Optimization of Spatial Join Operation using Map-Reduce & Indexing on Hadoop System

Topia Munira M. Salim
M.E. Student
Department of Computer Engineering
Silver Oak College of Engineering & Technology, Ahmadabad, India

Abstract— Performing Spatial Join operation on voluminous data is time consuming operation which degrades performance of entire systems. We propose an optimized approach to perform spatial join using map-reduce on HDFS (Hadoop Distributed File System) system. Currently various algorithms for computing spatial join with indexing and spatial data type support do exists, some of them work very efficient on spatial-hadoop systems. In this paper an indexing using quad-tree for spatial data has been proposed for non-spatial, simple hadoop system, to improve the performance.
Key words: Spatial Join Operation, Map-Reduce, Hadoop

I. INTRODUCTION

MapReduce [1] is a parallel programming model which is proposed by Google to process large scale data. Using an interface with two functions map and reduce, facilitates parallel program implementation for many real-world job on clusters. Using MapReduce programming model benefits are achieved such as load balance, task scheduling and fault tolerance.

Spatial join is one of the most widely used spatial query in applications such as spatial DBMS, robotics, game programming and VLSI designing, etc; which combines two spatial data sets with a spatial relationship between the objects. Efficient spatial join processing is extremely important since its execution time is directly proportional with the object numbers in data sets. It is a good idea to implement spatial join with MapReduce to take an advantage of parallel computing. With this we can greatly reduce the processing time for such a computing intensive operation [2].

Many difficulties have been found to parallelize spatial join with MapReduce such as, MapReduce focuses mainly on processing homogeneous data sets, while spatial join has to deal with two heterogeneous data sets [3]. Another, spatial objects are generally larger and more complex, plus the spatial join predicates are complex and time-consuming

Many solution to this problem has been provided such as (1) Parallel-Secondo a parallel spatial DBMS that uses Hadoop as a distributed task scheduler, (2) MD-HBase extends HBase, a non-relational database for Hadoop, to support multidimensional indexes, and (3) Hadoop-GIS extends Hive, a data warehouse infrastructure built on top of Hadoop with a uniform grid index for range queries and self-join [4]. Main drawback in all these systems is that they still deal with Hadoop as a black box and remain limited by the limitations of existing Hadoop systems.

Another Solution: SpatialHadoop, a full-fledged MapReduce framework with native support for spatial data; available as open-source [3]. SpatialHadoop overcomes the limitations of Hadoop-GIS and all previous approaches as: (1) SpatialHadoop is built-in Hadoop base code (around 14,000 lines of code inside Hadoop) that pushes spatial constructs and the awareness of spatial data inside the core functionality of Hadoop. This does takes advantage of indexing techniques such as Grid and R-tree partitioning. Other partitioning and indexing techniques do exists and generated within less time; Eldaway compared results [5] and found that Quad tree outperforms all other techniques as it minimizes the number of overlapping partitions between the two files by employing a regular space partitioning which is perfectly aligned across different files. This finding conforms with earlier work in the literature which showed that Quad tree partitioning outperforms both R-tree g when running a spatial join query in a traditional DBMS environment [6].

II. BACKGROUND & RELATED WORK

A. MapReduce

MapReduce is a programming model and platform suited for parallel computation [1]. In MapReduce, a program consists of a map function and a reduce function which are user-defined. The input data format is application related, and is specified by the user. The output is a set of < *key*, *value* > pairs. The MapReduce signatures are listed below:

$$map: (k1, v1) \rightarrow [(k2, v2)] \quad (2.11)$$
$$reduce: (k2, [v2]) \rightarrow [k3, v3]$$

As shown in the equation (1), the map function applies user defined logic on every input key/ value pair (*k1*, *v1*) and transforms it into a list of intermediate key/ value pairs [(*k2*, *v2*)]. Then the reduce function applies user-defined logic to all intermediate values [*v2*] associated with the same *k2* and produces a list of final output key/value pairs [*k3*, *v3*].

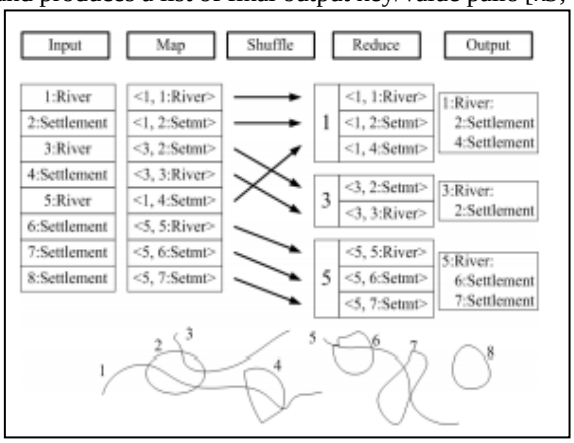

Fig. 1: MapReduce Application Calculating Settlements Passed through by Each River

MapReduce provides many benefits over other parallel processing models [2]. It allows simple data processing programs to benefit from MapReduce implementation for communication, load balance, fault tolerance, resource allocation, job startup, and file distribution.

B. Spatial Join

Given two sets of multidimensional objects in Euclidean space, a spatial join query can discover all pairs of objects satisfying a given spatial relation, such as intersection [7]

For example, a spatial join answers query like finding all the roads spanning any rivers, given a road table and a river table. Since the representation of a spatial object can be very large and complex, spatial join typically operate in two steps:

1) Filter Step

An approximation of each spatial object, such as its minimum bounding rectangle (MBR), is used to check with the spatial predication, eliminating tuples that cannot be parts of the result. This step produces candidates which are a superset of the actual result.

2) Refinement Step

Each candidate pair of the last step is examined to check whether its spatial properties satisfy the spatial predicate. A CPU-intensive computational geometry algorithm is generally used.

C. Indexing

existing techniques for spatial indexing in Hadoop fall in three broad categories: (1) Build only: A MapReduce approach is proposed to construct an R-tree [8], [9] but the R-tree has to be queried outside MapReduce using traditional techniques. (2) Custom on-the-fly indexing: With each query execution, a non-standard index is created and discarded after query completion (3) indexing in HDFS: Up to our knowledge, only [10] builds an HDFS-based index structure for a set of trajectories. Yet, the index structure can only support range queries on trajectory data, which is a very limited functionality for what we need for an operation.

We employ a two-layers indexing approach of global and local indexes as shown in Fig. 2[5].

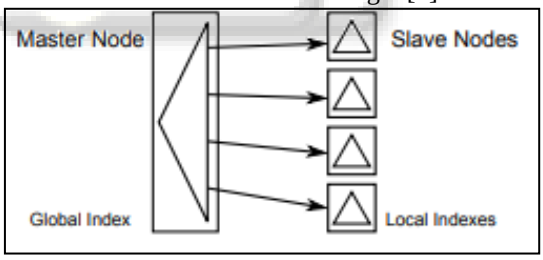

Fig. 2: Spatial Indexing in Spatial Hadoop

Index building approach is used similar to as in SpatialHadoop[5] is composed of three main phases, namely, partitioning, local indexing, and global indexing.

Phase I: Partitioning: This phase spatially partitions the input file into n partitions that satisfy three main goals: (1) Block fit; each partition should fit in one HDFS block of size 64 MB, (2) Spatial locality; spatially nearby objects are assigned to the same partition, and (3) Load balancing; all partitions should be roughly of the same size.

D. Quad Tree Partitioning

The Minimum Bounding Rectangle (MBR) of each object is stored along with its identifier within the list of quadrants object references. These references to the objects of the quadrant are ordered according to the MBR. The ordering aims at enhancing the intersection between two quadrants since a plane-sweep .This technique inserts all sample points into a quad tree [11] with node capacity of ⌊k/n⌋, where k is the sample size. The boundaries of all leaf nodes are used as cell boundaries. We use the replication method to assign records to cells.

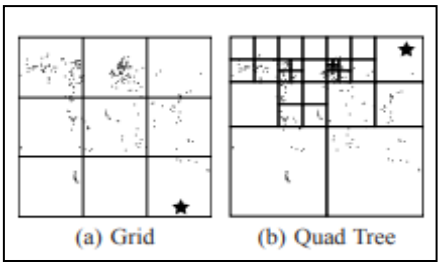

Fig. 3: Partitioning Techniques

III. Spatial Join with Map Reduce on Hadoop using Quad Tree

Before carrying out spatial join operation of two data sets R and S are first indexed using quad tree. Each data set is considered to represent data over an area covering large MBR (Considered this step as Normalization. Since we map data set over (0,0) to (100,100) region). Then sample from normalized data is drawn and large MBR is divided into four equal quadrants. Total 1% of original data set is enough for sample size, as it correctly partitions data set.

Quad-trees are known for their large storage requirements. To overcome this disadvantage we adopted the solution proposed in [12]: the utilization of linear quad trees. In this approach, a space-ordering method is chosen as z-order so each quadrant receives a code according to this order.

Algorithm we chose for joining is Distributed join which has three main steps, namely global join, local join, and duplicate avoidance

1) Step 1: Global join. Given two input spatial data sets R and S, this step produces all pairs of file blocks with overlapping MBRs. Result of which only an overlapping pair of blocks can contribute to the final answer of the spatial join since records in two non-overlapping blocks are definitely disjoint. This step resembles the filter of spatial join operation. Input to this step is global indexed files of two data sets.
2) Step 2: Local join. Given a combined split produced from the previous step, this step joins the records in the two blocks in this split to produce pairs of overlapping records. The result of the local join may contain duplicate results due to having records overlapping with multiple blocks.
3) Step 3: Duplicate avoidance. For each detected overlapping pair of records, the intersection of their MBRs is first computed. Then, the overlapping pair is reported as a final answer only if the top-left corner (i.e., reference point) of the intersection falls in the overlap of the MBRs with the two processed blocks.

IV. Performance Evaluation

In order to verify the effectiveness of the proposed spatial join query optimization algorithm, experiments on real data sets is made, and the algorithm is compared with the original join query processing algorithm SJMR [2] and DJ (Distributed

Join) of Spatial Hadoop[3]. Next we will give the experimental setup and results in detail.

A. Experimental Setup

The experimental environment consists of two IBM servers with processor eight core, 2.0 GHz, 64 GB memory and 1 TB SATA disk. Each machine has installed Centos 6.4 operating system, the Hadoop 2.0 cluster software and the related algorithm modules.

B. Experimental Results

We use the real data to test the performance of the proposed algorithm, the real data from Census 2000 TIGER map file data sets, among these the number of road data is 2092079, the number of hydrological data is 37950.

The above experiments on real datasets show that under the condition of the appropriate division dimensions, the proposed algorithm obviously performs better than the algorithm SJMR and approximately equal performance as of DJ on Spatial Hadoop. Proposed algorithm has a good scalability and applicability than SJMR. Although it is clearly noted that it does requires more time in index building and storing compared to R-tree and grid.

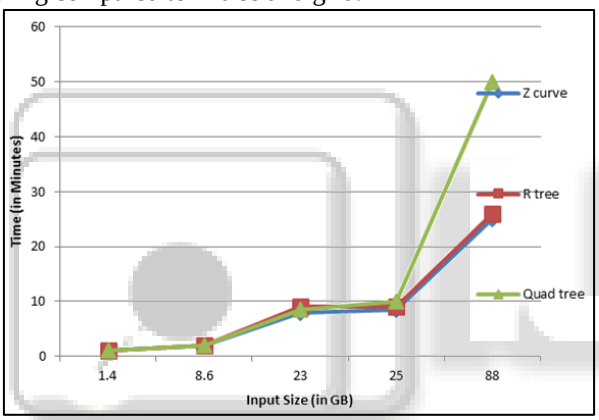

Fig. 4: Partitioning Time

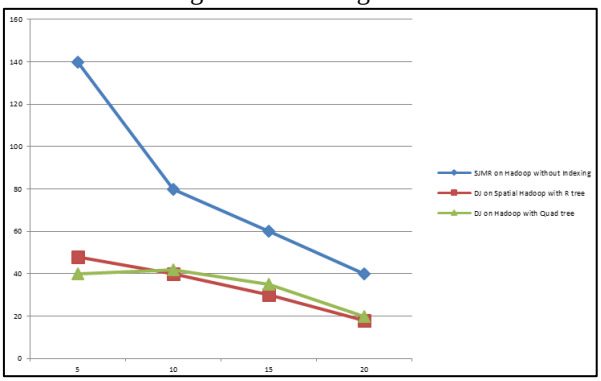

Fig. 5: Spatial Join

V. CONCLUSIONS

A common spatial data query, spatial join query occupies an important position in spatial applications, and its performance has important effects on the whole spatial application system. Quad trees have already been considered viable alternatives to R-trees as spatial indexing structures, being especially well-suited to parallel processing environments. We adapted this spatial index structures in traditional hadoop environment to achieve measurable higher throughput and performance in terms of execution time.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, vol. 51, no. 1, pp. 107, 2008.
[2] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu, "SJMR: Parallelizing spatial join with MapReduce on clusters," in CLUSTER, 2009.
[3] Ahmed Eldawy Mohamed F. Mokbel, SpatialHadoop: A MapReduce Framework for Spatial Data, USA, under Grants IIS-0952977 and IIS-1218168, IEEE, 2015.
[4] http://spatialhadoop.cs.umn.edu/.
[5] Ahmed Eldawy, Louai Alarabi, Mohamed F. Mokbel , Spatial Partitioning Techniques in SpatialHadoop, IEEE, 2150-8097/15/08 , 2015
[6] E. G. Hoel and H. Samet. Performance of Data-Parallel Spatial Operations. In VLDB, pages 156–167, 1994.
[7] E.H. Jacox and H. Samet, "Spatial join techniques," ACM Transactions on Database Systems (TODS), vol. 32, no. 1, 2007.
[8] Cary, Z. Sun, V. Hristidis, and N. Rishe, "Experiences on Processing Spatial Data with MapReduce," in SSDBM, 2009.
[9] H. Liao, J. Han, and J. Fang, "Multi-dimensional Index on Hadoop Distributed File System," ICNAS, vol. 0, 2010.
[10] Q. Ma, B. Yang, W. Qian, and A. Zhou, "Query Processing of Massive Trajectory Data Based on MapReduce," in CLOUDDB, 2009.
[11] R. A. Finkel and J. L. Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. Acta Inf., 4:1–9, 1974.
[12] Hod, E.G., Samet, H.: "A Qualitative Comparison Study of Data Structures for Large Line Segment Databases", Proc. ACM SIGMOD Itit. Cot$ On Managenlent of Data, San Diego, June, 1992, pp. 205-214

