Divide and Recombine

A Distributed Data Analysis Paradigm

Ryan Hafen

Workshop on Distributed Computing in R

January 27, 2015
GOAL: DEEP ANALYSIS OF LARGE COMPLEX DATA

» Large complex data has any or all of the following:

  – Large number of records

  – Many variables

  – Complex data structures not readily put into tabular form of cases by variables

  – Intricate patterns and dependencies that require complex models and methods of analysis

  – Not i.i.d.!
What we want to be able to do (with large complex data)

» Work completely in R

» Have access to R’s 1000s of statistical, ML, and vis methods ideally with no need to rewrite scalable versions

» Be able to apply any ad-hoc R code to any type of distributed data object

» Minimize time thinking about code or distributed systems

» Maximize time thinking about the data

» Be able to analyze large complex data with nearly as much flexibility and ease as small data
Divide and Recombine (D&R)

» Simple idea:
  – specify meaningful, persistent divisions of the data
  – analytic or visual methods are applied independently to each subset of the divided data in embarrassingly parallel fashion
  – Results are recombined to yield a statistically valid D&R result for the analytic method

» D&R is not the same as MapReduce (but makes heavy use of it)
Data

Subset
Subset
Subset
Subset

Output
Output
Output
Output

Result

Statistic Recombination
New Data for Analysis Sub-Thread
Analytic Recombination
Visual Displays

Divide
One Analytic Method of Analysis Thread
Recombine

Visual Recombination
How to Divide the Data?

» It depends!
» Random replicate division
  – randomly partition the data
» Conditioning variable division
  – Very often data are “embarrassingly divisible”
  – Break up the data based on the subject matter
  – Example:
    • 25 years of 90 daily financial variables for 100 banks in the U.S.
    • Divide the data by bank
    • Divide the data by year
    • Divide the data by geography
  – This is the major division method used in our own analyses
  – Has already been widely used in statistics, machine learning, and visualization for datasets of all sizes
Analytic Recombination

» Analytic recombination begins with applying an analytic method independently to each subset

  – The beauty of this is that we can use any of the small-data methods we have available (think of the 1000s of methods in R)

» For conditioning-variable division:

  – Typically the recombination depends mostly on the subject matter

  – Example:

    • subsets each with the same model with parameters (e.g. linear model)

    • parameters are modeled as stochastic too: independent draws from a distribution

    • recombination: analysis to build statistical model for the parameters using the subset estimated coefficients
Analytic Recombination

» For random replicate division:
  - Observations are seen as exchangeable, with no conditioning variables considered
  - Division methods are based on statistical matters, not the subject matter as in conditioning-variable division
  - Results are often approximations

» Approaches that fit this paradigm
  - Coefficient averaging
  - Subset likelihood modeling
  - Bag of little bootstraps
  - Consensus MCMC
  - Alternating direction method of multipliers (ADMM)

» Ripe area for research
Visual Recombination

» Data is split into meaningful subsets, usually conditioning on variables of the dataset

» For each subset:
  – A visualization method is applied
  – A set of cognostics, metrics that identify attributes of interest in the subset, are computed

» Recombine visually by sampling, sorting, or filtering subsets based on the cognostics

» Implemented in the trelliscope package
Data structures for D&R

» Must be able to break data down into pieces for independent storage / computation

» Recall the potential for: “Complex data structures not readily put into tabular form of cases by variables”

» **Key-value pairs**
  
  – represented as lists of R objects
  
  – named lists is a possibility but we want keys to be able to have potentially arbitrary data structure too
  
  – instead we can use lists for keys and values

  – can be indexed by, for example, character hash of key
```plaintext
[[1]]
$key
[1] "setosa"

$value
  Sepal.Length Sepal.Width Petal.Length Petal.Width
  1    5.1      3.5      1.4      0.2
  2    4.9      3.0      1.4      0.2
  3    4.7      3.2      1.3      0.2
  4    4.6      3.1      1.5      0.2
  5    5.0      3.6      1.4      0.2
...

[[2]]
$key
[1] "versicolor"

$value
  Sepal.Length Sepal.Width Petal.Length Petal.Width
  51   7.0      3.2      4.7      1.4
  52   6.4      3.2      4.5      1.5
  53   6.9      3.1      4.9      1.5
  54   5.5      2.3      4.0      1.3
  55   6.5      2.8      4.6      1.5
...
```
Distributed data objects (ddo)

- A collection of k/v pairs that constitutes a set of data
- Arbitrary data structure (but same structure across subsets)

> irisDdo

Distributed data object backed by 'kvMemory' connection

| attribute      | value            |
|----------------+--------------------------------------------|
| size (stored)  | 12.67 KB         |
| size (object)  | 12.67 KB         |
| # subsets      | 3                |

* Other attributes: getKeys()
* Missing attributes: splitSizeDistn
Distributed data frames (ddf)

» A distributed data object where the value of each key-value pair is a data frame

» Now we have more meaningful attributes (names, number of rows & columns, summary statistics, etc.)

```
> irisDdf

Distributed data frame backed by 'kvMemory' connection

| attribute      | value                                                                 |
|----------------+-----------------------------------------------------------------------|
| names          | Sepal.Length(num), Sepal.Width(num), and 3 more                          |
| nrow           | 150                                                                   |
| size (stored)  | 12.67 KB                                                               |
| size (object)  | 12.67 KB                                                               |
| # subsets      | 3                                                                     |

* Other attrs: getKeys(), splitSizeDistn(), splitRowDistn(), summary()
```
D&R computation

» MapReduce is sufficient for all D&R operations
  – Everything uses MapReduce under the hood
  – Division, recombination, summaries, etc.
TESSERA
Software for Divide and Recombine

» D&R Interface
  – datadr R package: R implementation of D&R that ties to scalable back ends
  – Trelliscope R package: scalable detailed visualization

» Back-end agnostic design

http://tessera.io
Supported back ends (currently)

- datadr / trelliscope

- Computation:
  - R
  - Multicore R
  - RHIPE / Hadoop

- Storage:
  - Memory
  - Local Disk
  - HDFS

- Small
- Medium
- Large

And more… (like Spark)
What does a candidate back end need?

» MapReduce
» Distributed key-value store
» Fast random access by key
» Ability to broadcast auxiliary data to nodes
» A control mechanism to handle backend-specific settings (Hadoop parameters, etc.)

» To plug in, implement some methods that tie to generic MapReduce and data connection classes
Distributed data types / backend connections

- `localDiskConn()`, `hdfsConn()`, `sparkDataConn()`

    connections to ddo / ddf objects persisted on a backend storage system

- `ddo()`: instantiate a ddo from a backend connection
- `ddf()`: instantiate a ddf from a backend connection

Conversion methods between data stored on different backends
datadr: data operations

» divide(): divide a ddf by conditioning variables or randomly
» recombine(): take the results of a computation applied to a ddo/ddf and combine them in a number of ways
» drLapply(): apply a function to each subset of a ddo/ddf and obtain a new ddo/ddf
» drJoin(): join multiple ddo/ddf objects by key
» drSample(): take a random sample of subsets of a ddo/ddf
» drFilter(): filter out subsets of a ddo/ddf that do not meet a specified criteria
» drSubset(): return a subset data frame of a ddf
» drRead.table() and friends
» mrExec(): run a traditional MapReduce job on a ddo/ddf
maxMap <- expression(
  for(curMapVal in map.values)
    collect("max", max(curMapVal$Petal.Length))
)

maxReduce <- expression(
  pre = {
    globalMax <- NULL
  },
  reduce = {
    globalMax <- max(c(globalMax, unlist(reduce.values)))
  },
  post = {
    collect(reduce.key, globalMax)
  }
)

maxRes <- mrExec(hdfsConn("path_to_data"),
  map = maxMap,
  reduce = maxReduce,
  control =
)
maxMap <- expression(
  for(curMapVal in map.values)
    collect("max", max(curMapVal$Petal.Length))
)

maxReduce <- expression(
  pre = {
    globalMax <- NULL
  },
  reduce = {
    globalMax <- max(c(globalMax, unlist(reduce.values)))
  },
  post = {
    collect(reduce.key, globalMax)
  }
)

maxRes <- mrExec(sparkDataConn("path_to_data"),
  map = maxMap,
  reduce = maxReduce
  control =
)
maxMap <- expression({
  for(curMapVal in map.values)
    collect("max", max(curMapVal$Petal.Length))
})

maxReduce <- expression(
  pre = {
    globalMax <- NULL
  },
  reduce = {
    globalMax <- max(c(globalMax, unlist(reduce.values)))
  },
  post = {
    collect(reduce.key, globalMax)
  }
)

maxRes <- mrExec(localDiskConn("path_to_data"),
  map = maxMap,
  reduce = maxReduce
  control =
)
maxMap <- expression({
    for(curMapVal in map.values)
        collect("max", max(curMapVal$Petal.Length))
})

maxReduce <- expression(
    pre = {
        globalMax <- NULL
    },
    reduce = {
        globalMax <- max(c(globalMax, unlist(reduce.values)))
    },
    post = {
        collect(reduce.key, globalMax)
    }
)

maxRes <- mrExec(data,
    map = maxMap,
    reduce = maxReduce
    control =
    )
datadr: division-independent methods

» drQuantile(): estimate all-data quantiles, optionally by a grouping variable

» drAggregate(): all-data tabulation

» drHexbin(): all-data hexagonal binning aggregation

» summary() method computes numerically stable moments, other summary stats (freq table, range, #NA, etc.)
# divide home price data by county and state
byCounty <- divide(housing,
    by = c("county", "state"), update = TRUE)

# look at at summary statistics for the variables
summary(byCounty)

# compute all-data quantiles of median list price
priceQ <- drQuantile(byCounty, var = "medListPriceSqft")

# apply transformation to each subset
lmCoef <- function(x)
    coef(lm(medListPriceSqft ~ time, data = x))[2]
byCountySlope <- addTransform(byCounty, lmCoef)

# recombine results into a single data frame
countySlopes <- recombine(byCountySlope, combRbind)

# divide another data set with geographic information
geo <- divide(geoCounty, by = c("county", "state"))

# join the subsets of byCounty and geo to get a joined ddo
byCountyGeo <- drJoin(housing = byCounty, geo = geo)
Considerations

» Distributed computing on large data means:
  – distributed data structures
  – multiple machines / disks / processors
  – fault tolerance
  – task scheduling
  – data-local computing
  – passing data around nodes
  – tuning

» Debugging

» Systems
  – Installation, updating packages, Docker,