REVOLUTION ANALYTICS

RHadoop: Additional Examples

RevolutionAnalytics

December 3, 2013

Outline

Overview

Let's use rmr2 to examine U.S. airline flight performance using publicly available data. Are planes flying more slowly or are airports getting farther away?

- **Objective:** Provide a slightly more complicated example using structured data, our own input formatter, and compound keys
- **What's new:**
  - Dealing with structured data
  - custom input formatter which parses and labels data
  - multi-value (compound) keys and values

airline: about the data

Each month, the US DOT publishes details of the on-time performance (or lack thereof) for every domestic flight in the country The ASA's 2009 Data Expo poster session was based on a cleaned version spanning 1987-2008, and thus was born the famous "airline" data set:

```
Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,U
FlightNum,TailNum,ActualElapsedTime,CRSElapsedTime,AirTime,ArrDelay,Dep
Dest,Distance,TaxiIn,TaxiOut,Cancelled,CancellationCode,Diverted,Carrie
WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay
2004,1,12,1,623,630,901,915,UA,462,N805UA,98,105,80,-14,-7,ORD,CLT,599,
[...]
```

airline: selecting meaningful keys and values

Hadoop may seem an odd choice for such nicely structured data, but it allows the flexibility to pick and choose meaningful identifiers (keys) and data to analyze on a case-by-case basis.

Since Hadoop keys and values needn't be single-valued, let's pull out a few fields from the data: scheduled and actual gate-to-gate times and actual time in the air keyed on year and airport pair

For a given day (3/25/2004) and airport pair (BOS & MIA), here's what the data might look like:

```
2004,3,25,4,1445,1437,1820,1812,AA,399,N275AA,215,215,197,8,8,BOS,MIA,1
2004,3,25,4,728,730,1043,1037,AA,596,N066AA,195,187,170,6,-2,MIA,BOS,125
[...]
```

airline: writing an input formatter

The input formatter is called by `mapreduce()` to parse the input. `rmr2` can parse text, CSV, JSON files out of the box, or you can use `make.input.format()` to pass in your own parameters to `read.table()` or to wrap your own function. For this example, we will pass our own parameters to split by commas and label each column:

```
asa.csv.input.format = make.input.format(format='csv', mode='text', str
sep=',', col.names = c('Year', 'Month', 'DayofMonth', 'DayOfWeek',
            'DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime',
            'UniqueCarrier', 'FlightNum', 'TailNum',
            'ActualElapsedTime', 'CRSElapsedTime', 'AirTime',
            'ArrDelay', 'DepDelay', 'Origin', 'Dest', 'Distance',
            'TaxiIn', 'TaxiOut', 'Cancelled', 'CancellationCode',
            'Diverted', 'CarrierDelay', 'WeatherDelay',
            'NASDelay', 'SecurityDelay', 'LateAircraftDelay'),
           stringsAsFactors=F)
```

airline: input

```
Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,Un
2004,3,25,4,1815,1820,2128,2136,UA,839,N613UA,373,376,337,-8,-5,JFK,LAX
2004,3,25,4,1304,1305,2107,2115,UA,840,N643UA,303,310,277,-8,-1,LAX,JFK
[...]
```

airline: output

```
'data.frame':   45 obs. of  29 variables:
 $ Year              : chr  "Year" "2004" "2004" "2004" ...
 $ Month             : chr  "Month" "3" "3" "3" ...
 $ DayofMonth        : chr  "DayofMonth" "25" "25" "25" ...
 $ DayOfWeek         : chr  "DayOfWeek" "4" "4" "4" ...
 $ DepTime           : chr  "DepTime" "1815" "1304" "1119" ...
 $ CRSDepTime        : chr  "CRSDepTime" "1820" "1305" "1120" ...
 $ ArrTime           : chr  "ArrTime" "2128" "2107" "1931" ...
[...]
```

Note that all lines of the split are included, including headers

airline: mapper

```
mapper.year.market.enroute_time = function(key, val.df) {

    # Remove header lines, cancellations, and diversions:
    val.df = subset(val.df, Year != 'Year' & Cancelled == 0 & Diverted =

    # We don't care about direction of travel, so construct a new 'mark
    # with airports ordered alphabetically (e.g, LAX to JFK becomes 'JF
    market = with( val.df, ifelse(Origin < Dest,
                        paste(Origin, Dest, sep='-'),
                        paste(Dest, Origin, sep='-')) )

    # key consists of year, market
    output.key = data.frame(year=as.numeric(val.df$Year), market=market

[...]
```

airline: mapper

```
[...]

    # emit data.frame of gate-to-gate elapsed times (CRS and actual) +
    output.val = val.df[,c('CRSElapsedTime', 'ActualElapsedTime', 'AirTi
    colnames(output.val) = c('scheduled', 'actual', 'inflight')

    # and finally, make sure they're numeric while we're at it
    output.val = transform(output.val,
                           scheduled = as.numeric(scheduled),
                           actual = as.numeric(actual),
                           inflight = as.numeric(inflight)
    )

    return( keyval(output.key, output.val) )
}
```

airline: mapper input

```
'data.frame':   45 obs. of  29 variables:
 $ Year              : chr  "Year" "2004" "2004" "2004" ...
 $ Month             : chr  "Month" "3" "3" "3" ...
 $ DayofMonth        : chr  "DayofMonth" "25" "25" "25" ...
[...]
 $ Origin            : chr  "Origin" "JFK" "LAX" "LAX" ...
 $ Dest              : chr  "Dest" "LAX" "JFK" "JFK" ...
 $ Cancelled         : chr  "Cancelled" "0" "0" "0" ...
 $ CancellationCode  : chr  "CancellationCode" "" "" "" ...
 $ Diverted          : chr  "Diverted" "0" "0" "0" ...
 $ UniqueCarrier     : chr  "UniqueCarrier" "UA" "UA" "UA" ...
 $ FlightNum         : chr  "FlightNum" "839" "840" "904" ...
 $ TailNum           : chr  "TailNum" "N613UA" "N643UA" "N657UA" ...
 $ ActualElapsedTime: chr  "ActualElapsedTime" "373" "303" "312" ...
 $ CRSElapsedTime    : chr  "CRSElapsedTime" "376" "310" "307" ...
 $ AirTime           : chr  "AirTime" "337" "277" "282" ...
[...]
 - attr(*, "rmr.input")= chr "local/airline/data/20040325-jfk-lax.csv"
```

airline: mapper output

```
> str(key)
'data.frame':   44 obs. of  2 variables:
 $ year  : num  2004 2004 2004 2004 2004 ...
 $ market: chr  "JFK-LAX" "JFK-LAX" "JFK-LAX" "JFK-LAX" ...
> str(val)
'data.frame':   44 obs. of  3 variables:
 $ scheduled: num  376 310 307 303 308 302 303 365 362 362 ...
 $ actual   : num  373 303 312 289 319 297 305 358 364 359 ...
 $ inflight : num  337 277 282 270 281 274 279 331 342 337 ...
```

airline: reducer

For each key, our reducer is called with a list containing all of its values:

```
# the reducer gets all the values for a given key
# the values (which may be multi-valued as here) come in the form of a
reducer.year.market.enroute_time = function(key, val.df) {

    output.key = key
    output.val = data.frame(flights = nrow(val.df),
                            scheduled = mean(val.df$scheduled, na.rm=T),
                            actual = mean(val.df$actual, na.rm=T),
                            inflight = mean(val.df$inflight, na.rm=T) )

    return( keyval(output.key, output.val) )
}
```

airline: reducer

Hadoop has collected all the mapped values for a given key (year & market):

```
> str(key)
'data.frame':   1 obs. of  2 variables:
 $ year  : num 2004
 $ market: chr "JFK-LAX"
> str(val)
'data.frame':   44 obs. of  3 variables:
 $ scheduled: num  376 310 307 303 308 302 303 365 362 362 ...
 $ actual   : num  373 303 312 289 319 297 305 358 364 359 ...
 $ inflight : num  337 277 282 270 281 274 279 331 342 337 ...
```

airline: reducer

Our reducer computes and emits the results:

```
> str(key)
'data.frame':   1 obs. of  2 variables:
 $ year  : num 2004
 $ market: chr "JFK-LAX"
> str(val)
'data.frame':   1 obs. of  4 variables:
 $ flights  : int 44
 $ scheduled: num 338
 $ actual   : num 338
 $ inflight : num 308
```

airline: submit job and fetch results

```
mr.year.market.enroute_time = function (input, output) {
    mapreduce(input = input,
              output = output,
              input.format = asa.csv.input.format,
              map = mapper.year.market.enroute_time,
              reduce = reducer.year.market.enroute_time,
              backend.parameters = list(
                hadoop = list(D = "mapred.reduce.tasks=10")
                ),
              verbose=T)
}
```

airline: submit job and fetch results

```
out = mr.year.market.enroute_time(hdfs.data, hdfs.out)
[...]
results = from.dfs( out )
results.df = as.data.frame(results, stringsAsFactors=F )
colnames(results.df) = c('year', 'market', 'flights', 'scheduled', 'actu

save(results.df, file="out/enroute.time.RData")
```

airline: execution notes

First-time Hadoop users are often surprised to learn (usually the hard way) that regardless of cluster size, the default configuration runs only a single reducer process.

Use `mapreduce`'s `backend.parameters` parameter to tweak such settings:

```
backend.parameters = list(
                                hadoop = list(D = "mapred.reduce.tasks=10")
                                )
```

airline: execution notes

Since this example actually took a while to run, let's save the resulting data.frame as a regular R object, locally:

```
save(results.df, file="out/enroute.time.RData")
```

airline: R can handle the rest

```
> nrow(results.df)
[1] 42612
> yearly.mean = ddply(results.df, c('year'), summarise,
                      scheduled = weighted.mean(scheduled, flights),
                      actual = weighted.mean(actual, flights),
                      inflight = weighted.mean(inflight, flights))
> ggplot(yearly.mean) +
   geom_line(aes(x=year, y=scheduled), color='#CCCC33') +
   geom_line(aes(x=year, y=actual), color='#FF9900') +
   geom_line(aes(x=year, y=inflight), color='#4689cc') + theme_bw() +
   ylim(c(60, 130)) + ylab('minutes')
```

Your turn: repeat analysis by year and airline

Instead of focusing on year and market, let's repeat our analysis by year and airline instead

Your mission: Write a mapper, reducer, and job to calculate the elapsed times by year and airline.

Airline name is stored in the data's "UniqueCarrier" field

Suggested function names:

```
mapper.year.airline.enroute_time()
reducer.year.airline.enroute_time()
mr.year.airline.enroute_time()
```

Use rmr2's local backend and supplied data file to test

Outline

Questions?

- Topics
- Models
- Etc.