A Herd of Unicorns

David Smith

R/Finance 2010, Chicago

Huge Data

Huge Data is big.

Really big.

You might think your tick database is big, but that's peanuts compared to Huge Data

With apologies to Douglas Adams

The Unicorn

Perform statistical analyses on Huge Data

in an environment that promotes exploration, visualization, transformation and evaluation

?

REvolution computing®

R and Huge Data : Problem #1

- Capacity
  - Base R is memory bound.   All data needs to be in memory in for R to process it.
  - Number of rows
  - Number of columns
  - Model parameters
    - Factor interactions

R and Huge Data sets : Problem #2

- Performance
  - Standard algorithms in R are run as a single process
    - They don't take advantage of multi-core architectures
    - They can't be distributed among nodes on a cluster
    - Explicit parallel programming helps … maybe
  - Standard file structures used with R are not efficient for accessing rows and columns in arbitrarily large data sets.

Breaking Capacity and Performance Barriers

- There are many open-source initiatives in R that address either the capacity or performance issues
  - http://cran.r-project.org/web/views/HighPerformanceComputing.html
  - There is no complete framework that solves **both** the capacity and performance issues
  - Programming and/or set-up costs can be high to implement

REvolution: Top down approach

- Design to solve both the Capacity and Performance constraints
  - Low cost of implementation
  - Allow for highly efficient access to data and for performing data transformations
  - Implement the most commonly used statistical algorithms for huge data sets
  - Provide a programming framework for extending and customizing data access and algorithm development for use in R

REvolution computing®

Fundamental Design Concepts for Huge Data: 1

- High performance data access
  - Disk I/O is a bottleneck
  - Efficient access to arbitrary selection of rows and columns
  - Block access to sequential rows of data using only the specified columns
  - External memory data management routines (sorts, merges, variable creation, etc.)
  - Smart I/O limits what has to be written to disk when appending rows or columns.

Fundamental Design Concepts for Huge Data: 2

- Parallel External Memory Algorithms
  - Take advantage of modern hardware (multi-core architectures, and multi-node networks)
  - RAM is a bottleneck: Do not require all data to be in memory
  - CPU is a bottleneck: Process parts of data independently, and combine results
  - Parallelism in inherit in most external memory algorithms

REvoAnalytics Package

- Collection of high performance analysis routines
  - Package of R functions
    - Data access (SAS, ODBC, flat files, etc.)
    - Data selection / transformation
    - Summarization / aggregation
    - Statistical models
    - Visualization
  - C++ Engine
  - Extensibility framework

Data Storage Considerations

- Relational
- Row-based
- Column-based (NoSQL)
- Distributed file systems

XDF File Format

- Compact binary format for N×p data
  - Fast throughput from disk to memory
  - Supports single bytes through 64-bit floats
- Data stored in blocks of rows by column
  - O(1): seek to specified data blocks (rows)
  - O(N): retrieve one column (independent of p!)
  - Add rows/columns or modify cells without rewriting entire file
- Independent metadata for columns
  - Factor levels

Data / Statistical Algorithms

- Operate directly from XDF file
  - Block updating, parallelized, distributed
- Handle factors efficiently
  - Sparse-matrix operations
- Allow multiple models to be computed simultaneously
  - Reuse algebraic terms
- Support in-line variable transformations
  - via model formula, R engine

REvolution computing®

Example: Mortgage Default

- Build a logistic regression model to predict mortgage defaults based on historical data
- Four benchmark scenarios:
  1. revoAnalytics' `rxLogit` logistic regression function
  2. `bigglm`, on in-memory data frame
  3. logistic regression performed by `bigglm` from the CRAN `biglm` package, block-updating
  4. `glm`, on in-memory data frame

REvolution computing®

Mortgage Default Model

- Mortgage default data for years 2000 to 2009
- 10 files of 1,000,000 records each
- Aggregate file:

  10,000,000 rows

  6 columns, 250Mb

Sample of Data

Record Number	Credit Score	House Age	Years Employed	Credit Card Debt	Debt Year	Default
1	615	10	5	2818	2000	0
2	780	34	5	3575	2000	0
3	735	12	1	3184	2000	0
4	713	15	5	6236	2000	1
5	689	10	5	6817	2000	0

Model

default ~ creditScore + yearsEmploy + ccDebt + year

Scenario 1: code

```
require(revoAnalytics)

ageLevels <- as.character(0:41)
yearLevels <- as.character(2000:2009)
colInfo=list(
  list(name="houseAge", type="factor", levels=ageLevels),
  list(name="year", type="factor", levels=yearLevels))

append= FALSE
for (i in 2000:2009)
{
    importFile <- paste("annualdata", i, ".csv", sep="")
    rxTextToXdf(importFile, "mortDefault", colInfo=colInfo,
        append=append)
    if (i == 2000) append=TRUE
}

rxLogit(default~creditScore + yearsEmploy + ccDebt + year,
    data="mortDefault"))
```

REvolution computing®

Scenario 2 code

```
# Use bigglm from the biglm package
# on in-memory data frame
library(biglm)

mydf <- rxReadData(data=dataFileName)
bigglm(default ~ creditScore + yearsEmploy +
                               ccDebt + year,
  data=mydf,
  family=binomial(,
  chunksize=20000,
  maxit=20,
  tolerance=1e-6))
```

REvolution computing®

Scenario 3 code

```
# explicitly block-updating bigglm
library(biglm)

dataHandle <- rxOpenData(data="mortData", blocksPerRead=15)
ff <- default~creditScore + yearsEmploy + ccDebt + year

df <- rxReadNext(dataHandle)
a <- bigglm(ff, df,chunksize=20000,maxit=10,
                   tolerance=1e-6,family=binomial())

moreData <- TRUE
while (moreData) {
  df <- rxReadNext(dataHandle)
  if (length(df) != 0) {
    a <- update(a, df)
  } else
    moreData <- FALSE
}

rxCloseData (dataHandle)
```

Scenario 4 code

```
# In-memory glm

mydf <- rxReadData(data=dataFileName)
glm(default ~ creditScore + yearsEmploy +
                             ccDebt + year,
  data=mydf,
  family=binomial(),
  maxit=20,
  tolerance=1e-6))
```

Benchmark Results*

Scenario	Description	Elapsed Time (seconds)
1	revoAnalytics rxLogit function applied to a .XDF formatted file	79.98
2	bigglm from biglm package applied to a data frame in memory	599.56
3	bigglm, block-updating method	677.85
4	glm applied to in-memory data frame	N/A

* Benchmark machine: Lenovo ThinkPad with 2 cores running at 2.5 GHz with 3 GB of RAM

REvolution computing®

Distributed Computing

- All algorithms designed to work in distributed mode
- Issue: Data access at nodes
  - Network reads to XDF file
  - Duplication on hard drives
- Distributed file systems
  - Hadoop DFS integration?
  - Metadata system means stat models can be fit on partial files

Thank you!

david@revolution-computing.com
http://blog.revolution-computing.com
@revodavid

End