Distributed Machine Learning and Graph Processing with Sparse Matrices

Shivaram Venkataraman*, Erik Bodzsar#
Indrajit Roy*, Alvin AuYoung*, Rob Schreiber*

*UC Berkeley, #U Chicago, +HP Labs
Big Data, Complex Algorithms

PageRank (Dominant eigenvector)
Recommendations

Machine learning + Graph algorithms

Anomaly detection (Top-K eigenvalues)
User Importance (Vertex Centrality)
Large-Scale Processing Frameworks

Data-parallel frameworks — MapReduce/Dryad (2004)
- Process each *record* in parallel
- Use case: Computing sufficient statistics, analytics queries

Graph-centric frameworks — Pregel/GraphLab (2010)
- Process each *vertex* in parallel
- Use case: Graphical models

Array-based frameworks — MadLINQ (2012)
- Process *blocks* of array in parallel
- Use case: Linear Algebra Operations
PageRank using Matrices

Simplified algorithm repeat { p = M*p }

Linear Algebra Operations on Sparse Matrices

Power Method
Dominant eigenvector

M = web graph matrix
p = PageRank vector
Presto

Large-scale machine learning and graph processing on sparse matrices

Extend R — make it scalable, distributed
Challenge 1 – Sparse Matrices
Challenge 1 – Sparse Matrices

Block density (normalized)

Block ID

LiveJournal, Netflix, ClueWeb-1B

1000x more data → Computation imbalance
Challenge 2 – Data Sharing

Sparse matrices $\rightarrow$ Communication overhead

Sharing data through pipes/network

**Time-inefficient** (sending copies)  
**Space-inefficient** (extra copies)
Outline

- Motivation
- Programming model
- Design
- Applications and Results
darray
foreach $f(x)$
M ← darray(dim=c(N,N),blocks=(s,N))
P ← darray(dim=c(N,1),blocks=(s,1))

while(..){
    foreach(i,1:len,
        calculate(m=splits(M,i),
            x=splits(P), p=splits(P,i)) {
            p ← m*x
        }
    )
}
PageRank Using Presto

\[ M \leftarrow \text{darray}(\text{dim}=c(N,N),\text{blocks}=(s,N)) \]
\[ P \leftarrow \text{darray}(\text{dim}=c(N,1),\text{blocks}=(s,1)) \]

while(\ldots){
    \textbf{foreach}(i,1:len,}
    \text{calculate}(m=\text{splits}(M,i),
                  x=\text{splits}(P), p=\text{splits}(P,i)) \{ \\
      p \leftarrow m \times x
    \}
    \}
}

Execute function in a cluster
Pass array partitions
Presto Architecture

Master

Worker

R instance

DRAM

Worker

R instance

DRAM
Repartitioning Matrices

Profile execution

Repartition

Partition if \[ \frac{\text{max}(t)}{\text{median}(t)} > \delta \]
Maintaining Size Invariants

\texttt{invariant(mat, vec, type=ROW)}
Sharing Distributed Arrays

Goal: Zero-copy sharing across cores

Immutable partitions → Safe sharing

Versioned distributed arrays
Data Sharing Challenges

1. Garbage collection

2. Header conflicts
Overriding R’s allocator

Allocate process–local headers. Map data in shared memory
Outline

- Motivation
- Programming model
- Design
- Applications and Results
5 node cluster 8 cores per node
PageRank on 1.5B edge Twitter data
## Applications Implemented in Presto

<table>
<thead>
<tr>
<th>Application</th>
<th>Algorithm</th>
<th>Presto LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRank</td>
<td>Eigenvector calculation</td>
<td>41</td>
</tr>
<tr>
<td>Triangle counting</td>
<td>Top-K eigenvalues</td>
<td>121</td>
</tr>
<tr>
<td>Netflix recommendation</td>
<td>Matrix factorization</td>
<td>130</td>
</tr>
<tr>
<td>Centrality measure</td>
<td>Graph algorithm</td>
<td>132</td>
</tr>
<tr>
<td>k-path connectivity</td>
<td>Graph algorithm</td>
<td>30</td>
</tr>
<tr>
<td>k-means</td>
<td>Clustering</td>
<td>71</td>
</tr>
<tr>
<td>Sequence alignment</td>
<td>Smith–Waterman</td>
<td>64</td>
</tr>
</tbody>
</table>

**Fewer than 140 lines of code**
Evaluation Overview

Evaluation Setup
- 25 machine cluster
- Machine: 24 cores, 96GB RAM, 10Gbps network

Data-sharing benefits — 1.5B edge Twitter graph
Repartitioning analysis — 6B edge Web-graph

Faster than Spark and Hadoop using in-memory data
Collaborative Filtering using Netflix dataset
Data sharing benefits

No sharing

<table>
<thead>
<tr>
<th>CORES</th>
<th>Compute</th>
<th>Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4.38</td>
<td>1.22</td>
</tr>
<tr>
<td>20</td>
<td>2.21</td>
<td>2.12</td>
</tr>
<tr>
<td>40</td>
<td>1.22</td>
<td>4.16</td>
</tr>
</tbody>
</table>

Sharing

<table>
<thead>
<tr>
<th>CORES</th>
<th>Compute</th>
<th>Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4.45</td>
<td>0.71</td>
</tr>
<tr>
<td>20</td>
<td>2.49</td>
<td>0.7</td>
</tr>
<tr>
<td>40</td>
<td>1.63</td>
<td>0.72</td>
</tr>
</tbody>
</table>
Repartitioning Progress

Split size (GB)

Iteration count

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Repartitioning benefits

No Repartition

Repartition

Workers

Transfer

Compute
Related Work

Large scale data processing frameworks — MapReduce, Dryad, Spark, GraphLab

Matrix Computations — Ricardo, MadLINQ

HPC systems — ARPACK, Combinatorial BLAS

Multi-core R packages — doMC, snow, Rmpi
Presto

Locality-based scheduling

Caching partitions

Co-partitioning matrices
Conclusion

Presto: Large scale array-based framework extends R

Challenges with Sparse matrices
Repartitioning, sharing versioned arrays
Backup Slides
Netflix Collaborative Filtering

Number of cores vs Time (seconds)

- **Load**
- **t(R)×R**
- **R×t(R)×R**

- 48 cores:
  - Load: 0
  - t(R)×R: 155.299
  - R×t(R)×R: 202.725

- 40 cores:
  - Load: 0
  - t(R)×R: 202.725
  - R×t(R)×R: 234.236

- 32 cores:
  - Load: 0
  - t(R)×R: 234.236
  - R×t(R)×R: 256.1

- 24 cores:
  - Load: 0
  - t(R)×R: 256.1
  - R×t(R)×R: 380.985

- 16 cores:
  - Load: 0
  - t(R)×R: 380.985
  - R×t(R)×R: 755.112

- 8 cores:
  - Load: 0
  - t(R)×R: 755.112
  - R×t(R)×R: 755.112
Repartitioning benefits

Graph showing the relationship between the number of repartitions and time to convergence and cumulative partitioning time.