Accelerating CSV Analysis Using a Databases Approach

Arjun Rawal
arjunrawal4@uchicago.edu

Motivation: Even with an increased focus on analytics and data storage size, CSV remains a popular choice for simple data storage as it is human readable and portable. Providing the same capabilities as optimized formats such as Parquet and HDF5 is a challenge as CSV requires more data movement and parsing. We use techniques from data compression and database theory to allow users to pass queries when reading a CSV, allowing larger files to be analyzed with limited memory, and providing fast access to small subsets of data that a user may want to view. To achieve these results, we use categorization and optimized datatypes to reduce memory size, and store in memory representation to remove text parsing overhead.

Approach: Very large CSV datasets (>5 GB) are commonly found, and deriving valuable insights from them requires extensive memory to read and parse the entire file at once, or user overhead to divide up computation and memory usage, increasing time to results. Pandas, a Python data analytics framework, provides a `read_csv` function which is configurable, but is fundamentally limited by it having to load and parse every line of the data. The CSV format has no fixed offsets or higher lever structure, so a query to just the last row will require seeking through the entire file, and doing some text parsing on each row. Reading a file that is larger than memory will require parsing in chunks, adding additional user development time to manage total memory consumption and aggregate results. Furthermore, if a secondary query needs to be run on a larger than memory dataset, the entire file has to be read and parsed again, increasing the overall user runtime. Other formats such as Parquet and HDF5 are better at dealing with larger than memory datasets, and perform better than CSV on parsing and querying. However, they are harder for non computer scientists to work with. We show that storing a secondary representation can speed selection query times by >67x and cardinality queries by >40x over the standard `read_csv()` and then `drop()` approach, while keeping additional storage overhead below 25%.

Memory Representation: We realize that latency overhead of multiple queries (due to repeated parsing) can be eliminated by storing a memory representation on disk, which eliminates the need to redo the text parsing on every read. However, this representation can be larger than the CSV file, requiring more data movement and degrading performance . Therefore, we create a procedure to reduce the in memory size of Pandas dataframes using best-fit datatypes and automatic categorization. We use categories to replace values that occur more than 10% of the time, as many values are repeated extensively (State, Country, etc).

We also reduce integer column memory by looking at the range of values in each column and matching to the smallest numpy datatype that fits it (int8, int16, etc). Because pandas natively stores all datatypes in the largest type, and does not remove duplicates, this can reduce the size of in memory representation by >90%. We use Apache Feather to write the reduced memory dataframe at >80x the speed of CSV writing, and store a third representation of individual columns to enable even faster cardinality queries. This one time data conversion is done with the first call to read.

Methodology: We implemented *ReadFast*[1] in Python 3.7.6 using Pandas 0.24.1 and Feather 0.4.0. All evaluation was run on a 2.9 GHz processor with 8GB RAM. We analyze on the Stanford Open Policing dataset from California (6.7GB). [2]

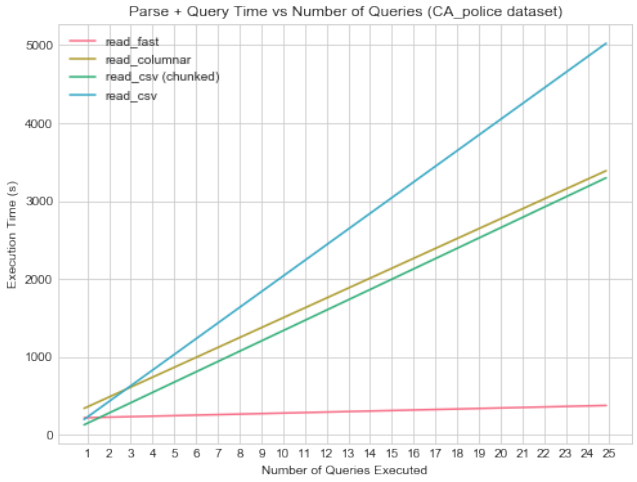

Figure 1: Benefit of Secondary Storage on Query Runtime

Results: We see that the storage overhead for the memory representation of both the entire frame and individual columns is 20% of the original CSV size. Queries on one column (SELECT * FROM ca_police WHERE COL = X) are 20x faster than regular read_csv, and 16x faster than iterated read_csv. Queries on 20 columns are >60x faster than read_csv. Additionally, the columnar storage enables cardinality queries, which are >500x faster than than read_csv on one column, and >50x faster on 3+ columns. The benefit on multiple columns decreases as the cost to intersect and union indices starts to take a substantial portion of the overall execution time. The overhead of the original parsing and write out for *ReadFast* is 1.5x the cost of a normal CSV query, so within 2 queries *ReadFast* has reduced the the overall execution time. As demonstrated in Figure 1 the cost of additional queries is extremely low when compared to other approaches. This approach is scalable and can reduce the time to execute queries when data is either numerical, or contains strings which can be categorized. Data sets that have less repetition may not benefit as much, but even in non optimal cases, this approach will not increase overall query time on multiple queries. Future work could focus on making a multi-threaded version of this software, and integrating fast compression libraries to reduce the size of Feather files.

[1] https://github.com/arjunrawal4/pandas-memdb
[2] https://openpolicing.stanford.edu/