Live Data Compression Using Word-Aligned Hybrid (WAH) Algorithm

Rachel Hirsch
Department of Computer Science, The University of Puget Sound, Tacoma, WA

BACKGROUND & IMPORTANCE

This project builds on previous research on the handling of Big Data, data management, and data compression algorithms, with the main focus being compression of live bitmap data. The project this research is building on relied primarily on test data rather than real-time data streaming. The primary part of this research investigates the optimal speed of streaming in bitmap data before compressing it using the already-existing Word-Aligned-Hybrid (WAH) bitmap compression algorithm.

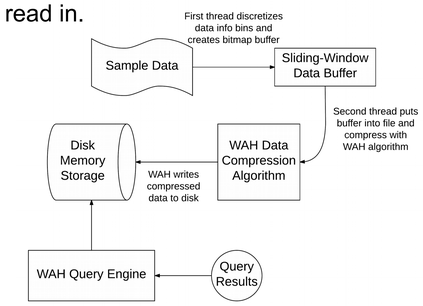
The amount of data which can potentially be queried from greatly increases every day, especially from systems like seismic sensors or websites like Google with large amounts of new data input every second. One of the biggest challenges of working with Big Data is how to deal with such an influx of information without immediately running out of memory too quickly or slowing down queries (i.e. data lookups). Fortunately bitmap compression, like WAH, can be used to make querying faster by condensing the data, thus making it easier to store much more data in one place.

Before this summer, the data being compressed was written directly to large, slow, albeit non-volatile storage on the hard disk. Usage of fast storage and retrieval mechanisms are critical in order to accelerate query time. Random accesses of the disk takes an exceptionally high amount of time, compared to searching for data in a buffer.

DATA COMPRESSION

To reduce disk access, I used two threads and a sliding-window data buffer for incoming live data. This buffer is actually comprised of two buffers of the same size (INPUTLEVEL) - one buffer is filled by one thread while another thread compresses the remaining buffer.

The first thread reads each line of the original data file, turns it into a bitmap representation of the data, then writes it to the read buffer. Once this buffer is full, it gets compressed using the WAH algorithm and the second buffer becomes the read buffer. Once compression finishes, the buffers swap positions again. The thread being compressed turns into the buffer being written to, and the process continues until the incoming data finishes being read in.



RESULTS & CONCLUSIONS

Test 1: Changing Input Level

- Time taken to compress a file of 4177 lines decreases quickly until approximately INPUTLEVEL = 100, where the time taken remains almost constant up through INPUTLEVEL = 1500.
- This may be because it takes approximately as long to read in greater than 100 lines as it does to compress the same number of lines in the compression buffer.
- For this reason, when testing line length, INPUTLEVEL = 750 was arbitrarily chosen.

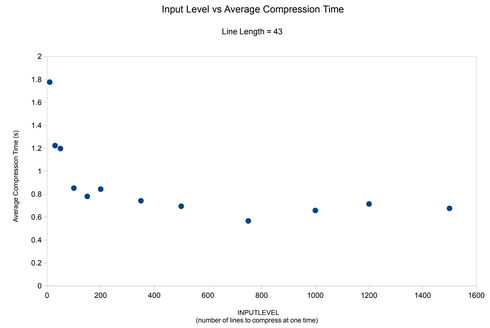


Figure 1. Line input amount (i.e.the number of lines read into the sliding buffer before compression) vs the average time taken to compress a file with 4177 lines.

Test 2: Changing Line Length

- N is the length of each line being compressed, i.e. the number of bins the data was discretized into by the read thread.
- Time taken to compress a file of 4177 lines increases steadily from N = 43 to N = 133.
- This was surprising, as the WAH algorithm intuitively seems like it would compress longer lines faster. However the presence of mostly literal words rather than fills may have something to do with this discrepancy.

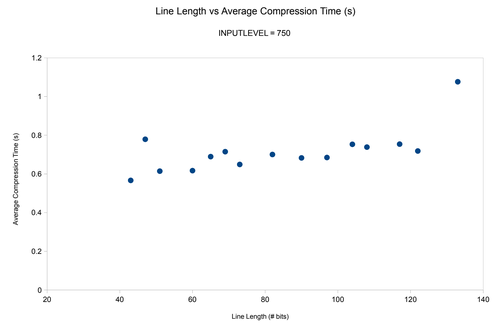


Figure 2. Line length (i.e. the number of bins per line) vs the average time taken to compress a file with 4177 lines.

BITMAP INDEXING

Bitmap indexing is a way of representing large data sets by grouping their attributes into bins - often spanning a range of values - and representing the values within each row as a 0 (False) or 1 (True). This kind of data may be seen in a database such as that of a national census or a social media platform.

To build a bitmap index of this database table we put the “Sex” and “Age” attributes into separate bins. The Sex column is split into two bins (“M[ale]” and “F[emale]”) while the Age category is split into three range-bins (0-18, 19-65, 66+), though we could have split this table any number of ways (e.g. 0-12, 13-19, 20-35, 36-60, etc). These columns contain only 0 or 1, depending on whether the original data falls into a given bin.

Sample Database Table:

Tuples	Name	Sex	Age
t1	Yuki	F	18
t2	Kiona	F	31
t3	Andy	M	47
. . .	. . .	. . .	. . .

Corresponding Bitmap Index:

Tuples	Name			Sex		Age		
	Yuki	Kiona	Andy	F	M	0-18	18-60	60+
t1	1	0	0	1	0	1	0	0
t2	0	1	0	1	0	0	1	0
t3	0	0	1	0	1	0	1	0
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

TESTS

- 1. We hypothesized that the data streaming algorithm could compress at least 60 lines of data streaming in real-time. If this level can not be handled, then queueing theory tells us that the data would be infinitely queued and catch-up would be nigh impossible.

- 2. We then tested whether changing the width of the bitmap being read into our program would affect how fast a 4177-line bitmap file takes to fully compress.

The original dataset we used came from UC Irvine's Machine Learning Database. It lists line number, then eight comma-separated attributes of abalone. The read thread grabs one line at a time and discretizes the data into bins. We began with 43 bins, then tested compression time when adding more and more bins.

ACKNOWLEDGEMENTS

Thank you to Alexia Ingerson for providing the WAH code from her Summer 2015 research project.

And many thanks to my research advisor Professor David Chiu for picking me for this project and helping me through it all summer and to the University of Puget Sound for providing resources to work on and present my research.

Sample data was obtained from UC Irvine's Machine Learning Database.

This project was funded by an Adam S. Goodman Research Scholar Award for Summer 2016.