Study of BKD tree performance with different values for max points per leaf

We present a study of the performance of the BKD tree in relation to different values for the parameter **maxPointsInLeafNode**. In order to run this experiment, we have used data from Lucene geo benchmarks for points (1) and for polygons (2). The tests are run for five experiments:

- Double point: We index the longitude from Lucene geo benchmarks (1) and execute 225 range queries (20 iterations).
- LatLonPoint: We index the points from the Lucene geo benchmarks (1) and execute 225 bounding box queries (20 iterations).
- Geo3DPoint: We index the points from the Lucene geo benchmarks (1) and execute 225 bounding box queries (20 iterations).
- LatLonShape (points): We index the points from the Lucene geo benchmarks (1) and execute 225 bounding box queries (20 iterations).
- LatLonShape (polygons): We index the polygons from the Lucene geo benchmarks (2) and execute 225 bounding box queries (20 iterations).

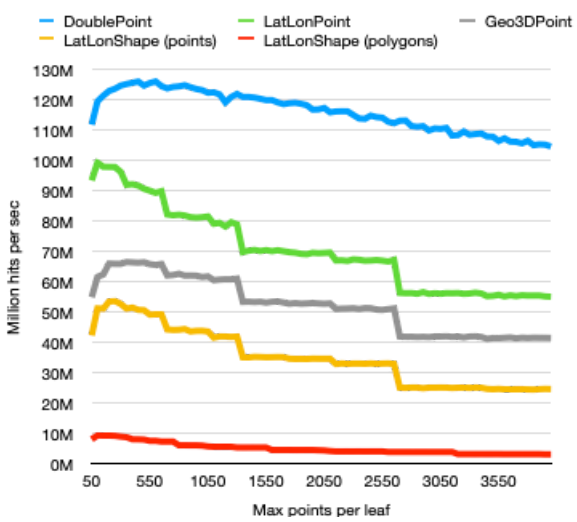
In all cases we capture the query performance, indexing time and index size. Results are presented with the raw values and the variation in each case with respect to the maximum value measured.

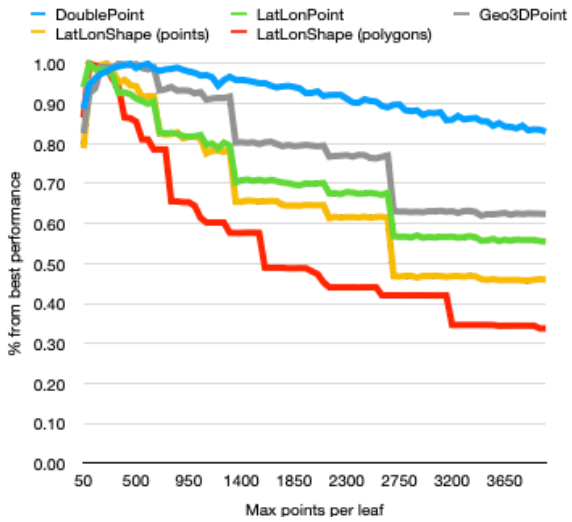
It is important to keep in mind that in the 1-dimensional case (double point), the tree is built as an unbalanced kd-tree, and in the other cases as fully balanced kd-tree. Therefore the profiles are very different.

(1) http://people.apache.org/~mikemccand/geobench.html
(2) http://people.apache.org/~ivera/geoshapebench.html

Query performance:

Execute 225 range / bounding box queries and measure the performance as million hits per second. The results are presented as well as the variation of performance compared to the best value measured in each case:

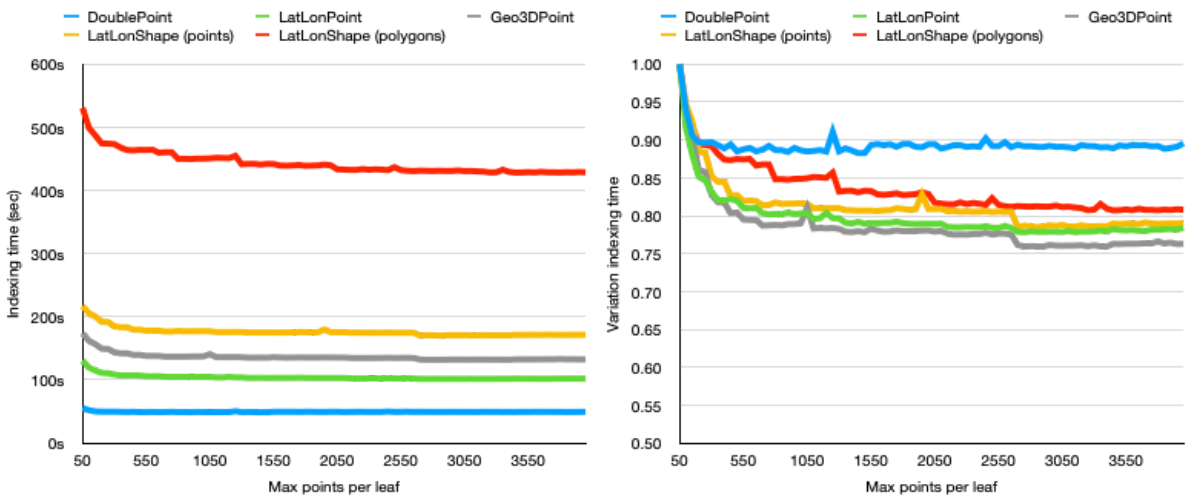




We can observe that if the max points per leaf is too small, the performance is bad as the tree is too deep but that quickly converges to the best performance. After that point the performance starts decreasing. In the 1D case, the decrease is more or less linear but in the nD case (n > 1), the decrease has jumps as it depends on the number of points stored in leaf nodes in order to build a fully balanced tree.

Indexing time

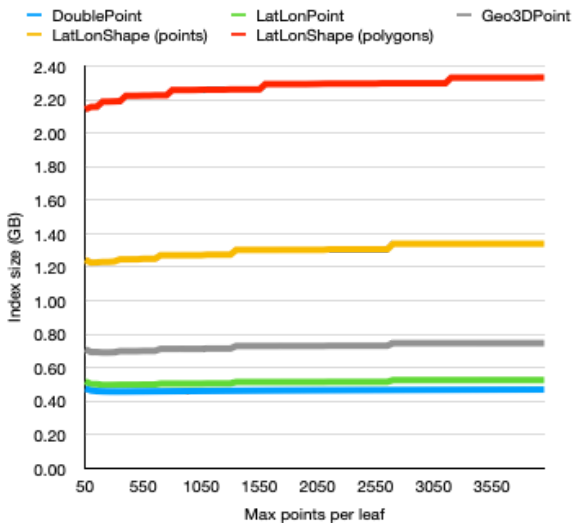
Time it took to index the data in each case, measured in seconds. The results are presented as well as the variation of indexing time compared to the slowest time measured in each case:

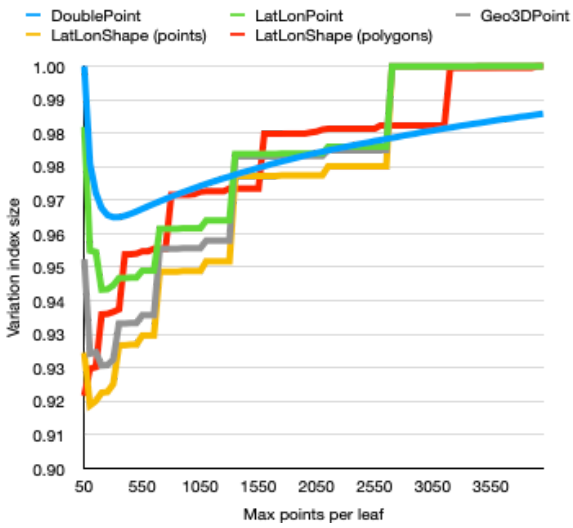


We can observe that the lower the value of max points per leaf, the longer it takes to index the data. if the max points per leaf is too small, the indexing time is bad as the tree is too deep. As the value of max points per leaf nodes increases, the decrease of indexing time is lower as in many cases we are building trees with the same levels. In the 1D case, the indexing time is almost constant.

Index size

Final size of the index, measure in GB. The results are presented as well as the variation of index size compared to the biggest size measured:



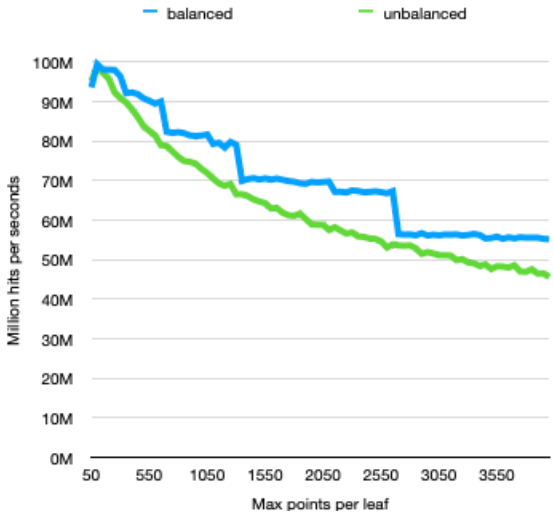


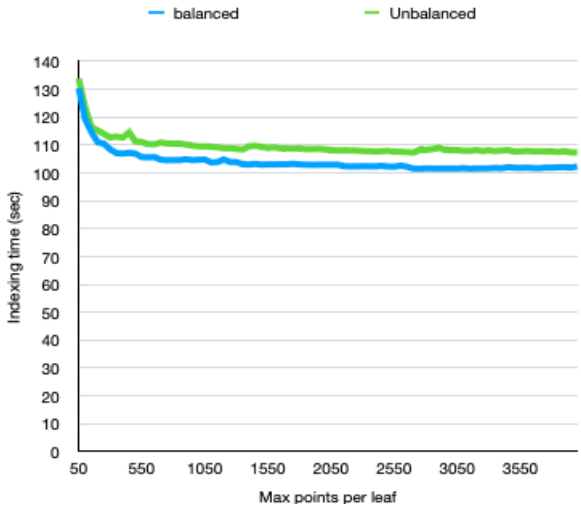
In general, index size variations are quite small. We presume that the lower the value of max points per leaf, the bigger is the tree but the compression of leaf nodes is better as well. As we increase the value, the size of the tree becomes smaller but the size of the leaf nodes increases as they do not compress as well as lower occupied nodes.

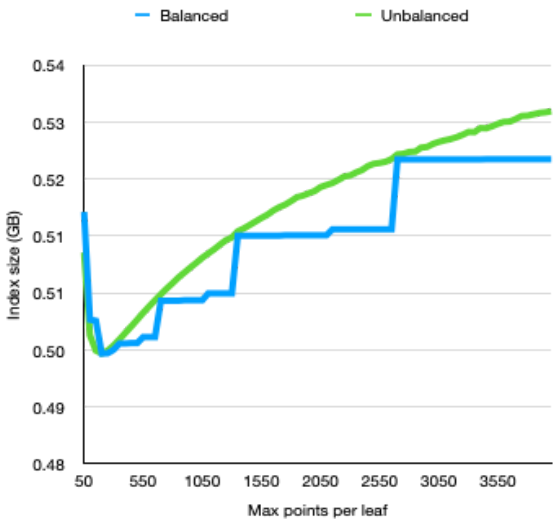
Building an unbalanced tree for dimensions > 1

It is not difficult to modify the construction of the tree to generate an unbalanced tree in all cases, similar to what we are already doing for the 1D case.

I have run the experiments for the LatLonPoint case and the results are the expected ones. It seems a bit slower at indexing time but the behavior of the tree becomes linear to the changes in max points per leaf.

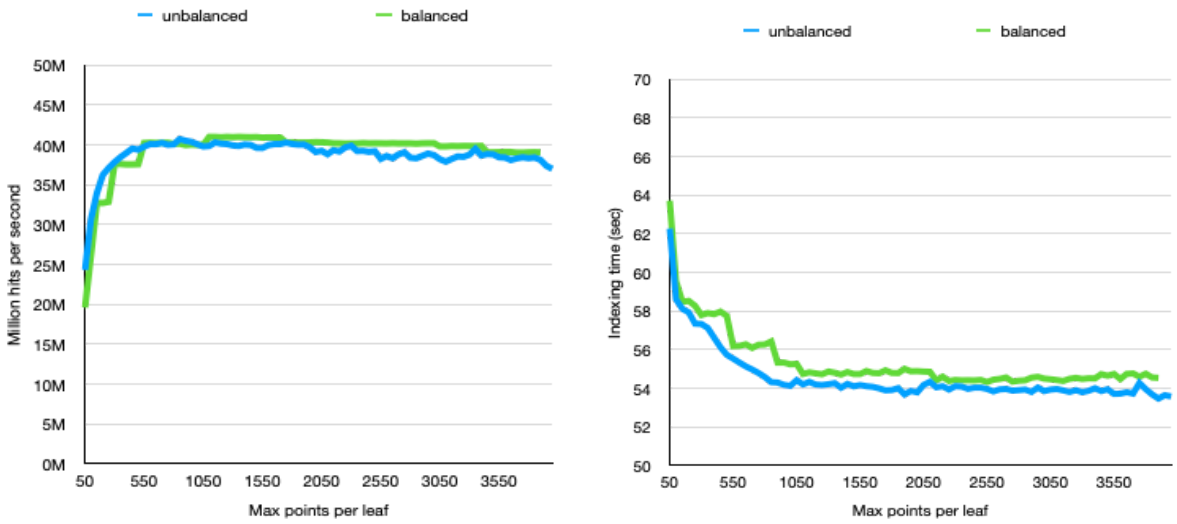


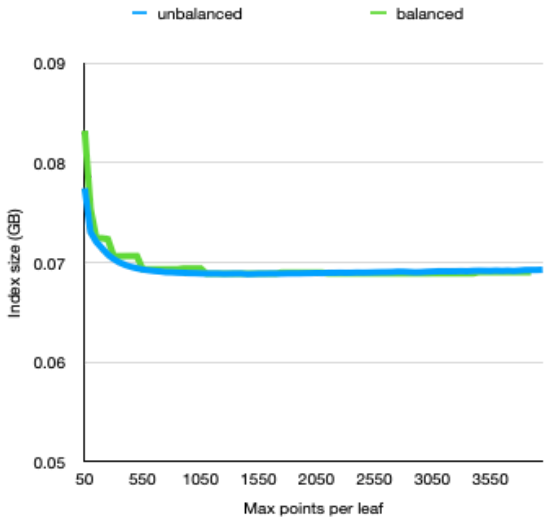




Edge case experiment

I have done one more experiment with polygon data that contains lots of duplicates.
And here the results for the experiment:





This case is interesting because it behaves better with more points per leaf as we are visiting many leaves. In this case having full leaves gives better control of the expected performance.