Programming with Scalding

Krishnan Raman

Data Scientist, Twitter

http://github.com/krishnanraman/bigdata

What is Scalding ?

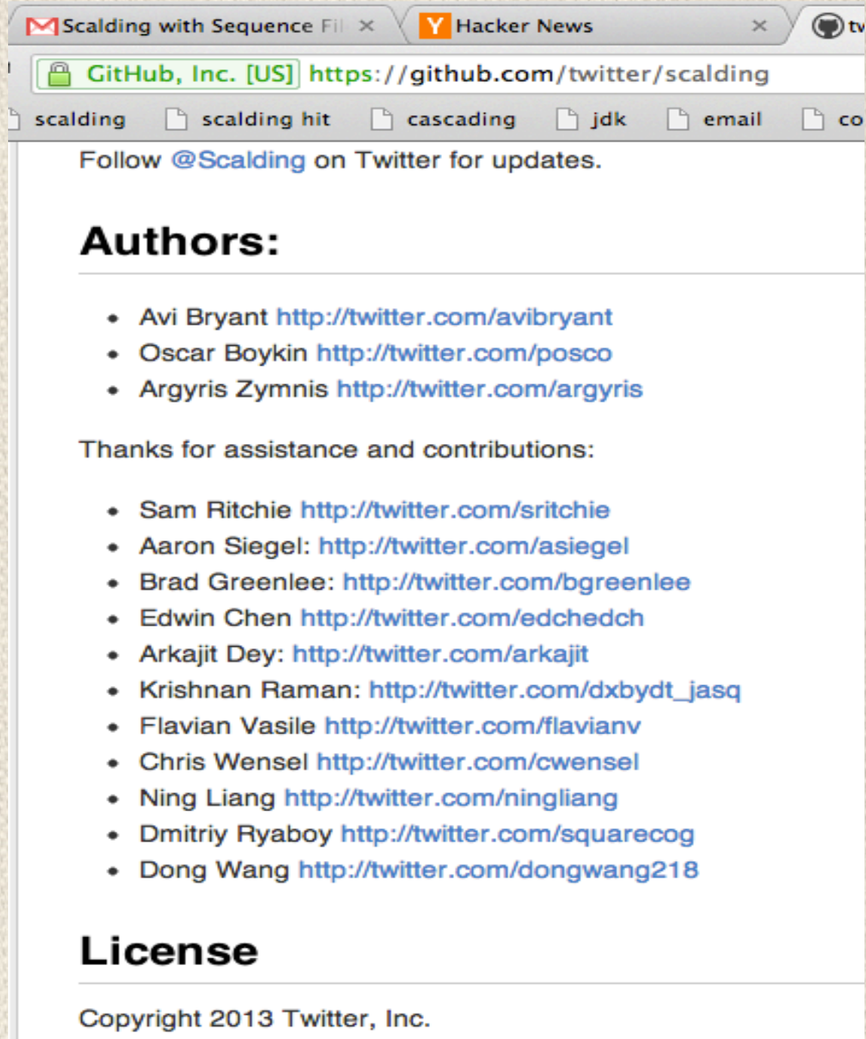
- Primary Usecase: Hadoop Map-Reduce Jobs in Scala
- Abstract away low-level Hadoop + Cascading details (but can still get to them …)
- Identical functions as Scala – map, flatMap, filter, project,takeWhile, groupBy….
- Work well with CT primitives – Twitter's Bijection libraries (Monoids, Groups, Rings, …)

Powered By

New Page | Edit Page | Page History

Want to be added to this page? Send a tweet to @scalding or open an issue.

Company	Scalding Use Case	Code
Twitter	We use Scalding often, for everything from custom ad targeting algorithms, market insight, click prediction, traffic quality to PageRank on the Twitter graph. We hope you will use it too!	-
Etsy	We're starting to use Scalding alongside the JRuby Cascading stack described here. More to come as we use it further.	-
eBay	We use Scalding in our Search organization for ad-hoc data analysis jobs as well as more mature data pipelines that feed our production systems.	-
Gatling	We've just rebuilt our reports generation module on top of Scalding. Handy API on top of an efficient engine.	GitHub
Sonar	Our platform is built on Hadoop, Scalding, Cassandra and Storm. See Sonar's job listings.	-



ssql : Gentle Intro to Scalding

```
open employees file1.txt
columns 3
column 1 name text
column 2 age number
column 3 income decimal
rows select age > 20
rows select age < 27
column add newincome income*1.1 + (age-20)*100
column remove age
column remove income
save file2.txt
```

```
fred 25 10000
sam 25 150000
oscar   39  200000
baker 27  234567
pascal 24 123456
ford 22 55000
haskell 56 45466
```

```
fred      11500.0
sam       165500.0
pascal  136201.6
ford      60700.0
```

ssql : Gentle Intro to Scalding

```
open employees file1.txt
columns 3
column 1 name text
column 2 age number
column 3 income decimal
rows select age > 20
rows select age < 27
column add newincome income*1.1 + (age-20)*100
column remove age
column remove income
save file2.txt
```

somescript

ssql

somescript.scala

```
import com.twitter.scalding._
class somescript(args : Args) extends Job(args){
  val employees =
    TextLine("file1.txt")
    .read
    .mapTo('line -> ('name,'age,'income) ){
    line:String =>
    val res = line.split(" ").map( _.trim).filter(_.length > 0)
    val name:String = res(0).toString
    val age:Int = res(1).toInt
    val income:Double = res(2).toDouble
    (name,age,income)
  }.filter('age) {
    age:Int =>
    (age > 20)
  }.filter('age) {
    age:Int =>
    (age < 27)
  }.map(('income,'age) -> ('newincome)) {
    columns:(Double,Int)=>
    val (income,age) = columns
    income*1.1 + (age-20)*100
  }.discard('age)
    .discard('income)
    .write(Tsv("file2.txt"))
}
```

```scala
import com.twitter.scalding._
class somescript(args : Args) extends Job(args){
  val employees =
    TextLine("file1.txt")
    .read
    .mapTo('line -> ('name,'age,'income) ){
    line:String =>
    val res = line.split(" ").map( _.trim).filter(_.length > 0)
    val name:String = res(0).toString
    val age:Int = res(1).toInt
    val income:Double = res(2).toDouble
    (name,age,income)
  }.filter('age) {
    age:Int =>
    (age > 20)
  }.filter('age) {
    age:Int =>
    (age < 27)
  }.map(('income,'age) -> ('newincome)) {
    columns:(Double,Int)=>
    val (income,age) = columns
    income*1.1 + (age-20)*100
  }.discard('age)
    .discard('income)
    .write(Tsv("file2.txt"))
}
```

Portfolio Management with Scalding

Divide $1000 among 4 stocks –

1. Kroger, $27
2. Abbott Labs, $64
3. Dollar Tree, $41
4. Monster Beverage, $52

in the “best” possible way.

These are 4 stocks with a low beta (risk) and high average volume.

Solve: $27x + 64y + 41z + 52w = 1000$

```scala
import com.twitter.scalding._
import com.twitter.scalding.mathematics.Combinatorics

class Portfolios(args : Args) extends Job(args) {

    val cash = 1000.0 // money at hand
    val error = 1 // its ok if we cannot invest the last dollar
    val (kr,abt,dltr,mnst) = (27.0,64.0,41.0,52.0) // share prices
    val stocks = IndexedSeq( kr,abt,dltr,mnst)

    Combinatorics.weightedSum( stocks, cash,error).write( Tsv("invest.txt"))
}
```

```scala
import cern.colt.matrix.{DoubleFactory2D, DoubleFactory1D }
import cern.colt.matrix.linalg.Algebra
import java.util.StringTokenizer

object BestPortfolio {
    def main(args:Array[String]) = {
      val alg = Algebra.DEFAULT

      // define the correlation matrix
      val data = Array(Array(0.448, 0.177, 0.0, 0.017),
        Array(0.177, 0.393, 0.177, 0.237),
        Array(0.0, 0.177, 0.237, 0.06),
        Array(0.017, 0.237, 0.06, 0.19))
      val corr = DoubleFactory2D.dense.make(data)
      val file = scala.io.Source.fromFile("invest.txt").getLines.toList

      // convert the tab-delimited weights in the file to a row vector
      def getWeights(s:String) = {
          val weights = s.split("\t").map( x=> x.toDouble)
          DoubleFactory1D.dense.make(weights)
      }
      // compute risks per tuple and sort by risk
      file.map(line=> {
          val w = getWeights(line)
          ( line, alg.mult( alg.mult(corr,w), w))
      }).sortBy(x=>x._2)
      .foreach( println )
  }
}
```

weighted sums...

```
$ cat invest.txt
0 0 13  9
0 1 0 18
0 1 19  3
0 2 1 16
0 2 20  1
0 3 7 10
0 4 8 8
0 5 9 6
0 6 15  0
0 8 3 7
0 9 4 5
0 10  5 3
0 14  0 2
1 0 6 14
1 1 12  8
1 2 13  6
1 3 0 15
1 3 14  4
1 4 1 13
1 5 2 11
1 6 8 5
1 7 9 3
1 8 10  1
1 11  4 2
1 12  5 0
2 0 18  4
2 1 5 13
2 1 19  2
2 2 6 11
2 3 7 9
2 4 13  3
2 5 14  1
2 6 1 10
2 7 2 8
2 8 3 6
2 9 9 0
3 0 11  9
3 1 12  7
3 2 18  1
3 3 0 14
3 4 6 8
3 5 7 6
3 6 8 4
3 9 2 5
3 10  3 3
3 11  4 1
4 0 4 14
```

```
4 2 11  6
4 3 12  4
4 4 13  2
4 5 0 11
4 6 1 9
4 7 7 3
4 8 8 1
4 12  3 0
5 0 16  4
5 1 17  2
5 2 4 11
5 3 5 9
5 4 6 7
5 5 12  1
5 7 0 8
5 8 1 6
5 9 2 4
6 0 9 9
6 1 10  7
6 2 11  5
6 5 5 6
6 6 6 4
6 7 7 2
6 10  1 3
6 11  2 1
7 0 2 14
7 0 16  3
7 1 3 12
7 2 4 10
7 3 10  4
7 4 11  2
7 7 0 7
7 8 6 1
8 0 9 8
8 1 15  2
8 2 16  0
8 3 3 9
8 4 4 7
8 5 5 5
8 9 0 4
8 10  1 2
9 0 2 13
9 1 8 7
9 2 9 5
9 3 10  3
9 6 4 4
9 7 5 2
9 11  0 1
```

```
10  1 15  1
10  2 2 10
10  3 3 8
10  4 9 2
10  5 10  0
11  0 7 8
11  1 8 6
11  2 14  0
11  4 2 7
11  5 3 5
11  6 4 3
12  0 0 13
12  1 1 11
12  2 7 5
12  3 8 3
12  4 9 1
12  7 3 2
12  8 4 0
13  0 12  3
13  1 13  1
13  2 0 10
13  3 1 8
13  4 2 6
13  5 8 0
14  0 5 8
14  1 6 6
14  2 7 4
14  5 1 5
14  6 2 3
14  7 3 1
15  0 12  2
15  2 0 9
15  3 6 3
15  4 7 1
15  8 2 0
16  0 5 7
16  1 11  1
16  4 0 6
16  5 1 4
17  1 4 6
17  2 5 4
17  3 6 2
17  6 0 3
17  7 1 1
18  0 10  2
18  4 5 1
19  0 3 7
19  1 4 5
```

```
19  6 0 2
20  2 3 4
20  3 4 2
21  0 8 2
21  1 9 0
22  0 1 7
22  1 2 5
22  2 3 3
23  0 8 1
23  3 2 2
23  4 3 0
24  0 1 6
24  1 7 0
25  1 0 5
25  2 1 3
25  3 2 1
26  0 6 1
26  4 1 0
27  1 0 4
28  3 0 1
29  0 4 1
34  0 2 0
37  0 0 0
```

So it looks like we have 169 unique ways to divvy up the cash.
Lets take one of the 169 tuples. How about (5,4,6,7) ?
So 5 shares of Kroger @ $27 = $135
4 shares of Abbott Labs @ $64 = $256
6 shares of Dollar Tree @ $41 = $246
7 shares of Monster @ $52 = $468

135+256+246+468 = $1001

But is (5,4,6,7) the best-posible tuple among the 169 choices ?

Correlations among equities…

4x4 correlation matrix

KR	ABT	DLTR	MNST
0.448	0.177	0	0.017
0.177	0.393	0.177	0.237
0	0.177	0.237	0.06
0.017	0.237	0.06	0.19

Given a vector of weights W, the net risk imposed by this matrix = W'CW
Goal: Want the 4-tuple with the least risk.

The 4-tuple with the least risk…

```
// compute risks per tuple and sort by risk
file.map(line=> {
    val w = getWeights(line)
    ( line, alg.mult( alg.mult(corr,w), w))
}).sortBy(x=>x._2)
```

```
res11: List[(String, Double)] = List(
(1  0 6 14,56.776),
(4  0 4 14,56.824),
(2  1 5 13,57.544),
(4 1 5 12,58.552),
(2  2 6 11,59.646),
.....
(37 0 0 0,613.312)
```

Under the hood...weightedSum(..)

```
1  def weightedSum( weights:IndexedSeq[Double], result:Double, error:Double)(implicit flowDef:FlowDef):Pipe = {
2      val numWeights = weights.size
3      val allColumns = (1 to numWeights).map( x=> Symbol("k"+x))
4
5      // create as many single-column pipes as the number of weights
6      val pipes = allColumns.zip(weights).map( x=> {
7        val (name,wt) = x
8        IterableSource( (0.0 to result by wt), name).read
9      }).zip( allColumns )
10
11     val first = pipes.head
12     val accum = (first._1, List[Symbol](first._2))
13     val rest = pipes.tail
14
15     val res = rest.foldLeft(accum)((a,b)=>{
16
17       val (apipe, aname) = a
18       val (bpipe, bname) = b
19       val allc = (List(aname)).flatten ++ List[Symbol](bname)
20       // Cross two pipes, Create a temp column that stores intermediate results
21       // Apply progressive filtering on the temp column
22       // Once all pipes are crossed, test for temp column within error bounds of result
23       ( apipe.crossWithSmaller(bpipe)
24         .map(allc->'temp){
25           x:TupleEntry =>
26           val values = (0 until allc.size).map( i=> x.getDouble( i.asInstanceOf[java.lang.Integer]))
27           values.sum
28         }.filter('temp){
29           x:Double => if( allc.size == numWeights) (math.abs(x-result)<= error) else (x <= result)
30         }.discard('temp), allc )
31     })._1.unique(allColumns)
32
33     (1 to numWeights).zip(weights).foldLeft( res) ((a,b) => {
34         val (num,wt) = b
35         val myname = Symbol("k"+num)
36         a.map( myname->myname){ x:Int => (x/wt).toInt }
37     })
38   }
```

US Population Data Analysis

Population Data Dictionary

Population Data Dictionary - 1969-2011

File Format

- Fixed length ASCII text records (28 bytes)
- One population per record/line
- Windows (CR/LF) line delimiters
- All numeric data is zero filled to the left

Variable Name and Values	Start Column	Length	Data Type
Year (1969, 1970, 1971...)	1	4	numeric
State postal abbreviation "KR" is used for the dummy state created to represent hurricane Katrina/Rita evacuees	5	2	character
State FIPS code Field is 9-filled for dummy state created to represent hurricane Katrina/Rita evacuees	7	2	numeric
County FIPS code Field is 9-filled for dummy state created to represent hurricane Katrina/Rita evacuees	9	3	numeric
Registry 01 = San Francisco-Oakland SMSA 02 = Connecticut 20 = Detroit (Metropolitan) 21 = Hawaii 22 = Iowa 23 = New Mexico 25 = Seattle (Puget Sound) 26 = Utah 27 = Atlanta (Metropolitan) 29 = Alaska Natives	12	2	numeric

Population data – 12 million records

```
1  1969AL01001991910000000159
2  1969AL01001991910100000657
3  1969AL01001991910200001137
4  1969AL01001991910300000956
5  1969AL01001991910400000721
6  1969AL01001991910500000424
7  1969AL01001991910600000585
8  1969AL01001991910700000637
9  1969AL01001991910800000607
10 1969AL01001991910900000523
11 1969AL01001991911000000466
12 1969AL01001991911100000386
13 1969AL01001991911200000357
14 1969AL01001991911300000268
15 1969AL01001991911400000179
16 1969AL01001991911500000126
17 1969AL01001991911600000076
18 1969AL01001991911700000049
19 1969AL01001991911800000031
20 1969AL01001991920000000151
21 1969AL01001991920100000566
22 1969AL01001991920200001037
23 1969AL01001991920300000979
24 1969AL01001991920400000683
25 1969AL01001991920500000557
26 1969AL01001991920600000705
27 1969AL01001991920700000669
28 1969AL01001991920800000614
29 1969AL01001991920900000496
30 1969AL01001991921000000395
31 1969AL01001991921100000372
32 1969AL01001991921200000377
33 1969AL01001991921300000320
```

US FIPS Codes.csv (3000+ records)

```
Alabama,Autauga,01,001
Alabama,Baldwin,01,003
Alabama,Barbour,01,005
Alabama,Bibb,01,007
Alabama,Blount,01,009
Alabama,Bullock,01,011
Alabama,Butler,01,013
Alabama,Calhoun,01,015
Alabama,Chambers,01,017
Alabama,Cherokee,01,019
Alabama,Chilton,01,021
Alabama,Choctaw,01,023
Alabama,Clarke,01,025
Alabama,Clay,01,027
Alabama,Cleburne,01,029
Alabama,Coffee,01,031
Alabama,Colbert,01,033
Alabama,Conecuh,01,035
Alabama,Coosa,01,037
Alabama,Covington,01,039
Alabama,Crenshaw,01,041
Alabama,Cullman,01,043
Alabama,Dale,01,045
Alabama,Dallas,01,047
Alabama,De Kalb,01,049
Alabama,Elmore,01,051
Alabama,Escambia,01,053
Alabama,Etowah,01,055
Alabama,Fayette,01,057
Alabama,Franklin,01,059
Alabama,Geneva,01,061
Alabama,Greene,01,063
```

Convert pop.txt to a legit Scalding Source

```
44  case class USPopulationSource(override val p:String) extends TextLine(p) {
45    override def transformForRead(pipe : Pipe) = {
46      import Dsl._
47      RichPipe(pipe).mapTo('line->( 'year,'state,'fips,'isWhite,'isBlack,
48                                    'isHispanic,'isMale,'age,'population)) {
49        record:String =>
50        val year:Int                = record.slice(0,0+4).toInt
51        val state:String            = record.slice(4,4+2)
52        val fips:String             = record.slice(6,6+5)
53        val isWhite:Boolean         = record.slice(13,13+1).toInt == 1
54        val isBlack:Boolean         = record.slice(13,13+1).toInt == 2
55        val isHispanic:Boolean      = record.slice(14,14+1).toInt == 1
56        val isMale:Boolean          = record.slice(15,15+1).toInt == 1
57        val age:Int                 = 5*(record.slice(16,16+2).toInt -1)
58        val population:Int          = record.slice(18,18+8).toInt
59
60        (year,state,fips,isWhite,isBlack, isHispanic,isMale,age,population)
61      }
62    }
63  }
```

```
65  Find THE FASTEST GROWING COUNTY IN THE UNITED STATES over the 1969-2011 timeframe
66
67  class PopulationStats(args:Args) extends Job(args) {
68    val people = USPopulationSource("pop.txt").read
69    val fipspipe = TextLine("US_FIPS_Codes.csv").read.mapTo('line->('state,'county,'fips)) {
70      line:String =>
71      var arr = line.split(",")
72      (arr(0),arr(1),(arr(2)+arr(3)))
73    }
74    people.groupBy('year, 'fips){
75        group => group.plus[Int]('population->'population)
76    }.groupBy('fips) {
77       val init = (0,0.0d)
78       type X = (Int,Double)
79       type T = (Int,Int)
80
81       // foldLeft[X,T](fieldDef : (Fields,Fields))(init : X)(fn : (X,T) => X)
82        group => group.foldLeft[X,T]( ('population,'year) -> ('dummy,'growth))(init:X) {
83
84          (x:X, t:T) =>
85            val (population,year) = t
86            val (dummy, growth ) = x
87            year match {
88            case 1969 => (population, 0.0d)
89            case 2011 => (population,(population-dummy)/(dummy+0.0d))
90            case _ => if (dummy==0) (population,0.0d) else (dummy,0.0d)
91        }
92      }
93    }.project('fips,'growth)
94    .joinWithSmaller(('fips-> 'fips), fipspipe)
95    .groupAll(_.sortBy('growth))
96    .write(Tsv("growth.txt"))
97  }
```

```
RESULTS:
13053  -0.5959766162310867      Georgia Chattahoochee
38083   -0.5825892857142857     North Dakota    Sheridan
38013   -0.575839766325892      North Dakota    Burke
54047   -0.5703608502224419     West Virginia   McDowell
48101   -0.5474033816425121     Texas   Cottle
38047   -0.5436781609195402     North Dakota    Logan
38023   -0.5362287210824968     North Dakota    Divide
38087   -0.5334632878492528     North Dakota    Slope
38037   -0.5310054184226369     North Dakota    Grant
48301   -0.5276381909547738     Texas   Loving

---- some 3000 counties here ----

13117  10.637015231025215       Georgia Forsyth
12097   10.880533448053345      Florida Osceola
48397   10.907133440749963      Texas   Rockwall
13135   11.063892016788289      Georgia Gwinnett
48157   11.168507788849015      Texas   Fort Bend
48491   11.323119312014695      Texas   Williamson
08117   11.39344262295082       Colorado        Summit
48085   11.507137247655564      Texas   Collin
12035   21.135939986360537      Florida Flagler
08035   36.49576488706366       Colorado        Douglas

CONCLUSION:
Over the 1969-2011 timeframe, Douglas County experienced the highest population growth ( 3600% )
The top-10 counties with the most pop growth are shown above - they are in Texas, Florida & Colorado.
```

```
Goal: You are a male between the age of (20,40).
Where should you go to maximize your chances of hooking up with the opposite sex ?

Scalding code: sexratio(2011,(20,40),people,fipspipe)

where sexratio as below:

def sexratio(baseYear:Int, agerange:(Int,Int), people:Pipe, fipspipe:Pipe) = {
    people.filter('year,'age) {
    yearage:(Int,Int) =>
    val (year,age) = yearage
    (year == baseYear) && (age>=agerange._1) && (age<=agerange._2) // marriageable
  }.groupBy('fips, 'isMale){
      group => group.plus[Int]('population->'population)
  }.groupBy('fips) {
    val init = 0.0d
    type X = Double
    type T = (Int,Boolean)
    // foldLeft[X,T](fieldDef : (Fields,Fields))(init : X)(fn : (X,T) => X)
    group => group.foldLeft[X,T]( ('population,'isMale) -> ('sexratio))(init:X) {
        (x:X, t:T) =>
        val ratio = x
        val (population,isMale) = t
        if(ratio==0.0) (population+0.0d)
        else {
         val sexratio = ratio/(population+0.0d)
         if (isMale) 1.0/sexratio else sexratio
        }
    }
  }.joinWithSmaller(('fips-> 'fips), fipspipe)
  .project('sexratio, 'state,'county)
  .groupAll(_.sortBy('sexratio))
  .write(Tsv("sexratio"+baseYear+".txt"))
}
```

```
SORTED RESULTS:
------------
SEX RATIO (M/F), State, County
0.5809725158562368  Georgia     Pulaski
0.6842327150084317      West Virginia   Summers
0.6924882629107981      Wyoming Niobrara
0.7     Missouri        Audrain
0.7070063694267515      Indiana Parke
0.7194008037997808      Missouri        Livingston
0.7434895833333334      Virginia        Fluvanna
0.764525993883792       North Dakota    Hettinger
0.7655172413793103      Texas   Oldham
0.7692307692307692      Texas   Loving
0.7831548198636806      Ohio    Union

....3000 more counties here .....

3.4205729166666665  Georgia     Wheeler
3.56390977443609        Colorado        Bent
4.472380952380952       Pennsylvania    Forest
4.957894736842105       Texas   Concho
5.52020202020202        Colorado        Crowley
------------

TL:DR;
Pulaski County, Georgia has the most advantageous sex ratio of 0.58 ( Male/Female in the (20,40) agegroup)
~ Two women for every man!
```

```
134   Goal: Race Analysis: find the county with the highest ratio of whites
135   to the total population. Do the same for African Americans.
136
137   /*
138     Want: white people in an agegroup
139   */
140   race(2011,(20,40),people,fipspipe,true)
141
142   /*
143     Want: black people in an agegroup
144   */
145   race(2011,(20,40),people,fipspipe,false)
146
147
148 def race(baseYear:Int, agerange:(Int,Int), people:Pipe, fipspipe:Pipe, white:Boolean) = {
149     val filename = (if (white) "white" else "black")+baseYear+".txt"
150
151     // first get total population ( ie. all races ) of agegroup for baseYear
152     val pipe1 = people.filter('year,'age){
153       yearage:(Int,Int) =>
154       val (year,age) = yearage
155       (year == baseYear) && (age>=agerange._1) && (age<=agerange._2)
156     }.groupBy('isMale,'fips){
157       // add (fe)male blacks, (fe)male hispanics, (fe)male whites etc.
158       group => group.plus[Int]('population->'population)
159     }.groupBy('fips){
160       group => group.plus[Int]('population->'population) // add males + females
161     }
```

```
163     // now count people of certain race in certain age group
164     val pipe2 = people.filter('year,'age, 'isWhite,'isBlack) {
165     yearagewb:(Int,Int,Boolean,Boolean) =>
166     val (year,age,isWhite,isBlack) = yearagewb
167     (year == baseYear) && (age>=agerange._1) && (age<=agerange._2) &&
168     (if (white) isWhite else isBlack)
169     }.groupBy('isMale,'fips){
170       // add (fe)male blacks, (fe)male hispanics, (fe)male whites etc.
171       group => group.plus[Int]('population->'population)
172     }.groupBy('fips){
173       group => group.plus[Int]('population->'mypop) // add males + females
174     }
175
176     // now compute ratio & sort by ratio
177     val pipe3 = pipe1.joinWithSmaller(('fips-> 'fips), pipe2)
178     .map( ('mypop, 'population)->'raceratio){
179       mypop_total:(Int,Int) =>
180       val( mypop, population) = mypop_total
181       mypop/(population+0.0d)
182     }
183     .project('population, 'raceratio,'fips)
184
185     // now join with verbose fips
186     pipe3.joinWithSmaller(('fips-> 'fips), fipspipe)
187     .project('raceratio,'population, 'state,'county)
188     .groupAll(_.sortBy('raceratio))
189     .write(Tsv(filename))
190   }
```

```
Black:
.....

0.7854832914121451  2783  Mississippi Humphreys
0.7856462369006033  3149  Alabama Wilcox
0.7889369592088998  3236  Alabama Lowndes
0.7913015254787407  3081  Georgia Hancock
0.794999045619393 10478 Virginia  Petersburg City
0.799015275849009 7921  Mississippi Coahoma
0.8423519386450788  2347  Alabama Greene
0.8453242514545196  7047  Alabama Macon
0.87166636835511  5587  Mississippi Holmes
0.8917171717171717  2475  Mississippi Jefferson

White
.....

0.9939713639788997  2654  West Virginia Clay
0.9940298507462687  670 Nebraska  Frontier
0.9946686976389947  1313  Tennessee Pickett
0.9950510361892979  6466  West Virginia Lincoln
0.9955357142857143  672 Nebraska  Sherman
0.9961240310077519  1032  Nebraska  Chase
0.9964850615114236  569 Nebraska  Greeley
1.0 93  Montana Petroleum
1.0 400 Nebraska  Garfield
1.0 155 Nebraska  Grant
1.0 152 Nebraska  Keya Paha
1.0 116 Nebraska  Loup


TL:DR; Racial makeup of youth in the 20-40 agegroup is:

100% white, in several counties of Nebraska - Loup, Keya Paha, Grant, Garfoeld etc.
In general, Nebraska has an overwhelming white population ( >99%) in several counties.

The racial makeup of Jefferson county in Mississippi is 90% African American.
In general, Mississippi & Alabama dominate AA rankings as seen above.
```

Algebird…Twitter's Abstract Algebra library

- Model a wide class of "reductions" as a sum on some iterator
- Average, moving average, max/min, set union, approximate set size (in much less memory with HyperLogLog), approximate item counting (using CountMinSketch).
- Implementations of Monoids for interesting approximation algorithms, such as Bloom filter, HyperLogLog and CountMinSketch.
- Use CMS, HLL within Storm (realtime aggregation) and Scalding (batchmode aggregation

Trivial Monoid: Employee Monoid

```
257  USECASE: Find the oldest (max age) employee with max tenure making the least salary
258
259  case class Monoid(e:Set[Employee]) {
260    def plus(a:Employee,b:Employee)= {
261    if(a.age > b.age) a else
262    if(b.age > a.age) b else
263    if(a.tenure > b.tenure) a else
264    if(b.tenure > a.tenure) b else
265    if (a.salary > b.salary) b else
266    if(b.salary > a.salary) a else
267    if( math.random > 0.5) a else b
268  }
269    val identity = Employee(0,0,0,"id")
270    val theUnluckyOne = e.foldLeft(identity)((a,b)=>plus(a,b))
271  }
272
273  scala> Monoid(empDB.toSet).theUnluckyOne
274  res2: Employee = Employee(35,6,85079,emp7435)
```

Nontrivial example: Abelian Group

```
234  trait Abelian
235
236  case class Zn(order:Int, zero:Int) extends Abelian {
237      def identity = zero
238      def size = order
239      def elements =  (1 to order).toSeq
240      def cayley = {
241          Vector.tabulate(order,order)((x,y)=> {
242              val idx = math.abs(zero - (x+1)) // difference between x & zero
243              val timesToShift = if ((x+1)>=zero) idx else (order-idx)
244              val row = shift(elements, timesToShift )
245              row(y)
246          })
247      }
248      private def shift(s:Seq[Int], pos:Int) = {
249          val n = 1+s.indexOf(pos)
250          s.slice(n,s.size)++s.slice(0,n)
251      }
252
253      def plus(x:Int, y:Int) = cayley(x-1)(y-1)
254      def inverse(x:Int) = cayley(x-1).indexOf(zero)+1
255  }
```

CMS Monoid for counting tweets

```scala
import com.twitter.algebird.{CountMinSketchMonoid, CMS}
object CMSTest {
  // sane CMS builder
  def mkCMS( probability:Byte = 90, EPS:Double = 0.01 ) = {
    assert( probability >0 && probability < 100 )
    val DELTA = 1 - probability/100.0
    val SEED = 1
    new CountMinSketchMonoid(EPS, DELTA, SEED)
  }

  def updateCMS( monoid:CountMinSketchMonoid, cms:Option[CMS], x:Long ):Option[CMS] = {
    val item = monoid.create(x)
    if( cms.isDefined )  Some(item ++ cms.get) else Some(item)
  }

  def main(args:Array[String]) = {

      // make items from 1 to n, assume k has been seen n-k times
      val n = 100
      val stream = List.tabulate[List[Long]](n)( k => {
        val freq = n - k
        List.fill[Long](freq)( k )
      }).flatten

      // assume this data comes in one at a time from some stream
      // update the CMS as the data arrives
      val monoid = mkCMS( args(0).toByte, args(1).toDouble )
      val cmsitem:Option[CMS] = None
      val result = stream.foldLeft(monoid, cmsitem)( (a,b) => {
        val (mymonoid, myitem) = a
        val number = b
        val newitem = updateCMS( mymonoid, myitem, number )
        val retval = (mymonoid, newitem)
        retval
      })._2.get

      // compare estimated frequencies vs actuals
      val estimate = result.frequency ( 34 ).estimate
      val actual = n - 34
      println( "Estimate: %d, Actual : %d" + estimate, actual)
  }
}
```

More goodies…

- https://github.com/twitter/scalding
- https://github.com/twitter/algebird
- https://github.com/twitter/bijections

Thank you! (Krishnan Raman, kraman@twitter.com)