Tools for uncomfortably big data

Edward Visel
LA R Users Group Meetup
2020-09-22

Edward Visel

The short version:

- Philosophy major
- Worked in campaigns
- Data scientist at Uptake, which uses ML to identify problems in assets in the industrial internet of things
- Have worked on:
    - Anomaly detection algorithms
    - Productionalizing data science
    - Delivering DS for our Fleet/Federal team for customers with truck-like assets
- Obsessions: fried egg tacos and clean code

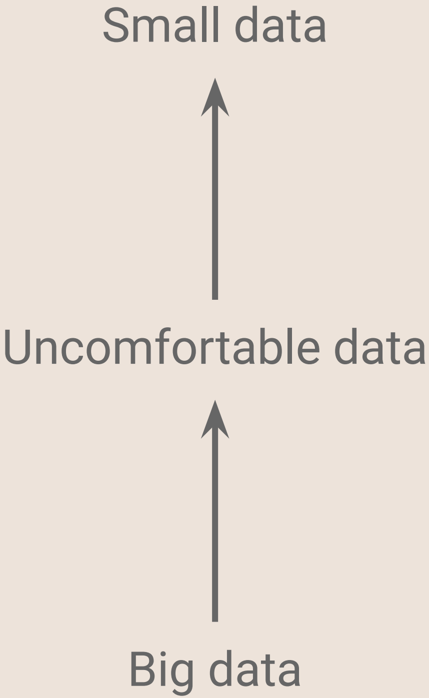
Uncomfortably big data

For the purposes of this talk:

- **Small data:** data you can work with locally without any concessions due to size
- **Uncomfortable data:** data you can work with locally with a little forethought
- **Big data:** data big enough to require significant tooling, probably stored in a managed database or data lake

<u>Maxim for Uncomfortable Data:</u>

Subset to what you need



From uncomfortable to small data

For on-disk data:

Strategy 1: Read it all in, then subset

- ➢ Simplest option for data which will fit in memory

Strategy 2: Subset on read

- ➢ Some reading functions will allow you to subset columns
  - `readr`'s col_types parameter
  - `data.table::fread`'s `select` and `drop` parameters

From uncomfortable

to

small data

Strategy 3: Read in batches, subset and aggregate

- ➢ Almost all reader function have `skip` and `nrows`/`n_max` params that can be used for batching
- ➢ Requires more setup
    - ○ Have to specify column names
    - ○ Be cautious about types so aggregation works as expected

Caution! Don't make 10,000 data frames in your global environment (😬) or try to grow a single data frame by [appending during iteration] (requires a lot of reallocation of memory); [make a list of data frames from the start] and combine at the end.

Demo 1

Subsetting strategies

[Code for demos]

Storing uncomfortable data

If you don't regularly **create large datasets**, you might think you can ignore how to write them...but that's wrong because:

- ➢ Better stored data can be **interacted with more efficiently**, to the point that it's often worthwhile to re-save data
- ➢ You may want to save large intermediate products that took a long time to create
- ➢ Nobody really wants to deal with a multi-gigabyte Excel file with unparsable garbage in there somewhere, or a 281 Gb CSV

Storing uncomfortable data

When to CSV:

- ➢ Your data is small enough that you can look at it all, and visual inspection is useful
- ➢ You want to use command-line tools on it
- ➢ You're giving it to someone who doesn't code

When not to CSV:

- ➢ Your data is 281 Gb
- ➢ You want to ensure types are maintained
- ➢ You care about how much space it takes up
- ➢ You often want to subset it without reading the whole thing

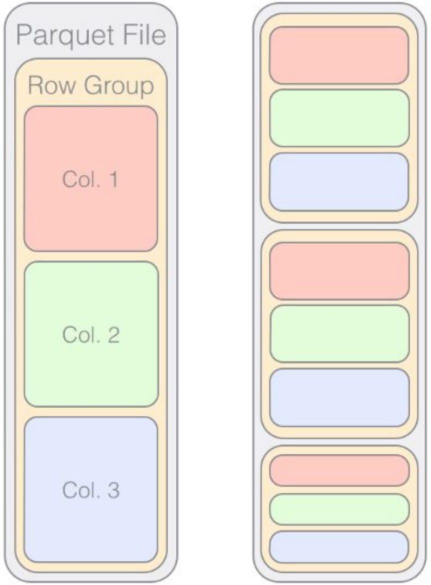
Storing uncomfortable data

Why Apache Parquet

- ➢ A serialized, compressed format like .Rds that **takes up less space** than plaintext
- ➢ Well-supported by **lots of languages** (R, Python, C++, Java, Scala, …) and tools
- ➢ **Maintains types**, so datetimes will stay datetimes and integers will stay integers without a reader function guessing
- ➢ Optimized so individual columns can be read without scanning the whole file, so **reads are fast**

Parquet in R

in 1 minute



Source

Read: `arrow::read_parquet()`

- `col_select` parameter accepts either a vector of names to keep, or a tidy selection specification

Write: `arrow::write_parquet()`

- `version` defaults to `"1.0"` to maximize compatibility, but `"2.0"` enables some cool features
- Default `compression` is `"snappy"`, which aims for reasonably small sizes very fast. `"gzip"` and others are also available, and `compression_level` can be set.
- `chunk_size` is the number of rows to put in a row group. (Think of row groups as files within the file.) When reading large files, smaller row group sizes can allow parallelism, reduce memory consumption, and when well optimized allow groups not to be read if irrelevant.

Storing uncomfortable data

Save one big file or lots of small files?

- One big file allows more familiar idioms—if the machine can handle it
- **More smaller files** are less likely to blow out the machine's resources
  - Partitioning makes **parallelization** easy
  - Tooling can make handling a directory of files as simple as a single file

Before splitting a dataset, consider your **partitioning scheme**

- How will the dataset be read? What **natural divisions** when used for partitioning will allow only the desired data to be read?

Demo 2

Storing data

Tools for working with on-disk data

When working with data on-disk that may be bigger than available memory, batching like in **Strategy 3** still works...but still takes time to write and optimize.

Tooling can help! Good tools let you *subset to what you need **faster***. Today's (non-exhaustive!) subjects:

- ➢ [Arrow Datasets]() are a quick, **lightweight** way to handle on-disk, partitioned data
- ➢ [Apache Drill]() is a robust tool that allows on-disk data to be **queried with SQL** like a database

Tools for working with on-disk data

Apache Arrow is a big project focused on standardizing how data is represented in-memory.

Arrow Datasets are a relatively new tool in R and Python to work with directories of on-disk data.

- ➢ Originally focused around **Parquet**, but now supports other formats including **CSV**
- ➢ Supports a very limited subset of **dplyr syntax**
- ➢ Great for *subsetting to what you need*; not great for calculating summary stats or sophisticated operations

Tools for working with on-disk data

To create an Arrow Dataset, use `arrow::open_dataset()`

- Pass `sources` a path to a directory of data files
- Use `format` to specify filetype. Default is `"parquet"`; other options include `"csv"`, `"tsv"`, and `"feather"`
- `partitioning` can be used to capture directory structure within the `source` path as variables
  - E.g. with a relative path of `2020/09/22/data.parquet` and `partitioning = c("year", "month", "day")`, three variables of those names will be appended to the dataset

Tools for working with on-disk data

Arrow Dataset dplyr syntax

- ➢ Accepts `select()`, `rename()`, `filter()` with very simple conditions, and `group_by()`
- ➢ Delayed evaluation, so use `collect()` to evaluate and get data, like working with a database from dplyr
- ➢ Does not yet accept `mutate()` or `summarise()` calls until after collection (at which point you are working with a normal, in-memory data frame)

☞ Great for *subsetting to what you need*; not great for your whole workflow

Demo 3

Arrow Datasets

Apache Drill

tl;dr

- Drill lets you write SQL against files without standing up a database
    - The files can be anywhere, including your hard drive
    - Lots of file formats are supported (CSV, TSV, Parquet, JSON, etc.)
    - Globbing works
        - e.g. `ls /tmp/*.csv`

What is it?

> Schema-free **SQL Query Engine** for Hadoop, NoSQL and Cloud Storage
>
> Drill supports a variety of NoSQL databases and file systems, including HBase, MongoDB, MapR-DB, HDFS, MapR-FS, Amazon S3, Azure Blob Storage, Google Cloud Storage, Swift, NAS and **local files**. A single query can join data from multiple datastores.

⟱

SQL on local files!

Installing Drill and accessories

Install Drill via a package manager on your OS. For macOS with homebrew,

```
$ brew install apache-drill
```

- ➢ Drill requires Java 1.8.
- ➢ To keep Drill available as a service, install zookeeper similarly and start it (instructions).

To use Drill from R, install sergeant (uses ODBC interface) or sergeant.caffeinated (uses JDBC; requires working rJava) as usual with `install.packages()`.

Starting Drill

To run Drill locally in embedded mode (not as a service), after installation start it from the command line with

```
$ drill-embedded
```

and you'll be greeted by a **Drill shell** where you can run queries and commands.

Once Drill is running, navigate to http://localhost:8047/ (adjust port if you've changed it) to see a website-based interface where you can query, configure, and administer.

Querying a filesystem with Drill

Storage plugins tell Drill where to look

- The local filesystem is available as `dfs`
- Can configure to change file handling behavior. Other storage plugins can be added, e.g `s3`

Workspaces are path shortcuts

- Defaults for `dfs` include `dfs.root` (`/`) and `dfs.tmp` (`/tmp`). Add your own!

Pass `FROM` clause of a query

1. Storage plugin, e.g. `dfs`
2. Workspace joined by `.`, e.g. `tmp`
3. Path, wrapped in `` `...` ``

...e.g. `` dfs.tmp.`flights/*` ``

Handy features

Use `SHOW FILES FROM` …to navigate

Set a default workspace with `USE`

- E.g. after `USE dfs.tmp;` can use `` FROM `flights` `` instead of `` FROM dfs.tmp.`flights` ``

Implicit columns offer access to path and filename

- Select `FQN` (fully qualified name), `FILEPATH`, `FILENAME`, or `SUFFIX`, and they'll magically appear as columns
- Subdirectories of the directory query will be appended as `dir0`, `dir1`, ...

Demo 4

Apache Drill

…ok cool but I thought this was a talk about tools I could use in R

Connect to Drill from R with sergeant

sergeant is an R package by Bob Rudis which defines a DBI interface and dbplyr backend for Drill

- ➢ Drill in a docker container with `drill_up()`
- ➢ Has both Drill **ODBC** and **REST** interfaces
- ➢ To use **JDBC**, use sergeant.caffeinated, which works the same way
- ➢ To use dbplyr, pass `tbl()` a DBI connection created with `dbConnect(Drill())` or `src_drill()` and a `FROM` clause argument
- ➢ Lots of nice helpers for SQL translation! See `?drill_custom_functions` and `sql_translate_env(src_drill()$con)`

dbplyr

basics

dbplyr: A database backend for dplyr

- Most dplyr verbs (e.g. `select`, `filter`, `mutate`, `summarise`) work as usual
- Operations in mutating verbs may not.
  - Recognized functions will be translated to SQL
  - Unrecognized ones will be passed through to Drill untranslated, allowing the use of SQL functions
- Queries are lazy and only return the first 10 rows. To return everything, run `collect()`
- To see what query will be run, use `show_query()`

Demo 5

Drill in R via sergeant

References and reading

- Querying across files with Apache Drill
- Drill docs
  - Drill in 10 Minutes
  - Drill SQL reference
- sergeant
  - sergeant.caffeinated
  - Using Apache Drill with R ← start here
- dbplyr
  - Getting started with dbplyr
  - Writing SQL with dbplyr
  - Function translation
- Arrow and parquet
  - arrow R package
  - Datasets vignette

Edward Visel @alistaire alistaire.rbind.io