Compressing Intermediate Keys between Mappers and Reducers in SciHadoop

Adam Cresen, Joe Bark, Carlos Maltzahn, Scott Brandt
University of California, Santa Cruz

PROBLEM
- Large amount of intermediate data in Hadoop
- Restricted key representation

“INCOMPRESSIBLE” BYTES

TRANFORM TIME

In the general case (such as with a sliding window), it is impossible to avoid key overlap. Even if we try to define alignment boundaries with some preconditions, there will still be keys that lie across the alignment boundaries.

UNAVOIDABLE OVERLAP

When we apply a filter to a file of keys from a 100 × 100 × 100 grid, the transform improves compression rates dramatically.

FRAMEWORK

Key aggregation virtually eliminates key overhead for a 100 × 100 × 100 grid of integers. Data size is reduced by 84.5%. The reduction in data is due to reduction in key data and reduction of file overhead. The file format used by Hadoop adds a non-zero overhead per key/value pair. Aggregation greatly reduces the number of key/value pairs.

RESULTS

We ran a query that finds the median of all values within a sliding 3 × 3 window, using its 800 × 800 × 800 grid of integers. The cluster has 5 nodes, with 5 reducers and 10 map slots. A custom codec applied the transform and then compressed the transformed data with the built-in zlib compressor.

We ran a query that finds the median within a sliding 3 × 3 window, using its 800 × 800 × 800 grid of integers. The cluster has 5 nodes, with 5 reducers and 10 map slots. A custom codec applied the transform and then compressed the transformed data with the built-in zlib compressor.

The reduce function requires that all values in a single invocation have the same key, and that all values for that key are present in that invocation. Overlapping but unsorted ranges break this constraint when used as keys. To fix this, overlapping ranges are split on overlap boundaries.

EFFECT OF KEY AGGREGATION

The reduce function requires that all values in a single invocation have the same key, and that all values for that key are present in that invocation. Overlapping but unsorted ranges break this constraint when used as keys. To fix this, overlapping ranges are split on overlap boundaries.

Conclusion

Aggregation can significantly reduce runtime for sorting over grid data. Compression is not beneficial, due to fixed overhead. Another benefit of aggregation versus compression is that keys stay aggregated across small write cycles, whereas compression requires to be applied each time. This is especially important during the sort phase.

Funding
This work was supported by Department of Energy grant DE-SC0016368 and the Los Alamos National Laboratory/University of California, Santa Cruz Institute for Sustainable Scientific Data Management (ISSEDM).