**CME 323: Distributed Algorithms and Optimization, Spring 2015**
`http://stanford.edu/~rezab/dao`.
Instructor: Reza Zadeh, Matroid and Stanford.

**Lecture 18, 5/25/2016. Scribed by Vishakh Hegde, Alex Williams and Patrick Landreman.**

Lecture outline:

- Matrix multiplication on a cluster
- Singular Value Decomposition

# 18 Matrix Multiplication on a Cluster

In the first half of this course, we have seen algorithms to perform matrix-matrix multiplication, assuming the PRAM model. In this lecture we will see how we can perform matrix-matrix multiplication in a distributed manner. In Spark, matrices are typically stored broken up for storage in three different ways:

- By entries (CoordinateMatrix): stored as a list of (i, j, value) tuples
- By rows (RowMatrix): each row is stored separately (e.g. Pagerank)
- By blocks (BlockMatrix): by storing submatrices of a matrix as dense matrices, block matrices can take advantage of low-level linear algebra library for operations like multiplications.

Consider two matrices A, of size $m \times k$ and B, of size $k \times n$. The matrix product $AB$ will have size $m \times n$. Let these matrices be saved as an RDD of $(i, j, value)$ corresponding to the $ij^{th}$ element of $A$ and $(p, q, value)$ corresponding to the $pq^{th}$ element of $B$. The $iq^{th}$ element of the product $AB$ is $(AB)_{iq} = \sum_{i=1}^{k} A_{ij}B_{jq}$. So it is clear that in order to perform a reduce operation (summation) in the MapReduce framework, we first need to get these values to have the same key. Therefore, we need to produce key-value pairs with the value being $A_{ij}B_{jq}$ and the key being $(i, q)$ for all possible values of $j$. The Map phase will constitute an all to all communication. We then do a ReduceByKey operation to sum up all values corresponding to the same key.

# 19 Singular Value Decomposition (SVD)

The rank-$r$ singular value decomposition (SVD) is a factorization of a real matrix $A \in \mathbf{R}^{m \times n}$, such that $A = U\Sigma V^T$, where $U \in \mathbf{R}^{m \times r}$ and $V \in \mathbf{R}^{n \times r}$ are unitary matrices holding the left and right singular vectors of $A$, and $\Sigma$ is a $r \times r$ diagonal matrix with non-negative real entries, holding the top $r$ singular values of $A$.

### 19.1 When $A$ is a `RowMatrix`

It is easy to see that the singular values and right singular vectors can be recovered from the SVD of the Gramian matrix $A^TA$:

$$A^TA = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T$$

We can exploit this property to efficiently compute the SVD of a tall-skinny matrix. If $n$ is small enough to fit on a single machine, then $A$ can be distributed as a one-dimensional block-row matrix (in Spark this is called a `RowMatrix`). We compute $A^TA$ using the methods described in the previous section, then solve the SVD of $A^TA$ locally on a single machine to determine $\Sigma$ and $V$. Finally, we can solve for $U$ by simple matrix multiplications:

$$A = U\Sigma V^T \;\Rightarrow\; U = AV\Sigma^{-1}$$

The key observation is that $A^TA$ is a much smaller matrix than $A$ when $n \ll m$. In general, computing the rank-$r$ SVD of $A$ will cost $O(mnr)$ operations (not to mention expensive communication costs), while this computation only costs $O(n^2r)$ operations for $A^TA$. Of course, if $A$ is short and fat (i.e. $n \gg m$), we can run this same algorithm on $A^T$. The complete algorithm is given below:

---

**Procedure 1** SVD of a `RowMatrix`

---

**Input:** $A \in \mathbf{R}^{m\times n}$ and desired rank $r$. $A$ is distributed as a `RowMatrix` across $p$ machines, machine $\ell$ holds a submatrix $A^{(\ell)} \in \mathbf{R}^{\frac{m}{p}\times n}$.

**Output:** The singular value decomposition of $A$. $U$ is distributed as a `RowMatrix` similar to $A$.

1: Each machine computes $A^{(\ell)T}A^{(\ell)}$

2: ***reduce:*** $A^TA = \sum_{\ell=1}^{p} A^{(\ell)T}A^{(\ell)}$

▷ The result is a small $n \times n$ matrix

3: $V\Sigma^2V^T = A^TA$

4: $\Sigma = \sqrt{\Sigma^2}$

5: ***broadcast:*** $V\Sigma^{-1}$

6: Each machine computes $U^{(\ell)} = A^{(\ell)}V\Sigma^{-1}$

---

*Communication costs:* We assume that we use combiners so that each machine locally computes its portion of $A^TA$ in line 1 separately before communication between machines occurs in line 2. The only communication costs are the all-to-one communication on line 2 with message size $n^2$, and the one-to-all communication on line 5 with message size $nr$. These require $O(\log p)$ messages when a recursive doubling communication pattern is used.

### 19.2 When $A$ is square

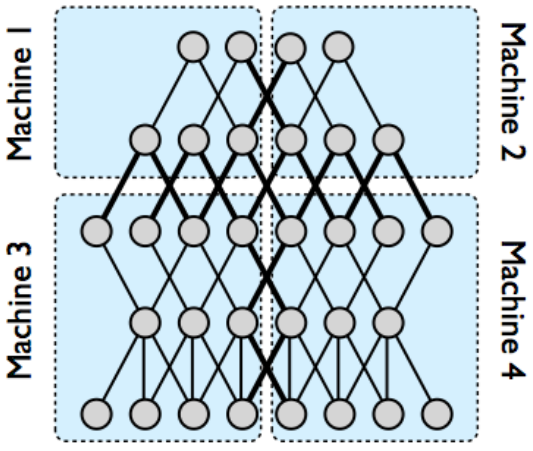
The algorithm which produces the $k$ largest eigenvectors of $A^TA$ is implemented with ARPACK and requires successive calls of matrix-vector multiplication. In the tall & skinny case above, $A^TA$ is relatively small and can be stored locally on the Driver machine. In the case where $A$ is square or nearly square, however, $A^TA$ remains a large matrix and the rows must be distributed. In this situation, the updated vector $(A^TA)v$ is produced by the following:

- Broadcast v to all workers
- Each worker computes the $i$th element of $Av = a_i^T v$ according to whichever rows of $A$ are present
- The $i$th column of $A^T$ is then scaled by the corresponding result of the previous dot product. Since the columns of $A^T$ are simply the rows of $A$, no additional data is needed
- The scaled columns are reduced by summation on the Driver

The above process is implemented using an `Aggregate` method. This distributed solution requires passing data over the network $O(k)$ times, compared to $O(1)$ when the Gramian matrix is stored locally. In exchange, the storage requirement is reduced from $O(n^2)$ to $O(n)$.

## 20 Distributed Neural Networks

Neural networks (NN) are an extremely active subject of research, enabled by advancements in data storage and parallel computing power. NNs may be distributed via two schemes: data parallelism or model parallelism. In the former, the nodes of the NN itself can reside on a single computer, while the training and input data (possibly large image files, etc) are scattered across a distributed file system. In model parallelism, the NN itself is too large to contain on a single computer. In this case, the model is clustered into blocks which communicate between each other using vectors - each element in a vector represents one input or output *"synapse"* of the block. Below is an example from DistBelief, a distributed NN developed by Google, inc. The figure illustrates a set of nodes which have been divided amongst four machines. The links in bold highlight synapses which communicate via input and output vectors.



## References


[1] L. Pu and R. Zadeh. (2014) Distributing the Singular Value Decomposition with Apache Spark. *Databricks Company Blog* https://databricks.com/blog/2014/07/21/distributing-the-singular-value-decomposition-with-spark.html

[2] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, MarcAurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng. (2012) Large Scale Distributed Deep Networks. *NIPS* http://research.google.com/pubs/pub40565.html