Experimental evaluations

A city-scale building dataset that covers an area of 20km\*20km and contains 173,746 polygon objects was used to test the performance of our MongoDB R-tree implementation. Two pieces of this dataset are shown in Figure 1. The dataset was downloaded from OSM website (Open Street Map, [www.openstreetmap.org](http://www.openstreetmap.org)), and then projected under Cartesian coordinate system. As a result, two versions about the building data are available: one is geodetic and the other is projected. All experiments have been carried out on a PC running Windows 10 with Intel Core i5-4200H 2.8GHz CPU and 16GB RAM, in which a simplest MongoDB cluster composed of one shard instance, one config instance and one route instance (including the R-tree module) was configured and built.

**Figure 1.** Two pieces of the test dataset

1. Construction

The first experiment aims at evaluating the efficiency of R-tree construction, during which five flattened R-trees with different branching factors are constructed from the projected dataset. The results are presented in Figure 2. On one hand, as branching factor increases from 8 to 256, the runtime initially decreases but turns to increase after reaching a minimum at the branching factor of 96, because the size of an R-tree node with 96 entries is mostly close to that of a disk page (namely 4096 bytes). On the other hand, the overall fluctuation is relatively small because of a two-level of caching, i.e., built-in structure caching in shard server and developed node caching in router server.

**Figure 2.** Construction with different branching factors

2. Query processing

Three sets of experiments were arranged to test the performance of query processing with flattened R-tree. The first experiment used MongoDB predicate $geoWithin to get objects that are wholly contained in a query window, the second experiment used MongoDB predicate $geoIntersect to get objects overlapping with a query window, while the last experiment executed MongoDB predicate $geoNear to fetch objects that fall into an specified distance interval with respect to a query point and output them in an order from nearest to farthest. The results are presented in Figure 3.

(a) $geoWithin

(b) $geoIntersect

(c) $geoNear

**Figure 3.** Query processing using different filtering predicts

One can see that the first two operators, i.e., $geoWithin and $geoIntersect, perform very similarly on not only processing time but also document throughput. As the query window enlarges, the response time increases dramatically, while the throughput initially increases then tends to be stable at the level of about 2,500 polygon objects per second. By comparison, the last operator of $geoNear, because of involving searching as well as ranking, is less efficient to be handled by flattened R-tree. As a result, the processing time is nearly doubled and the document throughput is reduced much more rapidly, when compared with $geoWithin and $geoIntersect. However, the document throughput keeps increasing, though the growth tends to be narrowed.

3. Comparison with 2dsphere

Finally, the group of query windows same to the above was carried out on the projected dataset with flattened R-tree as well as the geodetic dataset with 2dshpere. For each query, a mixture of $geoWithin and $geoIntersect was prepared and fed, and then the runtime took the average. The results are presented in Figure 4(a), which indicates that 2dshpere performs much better than flattened R-tree on the OSM building dataset, and the efficiency gap even becomes bigger and bigger as the window size increases. In addition, this experiment was repeated on the lake dataset of Wuhan, China that contains 4260 polygon objects, and the results are presented in Figure 4(b). With the increasing window size, our flattened R-tree initially outperforms built-in 2dshpere but turns to be caught up. The reason is that the input dataset is complex in shape and varies in coverage (from thousands of square meters to tens of square kilometers), and therefore, geometry calculation, particularly for 2dsphere, contributes a lot to the processing time when the query window is small or moderate, while data fetching starts to dominate the runtime when the query window becomes large.

(a) query processing on OSM building dataset (b) query processing on Wuhan lake dataset

**Figure 4.** Query processing of flattened R-tree and 2dsphere

4. Conclusion

MongoDB is a popular NoSQL product characterized as document-oriented, rich query language and high availability. Consider that MongoDB is LBS-oriented, only supporting spatial with WGS84 geographical coordinate, we provided an R-tree module within router server to manage planar spatial data that is still widely used in city-scale spatial applications, and developed corresponding algorithms to support CRUD operations. The experimental evaluation shows that our MongoDB-based flattened R-tree succeeds to manage planar spatial data and performs well on query processing, especially for complex and large spatial objects.

In the near future, we will consider how to push R-tree structures into shard server to accommodate the computation and storage of R-tree together, where the biggest challenge is how to efficiently map spatial data and aggregate results across storage nodes.