Physical data layout and storage 10/23/2019

Lab due today

Last time - performance
- python vs pandas vs C vs SQL
- quantifying performance
- fixing performance issues
  - indexing

This time: data layout
Key concepts - show slide

What do we mean by locality?
Locality: faster to access data near data you have already accessed, than data that is far away.

Locality matters, on disk and in memory
- show benchmark

Disk:

1K reads from 2GB file, purging buffer cache between reads

```
time for 1M random accesses = 7.107
bandwidth = 137.41 MB/sec, iops = 140709.87

time for 1M sequential accesses = 0.610
bandwidth = 1600.07 MB/sec, iops = 1638473.99
```

10x difference

Memory:

Read 1 8 byte integer

```
total = 0, time for 10M random accesses = 0.030
bandwidth = 1312.34 MB/sec

total = 0, time for 10M sequential accesses = 0.005
bandwidth = 7613.25 MB/sec
```

4x difference

Not totally trivial to figure out how to store data in a way to maximize locality; depends heavily on the types of operations we want to do on data. For example, suppose we want to find sales by region by date. How might I arrange my data to support that?

Is it better to order by date, or by region? Depends.

What if regions aren't defined by a name (like "NE"), but by lat/lon? How do we know which regions to look at? Need some way to map from lat/lon to regions (like and index!)

Ideas we're going to talk about make the most sense when you know you are going to be reaccessing data a lot. But even if you're only accessing it once, in some cases you will have control of the upstream process writing out the data, and can choose a more efficient representation. Just have to be careful because some kinds of upstream operations -- like sorting -- may take more time than they are worth.

Ok, so onto the layout question

Layout w/ tabular data is challenging because memory / disk are linear and data is at least 2D (tabular) -- so have to figure out how to linearize a data set. (Show slide)

Options:

- Row by row (Show slide)

Basic row store layout and representation

What queries will this be good for?

Lookups of individual records? Maybe, if data is ordered, or we have an index

Inserts -- especially if we are going to append to the end of the file -- are very efficient

<u>How can we do better?</u>

Vertical partitioning (Show slides)

When is vertical partitioning a good idea? When I only need to access a subset of my columns -- can simply avoid reading data from disk. With a row store, I can't "skip" the columns on disk I don't want because I'd have to do a lot of random I/O to do that, and that would kill performance as we showed.

But there are some challenges? What if I need to get the data for a particular record? Now I have to go to several places on disk? Doesn't that make performance worse? (Yes!)

So this representation only makes sense when I need to reconstruct many / most records, not just one or two.

Note that in order to keep relationship between columns the same, I need to store them in the same order on disk (kinda obvious).

Show query processing on column stores slides.

Introduce parquet -- an open source column oriented format for data science

Obtain benefits of column orientation, and overcome limitations of CSV

What's wrong with CSV?

CSV is a terrible storage format -- much better to have a representation where the schema is known up front, and where the layout is binary. Parsing is slow -- have to search through data for delimiters -- if instead fields are of known size, we bulk copy data from memory into disk.

May seem obvious, but....

Can have huge performance difference

Show parquet layout (actually allows groups of columns to be stored together) -- (show slide)

Putting it all together -- Parquet performance of parquet vs CSV for some data and formats (show demo and slide)

Other layout tricks

Horizontal partitioning (Show slide)

- Slice dataset according to some attribute
- Works well when I frequently order on a particular column
- Plays well with vertical partitioning -- even if sorted on date, I don't need the parts of the columns that correspond to dates I don't care about

Sorting (Show slide)

What if I don't have a single order for my data? A couple of ideas:

- tiling and spatial indexing (show slide)

- zorder (show slides)

Other ideas:

compression - black box, i.e., entropy coding

vs

light-weight compression, e.g., run length encoding, delta encoding, and dictionary encoding

Key benefit of dictionary encoding is that it makes variable length data fixed length, which can significantly improve performance of operating on data. (Show diagram)

Null suppression - sparse data encoding

Adding new data -- update in place vs store new partitions

New partitions preferable but data will accumulate over time

Introduce log-structured merge idea