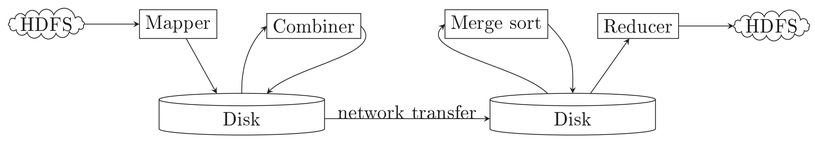
Compressing Intermediate Keys between Mappers and Reducers in SciHadoop

Adam Crume, Joe Buck, Carlos Maltzahn, Scott Brandt
University of California, Santa Cruz

Baskin Engineering
UC SANTA CRUZ

Problem

- Large amount of intermediate data in Hadoop
- Inefficient key representation



"Incompressible" bytes

```
00 00 00 00 00 00 00 00  00 00 00 00 0a 77 69 6e  |.............win|
64 73 70 65 65 64 31 00  00 00 01 00 00 00 01 6c  |dspeed1........l|
88 6c 80 01 01 1f 0e 00  00 00 04 00 00 00 8c 00  |.l..............|
00 00 00 00 00 00 00 00  00 00 01 0a 77 69 6e 64  |............wind|
73 70 65 65 64 31 00 00  00 01 00 00 00 01 6c 88  |speed1........l.|
6c 8a 01 01 1f 0e 00 00  00 04 00 00 00 8c 00 00  |l...............|
00 00 00 00 00 00 00 00  00 02 0a 77 69 6e 64 73  |...........winds|
70 65 65 64 31 00 00 00  01 00 00 00 01 6c 88 6c  |peed1........l.l|
94 01 01 1f 0e 00 00 00  04 00 00 00 8c 00 00 00  |................|
00 00 00 00 00 00 00 00  03 0a 77 69 6e 64 73 70  |..........windsp|
65 65 64 31 00 00 00 01  00 00 00 01 6c 88 6c 9e  |eed1........l.l.|
01 01 1f 0e 00 00 00 04  00 00 00 8c 00 00 00 00  |................|
00 00 00 00 00 00 00 04  0a 77 69 6e 64 73 70 65  |.........windspe|
```

Generic compression methods such as GZIP rely on repeating sequences of bytes. A stream of keys generated by walking a grid in a regular patterns creates *almost* identical sequences of bytes. The changing bytes greatly hamper compression by introducing bytes that must be literally encoded as well as forcing GZIP to use shorter sequences, rather than using long sequences such as multiples of the simple sequence.

Sequence detection

Stride	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4
1	1;0				
2	2;0	3;1			
3	-1;2	0;9	1;1		
4	0;5	2;4	0;5	1;5	
5	-1;0	-2;1	0;1	2;1	3;0

$\delta; r \equiv$ increment=δ, run length=r

Highest current r is for stride 4, so prediction uses $stride = 4, \delta = 1$. In other words, the prediction for byte x_i is $x_{i-4} + 1$.

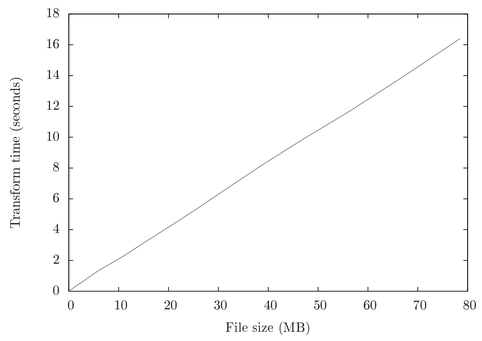
Linear prediction

Keys:	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)

Original:	1	1	1	2	1	3	1	4	1	5	2	1
Predictions:							1	4	1	5	1	6
Delta (output):	1	1	1	2	1	3	0	0	0	0	1	-7

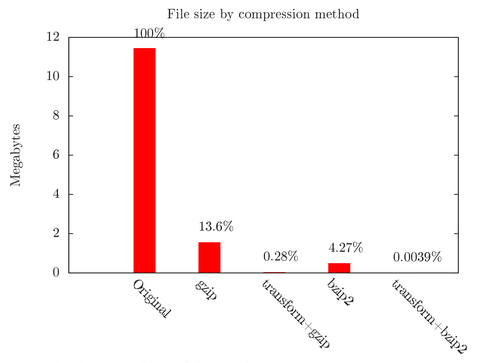
Noticing that the bytes form a linear sequence, we predict them, output deltas from our predictions, then run the result through a generic compression such as GZIP. Prediction and delta-encoding means that most of the linear sequences are replaced by zeros. Since the bytes are now constant, the stream is much more compressible with a generic compression scheme.

Transform time



The transform time scales linearly with the file size. This is expected, since the transform is based on a streaming algorithm.

Compression



When applied to a file of keys from a $100 \times 100 \times 100$ grid, the transform improves compression ratios dramatically.

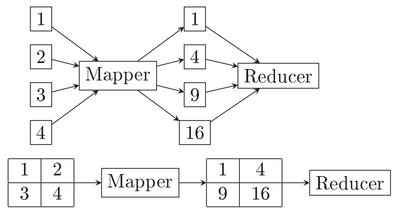
Results

We ran a query that finds the median within a sliding $3 \times 3 \times 3$ window, using an $800 \times 800 \times 800$ 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 intermediate data ("Map output materialized bytes") was reduced by 77.8% (from 55.5 GB to 12.3 GB).
- Unfortunately, total runtime increased by 106% (from 183 minutes to 377 minutes).

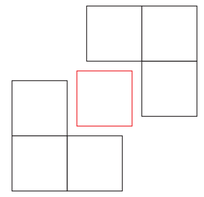
The runtime increase is due to the cost of the transform, which is roughly 2.9 times the cost of gzip alone.

Key redundancy



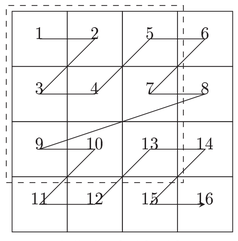
Hadoop assumes that all key/value pairs are independent. However, we know that (in this case) they form a dense, contiguous grid. The regular pattern means that most of the key information is redundant.

N-dimensional aggregation



When a cell may belong in multiple rectangles, the optimal choice is not obvious. Since the data may have any dimensionality, the problem is equivalent to finding a minimal hyperrectangle partitioning of an orthogonal polytope.

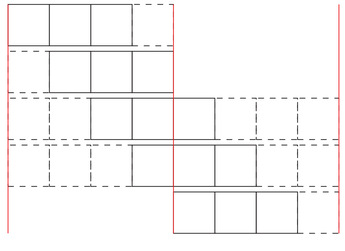
Space-filling curve



1-5, 7, 9-10, 13

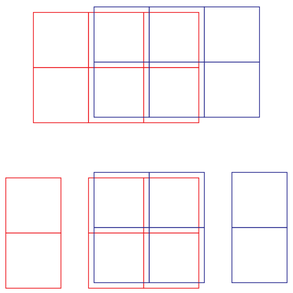
- Cells are numbered with a space-filling curve
- Contiguous numbers are collapsed into ranges
- Multiple curves are possible, including Z-order and Hilbert

Unavoidable overlap



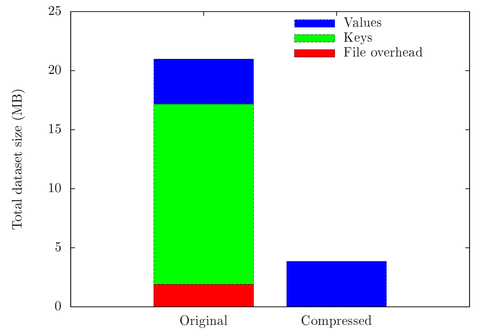
In the general case (such as with a sliding window), it is impossible to avoid key overlap. Even if we try to define alignments and extend keys with empty elements to fit the alignments, there will still be keys that lie across the alignment boundaries.

Key splitting



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 unequal ranges break this constraint when used as keys. To fix this, overlapping ranges are split on overlap boundaries.

Effect of key aggregation



Key aggregation virtually eliminates key overhead for a $100 \times 100 \times 100$ 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 within a sliding $3 \times 3 \times 3$ window, using an $800 \times 800 \times 800$ grid of integers. The cluster has 5 nodes, with 5 reducers and 10 map slots.

- Intermediate data ("Map output materialized bytes") was reduced by 60.7% (from 55.5 GB to 21.8 GB).
- Intermediate key/value pair count ("Reduce input records") was reduced by 73.3% (from 2,293,736,960 to 612,615,471).
- Total runtime was reduced by 28.5% (from 183 minutes to 131 minutes).

Conclusion

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

Funding

This work was supported by Department of Energy grant DE-SC0005428 and the Los Alamos National Laboratory/University of California, Santa Cruz Institute for Scalable Scientific Data Management (ISSDM).