Tessera: An Environment for the Analysis and Visualization of Large Complex Data

Amanda M. White, Ryan P. Hafen, Lisa M. Bramer, Landon H. Sego, Darren S. Curtis

45th Symposium on the Interface of Computing Science and Statistics
June 12, 2015
Motivation

Our goal in creating Tessera was to

- Enable data scientists to
  - visually explore large datasets, and
  - fit statistical models,
  - with minimal lines of code
- While providing
  - a familiar, interactive, desktop programming environment (R)
  - automatic management of the complicated tasks of distributed storage and computation required for big data
Tessera Fundamentals

- Consists primarily of two R packages:
  - datadr
  - trelliscope
- Open source
- All the power of R is available to you
- Developed by professionals who perform data analysis
Tessera Fundamentals

- Rapid development of code and models
- Excellent flexible statistical visualization capabilities
- Immense collection of statistical routines

- Hides messy details of parallelization
- Takes care of partitioning, scheduling, fault tolerance, data management, and execution
- Parallel programming paradigm (MapReduce) makes sense for many statistical algorithms

http://tessera.io
Tessera Fundamentals: datadr

Data

Subset → Output
Subset → Output
Subset → Output
Subset → Output
Subset → Output

Results
(Statistical Calculation)

New Data for Analysis Sub-Thread
(Analytic Recombination)

Visual Displays
(Visualization Recombination)

Divide

One Analytic Method or Analysis Thread

Recombine
Tessera Fundamentals: trelliscope

- Trelliscope: a viz tool that enables scalable, detailed visualization of large data
- Data is split into meaningful subsets, and a visualization method is applied to each subset
- The user can sort and filter plots based on "cognostics"—summary statistics of interest—to explore the data
Tessera Fundamentals: Connection Types

Regardless of what's underneath, the interface remains the same

**Single workstation**

- **Interface**: datadr / trelliscope
- **Computation**: R
- **Storage**: In-Memory / Local Disk

**Scaling with RHIPE / Hadoop**

- **Interface**: datadr / trelliscope
- **Computation**: RHIPE / Hadoop MapReduce
- **Storage**: HDFS

**In development:**

- **Scaling with distributed memory**
- **Interface**: datadr / trelliscope
- **Computation**: Spark
- **Storage**: RDD (in-memory)
The current Tessera distributed computing stack

- datadr: interface for divide and recombine operations
- RHIPE: The R and Hadoop Integrated Programming Environment
- Hadoop: Framework for managing data and computation distributed across multiple harddrives in a cluster
- HDFS: Hadoop Distributed File System
Tour of http://tessera.io

Tessera

Open source environment for deep analysis of large complex data

The Power of R with Big Data
Apply the thousands of statistical and visualization methods in the R language with simple commands over back ends like Hadoop - without being an expert in distributed computing.

Tessera components >

Get Started in Minutes
Tessera is a powerful computational environment for data large and small. From installation on a single workstation to the Amazon cloud, we've made it easy for you to get started.

Quickstart guide >

Resources to Learn & Join
Once you are up and running, check out our detailed documentation, join our mailing list, browse the code, and learn how to join the open source team!

Resources >
Introduction to datadr
Begin Hands-on Demonstrations

- Two ways to run the demonstrations:
  - Locally if you have already installed the Tessera tools and downloaded the data from [http://tessera.io/docs-csp2015](http://tessera.io/docs-csp2015)
  - Amazon Web Cluster
    - Use Firefox or Chrome (not Internet Explorer or Safari)
    - Security warning: untrusted connection
install.packages("devtools")  # if not installed
library(devtools)
install_github("tesseradata/datadr")
install_github("tesseradata/trelliscope")
install_github("hafen/housingData")  # demo data
Housing Data

- Housing sales and listing data in the United States
- Between 2008-10-01 and 2014-03-01
- Aggregated to the county level
- Zillow.com data provided by Quandl ([https://www.quandl.com/c/housing](https://www.quandl.com/c/housing))
## Housing Data Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fips</td>
<td>Federal Information Processing Standard a 5 digit count code</td>
</tr>
<tr>
<td>county</td>
<td>US county name</td>
</tr>
<tr>
<td>state</td>
<td>US state name</td>
</tr>
<tr>
<td>time</td>
<td>date (the data is aggregated monthly)</td>
</tr>
<tr>
<td>nSold</td>
<td>number sold this month</td>
</tr>
<tr>
<td>medListPriceSqft</td>
<td>median list price per square foot</td>
</tr>
<tr>
<td>medSoldPriceSqft</td>
<td>median sold price per square foot</td>
</tr>
</tbody>
</table>
datadr data representation

- Two main data types are
  - Distributed data frame (ddf):
    - A data frame that is split into chunks
    - Each chunk contains a subset of the rows of the data frame
    - Each subset may be distributed across the nodes of a cluster
  - Distributed data object (dao):
    - Similar to distributed data frame
    - Except each chunk can be an object with any structure
    - Every distributed data frame is also a distributed data object
- Both ddf and dao types use key/value pairs for their structure
Data storage

Data storage options:

- In memory
- Local disk
- HDFS
- Spark (coming soon)
Data ingest

```r
# similar to read.table function:
my.data <- drRead.table(
    hdfsConn("/home/me/dir/datafile.txt",
             header=TRUE, sep="\t")
)

# similar to read.csv function:
my.data2 <- drRead.csv(
    localDiskConn("c:/my/local/data.csv"))

# convert in memory data.frame to ddf:
my.data3 <- ddf(some.data.frame)
```
# Load necessary libraries
library(datadr)
library(trelliscope)
library(housingData)

# housing data frame is in the housingData package
housingDdf <- ddf(housing)
Division

- A common thing to do is to divide a dataset based on the value of one or more variables
- Another option is to divide data into random replicates
  - Use random replicates to estimate a GLM fit by applying GLM to each replicate subset and taking the mean coefficients
  - Random replicates can also be used for a bag of little bootstraps approach
Divide example

Divide the housing data set by the variables "county" and "state"

(This kind of data division is very similar to the functionality provided by the plyr package)

```r
byCounty <- divide(housingDdf,
                   by = c("county", "state"), update = TRUE)
```
Divide example

byCounty

##
## Distributed data frame backed by 'kvMemory' connection
##
## attribute | value
## ----------------+-----------------------------------------------------------
## names          | fips(cha), time(Dat), nSold(num), and 2 more
## nrow           | 224369
## size (stored)  | 15.73 MB
## size (object)  | 15.73 MB
## # subsets      | 2883

## * Other attributes: getKeys(), splitSizeDistn(), splitRowDistn(), summary()
## * Conditioning variables: county, state
Exercise: try the divide function

Now try using the divide statement to divide on one or more variables
Possible solutions

```r
byState <- divide(housing, by="state", update = TRUE)

byMonth <- divide(housing, by="time", update=TRUE)
```
Exploring the ddf data object

Data divisions can be accessed by index or by key name

byCounty[[1]]

## $key
## [1] "county=Abbeville County|state=SC"
##
## $value
##
## fips      time nSold  medListPriceSqft medSoldPriceSqft
## 1 45001  2008-10-01   NA       73.06226                NA
## 2 45001  2008-11-01   NA       70.71429                NA
## 3 45001  2008-12-01   NA       70.71429                NA
## 4 45001  2009-01-01   NA       73.43750                NA
## 5 45001  2009-02-01   NA       78.69565                NA
## ...

byCounty[["county=Benton County|state=WA"]]

http://tessera.io
Exploring the ddf data object

Partipants: try these functions on your own

- `summary(byCounty)`
- `names(byCounty)`
- `length(byCounty)`
- `getKeys(byCounty)`
Transformations

- The `addTransform` function applies a function to each key/value pair in a ddf.
  - E.g. to calculate a summary statistic
- The transformation is not applied immediately, it is deferred until:
  - A function that kicks off a map/reduce job is called (e.g. `recombine`)
  - A subset of the data is requested (e.g. `byCounty[[1]]`)
  - `drPersist` function explicitly forces transformation computation
Transformation example

```r
# Function to calculate a linear model and extract
# the slope parameter
lmCoef <- function(x) {
  coef(lm(medListPriceSqft ~ time, data = x))[2]
}

# Best practice tip: test transformation
# function on one division
lmCoef(byCounty[[1]]$value)

##          time
## -0.0002323686

# Apply the transform function to the ddf
byCountySlope <- addTransform(byCounty, lmCoef)
```

http://tessera.io
Transformation example

byCountySlope[[1]]

## $key
## [1] "county=Abbeville County|state=SC"

## $value
##                  time
##                  -0.0002323686
Exercise: create a transformation function

- Try creating your own transformation function
- Hint: the input to your function will be one value from a key/value pair (e.g. `byCounty[[1]]$value`)

```r
transformFn <- function(x) {
  ## you fill in here
}

# test:
transformFn(byCounty[[1]]$value)

# apply:
xformedData <- addTransform(byCounty, transformFn)
```
Possible solutions

# example 1
totalSold <- function(x) {
  sum(x$nSold, na.rm=TRUE)
}
byCountySold <- addTransform(byCounty, totalSold)

# example 2
timeRange <- function(x) {
  range(x$time)
}
byCountyTime <- addTransform(byCounty, timeRange)
Recombination

- Combine transformation results together again

Example

```r
countySlopes <- recombine(byCountySlope,
                          combine=combRbind)

head(countySlopes)
```

<table>
<thead>
<tr>
<th>time</th>
<th>county</th>
<th>state</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Abbeville County</td>
<td>SC</td>
<td>-0.0002323686</td>
</tr>
<tr>
<td>time1</td>
<td>Acadia Parish</td>
<td>LA</td>
<td>0.0019518441</td>
</tr>
<tr>
<td>time2</td>
<td>Accomack County</td>
<td>VA</td>
<td>-0.0092717711</td>
</tr>
<tr>
<td>time3</td>
<td>Ada County</td>
<td>ID</td>
<td>-0.0030197554</td>
</tr>
<tr>
<td>time4</td>
<td>Adair County</td>
<td>IA</td>
<td>-0.0308381951</td>
</tr>
<tr>
<td>time5</td>
<td>Adair County</td>
<td>KY</td>
<td>0.0034399585</td>
</tr>
</tbody>
</table>
Recombination options

`combine` parameter controls the form of the result

- `combine=combRbind`: `rbind` is used to combine results into `data.frame`, this is the most frequently used option
- `combine=collect`: results are collected into a list
- `combine=combDdo`: results are combined into a `ddo` object
Exercise: try the `recombine` function

- Apply `recombine` to the data with your custom transformation
- Hint: `combine=combrbind` is probably the simplest option
Exercise: divide

Divide two new datasets `geoCounty` and `wikiCounty` by county and state

```r
# look at the data first
head(geoCounty)
head(wikiCounty)

# use divide function on each
```
Solution

geoByCounty <- divide(geoCounty,
    by=c("county", "state"))

wikiByCounty <- divide(wikiCounty,
    by=c("county", "state"))
Data operations: drJoin

Join together multiple data objects based on key

```r
joinedData <- drJoin(housing=byCounty,
                     slope=byCountySlope,
                     geo=geoByCounty,
                     wiki=wikiByCounty)
```
Distributed data objects vs distributed data frames

- In a ddf the value in each key/value is always a data.frame
- A ddo can accommodate values that are not data.frames

```r
class(joinedData)
```

```r
## [1] "ddo"       "kvMemory"
```
Distributed data objects vs distributed data frames

joinedData[[176]]

## $key
## [1] "county=Benton County|state=WA"

## $value
## $housing
##     fips       time nSold medListPriceSqft medSoldPriceSqft
## 1  53005 2008-10-01   137         106.6351         106.2179
## 2  53005 2008-11-01    80         106.9650               NA
## 3  53005 2008-11-01    NA               NA         105.2370
## 4  53005 2008-12-01    95         107.6642         105.6311
## 5  53005 2009-01-01    73         107.6868         105.8892
## 6  53005 2009-02-01    97         108.3566               NA
## 7  53005 2009-02-01    NA               NA         104.3273
## 8  53005 2009-03-01   125         107.1968         103.2748
## 9  53005 2009-04-01   147         107.7649         102.2363
## 10 53005 2009-05-01   192         108.6823               NA
## 11 53005 2009-05-01    NA               NA         103.8925
## 12 53005 2009-06-01   256         108.5143         105.1873
Data operations: drFilter

Filter a ddf or ddo based on key and/or value

```r
# Note that a few county/state combinations do
# not have housing sales data:
names(joinedData[[2884]]$value)

## [1] "geo"  "wiki"

# We want to filter those out those
joinedData <- drFilter(joinedData,
  function(k,v) {
    !is.null(v$housing)
  })
```
Other data operations

- `drSample`: returns a ddo containing a random sample (i.e. a specified fraction) of key/value pairs
- `drSubset`: applies a subsetting function to the rows of a ddf
- `drLapply`: applies a function to each subset and returns the results in a ddo
Exercise: `datadr data operations`

Apply one or more of these data operations to `joinedData` or a `ddo` or `ddf` you created

- `drJoin`
- `drFilter`
- `drSample`
- `drSubset`
- `drLapply`
Using Tessera with a Hadoop cluster

Differences from in memory computation:

- Data ingest: use `hdfsConn` to specify a file location to read in HDFS
- Each data object is stored in HDFS
  - Use `output` parameter in most functions to specify a location in HDFS to store data

```r
housing <- drRead.csv(
  file=hdfsConn("/hdfs/data/location"),
  output=hdfsConn("/hdfs/data/second/location"))

byCounty <- divide(housing, by=c("state", "county"),
  output=hdfsConn("/hdfs/data/byCounty"))
```
Introduction to trelliscope
Trelliscope

- Divide and recombine visualization tool
- Based on Trellis display
- Apply a visualization method to each subset of a $d_1f$ or $d_2o$
- Interactively sort and filter plots
Trelliscope panel function

- Define a function to apply to each subset that creates a plot
- Plots can be created using base R graphics, ggplot, lattice, rbokeh, conceptually any htmlwidget

```r
# Plot medListPriceSqft and medSoldPriceSqft by time
timePanel <- function(x) {
  xyplot(medListPriceSqft + medSoldPriceSqft ~ time,
         data = x$housing, auto.key = TRUE,
         ylab = "Price / Sq. Ft.")
}
```
# Best practice tip: test the panel function on a single subset

timePanel(joinedData[[176]]$value)
Visualization database (vdb)

- Trelliscope creates a directory with all the data to render the plots
- Can later re-launch the Trelliscope display without all the prior data analysis

vdbConn("housing_vdb", autoYes=TRUE)
Creating a Trelliscope display

```r
makeDisplay(joinedData,
    name = "list_sold_vs_time_datadr",
    desc = "List and sold price over time",
    panelFn = timePanel,
    width = 400, height = 400,
    lims = list(x = "same")
)
```

## * Validating 'panelFn'...
## * Testing cognostics function on a subset ... ok
## * Precomputed limits not supplied. Computing axis limits...
## Testing 'prepanelFn' on a subset...
## Using 'trellis' panelFn to determine limits...
## At least one of the variables is not numeric. Casting as numeric for quantile calculations...
## * Storing display object...
## * Plotting thumbnail...
## * Updating displayList...
## * Display exists... backing up previous to /Users/d3l348/Files/CVS/Tessera/docs-UseR2015
## * Removing previous backup plot directory

view()
Trelliscope demo
Exercise: create a panel function

```r
newPanelFn <- function(x) {
  # fill in here
}

# test the panel function
timePanel(joinedData[[1]]$value)

vdbConn("housing_vdb", autoYes=TRUE)

makeDisplay(joinedData,
  name = "panel_test",
  desc = "Your test panel function",
  panelFn = newPanelFn)
```
Cognostics and display organization

- Cognostic:
  - a value or summary statistic
  - calculated on each subset
  - to help the user focus their attention on plots of interest
- Cognostics are used to sort and filter plots in Trelliscope
- Define a function to apply to each subset to calculate desired values
  - Return a list of named elements
  - Each list element is a single value (no vectors or complex data objects)
Cognostics function

```r
priceCog <- function(x) {
    st <- getSplitVar(x, "state")
    ct <- getSplitVar(x, "county")
    zillowString <- gsub(" ", "-", paste(ct, st))
    list(
        slope = cog(x$slope, desc = "list price slope"),
        meanList = cogMean(x$housing$medListPriceSqft),
        meanSold = cogMean(x$housing$medSoldPriceSqft),
        lat = cog(x$geo$lat, desc = "county latitude"),
        lon = cog(x$geo$lon, desc = "county longitude"),
        wikiHref = cogHref(x$wiki$href, desc="wiki link"),
        zillowHref = cogHref(
            sprintf("http://www.zillow.com/homes/%s_rbg/",
                zillowString),
            desc="zillow link")
    )
}
```
Use the cognostics function in trelliscope

# Best practice tip: test the cognostics function on a single subset
priceCog(joinedData[[176]]$value)

makeDisplay(joinedData,
    name = "list_sold_vs_time_datadr2",
    desc = "List and sold price with cognostics",
    panelFn = timePanel,
    cogFn = priceCog,
    width = 400, height = 400,
    lims = list(x = "same")
)
Trelliscope demo

Cognostics View / Sort / Filter

View cognostics in a table and specify sort order or filtering of panels.
Shift-click on the panel header sorting buttons for multi-column sorting.

<table>
<thead>
<tr>
<th>county</th>
<th>state</th>
<th>slope</th>
<th>meanList</th>
<th>meanSold</th>
<th>nObs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbeville County</td>
<td>SC</td>
<td>-0.0002323686</td>
<td>72.76927</td>
<td>NaN</td>
<td>66</td>
</tr>
<tr>
<td>Acadia Parish</td>
<td>LA</td>
<td>0.0019518411</td>
<td>68.11082</td>
<td>NaN</td>
<td>66</td>
</tr>
<tr>
<td>Accomack County</td>
<td>VA</td>
<td>-0.0092717711</td>
<td>123.79215</td>
<td>NaN</td>
<td>66</td>
</tr>
<tr>
<td>Ada County</td>
<td>ID</td>
<td>-0.0038197554</td>
<td>100.83173</td>
<td>NaN</td>
<td>66</td>
</tr>
<tr>
<td>Adair County</td>
<td>IA</td>
<td>-0.0008381951</td>
<td>65.79008</td>
<td>NaN</td>
<td>64</td>
</tr>
<tr>
<td>Adair County</td>
<td>KY</td>
<td>0.0034399585</td>
<td>68.54309</td>
<td>NaN</td>
<td>61</td>
</tr>
<tr>
<td>Adair County</td>
<td>MO</td>
<td>0.000980550</td>
<td>69.83997</td>
<td>NaN</td>
<td>66</td>
</tr>
<tr>
<td>Adair County</td>
<td>OK</td>
<td>-0.0048414082</td>
<td>69.12551</td>
<td>NaN</td>
<td>66</td>
</tr>
<tr>
<td>Adams County</td>
<td>CO</td>
<td>0.0219787578</td>
<td>113.26757</td>
<td>121.6475</td>
<td>66</td>
</tr>
<tr>
<td>Adams County</td>
<td>ID</td>
<td>-0.0362570575</td>
<td>112.44364</td>
<td>NaN</td>
<td>66</td>
</tr>
</tbody>
</table>

Showing entries 1 - 10 of 2883
**Exercise: create a cognostics function**

```r
newCogFn <- function(x) {
  #     list(
  #       name1=cog(value1, desc="description")
  #   )
}

# test the cognostics function
newCogFn(joinedData[[1]]$value)

makeDisplay(joinedData,
            name = "cognostics_test",
            desc = "Test panel and cognostics function",
            panelFn = newPaneFn,
            cogFn = newCogFn)
view()
```
Next session

Please return for Part 2 of our demo after the refreshment break

Three hands-on demos to learn about datadr and trelliscope
Tessera: Session 2

Amanda M. White, Ryan P. Hafen, Lisa M. Bramer, Landon H. Sego, Darren S. Curtis

45th Symposium on the Interface of Computing Science and Statistics
June 12, 2015
Three Hands-On Examples

- Power usage in retail buildings
  - Smallest data
  - Power consumption and outdoor air temperatures of 4 retail buildings in the U.S. during 2010, measured at 15 min intervals
- Housing sales data
  - Small data
- Network traffic data
  - Bigger data - uses Hadoop
  - A simulated dataset of computer network traffic
Hands-On Examples: Do it your way

- You may explore trelliscope views only
  - The first few lines of code in each demo opens the pre-configured trelliscope display of the data
- You may run and explore the code that creates the trelliscope display
- You may modify the code to create your own trelliscope display
  - Try making a different type of plot
  - Try creating new cognostics
Two ways to run the examples

- Locally on your own computer
  - Provided you have previously installed Tessera and the examples from http://tessera.io/docs-csp2015
- Amazon Web Service Cluster
  - URL to be provided
  - There are 2 clusters, each cluster has
    - 1 master node with 8 CPU-cores and 30 GB RAM
    - 3 compute nodes with 4 CPU-cores and 15 GB RAM

Download Demo Materials

The Tessera team will be presenting an interactive demo at the 2013 Conference on Statistical Practice in New Orleans on February 21, 2013.

Participants who would like to download the Tessera tools prior to the demo please follow these instructions. These tools will allow you to use and test Tessera on your own computer without Hadoop or a similar parallel processing backend.

1. If you do not already have the most recent version of R, version 3.1.2, please download and install it here: http://cran.rproject.org.
2. Optional: You may find the RStudio development environment an easier way to program in R, but it is not necessary. You may download it here: http://www.rstudio.com/.
3. You will need a non-Internet Explorer browser installed on your computer. Firefox, Safari, or Chrome will all work fine.
4. Open R and execute the following commands to install Tessera and other libraries you may need for the demos:

   ```r
   library(devtools)
   install_github("tessera/tessera")
   install_github("tessera/tessera-unit")
   install_github("tessera/tessera-utils")
   ```

For Windows users: when installing devtools, you may notice the following warning, which you can ignore:

```
warning: devtools 1.8.0 is required to build R packages, but no version of Rtools compatible with R 3.1 is available.
```

5. Download the CSP Tessera demo files and unzip them on your computer:

   ```
   Tessera_demo_CSP2013.zip
   ```

6. The zip file contains a folder called `demo`. Set your working directory in R to this folder, using something like `setwd("tessera/demo")`. The `demo` folder contains a folder for each of the three demos: `power-demos`, `netflow-demos`, and `housing-demos`. Each demonstration folder has a single `.R` file which contains the code for the demonstration. Open that file in your editor of choice and begin.

We look forward to seeing you at CSP!
Location of Demonstrations

- In each demonstration folder you'll see a single .R file
- Open that file and execute the contents (line by line) to begin the demo
- Read the code comments for information about the commands
Wrap-Up

- Tessera is available at http://tessera.io,
- Our goal is to build an open source community of users and contributors. Please post bugs or requests for new features:
  - datadr: https://github.com/tesseradata/datadr/issues
  - trelliscope: https://github.com/tesseradata/trelliscope/issues
- Subscribe to the mailing list by sending an email to tessera-users+subscribe@googlegroups.com
- Contact information:
  - Ryan Hafen rhafen@gmail.com
  - Amanda White amanda.white@pnnl.gov