Flyby: Improved Dense Matrix Multiplication (& other tasks)

Tom Vacek

Thomson Reuters R&D

Outline

- Matching use case
- 3 Algorithms
- Evaluation

Matching application

$$\begin{array}{cc} & \textbf{movies} \\ & \begin{bmatrix} m_1 & m_2 & m_3 & \dots \end{bmatrix} \\ \textbf{users} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \end{bmatrix} & \begin{bmatrix} f(m_1, u_1) & \dots \\ & \\ \vdots & \ddots \end{bmatrix} \end{array}$$

users cartesian movies map…combineByKey

cartesian matrix multiply

$$\left[\begin{array}{c} a'_1 \\ \hline a'_2 \\ \hline \vdots \end{array}\right] \left[\begin{array}{c|c|c|c} b_1 & b_2 & b_3 & \dots \end{array}\right] = \left[\begin{array}{c|c|c|c} a'_1 b_1 & a'_1 b_2 & a'_1 b_3 & \dots \\ \hline a'_2 b_1 & a'_2 b_2 & a'_2 b_3 & \dots \\ \hline \vdots & \vdots & \vdots & \ddots \end{array}\right]$$

aRows cartesian(bCols) map (a , b) => a*b

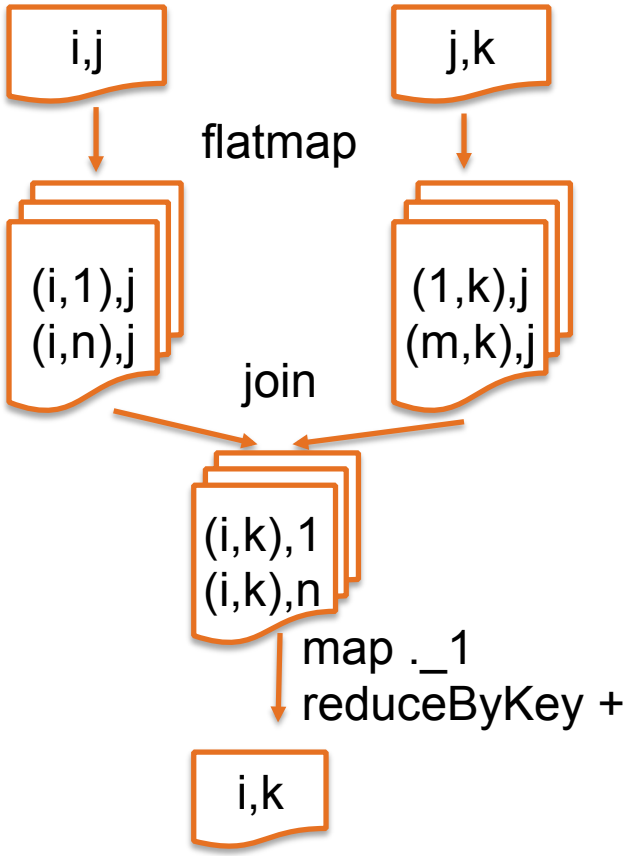
shuffleMultiply

- Local reduceByKey
- Optimal efficiency:

  $O(n^{1.5})$ workers

  each do $O(n^{1.5})$ work

  $O(n^{1.25})$ wire



left flyby right

$$
\begin{array}{cc|ccc}
 & & \multicolumn{3}{c}{\textbf{right}} \\
 & & r_1 & r_2 & r_3 \\
 & & \downarrow & \downarrow & \downarrow \\
\textbf{left} & l_1 \longrightarrow & f_0(l_1, r_1) & f_0(l_1, r_2) & f_0(l_1, r_3) \\
 & & \downarrow & \downarrow & \downarrow \\
 & l_2 \longrightarrow & f(l_2, r_1, \cdot) & f(l_2, r_2, \cdot) & f(l_2, r_3, \cdot) \\
 & & \downarrow & \downarrow & \downarrow \\
 & l_3 \longrightarrow & f(l_3, r_1, \cdot) & f(l_3, r_2, \cdot) & f(l_3, r_3, \cdot)
\end{array}
$$

Flyby matrix multiply

$$\begin{bmatrix} b_1 & b_2 & b_3 & \dots \end{bmatrix}$$

$$\begin{bmatrix} a'_1 \\ \hline a'_2 \\ \hline a'_3 \end{bmatrix} \longrightarrow \begin{bmatrix} a'_1 b_1 & a'_1 b_2 & a'_1 b_3 & \dots \end{bmatrix}$$

Flyby matrix multiply

$$\begin{bmatrix} b_1 & b_2 & b_3 & \dots \end{bmatrix}$$

$$\begin{bmatrix} a'_1 \\ \hline a'_2 \\ \hline a'_3 \end{bmatrix} \longrightarrow \begin{bmatrix} a'_1 b_1 & a'_1 b_2 & a'_1 b_3 & \dots \\ a'_2 b_1 & a'_2 b_2 & a'_2 b_3 & \dots \end{bmatrix}$$

Flyby implementation (hack)

```
def flyby[L,R,O] (left: RDD[L], right:RDD[R], fold0: (L,R)=>O, foldFun: (L,R,O)=>O))={
  var reduction:RDD[O] = null
  val leftLocal = getPartitionsAsLocalArray(left)
  for(part <- leftLocal) {
    val flyer = right.sparkContext broadcast (part)
      val nextReduction = if (reduction == null) {
        right.map { case rr =>
          val init = fold0(flyer.value.head, rr)
          flyer.value.tail.foldLeft(init) { case (acc, nextflyer) =>
          foldFun(nextflyer, rr, acc)} }
      } else {
        right.zip(reduction).map { case (rr, red) =>
          flyer.value.foldLeft(red) { case (acc, nextflyer) =>
           foldFun(nextflyer, rr, acc)} }
      }.persist
      nextReduction.count
      flyer.unpersist (false)
      reduction.unpersist(false)
      reduction = nextReduction
```

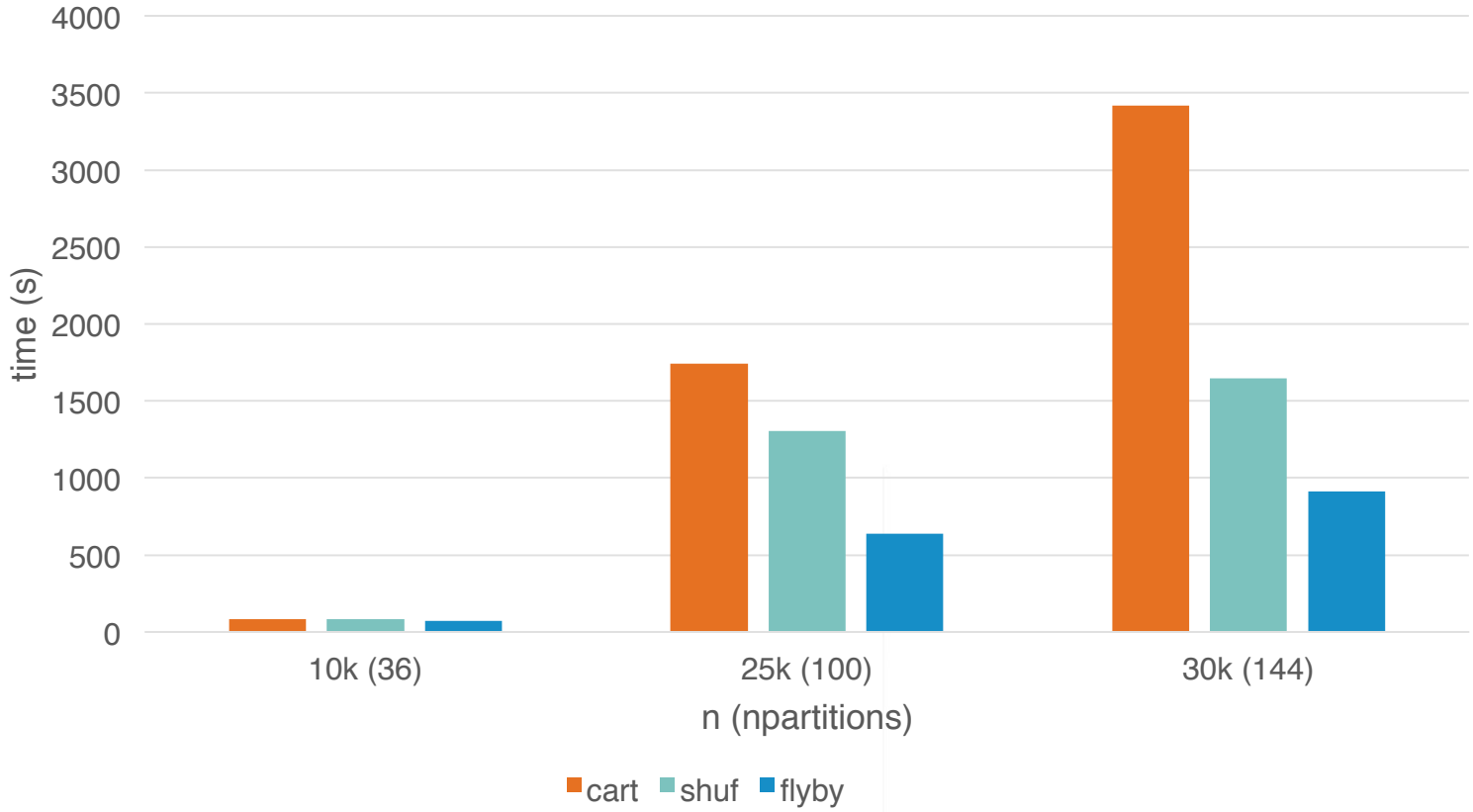
Matrix Experiments

- Implemented dist dense matrix class using Breeze
- 5 Nodes, quad socket Xeon E5-2690 (32 cores), 250GB. Cloudera 5.3 (Spark 1.2.0)
- Evaluate n×n matrices, npartitions
- Code: bitbucket.org/twvacek/spark-flyby

MLLib

- BlockMatrix introduced to mllib at version 1.3.0 (13 Mar 2015)
- Not available at time this work was done.
- Our matrix class very similar.
- Mllib multiply ~= shuffleMultiply

Results


4000
3500
3000
2500
2000
1500
1000
500
0
time (s)
10k (36)
25k (100)
30k (144)
n (npartitions)
cart
shuf
flyby


Spark
summit 2015

Discussion

- Flyby: signif improvement over cartesian. Gain for matching applications.
- Improvement over shuffle probably from memory caching. Asymptotics?
- FlybyRDD ?
- Code: bitbucket.org/twvacek/spark-flyby