Indexing Large Spatial Data using Hadoop, a closer look at spatialhadoop and another complementary QuadTree implementation

Donghan Miao

Goals

Spatial data are usually very large in volume, spatial object data sets that are of hundreds of gigabytes of are not uncommon. We want to find an efficient way of indexing the spatial objects set and run fast range query or other operations on the indexed data set. So we naturally turn to distributed system in hope that 1) we could manage the data of that scale and even larger data sets. 2) The system runs commodity hardware.

Hadoop, HDFS and MapReduce

The Apach Hadoop is an open-source project for reliable, scalable, distributed computing, which allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. It is designed to detect and handle failures at the application layer, so delivering a highly available service on top of a cluster of computers.

HDFS is short for hadoop distributed file system. It is a distributed, scalable file system for the Hadoop framework. HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines. It achieves reliability by replicating the data across multiple hosts. The file system is also fault tolerant – node failure is detected and handled in the application level. The file system uses the TCP/IP for communication between nodes.

MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a Map() procedure that performs filtering and sorting and a Reduce() procedure that performs a summary operation. The MapReduce System orchestrates by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

Spatialhadoop

SpatialHadoop is a project developed by University of Minnesota, a MapReduce extension to Apache Hadoop, designed specially to work with spatial data. It is designed to analyze huge spatial datasets on a cluster of machines. This report dissects spatialhadoop implementation details. In addition, I implemented another fully-fledged indexing and query system using quadtree on hadoop as a complement to spatialhadoop.

Index Building and data persistence

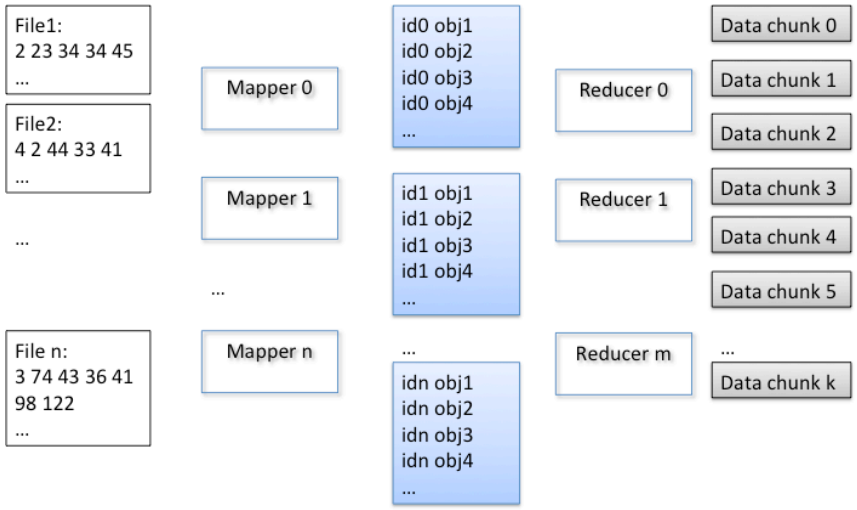
In general, database storage consists of persisted index structure and data storage. In big data environment, we leverage the advantage of data partitioning to achieve scalability. Indexing process partitions large data set into even-sized data chunks (the number of chunks is determined by the volume of spatial dataset). Each data chunk is a small but fully-fledged database storage.

By default, the block size of Hadoop distributed file system (HDFS for short hereafter) is 64MB. Therefore our default chunks should be close to but no larger than that size, in order to guarantee i/o efficiency.

The data chunk consists of two sections, and both are in binary format. The first section is spatial index structure, which may vary in user's own choice. In the very beginning, both parts are constructed in memory. When

the total size of the two sections grows close to the maximum size limit, we serialize the two parts into persisted file on HDFS.

We implement index building as a MapReduce job. In our case, input data are formatted texts where each line represents a polygon object. Mapper input takes <line number, line text string> as key-value pairs. Mapreduce framework will split the files into smaller splits before submitting them to mappers, so that multiple mappers will work in parallel to process the input splits. The level of parallelism is determined by the volume of data set or can be customized by the user.



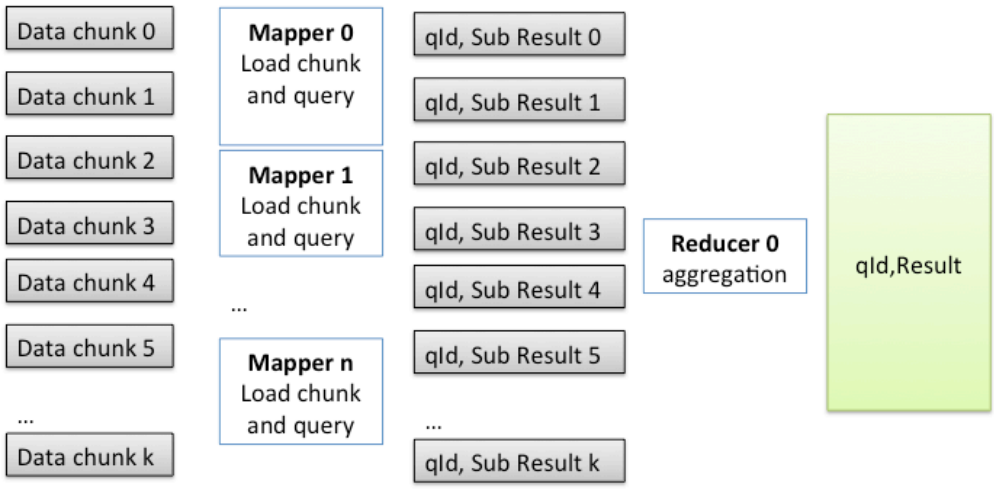
Graph 1, MapReduce process for Tree Building

Mappers convert the text string into a spatial object and emits <mapperInstanceID, object> pairs for reducers. The emitted key-value pairs sharing the same key will be sent to same reducer, and the reducers will collect an appropriate number of objects (a number adjusted to make data chunk close to 64MB) and build spatial index for them. Upon finishing index building, the index and raw data will be combined as a single data chunk and emit as a single binary file to the HDFS file system.

Executing a Query

Executing a range query is also a Mapreduce job, where input is a number of data chunks built from the first Mapreduce job. In each mapper, a customized input format reader was implemented according to our own data format to load the data chunks into memory, where their original data structure are restored – the first section of file restored as index structures, and the second section of file are simply loaded into a large byte block.

Query was executed in the mapper phase. Each mapper will conduct a search operation in each file scope. Index structure returns a set of object ids (id is a combination of file position and length as position in binary block) for any given range search. According to the object ids, the actual data of the object will be retrieved from the byte array.



Graph 2, Query In MapReduce

After that, a set of queried objects is sent to reducer with key being the queryID, value being a spatial object. In this way, results belong to the same query will be aggregated in the same reducer.

Spatial Indexing Data Structures

Spatial Hadoop supports R+tree, rtree, and grid indexing, lacking other popular spatial indexing structures like quadtree, kdtree, etc.

A quadtree is a tree data structure in which each internal node has exactly four children. Quadtrees partition a two-dimensional space by recursively subdividing it into four quadrants or regions. Some modification was made for efficiency. In one leaf node, more than one objects can be stored if the minimal bounding rectangle of an object has intersection with another object in that node. In this way, objects cluster will not result in over division of the quadtree.

Validation

I compared the result from range query from my implemented system against spatialhadoop structures. The results turned out consistent.

Source and Compile

Packages

io: This package includes implementation for customized input format and output format interface of Apache hadoop MapReduce.

mapReduceTasks: This package defines two MapReduce jobs, namely, index building job, query search job.

quadIndex: this package is an overall wrap-up for different spatial objects and quadtree structure.

Additional libraries

Libraries

Minimum subset of Apache hadoop standard library are included for compilation was included in the package for convenience of compilation.

Latest source code is available on Github.

>> git clone https://github.com/miaodonghan/QuadTreeHadoop.git

Deploy hadoop

The hadoop version we are using is 1.2.1 stable release. Please see Apache Hadoop official website for deployment instructions.

Single node deployment
http://hadoop.apache.org/docs/r1.2.1/single_node_setup.html

Cluster deployment
http://hadoop.apache.org/docs/r1.2.1/cluster_setup.html

Format

Raw data should be in plain text format. Each line indicates a spatial Object. Points, Rectangles and Polygons are supported. Now the latest version seamlessly support spatialhadoop data format, you can directly use the data source from spatialhadoop.org

Old data format is defined as follows:
Point:
1, x, y
Rectangle:
2, x, y, width, height
Polygon:
numberOfpoints, x1, y1, x2, y2, …, xn, yn

Commands to run the program

Indexing a data set
Command:
hadoop jar tb.jar mapReduceTasks.QuadTreeIndexer
Or alternatively, you can specify your input path and output path:
hadoop jar tb.jar mapReduceTasks.QuadTreeIndexer inputPath outputPath

Run a query on Mapreduce
Query format
queryID,x,y,width,height
where queryID is a unique integer
Command
>> hadoop jar tb.jar mapReduceTasks.QuadTreeQuery queryID,x,y,width,height
Or alternatively, you can specify your input path and output path
>> hadoop jar tb.jar mapReduceTasks.QuadTreeQuery queryID,x,y,width,height inputPath outputPath