[CP-04-007] Spatial MapReduce

Abstract

MapReduce has become a popular programming paradigm for distributed processing platforms. It exposes an abstraction of two functions, map and reduce, which users can define to implement a myriad of operations. Once the two functions are defined, a MapReduce framework will automatically apply them in parallel to billions of records and over hundreds of machines. Users in different domains are adopting MapReduce as a simple solution for big data processing due to its flexibility and efficiency. This article explains the MapReduce programming paradigm, focusing on its applications in processing big spatial data. First, it gives a background on MapReduce as a programming paradigm and describes how a MapReduce framework executes it efficiently at scale. Then, it details the implementation of two fundamental spatial operations, namely, spatial range query and spatial join. Finally, it gives an overview of spatial indexing in MapReduce systems and how they can be combined with MapReduce processing.

Author & citation

Eldawy, A. (2018). Spatial MapReduce. The Geographic Information Science & Technology Body of Knowledge (3rd Quarter 2018 Edition), John P. Wilson (Ed.). DOI:10.22224/gistbok/2018.3.9.

Explanation

1. Definitions
2. MapReduce Overview
3. MapReduce Framework
4. Spatial Operations in MapReduce
5. Spatial Indexing in MapReduce Frameworks

1. Definitions

Functional Programming: A programming paradigm in which the program is defined in terms of a set of mathematical functions or transformations. Each function is memoryless which means that the output depends only on its input value. Functions are connected together to define the program logic.

MapReduce: A functional programming style where the program is expressed mainly in two functions, map and reduce. This style can be supported in many programming languages and can be executed in different architectures including in-memory and disk-based architectures.

MapReduce Framework: A system that employs the MapReduce programming paradigm. It accepts user-defined map and reduce functions and applies them efficiently to an input file while hiding underlying technical details from users.

HDFS: Hadoop Distributed File System is a popular distributed file system used in

MapReduce which stores big files by splitting each file into equi-sized blocks and distributing the blocks across machines.

2. Map Reduce Overview

The continuous growth of data with the limited increase in hardware speeds, urged many developers to switch from single-machine processing to distributed processing. Unfortunately, that switch was not as simple as changing the algorithms as there are many complicated low-level system issues in distributed processing including distributed storage, network management, load balancing, and fault tolerance. While these issues might not be specific to each algorithm, developers had to spend much more time in handling them than the original algorithm which impeded their productivity.

Google was one of the first companies to identify this drawback in their internal use of distributed systems and they developed MapReduce (Dean and Ghemawat 2008) as a solution to this problem. MapReduce is a simple programming paradigm in which all parallel programs are expressed in two functions, map and reduce, described shortly. Developers embed all their program-specific logic in these two functions. Once the functions are defined, the MapReduce framework executes them in parallel while transparently handling all the common low-level system issues including parallelization, networking, and fault tolerance. The simplicity and flexibility of MapReduce allowed common developers to write production-ready distributed algorithms that run on hundreds of machines in a few hours instead of weeks.

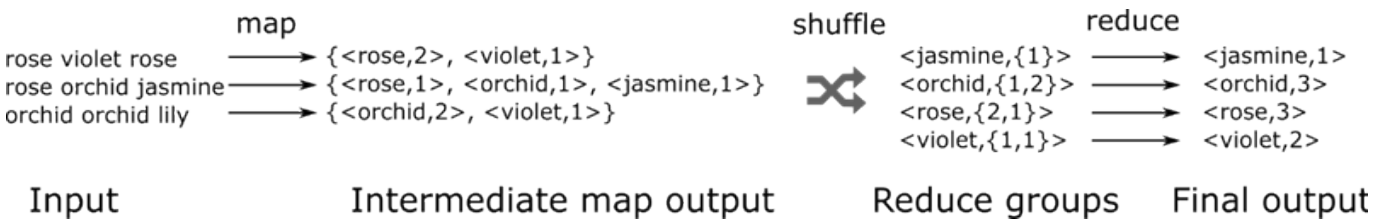


Figure 1. Word count example in MapReduce.

The logic of a MapReduce program is defined by two functions, map and reduce. Typically, the two functions are applied on large sets of records. The map function is applied to one input record and produces a set of intermediate key-value pairs (k, v). The reduce function is applied to each unique key k along with a set of all associated values {v} to produce the set of final records. The most popular example is the WordCount algorithm where the input is a set of text lines and the desired output is the set of all unique words in the input along with the number of occurrences for each one as illustrated in Figure 1. The map function takes as input one text line and produces a set of pairs (w, c) for each word w in the line where c is the number of occurrences of that word in the line. The reduce function takes each word along with all associated counts and adds all the count values (w{c}) to produce a final pair of (w, c) for each word in the input.

MapReduce has proven to be a very powerful tool especially when multiple MapReduce

stages, i.e., multiple MapReduce programs, are combined in one MapReduce program. In other words, the output of the reduce function in one stage can be used as the input to the map function of the next stage. MapReduce was used to develop distributed algorithms in many domains including relational operations, graph algorithms, machine learning, and computational geometry, which we further describe in this article.

3. MapReduce Framework

In order to run a MapReduce program, a MapReduce framework is required. The MapReduce framework is a system that takes an input file, the user-defined map and reduce functions, and applies them at scale on the input file to produce the final output. The framework automatically handles all the complicated low-level system issues mentioned earlier. One of the most popular open-source MapReduce frameworks is Hadoop.

Hadoop was initially developed by Yahoo! and later became a community-maintained open-source MapReduce framework that resembles Google's MapReduce system. The core of Hadoop contains the distributed file system (HDFS) and the MapReduce engine. On top of it, there is a plethora of systems and tools built on-top of it such as HBase, Hive, Pig, Mahout, and Giraph. All these systems together are referred to as the Hadoop ecosystem. In this article, we focus on how the core Hadoop components execute a MapReduce program.

To understand the role of the MapReduce framework, we will describe how Hadoop executes the WordCount example described above. The first step is to distribute the large input file among the machines in the cluster. Keep in mind that typically the input file is too large to fit on a single machine. A popular open source distributed file system is Hadoop Distributed File System (HDFS) in which the input file is split into equi-sized chunks, called HDFS blocks, with a default size of 128 MB (or 64 MB for older versions of Hadoop). By default, each block is replicated to three machines for fault tolerance but this replication factor is configurable. For efficiency and to avoid inconsistencies between replicas, HDFS does not allow a file to be modified once written in HDFS.

After the data is distributed and stored on the machines, Hadoop takes the user program, i.e., the map and reduce functions, and distributes that program to all the machines. This is called the compute-to-data shipment where the small program is moved over network to where the data is stored rather than moving the large amount of data to where the program resides. Then, it applies the map function in parallel on all the machines where each machine runs on the local data previously stored on that machine. For load balancing, Hadoop executes the map function on the fixed-size HDFS blocks one at a time. Since each block is stored on multiple machines, Hadoop can automatically choose the least loaded machine for each block. To parse text lines correctly from each HDFS block, Hadoop must handle lines that cross the block boundaries which are physically stored in multiple machines. To parse these lines correctly, Hadoop moves part of the line over network to the machine that will process the line. The map function is then applied to each line and the output is collected locally on each machine. If a machine fails while processing a block, Hadoop automatically reschedules that block on another machine.

The output of the map function is locally stored in sorted order by the key, i.e., the word w. After that, Hadoop shuffles the key-value pairs over network such that all pairs with the same key are assigned to the same machine. As the data is shuffled, Hadoop starts the

reduce phase in which each reducer machine collects the key-value pairs from mappers and merges them to produce one list of key-value pairs in sorted order by the key. The sorted order helps running the reduce function on each key efficiently. Once the merged lists are ready, Hadoop applies the reduce function on each key and its corresponding list of values to produce the final answer as shown in Figure 1. The output of the reduce function is stored back in HDFS to be ready to use by another MapReduce program. Similar to the mappers, if a reducer fails, Hadoop automatically reschedules it on another machine and ships the corresponding key-value lists from the mappers.

Regardless of the complicated execution of a MapReduce program, the developer needs only to worry about writing the map and reduce functions while the MapReduce framework, e.g., Hadoop, handles all the details of the complicated execution.

4. Spatial Operations in MapReduce

The flexibility of the MapReduce programming paradigm allows it to be used in many applications. Unlike traditional text files where the data can be processed by scanning the entire file once, some spatial operations need special treatments to execute efficiently. In particular, many spatial operations rely on the topology or the distance of the individual records to work correctly. For example, the k nearest neighbors operation relies on distance while the range query and spatial join operations rely on the topology. This part describes two fundamental algorithms that are widely used in spatial databases and GIS software, namely, range query and spatial join. Range query is an example of a very simple operation that is easy to implement in MapReduce. Spatial join is a more complicated query that requires spatial data partitioning. These two operations and many more spatial operations were implemented in several full-fledged systems such as SpatialHadoop (Eldawy and Mokbel 2015), Hadoop-GIS (Aji et al. 2013), and others (Li et al. 2017). Interested readers can refer to a comprehensive survey in this topic (Eldawy and Mokbel 2016).

4.1 Range Query

In range query, the input is a set of R objects, e.g., rectangles, and a query area q, and the output is all records in R that overlap the query range q. To run a range query in MapReduce, all we need to do is to express the operation in terms of map and reduce. In the case of range query, it can be expressed as a map-only program, i.e., no reduce function is needed. The input to the map function is one input geometry $\backslash fn_cmr \in R$ which is compared to the query rectangle q. If they overlap, the map function produces the output $\backslash fn_cm(\phi, r)$ where ϕ is the null key which is typically used to indicate that all the records belong to one group. Otherwise, if the record r does not overlap the query range q, the map function does not produce any output. Since there is no reduce function, the output of the map function represents the final answer and is directly written to the output.

4.2 Spatial Join

In spatial join, the input is two sets of geometries, R and S, and a spatial predicate $\backslash fn_cm\Theta$, e.g., overlaps or contains. The output is every pair of records $\backslash fn_cm(r \in R, s \in S)$ where $\backslash fn_cm\Theta$ (r, s) is true. For simplicity, we consider the case

where R and S are two sets of rectangles and $\backslash fn_c m\Theta$ is the overlap predicate. There are several MapReduce algorithms for spatial join. Below, we describe a popular algorithm based on the partition-based spatial-merge join PBSM (Patel and DeWitt 1996). PBSM partitions the input rectangles according to a fixed-size grid where each record is replicated to all overlapping grid cells. Then, it joins the contents of each grid cell using a traditional single-machine spatial join algorithm. Finally, it applies a duplicate avoidance technique to eliminate duplicates resulting from the replication.

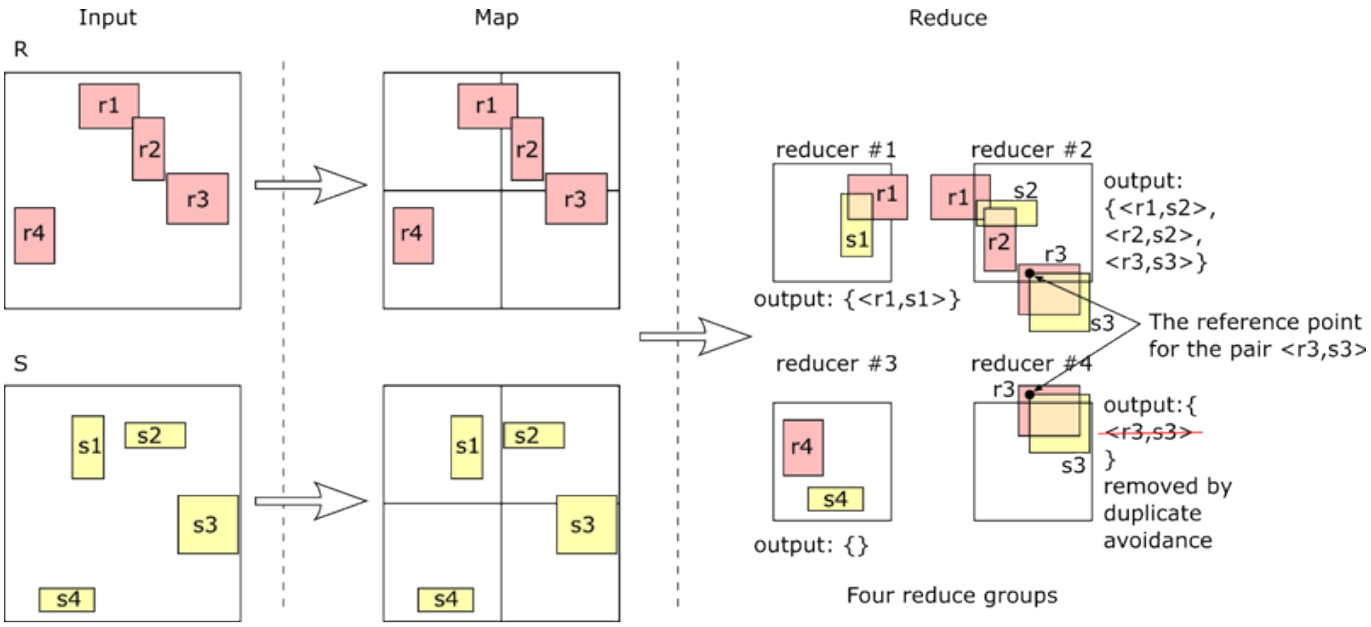


Figure 2. Partition-based Spatial-merge (PBSM) join in MapReduce. Source: author.

Figure 2 illustrates an example of how the PBSM algorithm is implemented in MapReduce where the map function carries out the partition step while the reduce function performs the join and duplicate avoidance steps. The map function takes as input one rectangle $\backslash fn_c mr \in R$ or $\backslash fn_c ms \in S$, and compares it to the grid to find all overlapping grid cells. The example in the figure uses a 2x2 uniform grid. For each overlapping cell c, the map function produces an output pair (ci,r) where ci is a unique cell ID. If a record overlaps more than one cell, it is replicated to all of them, e.g., r1 , r3, and s3 in the figure.

The reduce function runs in two steps. First, it collects all rectangles in one grid cell and applies a traditional single-machine join algorithm on them to find all pairs of overlapping rectangles. This step can produce duplicates if the same pair is replicated to multiple machines. For example, in Figure 2 the pair (r3, s3) can be produced by two reducers as both records are replicated to two cells. The second step applies the reference point duplicate avoidance technique (Dittrich and Seeger 2000) to ensure that each pair is reported only once. For a pair of rectangles r and s, the duplicate avoidance technique computes the top-left corner of the intersection and reports the pair to the answer only if the intersection lies within the grid cell being processed by that machine. In this example, the top-left corner of r3 $\backslash fn_c m\cap$ s3 (the intersection is indicated as an orange rectangle)

lies within the top-right cell, hence, it is reported by reducer #2 and avoided by reducer #4.

5. Spatial Indexing in MapReduce Frameworks

When a file is uploaded to HDFS, it is split into equi-sized chunks with a default size of 128 MB in the same order of the original file. This means that spatially nearby records will probably end up in two different blocks, and hence, two different machines. This might lead to inefficiency in most spatial operations where nearby records are correlated and need to be processed together. Recall from the first law of geography that near things are more related than distant things. Therefore, it will be more efficient to store nearby records, which are more related, in the same HDFS block. Spatial indexing is the technique used to organize records based on their location.

The main feature in spatially indexed files in HDFS is that each HDFS block is annotated with a minimum bounding rectangle (MBR) that encloses all its contents. In order for the index to be useful and efficient, the MBRs should have a minimal overall area with a little overlap between them. At the same time, each MBR should contain roughly 128 MB of data to ensure a good disk utilization and load balance.

While there are many traditional spatial indexes for single machine systems, e.g., R-tree and Quad-tree, they do not directly apply in MapReduce because of the limitations of HDFS. While traditional indexes are designed for traditional file systems which support random file access, HDFS files can only be written sequentially which makes spatial indexing very challenging. One simple technique to overcome this limitation is to first draw a random sample of the data and let one machine build an index layout that only contains the MBRs of the blocks without populating them with any data. Once the MBRs are computed, the index can be written sequentially by writing each record to the corresponding block based on the location of the record.

References

Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., and Saltz, J. (2013). Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. PVLDB 6(11): 1009-1020.

Dean, J., and Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51:107—113.

Dittrich, J-P., and Seeger, B., (2000). Data Redundancy and Duplicate Detection in Spatial Join Processing. In Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073), San Diego, CA. pages 535-546.

Eldawy, A., and Mokbel, M. F. (2015). SpatialHadoop: A MapReduce Framework for Spatial Data. In Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, Seoul, Korea (South), 2015, pp. 1352-1363.

Eldawy, A., and Mokbel, M. F. (2016). The Era of Big Spatial Data: A Survey. Foundations and Trends in Databases, 6(3-4):163—273.

Li, Z., Hu, F., Schnase, J. L., Duffy, D. Q., Lee, T., Bowen, M. K., and Yang, C. (2017). A spatiotemporal indexing approach for efficient processing of big array-based climate data with MapReduce. International Journal of Geographical Information Science 31(1).

Patel, J., and DeWitt, D. (1996). Partition Based Spatial-Merge Join. In Proceedings of the ACM International Conference on Management of Data, SIGMOD, pages 259—270.