

# A Parallel Spatial Co-location Mining Algorithm Based on MapReduce

Jin Soung Yoo\*, Douglas Boulware† and David Kimmey‡

\*‡Department of Computer Science

Indiana University-Purdue University Fort Wayne, Indiana 46805

Email: yooj, kimmdl01@ipfw.edu

†Air Force Research Laboratory

Rome Research Site, Rome, New York 13441

Email: douglas.boulware@us.af.mil

**Abstract**—Spatial association rule mining is a useful tool for discovering correlations and interesting relationships among spatial objects. Co-locations, or sets of spatial events which are frequently observed together in close proximity, are particularly useful for discovering their spatial dependencies. Although a number of spatial co-location mining algorithms have been developed, the computation of co-location pattern discovery remains prohibitively expensive with large data size and dense neighborhoods. We propose to leverage the power of parallel processing, in particular, the MapReduce framework to achieve higher spatial mining processing efficiency. MapReduce-like systems have been proven to be an efficient framework for large-scale data processing on clusters of commodity machines, and for big data analysis for many applications. The proposed parallel co-location mining algorithm was developed on MapReduce. The experimental result of the developed algorithm shows scalability in computational performance.

**Keywords**—spatial data mining; co-location pattern; spatial association analysis; cloud computing; MapReduce

## I. INTRODUCTION

So-called “Big data” is a fact of today’s world and brings not only large amounts of data but also various data types that previously would not have been considered together. Richer data with geolocation information and date and time stamps is collected from numerous sources including mobile phones, personal digital assistants, social media, vehicles with navigation, GPS tracking systems, wireless sensors, and outbreaks of disease, disaster and crime. The spatial and spatiotemporal data are considered nuggets of valuable information [1].

Spatial data mining is the process of discovering interesting and previously unknown, but potentially useful patterns from large spatial data and spatiotemporal data [2]. As one of spatial data mining tasks, spatial association mining has been widely studied for discovering certain association relationships among a set of spatial attributes and possibly some non-spatial attributes [3]. Spatial co-location represents a set of events (or features) which are frequently observed together in a nearby area [4]. Although many spatial association mining techniques [4]–[12] have been developed, the computation involved in searching spatially associated objects from large data is inherently too demanding of both processing time and memory requirements. Furthermore, explosive growth in the spatial and spatiotemporal data emphasizes the need for developing new and computationally efficient methods tailored for analyzing big data. As a solution, parallel/distributed data processing techniques are becoming a necessity to deal with the massive amounts of data. We propose to leverage the power

of parallel processing to achieve higher spatial data mining processing efficiency.

Modern frameworks facilitating the distributed execution of massive tasks are becoming increasingly popular since the introduction of MapReduce programming model, and Hadoop’s run-time environment and distributed file systems [13]. For large-scale data processing on clusters of commodity machines, MapReduce-like systems have been proven to be an efficient framework for big data analysis for many applications, e.g., machine learning [14] and graph processing [15]. In this paper, we present a MapReduce-based co-location mining algorithm.

Applying the MapReduce framework to spatial co-location mining presents several challenges. Because of limited communication among nodes in a cluster, all steps of co-location mining should be paralleled inter-dependently. However, for the parallel processing, it is non-trivial to divide the search space of co-location instances, i.e., a set of spatial objects which have neighbor relationships with each other. Explicit search space partitioning may lose co-location instances across different partitions and generate incomplete and incorrect mining results. When a client submits a job task, MapReduce automatically partitions the input data into physical blocks and distributes the blocks to the distributed file systems. However, this explicit data partition and distribution may lose some spatial relationships among objects because spatial objects are embedded on continuous space, and make various relationships with each other. Moreover, assigning balanced loads among nodes is a difficult problem. In this paper, we address these problems and propose a parallel/distributed algorithm for spatial co-location mining on MapReduce.

The remainder of this paper is organized as follows. Section II presents the basic concept of spatial co-location mining and the MapReduce paradigm, and describes the related work. Section III shows our search space partition strategy and Section IV presents the proposed MapReduce based co-location mining algorithm. Experimental results are reported in Section V and the paper will conclude in Section VI.

## II. PRELIMINARIES

This section presents the basic concept of spatial co-location mining and the MapReduce programming model, and describes the related work.

### A. Spatial Co-location Mining

Boolean spatial events (features) describe the presence of spatial events at different locations in geographic space. Exam-

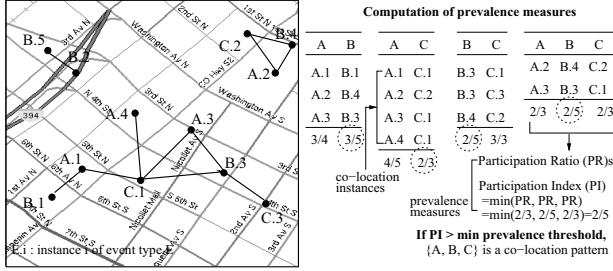


Figure 1. Spatial co-location mining

ples of such data include disease outbreaks, crime incidents, traffic accidents, mobile service types, climate events, plants and species in ecology, and so on. Let  $E = \{e_1, \dots, e_m\}$  be a set of different spatial events. A spatial data set  $S = S_1 \cup \dots \cup S_m$ , where  $S_i (1 \leq i \leq m)$  is a set of spatial objects of event  $e_i$ , where an object  $o \in S_i$  can be represented with a data record  $\langle \text{event type } e_i, \text{ instance id } j, \text{ location } x, y \rangle$ , where  $1 \leq j \leq |S_i|$ . Figure 1 shows an example where each data object is represented by its event type and unique instance id, e.g., A.1.

Spatial neighbor relationship  $R$  can be defined with metric relationship (e.g., Euclidean distance), topology relationship (e.g., within, nearest), and direction relationship (e.g., North, South). This work uses a distance-based neighbor relationship. Two spatial objects,  $o_i \in S$  and  $o_j \in S$ , are neighbors if the distance between them is not greater than a given distance threshold  $d$ ; that is,  $R(o_i, o_j) \Leftrightarrow \text{distance}(o_i, o_j) \leq d$ . In Figure 1, identified neighbor objects are connected by a solid line.

A **co-location**  $C = \{e_1, \dots, e_k\}$  is a subset of spatial events  $C \subseteq E$  whose instance objects are frequently observed in a nearby area. A **co-location instance**  $I \subseteq S$  of a co-location  $C$  is defined as a set of objects which includes all event types in  $C$  and forms a clique under the neighbor relationship  $R$ . In Figure 1,  $\{A.2, B.4, C.2\}$  is a co-location instance of  $\{A, B, C\}$ . The prevalence strength of co-locations is often measured by participation index [4].

**Definition 1:** The **participation index**  $PI(C)$  of a co-location  $C = \{e_1, \dots, e_k\}$  is defined as  $PI(C) = \min_{e_i \in C} \{PR(C, e_i)\}$ , where  $1 \leq i \leq k$ , and  $PR(C, e_i)$  is the **participation ratio** of event type  $e_i$  in the co-location  $C$  that is the fraction of objects of event  $e_i$  in the neighborhood of instances of co-location  $C - \{e_i\}$ , i.e.,  $PR(C, e_i) = \frac{\text{Number of distinct objects of } e_i \text{ in instances of } C}{\text{Number of objects of } e_i}$ .

The prevalence measure indicates wherever an event in  $C$  is observed, with a probability of at least  $PI(C)$ , all other events in  $C$  can be observed in its neighborhood. If the participation index of an event set is greater than a user-specified minimum prevalence threshold  $\text{min\_prev}$ , the event set is called a **co-location** or **co-located event set**. In the example of Figure 1, the participation index of an event set  $C = \{A, B, C\}$  is  $PI(C) = \min\{PR(C, A), PR(C, B), PR(C, C)\} = \min\{\frac{2}{3}, \frac{2}{5}, \frac{2}{3}\} = \frac{2}{5}$ .

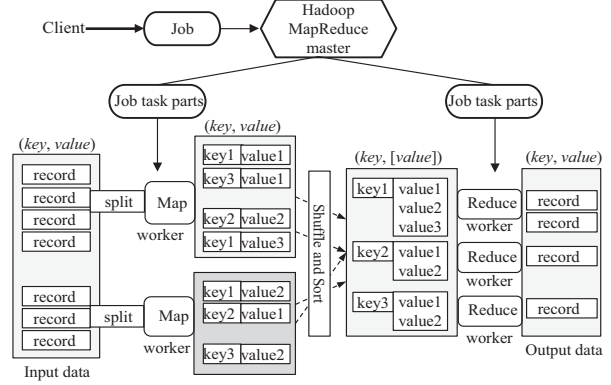


Figure 2. MapReduce Program and Execution Models

## B. MapReduce

MapReduce [16] is a programming model for expressing distributed computations on massive amounts of data and an execution framework for large-scale data processing on clusters of commodity servers. MapReduce simplifies parallel processing by abstracting away the complexities involved in working with distributed systems, such as computational parallelization, work distribution, and dealing with unreliable hardware and software.

The MapReduce model abstraction is presented in Figure 2. A MapReduce job is executed in two main phases of user defined data transformation functions, namely, *map* and *reduce*. When a job is launched, the input data is split into physical blocks and distributed among nodes in the cluster. Such division and distribution of data is called *sharding*, and each part is called a *shard*. Each block in the shard is viewed as a list of *key-value* pairs. In the first phase, the *key-value* pairs are processed by a mapper, and are provided individually to the map function. The output of the map function is another set of intermediate *key-value* pairs.

The values across all nodes, that are associated with the same key, are grouped together and provided as input to the reduce function in the second phase. The intermediate process of moving the *key-value* pairs from the map tasks to the assigned reduce tasks is called a *shuffle* phase. At the completion of the shuffle, all values associated with the same key are located at a single reducer, and processed according to the reduce function. Each reducer generates a third set of *key-value* pairs considered as the output of the job.

MapReduce can also refer to the execution framework that coordinates the execution of programs written in this particular style. MapReduce decomposes a job submitted by a client into small parallelized workers. A significant feature of MapReduce is its built-in fault tolerance. When a failure at a particular map or reduce task is detected, the keys assigned to that task are reassigned to an available node in the cluster and the crashed task is re-executed without the re-execution of the other tasks. MapReduce was originally developed by Google and has since enjoyed widespread adoption via an open-source implementation called Hadoop [13]. Currently, MapReduce is considered the actual solution for many data analytics' tasks over large distributed clusters [17], [18].

### C. Related Work

Since Koperski et al. [3] introduced the problem of mining association rules based on spatial relationships (e.g., proximity, adjacency), spatial association mining has been popularly studied in data mining literature [4]–[7], [9], [12], [19], [20]. Shekhar et al. [4] defines the spatial co-location pattern and proposes a join-based co-location mining algorithm. The instance join operation for generating co-location instances is similar to *apriori\_gen* [21]. Morimoto [5] studies the same problem to discover frequent neighboring service class sets but a space partitioning and non-overlap grouping scheme is used for finding neighboring objects. Xiao et al. [7] proposes a density based approach for identifying clique neighbor instances. Eick et al. [6] works on the problem of finding regional co-location patterns for sets of continuous variables in spatial data sets. Mohan et al. [20] proposes a graph based approach to regional co-location pattern discovery. Zhang et al. [11] presents a problem of finding a reference feature centric longest neighboring feature sets. Yoo et al. [22]–[25] proposes variant co-location mining problems to discover compact sets of co-locations. Wang et al. [10] presents techniques to find star-like and clique topological patterns from spatiotemporal data.

General data mining literature has formerly employed parallel methods since its very early days [26]. Zaki [27] presents the survey work of parallel and/or distributed algorithms for association rules mining. There are many novel parallel mining methods as well as proposals that parallelize existing frequent itemset mining techniques such as APriori [21] and FP-growth [28]. However, the number of algorithms that are adapted to the MapReduce framework is rather limited. Some works [29], [30] suggest counting methods to compute the support of every itemset in a single MapReduce round. Different adaptations of APriori to MapReduce are also shown in [31], [32]. Lin et al. [31] presents three different pass counting strategies that are adaptations of Apriori on multiple MapReduce rounds. An adaptation of FP-Growth to MapReduce is presented in [28], [33], [34]. The PARMA algorithm by Riondato et al. [35] finds approximate collections of frequent itemsets. Unfortunately, none of these frameworks have been directly adopted to spatial co-location mining. They are mainly developed for finding frequent itemsets and association rules from transaction databases like market-basket transaction data. However, spatial data does not have the natural transaction concept. Most of these algorithms feed input transaction records to mappers and execute the frequency (e.g., support) counting step in parallel. Whereas, our prevalence measures based on spatial neighborhoods cannot be simply computed with the frequency counting. To the best of our knowledge, our work is the first parallel algorithm of mining spatial co-location patterns on MapReduce.

### III. SEARCH SPACE PARTITION FOR PARALLEL PROCESSING

Spatial data objects and their neighbor relationships can be represented in various ways. Figure 3 (a) represents them with a neighbor graph  $G = (V, E)$  where a node,  $v \in V$ , represents a spatial object and an edge  $e \in E$  represents a neighbor relationship between two spatial objects of different event types. Note that we do not consider relationships among

same type events. Hence for every edge  $(u, v) \in E$  where  $u, v \in V, v.type \neq u.type$ . A subset  $C \subseteq V$  is a clique in graph  $G$  if for every pair of vertexes  $u, v \in C$ , the edge  $(u, v) \in E$ .

For the discovery of co-location patterns, we need to find all co-location instances forming cliques from the neighbor graph. However, this process is a computationally expensive operation. Even though we enumerate all maximal cliques in a graph, the number of maximal cliques is exponential in the number of vertexes [36]. In addition, the maximal clique enumeration problem has been proven to be NP-hard [37]. Furthermore, the neighbor graph of dense data may not fit in the memory of a single machine.

To take advantage of parallelism, the search space of co-location patterns should be divided into independent partitions so workers can process a given task with each partitioned data synchronously. However, splitting all neighbor relations into disjoint sets is not easy because spatial relationships are continuous in space. Furthermore, we do not want to miss any neighbor relationship during the partitioning process, and the partition cost needs to be inexpensive. In fact, partitions with minimal duplicate information are preferable.

We use a strategy to partition edges in the neighbor graph to divide the search space of co-location patterns. The edges of the neighbor graph are divided according to the following simple rule: each vertex  $v$  (a spatial object) keeps the relationship edge with other vertex  $u$  when  $v.type < u.type$ . Here, we assume there is a total ordering of the event type (i.e., lexicographic). This partition strategy divides neighbor relations without duplicating or missing any relationships needed for co-location mining as shown in our previous work [12]. Figure 3 (b) shows the status after the edge partitioning. For neighborhoods divided by the edge-based partitioning method, we define the following term.

**Definition 2:** Let  $G(V, E)$  be a neighbor graph. For  $v \in V$ , the **conditional neighborhood** of  $v$ ,  $\mathcal{N}(v)$ , is defined as the set of all the vertexes  $u$  adjacent to  $v$  including  $v$  itself which satisfies a certain condition  $\{v, u \in V : (u, v) \in E \text{ and } v.type < u.type\}$

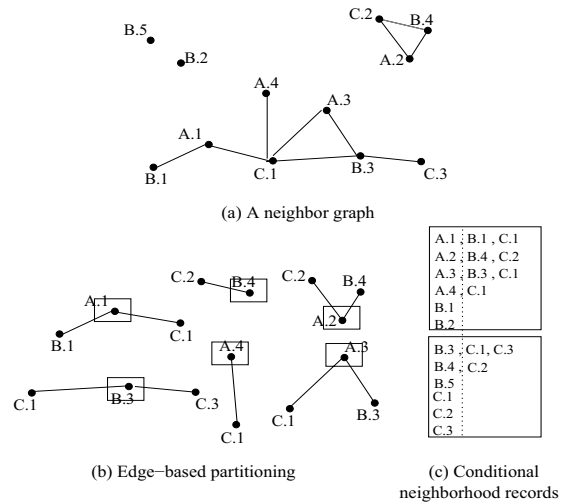


Figure 3. Spatial neighborhood partitioning

The conditional neighborhood of a spatial object  $v$  is a set of its neighbor objects whose event types are greater than the event type of  $v$  including the object itself,  $v$ . All objects in  $\mathcal{N}(v)$  are called *star neighbors* of  $v$ . Figure 3 (c) represents the spatial objects and their neighbor relations with conditional neighborhood records. The conditional neighborhood of A.3,  $\mathcal{N}(A.3)$ , is {A.3, B.3, C.1},  $\mathcal{N}(B.3)$  is {B.3, C.1, C.3}, and so on. When the neighborhood records are provided to the MapReduce job of co-location mining, MapReduce partitions the records into equal sized blocks and distributes the blocks evenly to the distributed file systems. Each mapper collects co-location instances from assigned neighborhood records, and the outputs are merged for the reducer which determines frequent co-located event sets.

#### IV. A MAPREDUCE BASED CO-LOCATION ALGORITHM

Given a spatial data set, a neighbor relationship, and a minimum prevalence threshold, the proposed algorithm discovers prevalent co-located event sets through two main tasks:

- *Spatial neighborhood partition* which divides the co-location search space.
- *Parallel co-located event set search*, where co-location instances are searched by each map worker synchronously, and then merged so that reducers find prevalent co-located event sets based on the merged instance sets.

Figure 4 shows the overall algorithmic framework of co-location mining on MapReduce.

##### A. Spatial Neighborhood Partition

For the spatial neighborhood partition task, we use two MapReduce jobs. The first job searches all neighboring pairs, and the second job generates the conditional neighborhood records from the neighboring pairs. Figure 5 and 6 show the pseudo codes.

The MapReduce framework splits the input data records into physical blocks without considering the geolocation of a data point. However, through the map function, we rearrange the data points for parallel neighbor search in the partitioned space. Space partitioning is the process of dividing a space into non-overlapping/overlapping regions. According to the geographic location of data point and the space partitioning strategy used, a grid (partition) number is assigned to each data point. The mapper then outputs a key-value pair  $\langle key' = gridno, value' = o \rangle$  where  $o \in S$  is a data point object. After all mapper instances have finished, for each  $key'$  assigned by the mappers, the MapReduce infrastructure collects corresponding values,  $[value']$  (here, a set of  $o$ ), and feeds the reducers with key-value pairs  $\langle key' = gridno, [value' = o] \rangle$ . The reduce function loads a given neighbor distance threshold ( $dist$ ), finds all neighbor object pairs in the  $[value']$ , and outputs  $\langle key'' = o_i, value'' = o_j \rangle$ , where  $o_i, o_j \in [value']$  and  $distance(o_i, o_j) \leq dist$ . The reduce uses a computational geometry algorithm, e.g., a plane-sweep algorithm [38], for the neighbor search. The reduce function of Figure 5 shows the plane sweep pseudo code of the neighbor search.

The next job is to generate the list of neighbors of each object according to the definition of the conditional neighborhood

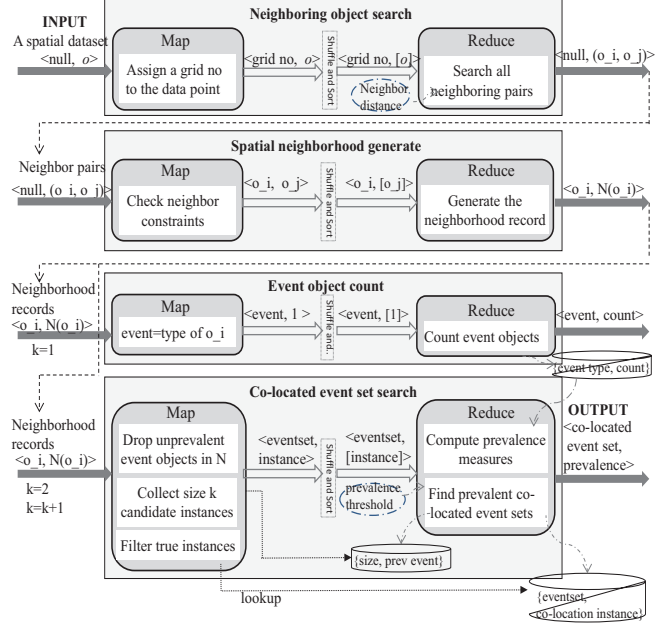


Figure 4. A MapReduce algorithmic framework for co-location mining

(Definition 2). As shown in Figure 4, the map function uses the output from the previous job. When each mapper instance starts, the instance is fed with the shards of neighbor pairs,  $\langle key = o_i, value = o_j \rangle$ . If  $o_j$  and  $o_i$  are the same object, or  $o_j$ 's event type is greater than  $o_i$ 's event type, the mapper outputs a key-value pair  $\langle key' = o_i, value' = o_j \rangle$ . A reducer sorts the objects in the value list  $[o_j]$  by their event type. The

```

1: procedure MAPPER(key, value= $o$ )
2:   gridno  $\leftarrow$  findRegion( $o$ )
3:   emit(gridno,  $o$ )
4: end procedure
5: procedure REDUCER(key= $gridno$ , value= $[o]$ )
6:   dist  $\leftarrow$  load('neighbor distance threshold')
7:   objectSet  $\leftarrow [o]$ 
8:   sort(objectSet, 'x')
9:   activeSet  $\leftarrow \emptyset$ 
10:  j  $\leftarrow$  1
11:  n  $\leftarrow$  length(objectSet)
12:  for i  $\leftarrow$  1, n do
13:    while (objectSet[i].x-objectSet[j].x) > dist do
14:      remove(activeSet, objectSet[j])
15:      j  $\leftarrow$  j+1
16:    end while
17:    range  $\leftarrow$  subset(activeSet, objectSet[i].y-dist, object-
Set[i].y+dist)
18:    for all obj  $\in$  range do
19:      if distance(objectSet[i], obj)  $\leq$  dist then
20:        emit(objectSet[i], obj)
21:      end if
22:    end for
23:    add(activeSet, objectSet[i])
24:    emit(objectSet[i], objectSet[i])
25:  end for
26: end procedure

```

Figure 5. Neighbor search

```

1: procedure MAPPER(key= $o_i$ , value= $o_j$ )
2:   if  $o_i == o_j$  or  $o_i.type < o_j.type$  then
3:     emit( $o_i$ ,  $o_j$ )
4:   end if
5: end procedure
6: procedure REDUCER(key= $o_i$ , value= $[o_j]$ )
7:   nRecord  $\leftarrow o_i \cup \text{sort}([o_j])$ 
8:   emit( $o_i$ , nRecord)
9: end procedure

```

Figure 6. Neighbor grouping

sorted list becomes the conditional neighborhood of  $o_i$ ,  $\mathcal{N}(o_i)$ . The reducer finally outputs  $\langle key'' = o_i, value'' = \mathcal{N}(o_i) \rangle$ .

### B. Parallel Co-located Event Set Search

After the preprocess task finishes, the parallel co-location mining task starts. Before this work, we have a job that counts and saves the number of object instances per event type for future prevalence calculation (Figure 7).

The prevalent co-located event set search is conducted in a level-wise manner. It begins with size  $k=2$  co-location discovery. The proposed approach finds co-location patterns without the candidate set generation. When each mapper instance starts, the mapper is fed with the shards of neighborhood records,  $\langle key, value = \mathcal{N}(o_i) \rangle$ . Per each  $\mathcal{N}(o_i) = \{o_i, o_1, \dots, o_m\}$ , the mapper performs the following steps: 1) For each event object  $o_j \in \mathcal{N}(o_i)$  where  $o_i \neq o_j$  and  $1 \leq j \leq m$ , remove  $o_j$  from  $\mathcal{N}(o_i)$  if the  $o_j$ 's type is not included in size  $k-1$  co-located patterns, 2) collect all size  $k$  candidate instances  $CI$  which start with  $o_i$ , i.e.,  $\{o_i, o_1 \dots, o_{k-1}\}$  from  $\mathcal{N}(o_i)$ , and 3) filter true co-location instances from the candidate instances, and outputs each instance with a key-value pair  $\langle key' = eventset, value' = instance \rangle$ . Note that the first object  $o_i$  of the candidate instance  $CI = \{o_i, o_1 \dots, o_{k-1}\}$  is the same as the first object  $o_i$  in  $\mathcal{N}(o_i)$ , and has a neighbor relationship with the other objects in  $CI$ . However, we cannot guarantee  $CI - o_i = \{o_1 \dots, o_{k-1}\}$  has neighbors with each other. We check the cliqueness of the subset with the information from instances of size  $k-1$  co-location patterns. If its sub-instance  $\{o_1, \dots, o_{k-1}\}$  is a co-location instance, the candidate instance  $\{o_i, o_1, \dots, o_{k-1}\}$  becomes a true co-location instance.

After all mapper instances have finished, the MapReduce collects the set of corresponding values per each  $key$ , and feeds the reducers with key-value pairs  $\langle key' = eventset, [value' = instance] \rangle$ , where  $[value']$  is a list of co-location instances of the event set,  $key'$ . A reducer computes the participation ratio of each event type with the instance set, and then computes the participation index of the event set. If the participation index satisfies a given prevalence threshold, the reducer outputs the frequent co-located event set  $\langle key'' = eventset, value'' = participation index \rangle$ . The reducer also saves the co-location instances of the co-located event set for the next size of pattern mining. Figure 8 represents the pseudo code of this MapReduce job. This job is repeated with the increase of pattern size  $k=k+1$ .

## V. EXPERIMENTAL EVALUATION

The proposed algorithm was implemented in Java, MapReduce library functions and HBase library functions. HBase [39] is a column-oriented database management system that runs on top of the Hadoop Distributed File System (HDFS). HBase was used to store the intermediate result such as prevalent co-located event types and co-location instances. The performance evaluation of the developed algorithm was conducted on Amazon Web Services' (AWS) Elastic MapReduce platform [40], which provides resizable computing capacity in the cloud. For this experiment, we used instance type *m1.large* in the AWS. The version of Hadoop used was 1.0.3, and the version of HBase was 0.94.8. We used real world data as well as synthetic data for the experiment. The synthetic data was generated using a spatial data generator used in [12].

In the first experiment, we compared the execution times of two main tasks: neighborhood partition and co-location search using synthetic datasets that differ in the number of data points. The number of distinct feature types in the data was 50. The neighbor distance was fixed to 10. We used a single node cluster for this experiment. Figure 9 (a) shows the result. There was no significant difference in the execution time of preprocessing spatial neighborhoods. However, the co-location pattern search time was increased with almost the same ratio of increased data points when the minimum prevalence threshold was 0.3. Although all the data points were included in a

```

1: procedure MAPPER(key= $o_i$ , value= $\mathcal{N}$ )
2:   emit( $o_i.type$ , 1)
3: end procedure
4: procedure REDUCER(key= $event$ , value= $[1]$ )
5:   count  $\leftarrow \text{sum}(value)$ 
6:   save( $event$ , count)
7:   emit( $event$ , count)
8: end procedure

```

Figure 7. Count of instance objects of event

```

1: procedure MAPPER(key= $o_i$ , value= $\mathcal{N}$ )
2:   for all  $o \in \mathcal{N}$  do
3:     if IsPrevalentType( $o$ 's type)  $\neq$  true then
4:        $N = N - o$ 
5:     end if
6:   end for
7:    $k \leftarrow \text{load}(\text{'current pattern size'})$ 
8:   candiInstSets  $\leftarrow \text{scanNTransaction}(\mathcal{N}, k)$ 
9:   for all instance  $\in$  candiInstSets do
10:    if checkCliqueness(instance)  $==$  true then
11:      eventset  $\leftarrow \text{eventTypesOf}(instance)$ 
12:      emit(eventset, instance)
13:    end if
14:  end for
15: end procedure
16: procedure REDUCER(key= $eventset$ , value= $[instance]$ )
17:    $\theta \leftarrow \text{Load}(\text{'min prevalence threshold'})$ 
18:   PI  $\leftarrow \text{compParticipationIndex}([instance])$ 
19:   if PI  $\geq \theta$  then
20:     emit( $eventset$ , PI)
21:     save( $eventset$ ,  $[instance]$ )
22:   end if
23: end procedure

```

Figure 8. Co-location pattern search

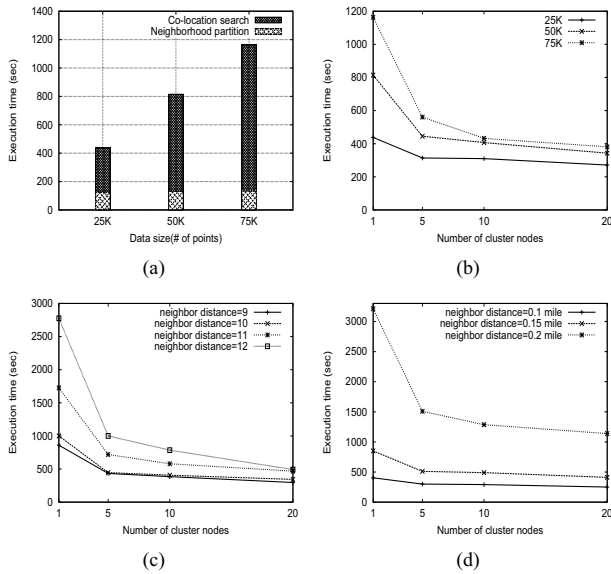


Figure 9. Experiment result: (a) Comparison of subtasks, (b) By number of data points, (c) By neighbor distance, and (d) Real-world data

single partition during the data preprocess, the plane sweeping method adopted to search all neighbor pairs showed a good performance. Next, we changed the number of nodes in the cluster to evaluate the speed increase of our algorithm. As shown in Figure 9 (b), the execution time was decreased with the increased number of nodes. The performance gain due to the parallel processing was high when the number of nodes was increased 1 to 5. After that, the gain was significantly decreased in this experimental setting; although, the overall execution time was decreased with the increase of nodes.

In the next experiment, we examined the effect of neighborhood size in the parallel processing performance. The neighborhood size was controlled with different neighbor distances. Figure 9 (c) shows the experimental result on 50K synthetic datasets with distances of 9, 10, 11, and 12. With an increase of the neighbor distance, the execution time of co-location mining is dramatically increased when the cluster has a single node. However, with an increase in the number of cluster nodes, the run time performance was significantly improved in all the neighbor distance settings.

In the last experiment, we used real-world data which was a set of 17K points of interest in the Washington D.C. area. The number of feature types was 87. Figure 9 (d) shows the results of the experiments with distances of 0.1, 0.15, and 0.2 miles, respectively. The minimum prevalence threshold was fixed to 0.4. When the neighborhood size is large, the experiment shows the computational performance of co-location mining is greatly improved with the parallel processing.

## VI. CONCLUSION

In this work, we have proposed to parallelize co-location pattern mining to deal with large-scale spatial data. We have developed a parallel/distributed co-location mining algorithm on Hadoop’s MapReduce infrastructure. The proposed framework partitions the spatial neighborhood without any missing

and duplicate neighbor relationships for co-location discovery. Each worker independently conducts the co-location mining process with a shard of neighborhood records. The co-location patterns are searched in a level-wise manner by re-using previously processed information and without the generation of candidate sets. The experimental results show that our algorithmic design approach is overall parallelizable and follows a significant increase in speed, with respect to an increase in nodes, when data size is large and the neighborhood is dense.

## ACKNOWLEDGMENT

This work is partially supported by Air Force Research Laboratory, Griffiss Business and Technology Park, and SUNY Research Foundation.

## REFERENCES

- [1] R. R. Vatsavai, A. Ganguly, V. Chandola, A. Stefanidis, S. Klasky, and S. Shekhar, “Spatiotemporal Data Mining in the Era of Big Spatial Data: Algorithms and Applications,” in Proceedings of ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, 2012, pp. 1–10.
- [2] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*. Prentice Hall, ISBN 0130174807, 2003.
- [3] K. Koperski and J. Han, “Discovery of Spatial Association Rules in Geographic Information Databases,” in Proceedings of the International Symposium on Large Spatial Data bases, 1995, pp. 47–66.
- [4] S. Shekhar and Y. Huang, “Co-location Rules Mining: A Summary of Results,” in Proceedings of International Symposium on Spatio-temporal Database, 2001, pp. 236–256.
- [5] Y. Morimoto, “Mining Frequent Neighboring Class Sets in Spatial Databases,” in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001, pp. 353–358.
- [6] C. F. Eick, R. Parmar, W. Ding, T. F. Stepinski, and J. Nicot, “Finding Regional Co-location Patterns for Sets of Continuous Variables in Spatial Datasets,” in Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2008, pp. 1–10.
- [7] X. Xiao, X. Xie, Q. Luo, and W. Ma, “Density based Co-location Pattern Discovery,” in Proceedings of ACM SIGSPATIAL international Conference on Advances in Geographic Information Systems, 2008, pp. 1–10.
- [8] Y. Huang, J. Pei, and H. Xiong, “Mining Co-Location Patterns with Rare Events from Spatial Data Sets,” *Geoinformatica*, 2006, pp. 239–260.
- [9] J. S. Yoo and S. Shekhar, “A Partial Join Approach for Mining Co-location Patterns,” in Proceedings of the ACM International Symposium on Advances in Geographic Information Systems, 2004, pp. 241–249.
- [10] J. Wang, W. Hsu, and M. L. Lee, “A Framework for Mining Topological Patterns in Spatio-temporal Databases,” in Proceedings of ACM International Conference on Information and Knowledge Management, 2005, pp. 429–436.
- [11] X. Zhang, N. Mamoulis, D. Cheung, and Y. Shou, “Fast Mining of Spatial Collocations,” in Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004, pp. 384–393.
- [12] J. S. Yoo and S. Shekhar, “A Join-less Approach for Mining Spatial Co-location Patterns,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, 2006, pp. 1323–1337.
- [13] “Apache Hadoop,” <http://hadoop.apache.org/>.
- [14] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan, “SystemML: Declarative Machine Learning on MapReduce,” in Proceedings of International Conference on Data Engineering, 2011, pp. 231–242.
- [15] “Giraph,” <http://giraph.apache.org/>.
- [16] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, 2008, pp. 107–113.

- [17] T. Elsayed, J. Lin, and D. W. Oard, "Pairwise Document Similarity in Large Collections with MapReduce," in Annual Meeting of the Association for Computational Linguistics on Human Language Technologies, 2008, pp. 265–268.
- [18] A. Ene, S. Im, and B. Moseley, "Fast Clustering using MapReduce," in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2011, pp. 681–689.
- [19] W. Ding, R. Jiamthapthaksin, R. Parmar, D. Jiang, T. F. Stepinski, and C. F. Eick, "Towards Region Discovery in Spatial Datasets," in Proceedings of International Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2008, pp. 88–99.
- [20] P. Mohan, S. Shekhar, J. Shine, J. Rogers, Z. Jiang, and N. Wayant, "A Neighborhood Graph based Approach to Regional Co-location Pattern Discovery: A Summary of Results," in Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2011, pp. 122–132.
- [21] R. Agarwal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in Proceedings of International Conference on Very Large Data Bases, 1994, pp. 487–499.
- [22] J. S. Yoo and M. Bow, "Mining Top-k Closed Co-location Patterns," in Proceedings of IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services, 2011, pp. 100–105.
- [23] J. S. Yoo and M. Bow, "Finding N-Most Prevalent Colocated Event Sets," in Proceedings of International Conference on Data Warehousing and Knowledge Discovery, 2009, pp. 415–427.
- [24] J. S. Yoo and M. Bow, "Mining Spatial Colocation Patterns: A Different Framework," Data Mining and Knowledge Discovery, 2012, pp. 159–194.
- [25] J. S. Yoo and M. Bow, "Mining Maximal Co-located Event Sets," in Proceedings of Pacific-Asia International Conference on Knowledge Discovery and Data Mining, 2011, pp. 351–362.
- [26] R. Agrawal and J. Shafer, "Parallel Mining of Association Rules," IEEE Transactions on Knowledge and Data Engineering, 1996, pp. 962–969.
- [27] M. Zaki, "Parallel and Distributed Association Mining: A Survey," Concurrency, IEEE, vol. 7, no. 4, 1999, pp. 14–25.
- [28] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns Without Candidate Generation," SIGMOD Record, vol. 29, 2000, pp. 1–12.
- [29] L. Li and M. Zhang, "The Strategy of Mining Association Rule based on Cloud Computing," in Proceedings of International Conference on Business Computing and Global Information, 2011, pp. 475–478.
- [30] X. Y. Yang, Z. Liu, and Y. Fu, "MapReduce as a Programming Model for Association Rules Algorithm on Hadoop," in Proceedings of International Conference on Information Sciences and Interaction Sciences (ICIS), 2010, pp. 99–102.
- [31] M. Y. Lin, P. Y. Lee, and S. C. Hsueh, "Apriori-based Frequent Itemset Mining Algorithms on MapReduce," in Proceedings of the International Conference on Ubiquitous Information Management and Communication, 2012, pp. 1–8.
- [32] N. Li, L. Zeng, W. He, and Z. Shi, "Parallel Implementation of Apriori Algorithm based on MapReduce," in Proceedings of ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2012, pp. 236–241.
- [33] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "PFP: Parallel FP-Growth for Query Recommendation," in Proceedings of ACM Conference on Recommender systems, 2008, pp. 107–114.
- [34] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Huang, and S. Feng, "Balanced Parallel FP-Growth with MapReduce," in Proceedings of International Conference on Information Computing and Telecommunications, 2010, pp. 243–246.
- [35] M. Riondato, J. A. DeBrabant, R. Fonseca, and E. Upfal, "PARMA: A Parallel Randomized Algorithm for Approximate Association Rules Mining in MapReduce," in Proceedings of ACM international Conference on Information and knowledge management, 2012, pp. 85–94.
- [36] J. Moon and L. Moser, "On Cliques in Graphs," Israel Journal of Mathematics, vol. 3, 1965, pp. 23–28.
- [37] J. L. E. Lawler and A. R. Kan, "Generating All Maximal Independent Sets: NP-Hardness and Polynomial-time Algorithms," Israel Journal of Mathematics, vol. 3, 1965, pp. 23–28.
- [38] L. Arge, O. Proceedingspiuc, S. Ramaswamy, T. Suel, and J. Vitter, "Scalable Sweeping-Based Spatial Join," in Proceedings of International Conference on Very Large Data Bases, 1998, pp. 570–581.
- [39] "Apache HBase," <http://hbase.apache.org/>.
- [40] "Amazon Elastic MapReduce (Amazon EMR)," <http://aws.amazon.com/elasticmapreduce/>.