Experimental Evaluation of Multi-Round Matrix Multiplication on MapReduce

Matteo Ceccarello\textsuperscript{1} \quad Francesco Silvestri\textsuperscript{1}

Dip. di Ingegneria dell’Informazione, Università di Padova, Italy

5/1/2015 - ALENEX 2015
Why matrix multiplication?

- MapReduce is a common choice when dealing with Big Data
- The computation is organized in rounds

Map $\rightarrow$ Shuffle $\rightarrow$ Reduce

**Monolithic algorithms**
Execute in a constant number of rounds

**Multi-round algorithms**
The number of rounds depends on the input size
Why matrix multiplication?

Not a performance benchmark
We are interested in investigating the behaviour of multi-round algorithms w.r.t monolithic ones.

Why multi round?

Service market  Start/stop computations waiting for better prices
Resource requirements  Monolithic algorithms may require too many resources
Fault tolerance  Easier checkpointing
Outline

1. MapReduce
2. Our results
3. Algorithm
4. Implementation
   ▶ Map and Reduce
   ▶ Partitioner
5. Experiments
   ▶ In-house cluster
   ▶ Amazon EMR
6. Conclusions
Previous work

- Original work
  - “MapReduce: Simplified Data Processing on Large Clusters” [Dean, Ghemawat 2004]

- Models
  - “A model of computation for MapReduce” [Karloff, Suri, Vassilvitskii 2010]
  - “Sorting, searching, and simulation in the MapReduce framework” [Goodrich, Sitchinava, Zhang 2011]
  - “Space-round tradeoffs for MapReduce computations” [Pietracaprina, Pucci, Riondato, Silvestri, Upfal 2012]

- Experimental work
  - “HAMA: An Efficient Matrix Computation with the MapReduce Framework” [Sangwon, Yoon, Jaehong, Seongwook, Jin-Soo, Seungryoul]
MapReduce - Model

$MR(m, \rho)$ [Pietracaprina+ 2012]

- $m$ local memory: constraints the amount of local work
- $\rho$ replication: constrains the communication in a single round
- The complexity is measured in number of rounds
Our results

- M₃ - Matrix Multiplication on MapReduce
  - Hadoop library for dense and sparse matrix multiplication

- Multi-round algorithms have performance comparable with monolithic ones

- Concentrating only on round number for performance evaluation may not be the best strategy
Algorithm: 3D dense

Data representation

- Input matrix $\rightarrow \sqrt{n} \times \sqrt{n}$, blocks of size $\sqrt{m} \times \sqrt{m}$
- Input represented as a collection of pairs

$$\langle (i, \ell, j); A_{ij} \rangle \quad \text{for } 0 \leq i, j < \sqrt{n/m}$$

Algorithm

- $C_{ij} = \sum_{h=0}^{\sqrt{n/m}-1} A_{ih} \cdot B_{ih}$
- There are $(n/m)^{3/2}$ products, partitioned into $\sqrt{n/m}$ groups
- Round $r$: groups $G_\ell$ for $r \rho \leq \ell < (r + 1) \rho$
Algorithm: 3D dense

\[ C_{ij} = \sum_{h=0}^{\sqrt{n/m}-1} A_{ih} \cdot B_{ih} \]

\[ (n/m)^{3/2} \text{ products } \Rightarrow \sqrt{n/m} \text{ groups} \]

\[ \text{Round } r: \text{ groups } \in [r\rho, (r+1)\rho) \]
Complexity

Theorem 2 [Pietracaprina+ 2012]

The above MR(m, ρ) algorithm multiplies two $\sqrt{n} \times \sqrt{n}$ dense matrices in

$$R = \frac{\sqrt{n}}{\rho \sqrt{m}} + 1$$

rounds.

$\rho = \sqrt{\frac{n}{m}} \Rightarrow$ monolithic algorithm

$\rho = 1 \Rightarrow$ multi-round algorithm
Algorithm: 2D dense (HAMA)

- $C_{i,j} = A_i \cdot B_j$
- The round complexity is higher

$$R = \frac{n}{\rho m} \quad \text{vs.} \quad \frac{\sqrt{n}}{\rho \sqrt{m}} + 1$$
Algorithm: 3D sparse

- Applies to Erdös-Rényi matrices
- The *density* is $\delta$
- Adaptation of the 3D algorithm
- Can be extended to generic sparse matrices

**Theorem**

*The above algorithm requires*

$$R = \frac{\delta n^{3/4}}{(\rho \sqrt{m})} + 1$$

*rounds, the expected shuffle size is $3\rho\delta^2 n^{3/2}$, and the expected reducer size is $3m$.***
Implementation - Partitioner

\[(i, h, j) \rightarrow z = i\rho \frac{n}{m} + j\rho + (h \mod \rho)\]

where \(\rho = \) replication
\(m = \) subproblem size
Questions

Q1: How does the subproblem size affect the performance?

Q2: Replication affects performance?

Q3: Which are the major factors affecting the running time?

Q4: Does the algorithm scale efficiently?

Q5: Is 3D better than 2D?

Q6: Does the sparse algorithm efficiently exploit the input sparsity?
Experimental setup

In-house cluster

- 16 machines
- 4-Core Intel I7 Nehalem @ 3.07GHz
- 12 GB RAM
- 6 disks of 1TB and 7200 RPM in RAID0

Amazon EMR

- c3.8xlarge
  - Compute-optimized instance
- i2.xlarge
  - Storage-optimized instance
Q1: Time vs. subproblem size

Increasing the subproblem size reduces the time

Local memory size

- max 16000
- max 32000
- min 16000
- min 32000
Q2: Time vs. replication

Multi round algorithms are comparable with monolithic ones.
Q3: Components cost

The most expensive component is communication.

![Graph showing the cost of components over time for different replication levels. The x-axis represents replication (inverse of the number of rounds), and the y-axis represents time (s). The chart shows that communication costs are significantly higher than computation and infrastructure costs.](image-url)
Q4: Scalability

The algorithm scales well with the number of machines.
Q5: 3D vs. 2D

The 3D algorithm is faster than the 2D approach, as predicted by the analysis.
Q6: Sparse

We can effectively leverage the sparsity of input matrices.

Sparse matrices

- Dimension: $2^{20}$
- Dimension: $2^{22}$
- Dimension: $2^{24}$

Graph showing time (s) vs replication for different dimensions.
A sneak peek at the future

Changing the platform makes space/round tradeoffs even more evident
Conclusions

Results

- Multi-round algorithms can be comparable with monolithic ones
- Focusing only on round number may not lead to the best performance if it implies a large amount of communication
- Open source implementation:
  - M₃ - Matrix Multiplication on MapReduce
  - http://www.dei.unipd.it/m3

Ongoing work

- Test algorithms on other MapReduce implementations (Spark)