Parallel Spatial Join Aggregation with Two Combine Stages in Map-Reduce Framework

Chuang Yao , Luo Chen [a] , and JinXin Shen

College of Electronic Science and Engineering, National University of Defense Technology, China
[a]luochen@nudt.edu.cn

Keywords: Spatial join aggregation, Map-Combine-Reduce, Cloud computing

Abstract. Current analysis show that Map-Reduce cannot directly support spatial join operations with secondary reduction features efficiently. This paper proposes a Map-Reduce based parallel strategy---Map-Combine-Reduce (MCR)---to treat the join aggregation of large-scale spatial data. MCR takes advantage of combine stage in Map-Reduce model to integrate the partial aggregation results distributed at various maperp and reducers. Filter optimization was proposed to facilitate single-allocation issues of spatial objects in division of parallel tasks to further improve the efficiency of join aggregation querying. Experimental results have proven its efficiency, effectiveness, and simplification during processing spatial join aggregation query.

1 Introduction

Spatial join aggregation (SJA) query is a common but important operation. Based on spatial attribute information, SJA returns aggregation information (based on the sum, mean, maximum and minimum value) in two or more spatial datasets. The SJA query of statistic information on return joining is more popular with customers sometimes than pure spatial joining. For example, in traffic monitoring systems, the space including join query results (looking for vehicles in traffic areas) are usually meaningless because of the frequent movement of vehicles, but the records of number of vehicles in specific traffic areas are more helpful.

SJA queries take more time and space cost than non-spatial join aggregation queries and requires expensive join and aggregate operations on massive spatial data [1]. The classic algorithms [2, 3] studied so far are performed in standalone machine environments, and it can be used to solve aggregation query of small-scale spatial datasets. However, there is a recent explosion in the amounts of spatial data. For example, satellite data archives in China Center for Resource Satellite Data and Application exceeded 5PB and is still growing; taxi trajectory data collected from Changsha city, a typical medium size city in south China, is about 20 million records per day [4]. This brought up severe challenges to traditional strategy to process SJA queries in standalone machine environments.

Parallel computation on clusters is an effective approach to process large-scale spatial data. Map-Reduce (MR) [5] is a distributed programming framework as well as a programming model which has been popularized by Google for processing a massive volume of data in parallel with many cheap configured computing nodes. Scholars have started to study spatial query techniques on MR framework [6], including spatial join [7], spatial kNN [8] and polygonal overlay processing [9]. However, SJA query refers to multiple heterogeneous data sources, and it needs secondary reduction process due to the multi-distribution features in the division of parallel join tasks. However, the MR model is applicable to single reduction processing, it does not directly support the join aggregation of multi-source heterogeneous data or secondary reduction processing in SJA.

In this paper, we propose an extended framework named MCR by taking the advantage of combine stage in traditional Map-Reduce model. The MCR model is used to enhance the ability to support multi-stage reduction and produces a parallel computing model to solve the data processing issues that require multi-source heterogeneous inputs and secondary reduction. Subsequently, we propose a multi-stage spatial join aggregation algorithm and filter optimization strategy with consideration of the distribution features of spatial object in parallel SJA processing. Experimental results indicate that the improved parallel computing framework is better than the original algorithms.

2 MCR model

The secondary reduction for computation assignment needs to add an additional MR process, which results in unnecessary serial communication costs between data transmission and MR. The Map-Combine-Reduce (MCR) model extends the Map-Reduce model and adds two combine processes to support secondary reduction. It also provides a data division mode to process heterogeneous joining.

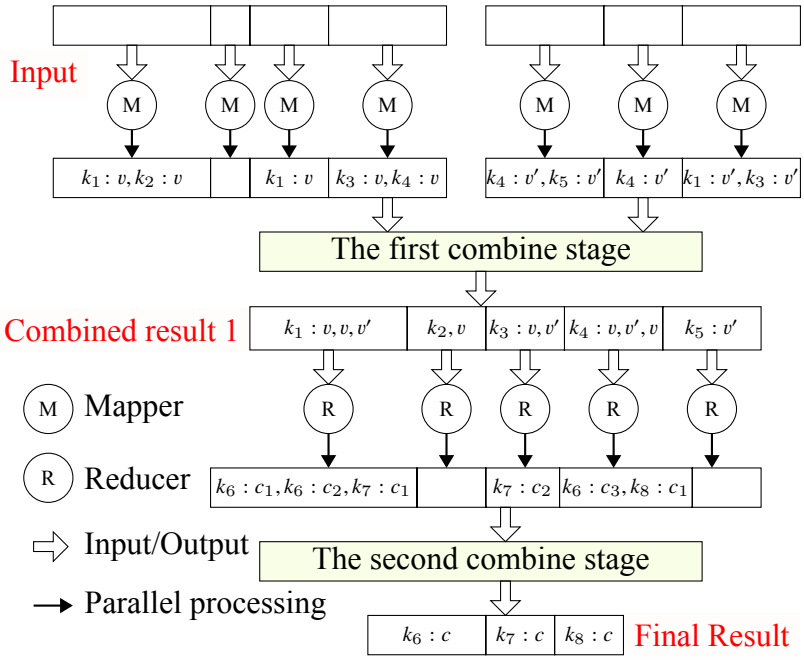

Fig. 1: MCR model

Fig. 1 describes this process as represented as follows:

- **map**:$\{(k_a, v_a)_i\} \mapsto \{[(k_b, v_b)]_i\}$
- **1st combine**: $\{[(k_b, v_b)]_i\} \mapsto \{(k_p : v_1, v_2, \ldots, v_p)\}$
- **reduce**: $\{(k_p : v_1 \ldots v_p)\} \mapsto \{[(k_c, v_c)]_j\}$
- **2nd combine**:$\{[(k_c, v_c)]_j\} \mapsto \{[(k_q, v)]_l\}$

where i, j and l represent different data sources, respectively. Let k be the key, and v the value. In the MCR model, the map function firstly transfer input key/value pair (k_a, v_a) to a series of intermediate results in the form of key/value pair $[(k_b, v_b)]$ in each data source. The first combine stage exerts homogenization processing (HP) according to the data source, and groups values from all pairs with k_p. Reduced function gathers all value set related to k_p (v_p may come from different data sources) in 1st combine stage, and executes spatial join aggregation operations on ($[v_p]$) to generate a series of key/value pair result sets $[(k_c, v_c)]_j$. After the reduce stage of the MR framework, the 2nd combine function gathers k_q related value sets and generates final key/value pair result sets $[(k_q, v)]_l$ in data source l. Unlike the Map-Reduce framework, the new model supports and processes multiple sources, and uses two combine stages to support secondary reduction processing. The new model is a four-stage (map-combine-reduce-combine) independent parallel and serial synchronization of the computational model. The internal asynchronous parallel assignment during the combine stage also operates under the independent parallel ideal status.

3 Spatial Join Aggregation with MCR

According to the MCR model, parallel spatial join aggregation query is divided into four serial parallel synchronous stages: 1) map stage in which the whole task is decomposed into parallel sub-tasks based on spatial distribution of objects; 2) first combine stage in which parallel join sub-tasks were generated; 3) reduce stage in which spatial join aggregation is processed in parallel in each sub-task; 4) second combine stage in which the final result is obtained with the integration of the results of each reducer.

3.1 Map stage for SJA In map stage, we decomposed the join task into several sub-tasks. Suppose the input datasets are R and S, the join region is D. We first divide D into N non-intersected tiles d_i; each d_i will be assigned to a processing task. Objects in R and S are distributed to each d_i according to their MBR. Fig. 2 shows an example, where D is divided into 4×4 tiles and tile d_0 contains $\{s_1, r_1\}$ which will be process in one sub-task.

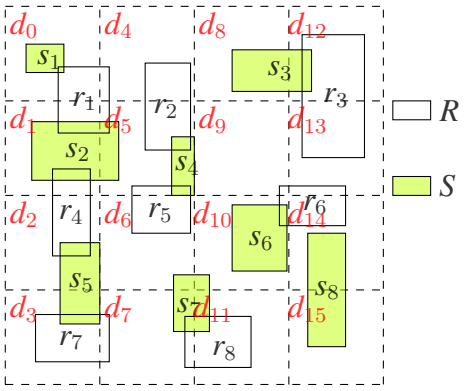

Fig. 2: Devision of join region

Thus we define the map function as follows in Algorithm 1.

Algorithm 1: Map function of SJA

Input : (*oid*: spatial object id, *geo*: geometry of the object)
Output: (di, oid)

```
1  MBR = GetMBRfromGeometry(geo);
2  outputlist = null;
3  for i = 0 to N − 1 do
4      if d_i intersect MBR then
5          outputlist.append(d_i, oid);
6      end
7  end
8  return outputlist
```

Surely we may improve the algorithm if there is a spatial index for D.

3.2 The first combine stage for SJA

In the first combine stage, we group the result of the map stage into N tuples by the key d_i in each output item of map function. The N tuples are correspond to N sub-tasks. Suppose we have P reducers ($P << N$), as in the actual scenario, we need to divide N sub-tasks into P groups to distribute them on those reducers and to ensure the balance of loads. Considering spatial proximity effect, we take a set of space filling curves to mark each sub-task, and define the combine algorithm as in Algorithm 2.

where R_i represents oid set from R in d_i and S_i represents oid set from S in d_i. In line 13, the $f(d_i)$ is the space filling curve mapping function. In the following experiments, we used 3 typical space filling cures which are: CMD curve, Z-curve and Hilbert curve.

Algorithm 2: First combine function of SJA

Input : M: result of map stage $\{(d_i, oid)\}$
Output: $\{(reducerID_i, (R_i, S_i))\}$

```
MBR = GetMBRfromGeometry(geo);
initiate groupedlist[i] (i = 0 to N − 1);
for j = 0 to item number of M − 1 do
    if d_i intersect MBR then
        append (oid) to groupedlist[M[j].d_i].R_i;
    end
    else
        append (oid) to groupedlist[M[j].d_i].S_i;
    end
end
initiate outputlist[i] (i = 0 to N − 1);
for i = 0 to N − 1 do
    outputlist[i].reducerID = f(d_i) mod P;
    outputlist[i].R_i = groupedlist[i].R_i;
    outputlist[i].S_i = groupedlist[i].S_i;
end
return outputlist
```

3.3 The second combine stage and its optimization In the second combine stage, we merge $(key, value)$ pairs of the reduce stage with the same key, and produce the final result as in Algorithm 3.

Algorithm 3: Second combine function of SJA

Input : *SJALst*:A list of (*object*,*count*) pairs from reduce stage
Output: *SJAResult*:A list of (*object*,*count*) pairs with unique object id

```
sort SJALst by object;
initiate SJAResult;
previousTuple=(nullkey, 0);
for i = 1 to item number of SJALst do
    if SJALst[i].object ≠ previousTuple.object then
        append (previousTuple) to SJAResult;
    end
    else
        previousTuple.count = previousTuple.object + SJALst[i].count;
    end
end
return SJAResult
```

3.4 The filtering optimization of the framework We noticed that in the map stage, objects in R and S are distributed to each d_i according to their MBR. Some of the objects was contained in only one tile, and would be processed only in one sub-task. We called this scenario the single assignment (SA). While others intersected with multiple tiles and would be processed in multiple sub-tasks. We called this scenario the multiple assignments (MA). This inspired us to develop a strategy to reduce the cost of the second combine stage. We added a mark at to object tuples according to scenario SA or MA during the map stage, and the output of Map function was converted to $(d_i, oid + at)$. The Reduce function was used to determine the key of output result. If at was SA, the Reduce function wrote it

directly into the result set. Otherwise, it was sent to the Combine stage for processing. As the strategy filtered SA objects after the reduce stage, we call this strategy the MCRF strategy.

4 Experiments

4.1 Experimental setup

We next present an experimental study of our algorithms. Our Hadoop cluster consisted of one master node and 32 slave nodes. Each node was equipped with 2 quad-core 2.4G Hz CPUs, 16 GB RAM, and a 167 GB local disk. Nodes were connected using 10-Gigabit Infiniband network. Datasets used in the experiments were the road network data of the United States (except Alaska and Hawaii)(R) with 29,692,784 objects, hydrograph data (H) with 7,066,849 objects, and population census data (C) with 8,137,053 objects.

The experiments evaluate the algorithm performance varying space division $P(T)$, node number N, and division function. In our experiments, $P(T)$ divided the join area into $2^k \times 2^k$ uniform grids, of which k ranged from 7 to 10 with 8 by default. the division functions included CMD division, Z-curve division, and Hilbert division with CMD division by default. N ranged from 4 to 32 with a default value of 32. The combiner number was set to 8.

We conducted two experiments, the spatial intersection join aggregation and the spatial contain semi-join aggregation. Both were commonly used join processing in spatial application. With the experiments, we tested the overall performance of the execution of primary Map-Reduce framework (MR) , the new parallel computing framework (MCR) and filtration optimization under new parallel framework (MCRF).

4.2 Evaluation of Spatial Intersection Join Aggregation

Experiments were conducted to evaluate the performance of spatial intersection join aggregate (SIJA) on dataset R and H.

Fig. 3 shows the variation of SIJA versus k values. Results shows that with the increase of k, the join area of spatial object became smaller, therefore the performance increased. But instead of decrease, the execution time of SIJA increased when $k > 8$. This indicates that when k is excessively large, the redundancy of spatial objects will increase, and the computation cost also increases during the reduction stage. As shown in Fig. 3, the performance of the SIJA algorithm based on MCR was at least 16% better than that based on primary MR. The filtration optimization under MCR resulted further improved the performance of processing SIJA. The experimental results are consistent with the theoretical analysis given before. Now that MCRF achieved optimal performance, the algorithm in the following experiments utilized MCRF.

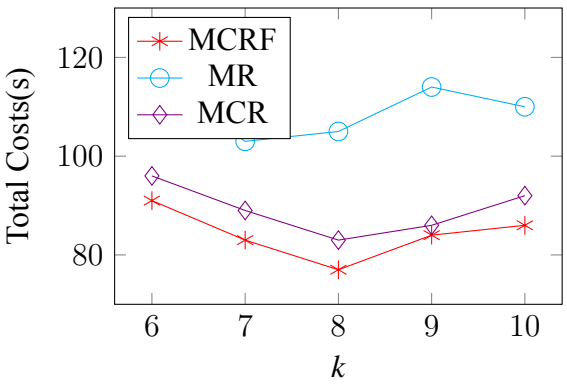

Fig. 3: Processing time of three algorithms with different k in SIJA

Fig. 4 shows the MCRF performance under various task divisions. Results showed that Hilbert-k and CMD division performed better than z-k division under the conditions of most k values. This indicated that the loads of distributed tasks were more balanced under Hilbert-k and CMD divisions.

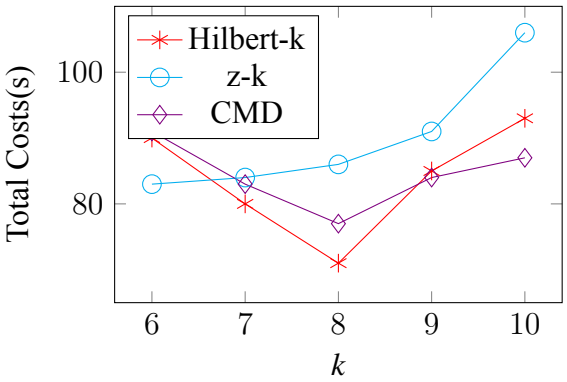

Fig. 4: Processing time of division strategies with different k in SIJA

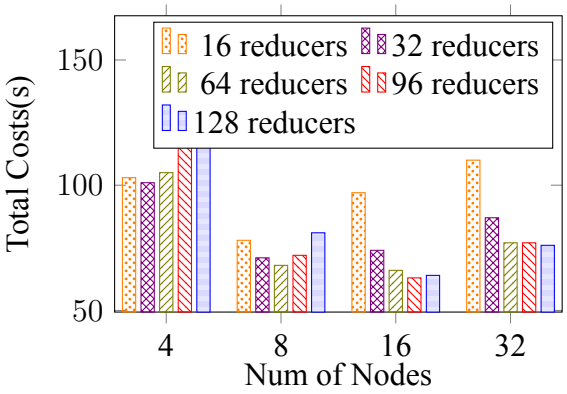

Fig. 5: Performance of MCRF algorithm with different node number in SIJA

Fig. 5 shows the variations in MCRF versus nodes. As the number of nodes increased, the SIJA execution time tended to decrease. This was because both the parallelism and processing capacity of the system increased as the number of nodes increased; but when $N > 16$, the SIJA execution time rebounded. This was because the processing capacity in 16 nodes was found to reach 128 cores of computational capacity. For SIJA under fixed computational strength, the excessive assignment on its computational tasks was found to contrarily increase its communication costs and depreciate its performance. Under the situation of fixed N, the performance of MCRF increased as r increased when the Reducer number met the conditions of $r \leq 8N$. This was primarily because each node had 8 cores and could simultaneously execute 8 tasks. When $r \geq 8N$, the Reducer task at each node cannot complete in one cycle, which resulted in the decrease of performance.

5 Summary

The purpose of this paper was to address the deficiency of current Map-Reduce parallel computing models in processing the join aggregation of multi-source heterogeneous spatial data. By using two combine stages on the basis of Map-Reduce framework, the system performed better in supporting secondary reduction. The addition of the combine stages reduced the computational cost in implementing secondary reduction, and provided a simple but highly efficient parallel computing mode for distribution of spatial join aggregation processing. In the proposed filtering optimization strategy, the join aggregation results of the SA objects were directly pushed into the Reduce stage. This method prevented any unnecessary operations in the data communication and combination stages and further improved the performance of the whole process. Experiments confirmed that, the SJA algorithm under proposed framework was more simple and efficient than the traditional one, which was based on primary Map-Reduce model.

Acknowledgments

This work is supported in part by the National High Technology Research and Development Program of China (Grant No. 2015AA123901).

References

[1] A. Stougiannis, F. Tauheed, T. Heinis, and A. Ailamaki, "Accelerating spatial range queries," Proceedings of the 16th International Conference on Extending Database Technology, EDBT '13, New York, NY, USA, pp.713–716, ACM, 2013

[2] N. Adrienko and G. Adrienko, "Spatial generalization and aggregation of massive movement data," IEEE Transactions on Visualization and Computer Graphics, vol.17, no.2, pp.205–219, 2011.

[3] L. Gmez, S. Haesevoets, B. Kuijpers, and A.A. Vaisman, "Spatial aggregation: Data model and implementation," Information Systems, vol.34, no.6, pp.551 – 576, 2009.

[4] S. Liao, L. Chen, J. Li, W. Xiong, and Q. Wu, "A spatiotemporal aggregation query method using multi-thread parallel technique based on region division," ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol.II-4/W2, pp.1– 5, 2015.

[5] J. Dean and S. Ghemawat, "Mapreduce: A flexible data processing tool," Commun. ACM, vol.53, no.1, pp.72–77, Jan. 2010.

[6] U. Bellur, "On parallelizing large spatial queries using mapreduce," in Web and Wireless Geographical Information Systems, ed. D. Pfoser and K.J. Li, Lecture Notes in Computer Science, vol.8470, pp.1–18, Springer Berlin Heidelberg, 2014.

[7] H. Gupta, B. Chawda, S. Negi, T.A. Faruquie, L.V. Subramaniam, and M. Mohania, "Processing multi-way spatial joins on mapreduce," Proceedings of the 16th International Conference on Extending Database Technology, EDBT '13, New York, NY, USA, pp.113–124, ACM, 2013.

[8] C. Zhang, F. Li, and J. Jestes, "Efficient parallel knn joins for large data in mapreduce," Proceedings of the 15th International Conference on Extending Database Technology, EDBT '12, New York, NY, USA, pp.38–49, ACM, 2012.

[9] S. Puri, D. Agarwal, X. He, and S.K. Prasad, "Mapreduce algorithms for gis polygonal overlay processing," Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, pp.1009–1016, IEEE, 2013.