Apache Arrow Memory Use

Sabrina Li

The goal for this Apache Arrow experiment is to find how effective dictionary encoding and storing in-memory Arrow objects as Feather using compression configurations (LZ4 or ZSTD) is in reducing Arrow object memory consumption. To do this, various datasets were loaded into Arrow and turned into Arrow objects of which the sizes with dictionary encoding, without dictionary encoding, with compression and without dictionary encoding, and with both dictionary encoding and compression were compared using pyarrow code.

To give a summary of the hardware and software setup, all code and values were written and recorded on a MacBook Air version 10.14.5 with a 1.8 GHz Intel Core i5 Processor, 8 GB 1600 MHz DDR3 Memory, and 32.38 GB available storage.

Ten datasets from Kaggle were taken for comparison ranging from 100 MB to 3 GB in size and varying in number of columns as well as data types. The predicted outcome is for both dictionary encoding and compression to reduce the size in memory of the objects and for the two combined to dramatically decrease the size. The different sizes in memory were recorded (Table 1) and compared. Although the results for most datasets align with what is expected, dictionary encoded objects being smaller in size than non-dictionary encoded objects, compressed objects being smaller in size than non-compressed objects, and dictionary encoded and compressed objects being smaller than either, some anomalies regarding memory consumption can be seen in the files of size .998 GB, 1.33 GB, and .318 GB. The size in memory of the encoded and compressed versions of these files were shown by pyarrow code to be larger in size than the version of the file that was just encoded.

Table 1: Sizes of Arrow Objects in Memory

File Number	1	2	3	4	5	6	7	8	9	10
Size of File (GB)	1.33	2.63	.998	1.33	.318	.772	.420	.135	.215	1.19
Size w/out Encoding w/out Compression (GB)	1.745	3.903	1.033	1.435	.412	.879	.540	.241	.389	2.117
Size w/ Encoding w/out Compression (GB)	.683	1.694	.0527	.453	.159	.469	.231	.141	.219	1.254
Size w/out Encoding w/ Compression (GB)	1.12	2.52	.549	.923	.185	.484	.314	.204	.313	1.9
Size w/ Encoding w/ Compression (GB)	.448	1.1	.507	.672	.212	.186	.069	.090	.131	.621

To further investigate this irregularity we looked into the datasets themselves; the number of columns, data types of columns, and the percentages of null, mismatched, and missing data were taken down(Table 2). As a result, the datasets containing only numerical data and no null, mismatched, or missing data seem to be the ones adhering to expected size changes. However, datasets that contain null or mismatched values and nonnumerical values seem to be the ones that go contrary to expectation.

Table 2: Columns and Data Types

File Number	1	2	3	4	5	6	7	8	9	10
Size w/	.683	1.694	.0527	.453	.159	.469	.231	.141	.219	1.254

encoding										
Size w/ encoding w/ compression	.448	1.10	.507	.672	.212	.186	.069	.090	.131	.621
# columns	2	3	19	11	8	11	5	79	79	3136
Data types	numbers	numbers	Dates and strings	Dates and T/F's	Numbers and NaN values	Dates and strings	numbers	numbers	numbers	numbers
Contains null values	None	None	[68%, 49%, 67%, 56%, 26%, 2%, 6%, 100%, 54%, 99%, 68%, 74%, 46%, 8%]	[70%, 5%]	None	None	None	None	None	None
Contains mismatched data	None	None	None	None	[26%, 26%, 26%, 26%, 26%, 26%, 26%]	None	None	None	None	None
Contains missing data	None	None	[68%, 49%, 67%, 26%, 2%, 6%, 100%, 54%, 99%, 68%, 74%, 46%, 8%]	[70%, 5%, 1%]	None	[5%, 100%, 100%]	None	None	None	None

When we look at File 3 and File 6, we see that although the data types are the same and File 6 contains missing data, the size of the File 6 object with encoding is larger than the size of the object with encoding and compression as it intuitively should. File 3 on the other hand has a larger encoded and compressed object. It can then be reasoned that the existence of null values might be an underlying cause of the irregularity. If we take a look at Files 4 and 5, we can see something similar. The encoded and compressed objects are larger than the objects with just encoding and the files both contain null values or mismatched data. Therefore it can be reasonably concluded that the existence of null and mismatched data values may be what is causing compression methods to be undermined.

In conclusion, we can say that arrow is effective in optimizing space using dictionary encoding as well as compression given that the datasets do not contain errors in its data values.