficiently process multiway spatial join queries in distributed systems.

We plan to investigate new metrics to be included in the histogram, such as the current workload that results from the execution of concurrent multiway spatial join queries. Besides having the potential to produce more precise estimates, this may lead to a more realistic model for production-grade spatial databases and to address interesting challenges for the scheduling of distributed queries in a cluster. We also plan to replace the greedy algorithm by more elaborate combinatorial methods in order to achieve better query scheduling in the cluster.

REFERENCES

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, USA, pp. 309–318, 2012.

Brinkhoff, T., Kriegel, H. P., and Seeger, B. Efficient Processing of Spatial Joins Using R-trees. *SIGMOD Record* 22 (2): 237–246, 1993.

Brinkhoff, T., Kriegel, 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, USA, pp. 258–265, 1996.

Carvalho, L. O., Oliveira, W. D., Pola, I. R. V., Traina, A. J. M., and Traina Jr, C. A 'Wider' Concept for Similarity Joins. *Journal of Information and Data Management* 5 (3): 210–223, 2014.

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.

Cormode, G., Garofalakis, M., Haas, P. J., and Jermaine, C. Synopses for Massive Data: samples, histograms, wavelets, sketches. *Foundations and Trends in Databases* 4 (1-3): 1–294, 2012.

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 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Vitória, 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 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Brasília, Brazil, pp. 1009–1022, 2013.

de Oliveira, T. B., Costa, F. M., and Rodrigues, V. J. S. Definição de Planos de Execução Distribuídos para Consultas de Junção Espacial usando Histogramas Multidimensionais. In *Proceedings of the Brazilian Symposium on Databases*. Petrópolis, Brazil, pp. 89–100, 2015.

de Oliveira, T. B., do Sacramento Rodrigues, V. J., de Oliveira, S. S. T., de Albuquerque Lima, P. I., and de C. Cardoso, M. DSI-RTree - Um Índice R-Tree Escalável Distribuído. In *Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Campo Grande, 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, 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, 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.

Ioannidis, Y. E. and Kang, Y. Randomized Algorithms for Optimizing Large Join Queries. *SIGMOD Record* 19 (2): 312–321, 1990.

Jacox, E. H. and Samet, H. Spatial Join Techniques. *ACM Transactions on Database Systems* 32 (1): 1–44, 2007.

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.

Mishra, P. and Eich, M. H. Join Processing in Relational Databases. *ACM Computing Surveys* 24 (1): 63–113, 1992.

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, 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*. Philadephia, 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, USA, pp. 56–61, 2000.

Ray, S., Simion, B., Brown, A. D., and Johnson, R. Skew-resistant Parallel In-memory Spatial Join. In *Proceedings of the International Conference on Scientific and Statistical Databases Management*. Aalborg, Denmark, pp. 1–12, 2014.

Roh, Y. J., Kim, J. H., Chung, Y. D., Son, J. H., and Kim, M. H. Hierarchically Organized Skew-tolerant Histograms for Geographic Data Objects. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Indianapolis, USA, pp. 627–638, 2010.

Sivasubramaniam, A. Selectivity Estimation for Spatial Joins. In *Proceedings of the IEEE International Conference on Data Engineering*. Berlin, 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, 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.