Multiple-Site Distributed Spatial Query Optimization using Spatial Semijoins

Wendy OSBORN [a,1], and Saad ZAAMOUT [a]
[a] *Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta T1K 3M4, Canada*

Abstract. In this paper, we present our strategy for distributed spatial query optimization that involves multiple sites. Previous work in the area of distributed spatial query processing and optimization focuses only on strategies for performing spatial joins and spatial semijoins, and distributed spatial queries that only involve two sites. We propose a strategy for optimizing a distributed spatial query using spatial semijoins that can involve any number of sites in a distributed spatial database. In this initial work we focus on minimizing the data transmission cost of a distributed spatial query by identifying and initiating semijoins from the smaller relations in order to reduce the larger relations and minimize the cost of data transmission. We compare the performance of our strategy against the naïve approach of shipping entire relations to the query site. We find that our strategy minimizes the data transmission cost in all cases, and significantly in specific situations.

Keywords. Spatial data, distributed spatial queries, optimization, performance, data transmission cost

1. Introduction

A distributed spatial database system [12] consists of several spatial database sites that are dispersed geographically. Each site manages its own collection of spatial data, but work collectively for processing inter-site data requirements. An important requirement of a distributed spatial database is the ability to efficiently process a query that requires spatial data from multiple sites. Historically, research in distributed relational databases focused on generating query execution plans that minimized the cost of data transmission over the network [3,4,11]. However, spatial data is more complex than alphanumeric data. This increases the complexity of joining spatial relations. Therefore, CPU and I/O costs should also be considered when processing a distributed spatial query [12].

Most existing strategies that process a distributed spatial query only work for two sites. The one exception to this does not handle spatial joins. Therefore, this preliminary work begins to address this shortcoming by propose a strategy for processing and optimizing a distributed spatial query that handles more than two sites. Our strategy focuses on minimizing the data transmission cost of a query by applying spatial semijoins in a cost-effective manner. An evaluation of our strategy shows reduction in the data transmission cost over the naïve approach (i.e. the approach that involves shipping entire relations directly to the query site). In specific situations, this reduction in cost is significant.

[1] Corresponding Author

2. Background and Related Work

Most research in distributed spatial query processing focuses on spatial join algorithms, spatial semijoins algorithms, and the use of Bloom filters for processing distributed spatial queries. A spatial join [12] takes two relations R and S, each with a spatial attribute, and relates pairs of tuples between R and S based on a spatial predicate that is applied to the spatial attribute values. Examples of spatial predicates include the overlap, containment, and adjacency of two spatial attributes. A spatial semijoin [2] is performed by projecting the spatial attribute from one relation, transmitting it to the site that contains the other spatial relation, and performing a spatial join of the spatial projection and relation. Then, the qualifying tuples from second site are shipped back to the first site and joined with the spatial relation on that site. A Bloom filter [1] is a hashed bit array that provides a compact but imprecise representation of the values of a joining attribute. A '1' bit represents the possible existence of a joining attribute value, while a '0' bit represents the absence of the value.

With the exception of [6], all proposed strategies for processing distributed spatial queries work for a two-site database only. We summarize these works below.

2.1. Spatial Join

A significant majority of spatial join algorithms are designed for a centralized system [7]. This section focuses on spatial join strategies for distributed spatial queries.

Kang et al. [8] propose a parallel spatial join strategy that is adapted to a distributed spatial database environment. Their strategy has two phases: data redistribution, and filter and refinement. In the data redistribution phase, on each site, the space that contains objects is partitioned into regions (i.e. buckets). A subset of regions is transmitted between servers so that each server has the same regions for both data sets. This subset is chosen by estimating which will result in the lowest overall response time (although it is unclear if I/O costs are considered). Then, the filter and refinement phase is carried out on both sites by performing a spatial join. An experimental evaluation shows that the parallel spatial join technique has a significantly faster response time – up to a 33% improvement over a semijoin-based strategy.

2.2. Spatial Semijoin

Abel *et al.* [2,13] propose a spatial semijoin operator that combines the conventional semijoin operator with the filter stage of spatial query processing in order to reduce the data transmission, I/O and CPU costs. Their work explores two adaptations of the spatial semijoin. In the first, a "projection" of a set of MBRs from one spatial relation is transmitted to the second site and applied to the other relation using a spatial join. In the second, the "projection" is a single-dimensional mapping that represents the objects in each relation. A performance evaluation between these approaches shows that 1) for datasets with very large spatial descriptions, both strategies perform the same, 2) for datasets with smaller spatial descriptions, a semijoin that uses single-dimensional mapping works best, 3) using the R-tree for retrieving MBRs incurs significant CPU costs, and 4) single-dimensional mapping causes more false drops than MBRs.

Karam and Petry [10] propose a spatial semijoin, which differs from [2,13] in that MBRs from different levels of the R-tree are chosen for the spatial semijoin, instead of

requiring that all come from the same level. A performance evaluation shows that their spatial semijoin outperforms the naïve spatial join (i.e. the whole relation is shipped to the other site for joining) when applied to real world data, but not when applied to randomly distributed rectangle sets. Limitation of their work are: 1) no comparison versus other strategies, 2) no consideration of CPU time.

2.3. Bloom Filters

Karam [9] propose a 2-dimensional bit-matrix approach for performing a semijoin of two relations that focuses on minimizing the data transmission, I/O and CPU costs. A 2-dimensional space is partitioned into equal-sized regions, with each region mapping to a bit in a 2-dimensional array. If a region contains objects, the corresponding bit is set to 1. This bit-matrix is transmitted to the site containing the other relation, and is applied by testing each region containing objects for the existence of a '1' bit in the bit-matrix. Any qualifying objects are sent to the first site. A performance evaluation shows that this approach shows the best improvement when applied to real world data. Limitation of this work are: 1) an evaluation against the spatial join, and not versus a spatial semijoin, which in functionality is a closer match to the bit-matrix approach, and 2) no compression of the bit-matrix – a bit-matrix that contains many zeros is still transmitted in its entirety.

Hua et al. [6] propose the BR-tree, which is an R-tree that is augmented with Bloom filters to support exact-match queries. Each node entry contains 1) a minimum bounding rectangle (MBR) that approximates an object or a subset of objects, and 2) a Bloom filter that also represents one or more objects. In a leaf node, a Bloom filter is created by taking each object and producing k bits in the filter using different hash functions. In a non-leaf node, a Bloom filter is created by intersecting the Bloom filters in its child node. Although the BR-tree supports exact-match queries by using Bloom filters, it still requires the MBRs for region and point queries. A strategy for processing distributed region, point, and exact-match queries is proposed. The algorithm duplicates the root of every BR-tree across every site in the database. Any objects that pass the test against a root node is shipped to the site containing the original BR-tree. This strategy works for any number of sites. A significant limitation is a lack of support for spatial joins.

3. Distributed Spatial Query Processing Strategy

In this section we present our algorithm for processing a distributed spatial query. The focus is to reduce the cost of data transmission over the network by using spatial semijoins. In the future we will also consisder I/O and CPU costs.

3.1. Preliminaries

We apply spatial semijoins by shipping the smaller spatial attributes to other sites and applying them to the larger relations in order to eliminate a significant amount of data that will not participate in the final result. We utilize a modified version of the approximation-based spatial semijoin that is proposed in [13].

In our implementation of the spatial semijoin, we use the following approach. We have two sites R and S that each contain a spatial relation. First, we obtain the projection of the spatial attribute from R. Then, the spatial relation is transferred to S and joined

with the spatial relation on S. Our semijoins differs after this point. Instead of sending the qualifying tuples from S back to R for the final join, we send back to R the identifiers from the spatial projection, which are used to select tuples from the relation on R to ship to the final query site. In addition, the tuples on site S that qualified in the semijoin are also shipped to the query site. Using this semijoin strategy allows us to incorporate more than two sites when processing a distributed query.

We make the following assumptions in our work:

1. Every spatial object is represented using its minimum bounding rectangle (MBR).
2. Every site that participates in the distributed spatial query has one spatial relation. If a site contains other relations that are required for the query, it is assumed that all local processing has taken place and one spatial relations remains.
3. All spatial relations have one spatial attribute.
4. All objects (and corresponding MBRs) in all spatial attributes are drawn from the same "spatial domain" (i.e. same region of space).
5. The cardinality of each spatial attribute is equal to the number MBRs in the relation. That is, we assume that all MBRs in a spatial attribute are distinct.
6. The number of sites participating in the distributed spatial query is a multiple of two. The reason for this will be made clear when the algorithm is presented.
7. The spatial attribute for every spatial relation is already indexed by an R-tree (or a similar index that places the minimum bounding rectangles for all objects in its leaf level).

3.2. The Algorithm

Given n sites that will be participating in processing a distributed query, where each site has one spatial relation, our strategy has four main steps:

1. Sorting and grouping by spatial attribute cardinality,
2. Transmission of spatial attributes,
3. Semijoin execution,
4. Transmission of qualifying tuples to query site for the final join and processing.

Each step is described next. First, the sites will be ordered by increasing spatial attribute cardinality. After ordering, the first $n/2$ sites of the ordered list are placed in a set P, while the remaining $n/2$ sites are placed in a set Q.

Then, the spatial attribute from the relation on each site in P are transmitted to a site in Q in the following manner:

- The attribute from the site with the smallest spatial cardinality in P is sent to the site with the smallest spatial attribute in Q,
- The attribute from the site with the next smallest spatial cardinality in P is sent to the site with the next smallest spatial attribute in Q,
- and so on... until,
- The attribute from the site with the largest spatial cardinality in P is sent to the site with the largest spatial attribute in Q.

Next, on each site in Q, a spatial semijoin is performed between the existing spatial relation and the spatial attribute sent from the corresponding site in P. The results of of

the semijoin are: 1) the set of tuples on the site that qualify in the semijoin, and 2) a set of identifiers from the spatial attribute whose MBRs also qualify in the semijoin. The set of identifiers is sent back to the corresponding site in P.

Finally, for all sites in P, the tuples whose identifiers match the ones obtained from Q are shipped to the query site. In addition, for each site in Q the set of tuples that qualified in the semijoin are sent to the query site. At the query site, the final join is performed.

3.3. Example

Suppose we have a distributed spatial database with six sites. Each site contains a spatial relation with 100, 200, 400, 600, 800, and 1000 tuples respectively. Our strategy for processing a query that involves these sites proceeds as follows. First, our sites are ordered by increasing spatial attribute cardinality. Then, the list is divided into the two sets. The set P will contain the sites with the 100-, 200- and 400-tuple relations, while the set Q will contain the sites with the 600-, 800-, and 1000-tuple relations.

Next, the spatial attributes from the sites in set P are sent to sites in Q in the following manner. First, the spatial attribute from the site containing 100 tuples is sent to the site that contains 600 tuples. Similarly, the spatial attribute from the 200-tuple sites is set to the 800-tuple site, and the spatial attribute from the 400-tuple site is sent to the 1000-tuple site. Then, on the 600-, 800-, and 1000-tuple sites, a semijoin is performed between the local spatial relation and the spatial attribute that was shipped to it. During this process, the identifiers that correspond to the MBRs in the spatial attribute that qualify for the semijoin are sent back to originating site. For example, on the 600-tuple site, the identifiers for the qualifying MBRs are sent back to the 100-tuple site, and are used to select the corresponding tuples. Finally, all qualifying tuples from all sites are shipped to the query site.

4. Experimental Evaluation

Here, we present our empirical evaluation of our distributed query processing algorithm. We compared our strategy for optimizing a distributed spatial query with the naïve approach which transfers all unreduced relations to the query site. First, we present the data sets and cost formulas used in our evaluation. Then, we present the results and discussion of our tests.

We simulated a six-site distributed spatial database, where each site contains one spatial relation. Each spatial relation has one spatial attribute, which consists of four values (lx, ly, hx, hy) that represent the extents of an MBR. In addition, each spatial relation has the following non-spatial attributes: identifier, region name, population and a line slope indicator. Each spatial relation has 100, 200, 400, 600, 800 and 1000 tuples respectively. We opted to use smaller relations for our experiments because of the preliminary nature of the work and the use of a simulated (and not real) distributed environment.

4.1. Data Transmission Cost Calculation

In our experiments, we estimated the cost of data transmission as the total number of bytes that are transmitted. We assume that the data transmission rate is constant and therefore is not added to our calculations. In addition, we assume a integer size of two

bytes, a double-precision floating point size of eight bytes, a long integer size of 8 bytes and a character size of one byte.

The various costs of data transmission are calculated in the following manner. There are several calculations required. First, the transmission cost for transmitting an MBR is equal to the number of bytes used to represent an MBR:

$$cost(MBR) = 4 * sizeof(double) + sizeof(int) \tag{1}$$

which encompasses the co-ordinate values (lx, ly, hx, hy) and the tuple identifier. Similarly, the cost for transmitting a tuple is:

$$\begin{aligned} cost(tuple) = {} & sizeof(MBR) + 20 * sizeof(char) + sizeof(longint) \\ & + sizeof(int) \end{aligned} \tag{2}$$

which encompasses the region name, population and line slope indicator. In addition, 1) the cost of transmitting an identifier back to the original site from which it came is $cost(ID) = sizeof(int)$, and, 2) the function $number\_of\_qualifiers(relation)$ returns the number of tuples from a relation that participate in the result of a spatial semijoin operation.

Finally, given spatial attribute X from relation Y (i.e. site Y from the set P above) that is shipped to relation Z (i.e. site Z from set Q above), the cost of processing the spatial semijoin is:

$$\begin{aligned} cost(X, Y, Z) = {} & number\_of\_tuples(Y) * cost(MBR) \\ & + number\_of\_qualifiers(X) * (cost(ID) + cost(tuple)) \\ & + number\_of\_qualifiers(Z) * cost(tuple) \end{aligned} \tag{3}$$

The first term is the cost of transmitting the spatial attribute X from site Y to site Z. The second term is the cost of both transmitting back to Y the corresponding tuple identifiers for the qualifying MBRs in X, and then transmitting the tuples that correspond to those tuple identifiers to the query site. Finally, the third term is the cost of transmitting qualifying tuples from Z to the query site. This cost is calculated for every pair (Y,Z) of sites that are involved in the query, with all costs summed together to obtain the total cost of the query.

4.2. Two-Site Query Test

The first set of tests we performed are for distributed queries that involve two sites. Table 1 shows the pairs of relations (i.e. sites) that were evaluated, along with the total cost (in bytes) of both our optimized strategy (column Optimized) and the naïve approach (column Naïve). We opted to report the cost of data transmission in bytes so that determining the number of physical disk blocks in a page of secondary storage would not be required at this point.

In all cases, our strategy results in a lower data transmission cost over the naïve approach. In particular, the most significant improvement is achieved when there exists a significant difference in the size of the spatial relations between the two sites. For example, when the query involves the sites that contain the 100- and 1000-tuple spatial relations, we have almost 80% less data that is being transmitted when our strategy is being used to process the query.

Table 1. Two-Site Query Test

Site 1	Site 2	Optimized	Naïve	%Improvement
100	400	16010	32000	50
100	600	16270	44800	64
100	800	15750	57600	73
100	1000	14580	70400	79
200	400	32150	38400	17
200	600	31760	51200	38
200	800	32020	64000	50
200	1000	31890	76800	59

Table 2. Four-Site Query Test

Site 1	Site 2	Site 3	Site 4	Optimized	Naïve	%Improvement
100	200	400	600	52264	83200	37
100	200	800	1000	53410	134400	60
400	600	800	1000	162604	172900	6

4.3. Four-Site Query Test

For our second set of tests, we compared the evaluation of the strategies for four-site queries. Table 2 shows the sites involved and the total costs in bytes from both strategies. Again, we find that our strategy outperforms the naïve approach. In addition, we also find that in the situation where a significant size difference exists between the relations - in this case, 100, 200, 800, and 1000 tuples - the greatest improvement is achieved.

4.4. Six-Site Query Test

Finally, we performed one test that compares our strategy with the naïve approach when all six sites are involved. We found the transmission cost from the query optimization strategy to be 127,456 bytes and that from the naïve strategy to be 198,400. This gives an improvement of approximately 36%.

4.5. Discussion

In all cases, we discovered a lower data transmission cost from our strategy over the naïve approach. In addition, we discovered the following trends. First, the queries with the largest difference in the number of tuples between the participating relations, the greater the reduction that our strategy achieves. Second, we discovered that as the difference in the number of tuples between participating relations increases, the improvement that our strategy achieves increases as well.

5. Conclusion and Future Work

In this paper, we propose a strategy for optimizing queries in a distributed spatial database that involves relations on multiple sites. Our strategy focuses on minimizing the cost of data transmission by applying spatial semijoins. Smaller spatial attributes are

chosen for transmission and application to larger relations so that overall data transmission costs are reduced. A empirical evaluation of our strategy against the naïve approach shows that our strategy achieves a reduction in the data transmission cost in all cases. In particular, as the size difference between relations increases, the savings achieved by our strategy over the naïve strategy are very significant.

As mentioned, one important direction of future work that we are currently exploring is the resulting I/O and CPU costs from our optimization strategy. It is important to determine if the I/O and CPU costs are minimal or outweigh any benefits of our strategy. Other directions of future work include the following. One is to create a real distributed database system with multiple sites, which will provides a means for better evaluation of our strategy. Another is to evaluate the two-site version of our strategy (i.e. when only two sites are involved) versus other existing strategies. Although the focus of this work was to extend the number of sites involved in a distributed spatial query, evaluating the efficiency of our algorithm in the two-site case versus existing strategies is also important and would better identify if our strategy is superior in this situation. A final research direction is to develop and evaluate other strategy for processing and optimizing a distributed spatial query. As discussed, very limited work has been proposed, which leads to many exciting opportunities for research in the area of distributed spatial query processing.

References

[1] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM* **13** (1970), 422–426.

[2] D.J. Abel, B.C. Ooi, K.-L. Tan, R. Power and J.X. Yu, Spatial join strategies in distributed spatial DBMS, *Proceedings of the 4th International Symposium on Advances in Spatial Databases*, 1995.

[3] P.M.G. Apers, A.R. Hevner and S.B. Yao, Optimization algorithms for distributed queries, *IEEE Transactions on Software Engineering* **9** (1983), 57–68.

[4] P. Bodorik, J.S. Riordon and J.S. Pyra, Deciding to correct distributed query processing, *IEEE Transactions on Knowledge and Data Engineering* **4** 1992, 348–357.

[5] T. Brinkhoff, H.-P. Kriegel and B. Seeger, Efficient processing of spatial joins using R-trees, *Proceedings of the 1993 ACM Sigmod International Conference on Managment of Data*, New York, USA, pp. 237–246, 1993.

[6] Y. Hua, B. Xiao and J. Wang, BR-Tree: a scalable prototype for supporting multiple queries of multidimensional data, *IEEE Transactions on Computers* **58** (2009), 1585–1597.

[7] E. Jacox and H. Samet, Spatial join techniques, *ACM Transactions on Database Systems* **34** (2007), 1–44.

[8] M.-S. Kang, S.-K. Ko, K. Koh and Y.-C. Choy, A parallel spatial join algorithm for distributed spatial databases, *Proceedings of the 5th International Conference on Flexible Query Answering Systems*, pp. 212–225, 2002.

[9] O. Karam, *Optimizing Distributed Spatial Joins using R-trees*, Ph.D. Thesis, Tulane University, 2001.

[10] O. Karam and F. Petry, Optimizing distributed spatial joins using R-trees, *Proceedings of the 43rd ACM Southeast Regional Conference*, pp. 222-226, 2006.

[11] M.T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, Springer, New York, 2011.

[12] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*, Prentice Hall, New Jersey, 2003.

[13] K.-L. Tan, B.C. Ooi and D.J. Abel, Exploiting spatial indexes for semijoin-based join processing in distributed spatial databases, *IEEE Transactions on Knowledge and Data Engineering* **12** (2000), 920–937.