CMSC724: Literature Survey
Applying Compression Techniques in MapReduce

Greg Benjamin, Samet Ayhan, Kishan Sudusinghe
University of Maryland, College Park

1 Inspiration: Floratou et al.

Our starting point for this project was a 2011 paper by Floratou et al., entitled *Column-oriented storage techniques for MapReduce* [1]. This work seeks to address many of the issues related to performance in the Hadoop implementation of MapReduce by incorporating techniques used in column-oriented and parallel database systems. In particular, contributions of the paper include:

- A novel column-oriented data storage format called `ColumnInputFormat`, which improves on existing column-storage formats like `RCFile` [5] by allowing each column of a relation to be stored in separate disk blocks, increasing efficiency payoffs in caching and compression ratio.

- A *lazy deserialization* scheme based on the SkipList [4] data structure. The scheme is designed so that column values need only be read in from disk if they are actually used, and blocks of unused values may be skipped over using pointer information embedded in the column file. This avoids the heavy cost of deserializing every value in the column, whether or not that value is actually needed by the MapReduce task.

- Two techniques for compressing column files in a chunked manner, such that the SkipList deserialization scheme also avoids having to decompress a chunk if the values it contains are not needed. These techniques make use of the lightweight, non-optimal LZO algorithm [2] [3] to do the compression of individual chunks.

- An experimental analysis of the above techniques, in comparison with other commmonly-used storage formats and compression techniques. In general, `ColumnInputFormat` and the compression schemes presented here are found to provide an order-of-magnitude speedup over the other configurations.

Of particular interest to us is the authors' claim that using heavier compression algorithms with better compression ratios does not lead to any speedup. Apparently this is because such algorithms incur too much CPU overhead during decompression, and this outweighs any benefit gained by having to read in less data. However, such claims were *only* tested using the `zlib` compression library, which is an implementation of the Lempel-Ziv '77 algorithm [2].

2 Background

2.1 MapReduce

2.2 Column-stores

2.3 Parallel Databases

2.4 Compression Techniques

3 MapReduce Performance

3.1 Serialization and Data Format

3.2 Data Layout

3.3 Indexing

3.4 Data Compression

4 Other Concerns

4.1 Energy Efficiency

4.2 anything else i'm forgetting?

5 Next Steps

References

[1] Avrilia Floratou, Jignesh M. Patel, Eugene J. Shekita, and Sandeep Tata. 2011. Column-oriented storage techniques for MapReduce. Proc. VLDB Endow. 4, 7 (April 2011), 419-429.

[2] Jacob Ziv and Abraham Lempel; A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, 23(3), pp. 337343, May 1977.

[3] Jacob Ziv and Abraham Lempel; Compression of Individual Sequences via Variable-Rate Coding, IEEE Transactions on Information Theory, 24(5), pp. 530536, September 1978.

[4] W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. CACM, 33(6):668676, 1990.

[5] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. RCFile: A Fast and Space-efficient Data Placement Structure in MapReduce-based Warehouse Systems. In ICDE, 2011.