Bigger than RAM data

Statistical Computing & Programming

Shawn Santo

Supplementary materials

Full video lecture available in Zoom Cloud Recordings

Additional resources

- disk.frame website
- useR presentation - 2019
- useR presentation slides - 2019
- fst website

Introduction to `disk.frame`

What is `disk.frame`?

- R package that will allow you to manipulate larger-than-RAM tabular data efficiently
- Rather than be limited by RAM, you are only limited by the amount of disk space you have available
    - In most cases you won't be able to bring objects into memory in R that exceed 4GB

    ```
    Error: vector memory exhausted (limit reached?)
    ```

`disk.frame` **versus** `data.frame`

- A data frame is an in-memory list with attributes. Hence, it requires your computer's RAM.
- `disk.frame` manipulates data on your hard drive. It employs a chunking strategy so only small parts are loaded into RAM.
- Recall, with data frames we have a row limit of 2^{31}; there is no limit for `disk.frame` objects other than your hard drive capacity.

How `disk.frame` works

`disk.frame` leverages the packages `future` and `fst`. It is also compatible with `dplyr` and `data.table` syntax for data wrangling. The biggest challenge is getting set-up and understanding how your code is processed.

It leverages two key concepts:

1. Split larger than RAM datasets into chunks and store these chunks in separate files (`.fst` format)

2. Utilize an API to manipulate each of the chunks

A framework for efficient use

We are going to use `dplyr` to manipulate our data. If you know the `data.table` syntax you can use that as well.

The general idea is as follows.

```
df %>%
  some_dplyr_fcn() %>%
  another_dplyr_fcn() %>%
  yet_another_dplyr_fcn() %>%
  collect()
```

The `collect()` function will row-bind the results from the `dplyr` function calls and your main session will receive the results. You should minimize the amount of data passed from the workers to your main session.

The manipulations are done on each chunk.

Toy examples

Set-up

```r
library(tidyverse)
library(nycflights13)
library(disk.frame)
```

```r
setup_disk.frame(workers = 4)

# this will allow unlimited amount of data to be
# passed from worker to worker
options(future.globals.maxSize = Inf)
```

Create a `disk.frame`

In this toy example, we'll create a `disk.frame` object from the in-memory data frame `flights` and mimic some of our manipulations from the `dplyr` lecture.

```
flights_disk <- as.disk.frame(df = flights, outdir = "tmp_flights.df",
                              nchunks = 10, overwrite = TRUE)
```

```
class(flights_disk)
```

```
#> [1] "disk.frame"        "disk.frame.folder"
```

```
class(flights)
```

```
#> [1] "data.table" "data.frame"
```

Examples

```
nrow(flights_disk)
```

```
#> [1] 336776
```

```
ncol(flights_disk)
```

```
#> [1] 19
```

```
names(flights_disk)
```

```
#>  [1] "year"           "month"          "day"            "dep_time"
#>  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
#>  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
#> [13] "origin"         "dest"           "air_time"       "distance"
#> [17] "hour"           "minute"         "time_hour"
```

```
flights_disk %>%
  filter(dest == "LAX" | dest == "RDU", month == 3) %>%
  collect() %>%
  tibble()
```

```
#> # A tibble: 1,935 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     3     1      607            610        -3      832            925
#>  2  2013     3     1      608            615        -7      737            750
#>  3  2013     3     1      623            630        -7      753            810
#>  4  2013     3     1      629            632        -3      844            952
#>  5  2013     3     1      657            700        -3      953           1034
#>  6  2013     3     1      714            715        -1      939           1037
#>  7  2013     3     1      716            710         6      958           1035
#>  8  2013     3     1      727            730        -3     1007           1100
#>  9  2013     3     1      803            810        -7      923            955
#> 10  2013     3     1      823            824        -1      954           1014
#> # … with 1,925 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights_disk %>%
  filter(month == 3, day == 2) %>%
  select(origin, dest, tailnum) %>%
  collect() %>%
  arrange(desc(origin), dest)
```

```
#>      origin dest tailnum
#>   1:    LGA  ATL  N928AT
#>   2:    LGA  ATL  N623DL
#>   3:    LGA  ATL  N680DA
#>   4:    LGA  ATL  N996AT
#>   5:    LGA  ATL  N510MQ
#>  ---
#> 761:    EWR  TPA  N41135
#> 762:    EWR  TPA  N625JB
#> 763:    EWR  TPA  N37408
#> 764:    EWR  TPA  N569UA
#> 765:    EWR  TPA  N73291
```

Why did we arrange after `collect()`?

```r
flights_disk %>%
  group_by(origin) %>%
  summarize(
    n = n(),
    min_dep_delay = min(dep_delay, na.rm = TRUE),
    max_dep_delay = max(dep_delay, na.rm = TRUE)
  ) %>%
  collect() %>%
  as_tibble()
```

```
#> # A tibble: 3 x 4
#>   origin      n min_dep_delay max_dep_delay
#>   <chr>   <int>         <dbl>         <dbl>
#> 1 EWR    120835           -25          1126
#> 2 JFK    111279           -43          1301
#> 3 LGA    104662           -33           911
```

Is this correct? How is this possible?

Before v0.3.0 of `disk.frame`, one-stage group-by was not possible, and the user had to rely on two-stage group-by even for simple operations like mean. Some functions in summarize will not work exactly.

Check our answer with regards to `n()`, `min()`, and `max()` using `flights`.

```
flights %>%
  group_by(origin) %>%
  summarize(
    n = n(),
    min_dep_delay = min(dep_delay, na.rm = TRUE),
    max_dep_delay = max(dep_delay, na.rm = TRUE)
  )
```

```
#> # A tibble: 3 x 4
#>   origin      n min_dep_delay max_dep_delay
#>   <chr>   <int>         <dbl>         <dbl>
#> 1 EWR    120835           -25          1126
#> 2 JFK    111279           -43          1301
#> 3 LGA    104662           -33           911
```

Exercises

1. Use `flights_disk` and compute the mean, median, and IQR for departure delay for each carrier. Arrange the carriers alphabetically. Compare your result to using `flights`.

2. Run the following code. How do you think the sampling is being done?

```
flights_disk %>%
  sample_frac(size = .01) %>%
  collect() %>%
  as_tibble()
```

Chunk distribution

`disk.frame` uses the *sharding* concept to distribute the data into chunks.

```
flights_disk <- as.disk.frame(df = flights, outdir = "tmp_flights.df",
                              shardby = "carrier", nchunks = 10,
                              overwrite = TRUE)
```

All flights with the same carrier are linked together.

Example

```r
flights_disk %>%
  group_by(carrier) %>%
  summarise(med_dep_delay = mediar
  collect() %>%
  arrange(carrier) %>%
  slice(1:7)
```

```
#> # A tibble: 7 x 2
#>   carrier med_dep_delay
#>   <chr>           <dbl>
#> 1 9E               -2
#> 2 AA               -3
#> 3 AS               -3
#> 4 B6               -1
#> 5 DL               -2
#> 6 EV               -1
#> 7 F9                0.5
```

```r
flights %>%
  group_by(carrier) %>%
  summarise(med_dep_delay = mediar
  collect() %>%
  slice(1:7)
```

```
#> # A tibble: 7 x 2
#>   carrier med_dep_delay
#>   <chr>           <dbl>
#> 1 9E               -2
#> 2 AA               -3
#> 3 AS               -3
#> 4 B6               -1
#> 5 DL               -2
#> 6 EV               -1
#> 7 F9                0.5
```

Joins

Joins also work, but the left data object must be a `disk.frame` and the right data object can be a `disk.frame` or data frame.

```
flights_disk %>%
  left_join(airlines, by = "carrier") %>%
  select(name, carrier, ends_with("delay")) %>%
  as_tibble()
```

```
#> # A tibble: 336,776 x 4
#>    name                   carrier dep_delay arr_delay
#>    <chr>                  <chr>       <dbl>     <dbl>
#>  1 United Air Lines Inc. UA              2        11
#>  2 United Air Lines Inc. UA              4        20
#>  3 United Air Lines Inc. UA             -4        12
#>  4 United Air Lines Inc. UA             -2         7
#>  5 United Air Lines Inc. UA             -2       -14
#>  6 United Air Lines Inc. UA             -1        -8
#>  7 United Air Lines Inc. UA              0       -17
#>  8 United Air Lines Inc. UA             11        14
#>  9 US Airways Inc.       US             -8         3
#> 10 United Air Lines Inc. UA             -4         1
#> # … with 336,766 more rows
```

No `collect()` is needed as the joins are evaluated eagerly.

Supported `dplyr` verbs with `disk.frame`

- `select()`
- `rename()`
- `filter()`
- `mutate()`
- `transmute()`
- `left_join()`
- `inner_join()`
- `full_join ()`
- `semi_join()`
- `anit_join()`
- `chunk_arrange()`
- `chunk_group_by()`
- `chunk_summarize()`
- `group_by()`
- `summarize()`

Realistic example

Set-up

```r
library(tidyverse)
library(lubridate)
library(disk.frame)

setup_disk.frame(workers = 6)

options(future.globals.maxSize = Inf)
```

We are going to use the 2009 TLC Trip Record Data. The yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.

Each 2009 CSV file is about 2.4GB and contains about 14 million taxi trips.

Todd Schneider has a GitHub repo will shell files to download all the taxi data -- ~2.8 billion trips in total.

A `disk.frame` from many CSVs

After downloading the CSV files, save them in a folder `data/taxi/`.

```
file_list <- list.files("data/taxi/", full.names = TRUE)

file_list
```

```
#>  [1] "data/taxi//yellow_tripdata_2009-01.csv"
#>  [2] "data/taxi//yellow_tripdata_2009-02.csv"
#>  [3] "data/taxi//yellow_tripdata_2009-03.csv"
#>  [4] "data/taxi//yellow_tripdata_2009-04.csv"
#>  [5] "data/taxi//yellow_tripdata_2009-05.csv"
#>  [6] "data/taxi//yellow_tripdata_2009-06.csv"
#>  [7] "data/taxi//yellow_tripdata_2009-07.csv"
#>  [8] "data/taxi//yellow_tripdata_2009-08.csv"
#>  [9] "data/taxi//yellow_tripdata_2009-09.csv"
#> [10] "data/taxi//yellow_tripdata_2009-10.csv"
#> [11] "data/taxi//yellow_tripdata_2009-11.csv"
#> [12] "data/taxi//yellow_tripdata_2009-12.csv"
```

Read in 1 row of the January 2009 CSV to get the variable names.

```
header_names <- read_csv(file_list[1], n_max = 1) %>%
  janitor::clean_names() %>%
  names()

header_names
```

```
#>  [1] "vendor_name"            "trip_pickup_date_time"  "trip_dropoff_date_time
#>  [4] "passenger_count"        "trip_distance"          "start_lon"
#>  [7] "start_lat"              "rate_code"              "store_and_forward"
#> [10] "end_lon"                "end_lat"                "payment_type"
#> [13] "fare_amt"               "surcharge"              "mta_tax"
#> [16] "tip_amt"                "tolls_amt"              "total_amt"
```

Convert all CSVs to a `disk.frame` with 100 chunks.

```
taxi_disk <- csv_to_disk.frame(file_list, outdir = "tmp_taxi.df",
                               overwrite = TRUE, header = FALSE,
                               nchunks = 100, col.names = header_names)
```

```
csv_to_disk.frame: Reading multiple input files.

Converting CSVs to disk.frame -- Stage 1 of 2 took: 00:04:49 elapsed
 -----------------------------------------------------

Row-binding the 100 disk.frames together to form one large disk.frame:
Creating the disk.frame at tmp_taxi.df

Appending disk.frames:
Stage 2 of 2 took: 00:03:20 elapsed
 -----------------------------------------------------
Stage 1 & 2 in total took: 00:08:09 elapsed
```

```
taxi_disk
```

```
path: "tmp_taxi.df"
nchunks: 100
nrow (at source): 170896055
ncol (at source): 18
```

Wrangle

Compute the mean tip percentage for each day in 2009. How many rows should our resulting tibble contain?

```
taxi_tips <- taxi_disk %>%
  srckeep(c("trip_pickup_date_time", "tip_amt", "total_amt")) %>%
  mutate(trip_pickup_date = as_date(trip_pickup_date_time),
         tip_pct          = tip_amt / total_amt) %>%
  group_by(trip_pickup_date) %>%
  summarise(mean_tip_pct = mean(tip_pct, na.rm = TRUE)) %>%
  collect() %>%
  as_tibble()
```

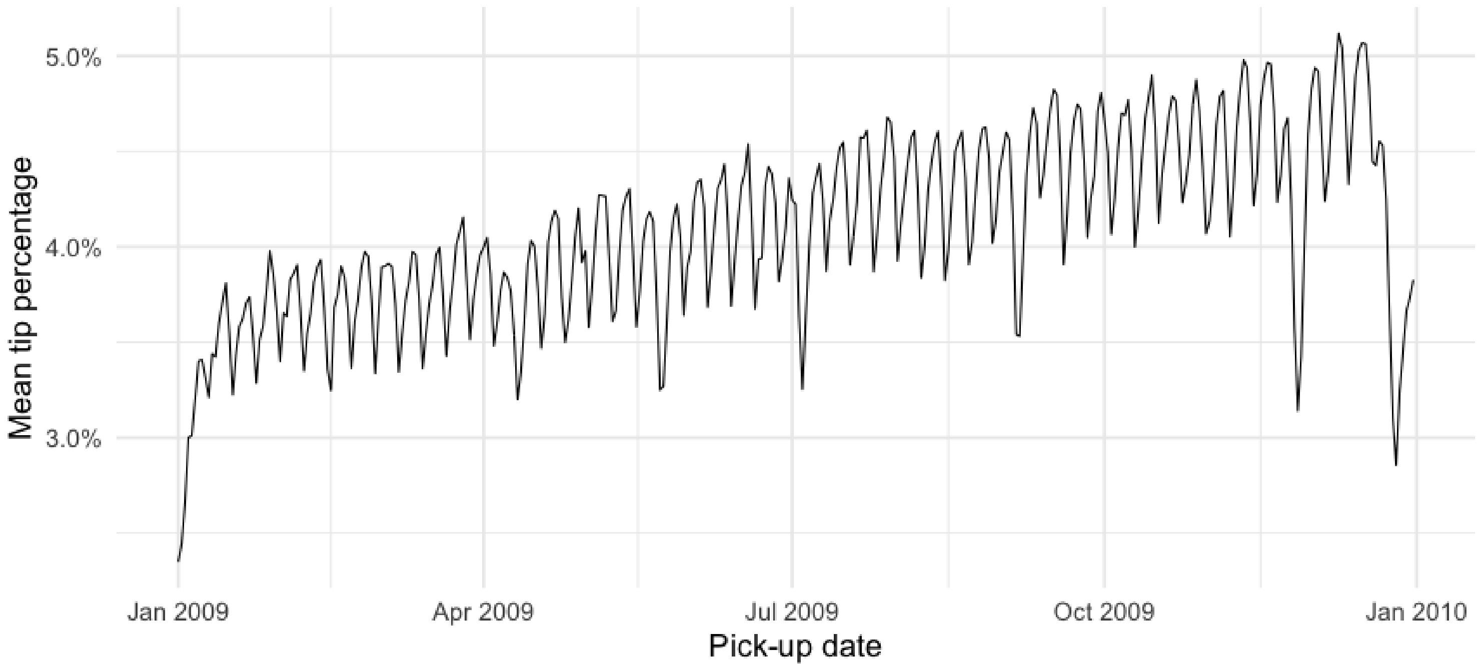
Again, we only `collect()` at the end to minimize the amount of data brought into memory.

```
glimpse(taxi_tips)
```

```
#> Rows: 365
#> Columns: 2
#> $ trip_pickup_date <date> 2009-01-01, 2009-01-02, 2009-01-03, 2009-01-04, 2009…
#> $ mean_tip_pct     <dbl> 0.02349496, 0.02444464, 0.02669761, 0.03000917, 0.030…
```

Visualize

```
taxi_tips %>%
  ggplot(aes(x = trip_pickup_date, y = mean_tip_pct)) +
  geom_line() +
  scale_y_continuous(labels = scales::percent) +
  labs(x = "Pick-up date", y = "Mean tip percentage") +
  theme_minimal(base_size = 18)
```



Restrict columns for faster processing

One can restrict which input columns to load into memory for each chunk; this can significantly increase the speed of data processing. To restrict the input columns, use the `srckeep()` function which only accepts column names as a string vector.

With `srckeep()`

```
system.time({
taxi_tips <- taxi_disk %>%
  srckeep(c("trip_pickup_date_time", "tip_amt", "total_amt")) %>%
  mutate(trip_pickup_date = as_date(trip_pickup_date_time),
         tip_pct          = tip_amt / total_amt) %>%
  group_by(trip_pickup_date) %>%
  summarise(mean_tip_pct = mean(tip_pct, na.rm = TRUE)) %>%
  collect() %>%
  as_tibble()
})
```

```
user  system elapsed
1.234   0.187 107.922
```

Without `srckeep()`

```
system.time({
taxi_tips <- taxi_disk %>%
  # srckeep(c("trip_pickup_date_time", "tip_amt", "total_amt")) %>%
  mutate(trip_pickup_date = as_date(trip_pickup_date_time),
         tip_pct          = tip_amt / total_amt) %>%
  group_by(trip_pickup_date) %>%
  summarise(mean_tip_pct = mean(tip_pct, na.rm = TRUE)) %>%
  collect() %>%
  as_tibble()
})
```

```
user  system elapsed
1.698   0.155 205.543
```

Cleaning up

When you are done, `delete()` your `disk.frame`

```
delete(flights_disk)
delete(taxi_disk)
```

Exercise

On the server, copy the capital bikeshare datasets to your home directory with

```
cp -rf cbs_data/ ~/
```

Create a `disk.frame` object using all the CSV files. Check how many rows and variables you have. Finally, create a visualization showing the mean duration bike ride for each station by member type. However, only show the 10 stations with the longest average.

References

1. "Larger-Than-RAM Disk-Based Data Manipulation Framework". Diskframe.Com, 2021, https://diskframe.com/index.html.