INTRO

By: Abhinav Prasad, Ethan Cohen, and Michael Feigen

In today's world, large scale analytical data processing has become increasingly popular. Many businesses and researches have begun taking steps to collect as much data as possible knowing that, even studying seemingly unnecessary variables, data scientists can possibly find latent trends that can help improve their findings. However, collecting an enormous amount of is nonsensical if the data can't be processed and understood in a reasonable amount of time. While efficient coding techniques are tremendously important and isn't something we can just overlook, it is not the only way to make data science more efficient. By looking at different file formats and how we read and write the data, rather than how we manipulate it, we can find many more ways to boost efficiency.

We will be considering 3 different file formats, CSV, Parquet and Avro. This paper will give you a brief understanding of how each of them work and explain why we have determined that for data science purposes, Parquet is the optimal choice.

CSV

CSV (Comma Separated Values) files tend to be the most popular file format today due to its simplicity. However, while it is used throughout the world by many businesses and students, it is not practical for any data over a few gigabytes. The main issue with CSV is that it is stored in a row based fashion as you can see in Figure 1. Row based storage is poor for two reasons. First it means that during a search of the data, every row must be checked. Typically, each row represents a data point, so when working with big datasets this could mean searching through hundreds of thousands of rows. This also means that the entire file must be read, even if you only want to work with a specific column. This is because after finding the first value for that column, the next value is not adjacent; it must be found be reading through all the other data in-between. Secondly, with this storage format, there is little room for any compression. The adjacent values may not be of the same type since they represent different columns, and this makes the file larger than it has to be. Additionally, there is no real definition of data or schema,

meaning that all the values have to be stored. CSV does have one major benefit. Most people know how to use CSV in Microsoft Excel and therefore it is easy for people unfamiliar with data science to use and works very well for business communication with clients.

Figure 1:

DataFrame

Name	Value
A	1
B	2
C	3
D	4

Row Based Store: < A , 1 , B , 2 , C , 3 , D , 4 >

Column Based Store: < A , B , C , D , 1 , 2 , 3 , 4 >

Additionally, CSV has no elegant solution to store nested structures. For example, looking at Figure 2, a CSV just essentially stores a nested data structure identically to a flat one, save for additional column demarcation. This makes the parsing of a nested structure even more tedious. Nested structures are quite important, in big data analysis; in fact, it was the nested structure which led Google to develop Dremel and Apache to develop Parquet.

Additionally, CSVs must also physically store a null indicator, as show in Figure 2. For sparse data sets, often found in many real-world scenarios (for examples reviews and Google's data about users) may contain many missing or null fields. In a CSV, each single value must be stored, which again is a massive wastage of space, when considering that often billions (or even trillions of records) with millions of columns may have many null/empty values.

Figure 2:

DataFrame

Nested Storage: < A, 1, <z, y, x>, B, 2, <w, r, *null*>>

A	1	z y x
B	2	w r *null*

Avro

The Avro file format has many benefits, and depending on the application, it can be more reliable and efficient than Parquet. Avro uses row based store which is the main reason why it does not reach the speeds of Parquet files when reading data. Like mentioned earlier with CSVs, row based means that in order to search through specific columns, you must search through all of the data. This is because all the values of that row are stored in-between two values that are from the same column; you need to look through a lot of useless data to find what you are looking for. Also mentioned earlier, compression is more difficult with row based store since adjacent values may not be of the same type. Avro does use some compression, and is much more efficient than a CSV file; however, the lack of a columnar format decreases the opportunity for the file to be more efficient. The benefit to Avro is that, because it is stored in rows, it writes data quicker than Parquet. This may be beneficial for companies that are continuously updating their algorithms and models as new data comes in. However, they still must reread the files in order to update models, so it is unclear how much more efficient Avro would be in these circumstances.

Parquet

The parquet file format is significantly better than CSV when it comes to efficiency and for Data Science purposes, we feel that Parquet is also more efficient than Avro due to its Column Based storage system. The Parquet format is based on the Dremel Paper, written by researchers at Google, and many of the largest technology companies have begun using Parquet including Google, Facebook, and Twitter. The main advantage of Parquet is that it uses column based storage, which is demonstrated in figure 1. Column based storage is helpful for several reasons. Firstly, it makes it possible to search through values in a specific column without having to look through all the data, because all the values of the same column are stored adjacent to each other. This also means that within a specific index, all the values are the same type and you can skip right to the beginning of the index of the column you are requesting. Additionally, modern data compression techniques work best on the same data type. Since columns are

of the same data types, each column can be efficiently compressed when stored, greatly reducing storage. However, the greatest benefit of the parquet is the schema it uses to define its structure, which greatly reduces the space needed for data storage.

THE BASICS

Parquet only uses two ways to store data: a field and a group, which itself consists of fields. All data sets, even extremely complex ones can be represented as a combination of this fields. Figure 3 shows the CSV example in this form. This makes the schema easy to represent, using only these two fields.

Each field has an additional category called the repetition. Repetition refers to how many times the field occurs or can occur in a record. The repetition categories are required (once), optional (either 0 or 1 occurrence), and repeated (0+ occurrences). This makes the parsing easier unnecessary fields don't have to be looked at.

Figure 3: Example of a parquet schema

```
schema{
        Repeated field letter;
        Repeated field integer;
        Repeated group letter_group{
                Repeated field letter_end;
        }
}
```

One main advantage of parquet is that it stores nested structures in a flat manner. For example, referring back to the CSV example, we had to see that each null value had to be stored, leading to a massive waste of space. The Parquet/Dremel schema handles this via a header field called the definition level. The definition level is to define is "value of a field with path p, esp. every NULL, has a definition level specifying how many fields in p that could be undefined (because they are optional or repeated) are actually present in the record." Essentially, this refers to the number of fields that could be null in a nested structure. The point of this is in nested columns. For example, in the null value in figure 2, the definition level would be 1, as the group exists but the field is actually null. w would have a definition

level of 2 (as its actually defined). Definition levels are stored in the header and allow the data to be reconstructed quite easily. Additionally, null values do not be stored due to this. In the first example (max definition level was 2), since null has a definition level of less than 2, then it means there are null values. Furthermore, definition levels can be represented by bits quite easily. Using a byte can represent up to 255 definition levels. This allows a cheap way of storing nulls.

Repetition level is another important part of Parquet. Repetition level refers to "what repeated field in the field's path the value has repeated," or essentially the marking of a new list. The repetition level is similar to the definition level in the way its defined. In simple terms, the repetition level *n* of a field demarcates the start of an *n+1* list (a record at the *n+1* level), which also implies the creation of level *n +2, n+3, …* lists as well. For example, the repetition level of the CSV example is shown again.

Value	Repetition Level
A	0
1	0
z	0
y	1
x	1
B	0
2	0
w	1
r	1

While in this example, its not too helpful, in highly nested structures the repetition level makes it easy to restructure the data. Additionally, markers for each new list start do not have to be stored, as the repetition level covers that, which again reduces the data to be stored.

All these aspects of the Parquet file are represented via bits, which allow for run-length encoding and various other compression strategies that greatly reduce the space required to store it.

Conclusion and Testing

Overall, we focused a lot on Parquet due to fact Parquet to CSV conversions are relatively simple. Additionally, due to the fact that Spark does lazy reading, the nature of Parquet is extremely better for CDS's purposes. Additionally, we didn't focus too many resources on Avro and testing in general, due to the fact that many online sources had done testing using vast amounts of data. For all types of testing, the Parquet format was significantly better than Avro, CSV, or any other file format. As a result, we suggest that Parquet is the best to use, not even considering the fact that most industrial applications use Parquet simply due to the fact that its not tied down to a language or framework and can easily be working into MapReduce and the like.