RHIPE

Experiences from Analyzing Large Data using R, Hadoop and MapReduce

Saptarshi Guha, Mozilla, 2015
MapReduce Programming Model

● transform input data to a key and value pair
● aggregate values belonging to same key
● distributed versions of R's Map, Filter, and Reduce
z <- rhwatch(map=function(a,b){

}, reduce=rhoptions()$templates$X
, input=rhfmt()
[, output=rhfmt([]
[, shared=]
[, debug=],[[setup=]]

Cluster Setup

- Hadoop Cluster with HDFS, or DFS e.g. LustreFS on which home folders are mounted.
- Need basic systems background to install across nodes
  - so many users have installation woes: Hadoop and R(shared libs, see *)
  - Cloudera
  - Stock Apache
  - Horton
  - MR v1 vs v2
  - RHEL Linux and others ...
Cluster Setup

- might have access to cluster wide accessible database e.g. HBase/Postgres etc.
  - how to dynamically share the running environment across nodes?
- nodes might be ‘bare’ or have software installed
  - how do R users install packages?
  - [*] do i need to distribute R to the nodes?

Cannot guarantee software environment which implies a plugin designed architecture
Paradigms used by RHIPE users

- Process data on disk
  - Often: transform to data.table/data.frames or small list structures
  - RHIPE has a limited random access by keys feature. Store the data using different keys
  - Compute across objects stored on disk
- Run Simulations: cpu hungry rather than data
- Common to all: all nodes need to access resources
Paradigms used by RHIPE users

\[
\text{rhwatch}(\text{map=}, \text{reduce=\text{rhoptions}()}\cdot\text{templates}()\cdot\text{X},..)
\]

where X can be an expression or pre-defined ones:

- \text{scalarsummer}: adding counts
- \text{colsummer}: adding vectors
- \text{range}: computing max/min
- \text{rbinder}(): for binding data.tables/frames. Used for creating data table objects for a key e.g. longitudinal history
- \text{raggregate}(F): aggregates values into a big list, F is then applied to it
Paradigms used by RHIPE users

● Some paradigms (e.g. summing) do not need an understanding of MapReduce/Hadoop etc.
  ○ Introduce user to rhwatch, the concept of grouping by key and adding values.
  ○ A lot of RHIPE codes could be expressed in SQL, yet cognitively coding with in the MR key-value paradigm is easier
● Others, e.g. rbinder do require some knowledge e.g. encourage small value sizes etc.
Paradigms used by RHIPE users

- Quantiles by covariates (so useful)
- Summary statistics by covariates
- Optimization frameworks (though iterations are slow, could be alleviated using YARN and Spark’s Tachyon)
- Compute models across subsets of data (e.g. Firefox profiles)

Encourage users to program with templates. More thought for data problem than programming problem.

Controls the paradigms.
Paradigms used by RHIPE users

- Simulations (more cpu bound as opposed to size of input data bound) are as straight forward as
  \[ \text{rhwatch(map=F, input=c(N1,N2),...)} \]
- N1 is the # of trials, N2 is the number of tasks to split it across
  - user manually balances cost of F when choosing N1, N2
- Has a built in parallel RNG (user can provide a seed)
Live Within R

- Submit Query
  
z <- rhwatch(map=function(a,b){ rhcollect(X,1) }, reduce=summer, input=X)
  
  ○ code detects references functions and variables, copies them automatically
- Progress can be watched, or run asynchronously
- Results returned within R
- Errors returned in R
- Data can be optionally streamed into R (or use MapFiles for random access)

To the extent possible, stay within R, expose little of Hadoop.
Debugging

- Errors are typically the same but multiple nodes will return the same error.
- Need to use regex to extract unique types of errors
  - don’t confuse user with the facts of a distributed environment
- RHIPE 0.75 master now saves the call trace of errors for ‘in place’ debugging
  - to the extent possible: replicate a single node environment
What Hadoop Gives Us

1. One infrastructure: 5 analysts are sharing the cluster with other programmers.
   a. scheduling algorithms could be more nuanced
2. Failure recovery.
3. Resource Sharing e.g. cpu limits, time limits etc.
4. It scales: yes it might be slow, but almost always completes.
5. Trivial but useful: A measure of progress. User can see how much of computation is over.
6. With YARN, a framework for diverse paradigms (beyond MapReduce) e.g. BSP.
What Works

● Once installation is a success …. :)  
● … and one accepts that key-values should be ‘small’  
● Things just work within R and RHIPE (almost)  
● We want the user to freely compute with all the data (even if it be 40GB sample)  
  ○ with the comfort of analyzing small data  
● Almost seamlessly move problem to AWS clusters for arbitrary cluster sizes  

but
What we want

- Better installation
- Interrupt computation without killing it
- Agnostic to installed software environment
- Better defaults: # of maps (for simulations, # of task), # of reducers (dynamically allocate more reducers depending on size of data being sent)
- APIs that dont overpromise (e.g. APIs that are misleadingly intuitive ..)
- Sometimes just need faster code (not distributed) - better extension language till R becomes faster e.g. Rcpp, rterra (Terra extensions)

and
Despite All This ...  

- A lot of data analysis/statistics involves understanding the data through exploration  
  - RHIPE is *very* good for handling this but less so for running ML algorithms on *entire* data  
- Yet, we are constrained by the tools/methodology we use  
  - how to study tens of thousands of subsets across many covariates?  
  - how to store and navigate the visualizations?  
  - how to cognitively absorb?  
- Instead of a deep analysis of data we  
  - resort to best possible predictive models without a deeper understanding of the data.  
  - though on ‘big’ data this works well because we get a huge number of cases in nearly every possible subspace of the data.