# Matrix-Vector Multiplication by MapReduce

From Rajaraman / Ullman- Ch.2

Part 1

## Google implementation of MapReduce

- created  to execute very large matrix-vector multiplications
- When ranking of Web pages that goes on at search engines, n is in the tens of billions.
- Page Rank- iterative algorithm
- Also, useful for simple (memory-based) recommender systems

# Problem Statement

- Given,
- n × n matrix **M**, whose *element* in row i and column j will be denoted $m_{ij}$ .
- a vector **v** of length n.
  - $x_i = \sum_1^n m_{ij} \, . \, v_j$
- Assume that
  - the row-column coordinates of each matrix element will be discoverable, either from its position in the file, or because it is stored with explicit coordinates, as a triple (i, j, $m_{ij}$).
  - the position of element $v_j$ in the vector v will be discoverable in the analogous way.

## Case 1: n is large, but not so large that vector v cannot fit in main memory

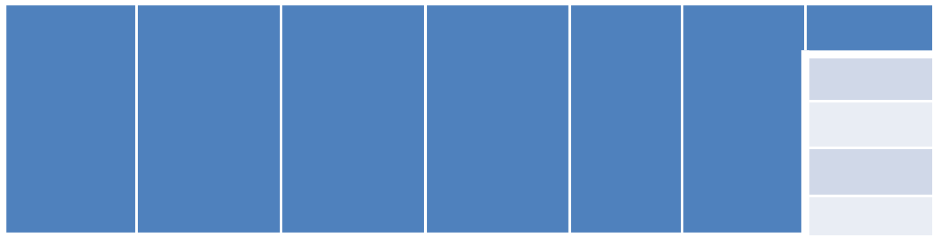
- **The Map Function:**
- Map function is written to apply to one element of **M**
- **v** is first read to computing node executing a Map task and is available for all applications of the Map function at this compute node.
- Each Map task will operate on a chunk of the matrix **M**.
- From each matrix element $m_{ij}$ it produces the key-value pair (i, ( $m_{ij}$ . $v_j$ ) ).
- Thus, all terms of the sum that make up the component $x_i$ of the matrix-vector product will get the same key, i.

## Case 1 Continued …

- **The Reduce Function:**
- The Reduce function simply sums all the values associated with a given key i.
- The result will be a pair (i,$x_i$ ).

## Case 2: n is large to fit into main memory

- **v** should be stored in computing nodes used for the Map task
- Divide the matrix into vertical stripes of equal width and divide the vector into an equal number of horizontal stripes, of the same height.
- Our goal is to use enough stripes so that the portion of the vector in one stripe can fit conveniently into main memory at a compute node.



Matrix **M** vector **v**

Figure 2.4: Division of a matrix and vector into *five stripes*

- The ith stripe of the matrix multiplies only components from the ith stripe of the vector.
- Divide the matrix into one file for each stripe, and do the same for the vector.
- Each Map task is assigned a chunk from one of the stripes of the matrix and gets the entire corresponding stripe of the vector.

The Map and Reduce tasks can then act exactly as was described, as case 1.

- Recommender is one of the most popular large-scale machine learning techniques.
  - Amazon
  - eBay
  - Facebook
  - Netflix
  - …

- Two types of recommender techniques
  - Collaborative Filtering
  - Content Based Recommendation
- Collaborative Filtering
  - Model based
  - Memory based
    - User similarity based
    - Item similarity based

# Item-based collaborative filtering

- Basic idea:
  - Use the similarity between items (and not users) to make predictions
- Example:

| | Item1 | Item2 | Item3 | Item4 | Item5 |
|---|---|---|---|---|---|
| Alice | 5 | 3 | 4 | 4 | ? |
| User1 | 3 | 1 | 2 | 3 | 3 |
| User2 | 4 | 3 | 4 | 3 | 5 |
| User3 | 3 | 3 | 1 | 5 | 4 |
| User4 | 1 | 5 | 5 | 2 | 1 |

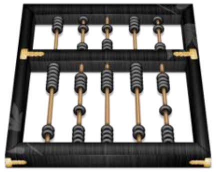
5

predict the

# The cosine similarity measure

- **Produces better results in item-to-item filtering**
- **Ratings are seen as vector in n-dimensional space**
- **Similarity is calculated based on the angle between the vectors**

$$sim(\vec{a},\vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|}$$



- **Adjusted cosine similarity**
  - take average user ratings into account, transform the original ratings
  - $U$: set of users who have rated both items $a$ and $b$

$$sim(\vec{a},\vec{b}) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_u})^2}}$$

## Making predictions

- A common prediction function:

$$pred(u,p) = \frac{\sum_{i \in ratedItem(u)} sim(i,p) * r_{u,i}}{\sum_{i \in ratedItem(u)} sim(i,p)}$$

- u –> user; i,p -> items; user has not rated item p.
- Neighborhood size is typically also limited to a specific size
- Not all neighbors are taken into account for the prediction
- An analysis of the MovieLens dataset indicates that "in most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable" (Herlocker et al. 2002)

## Pre-processing for item-based filtering

- Item-based filtering does not solve the scalability problem itself
- Pre-processing approach by Amazon.com (in 2003)
  - Calculate all pair-wise item similarities in advance
  - The neighborhood to be used at run-time is typically rather small, because only items are taken into account which the user has rated
  - Item similarities are supposed to be more stable than user similarities
- Memory requirements
  - Up to $N^2$ pair-wise similarities to be memorized (N = number of items) in theory
  - In practice, this is significantly lower (items with no co-ratings)
  - Further reductions possible
    - Minimum threshold for co-ratings
    - Limit the neighborhood size (might affect recommendation accuracy)

## Mahout’s Item-Based RS

- Apache Mahout is an open source **Apache Foundation** project for scalable machine learning.
- Mahout uses **Map-Reduce** paradigm for scalable recommendation

## Mahout’s RS

### Matrix Multiplication for Preference Prediction

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 16 & \mathbf{9} & \mathbf{16} & \mathbf{5} & \mathbf{6} \\ 2 & 9 & 30 & \mathbf{19} & \mathbf{3} & \mathbf{2} \\ 3 & 16 & 19 & 23 & \mathbf{5} & \mathbf{4} \\ 4 & 5 & 3 & 5 & 10 & \mathbf{20} \\ 5 & 6 & 2 & 4 & 20 & 9 \end{array} \times \begin{array}{c} \\ 0 \\ 5 \\ 5 \\ 2 \\ 0 \end{array} = \begin{array}{c|c} & \\ \mathbf{135} & 1 \\ 251 & 2 \\ 220 & 3 \\ 60 & 4 \\ 70 & 5 \end{array}$$

Item-Item Similarity Matrix

Active User’s Preference Vector

# As matrix math, again

Inside-out method

$$5\begin{bmatrix}9\\30\\19\\3\\2\end{bmatrix} + 5\begin{bmatrix}16\\19\\23\\5\\4\end{bmatrix} + 2\begin{bmatrix}5\\3\\5\\10\\20\end{bmatrix} = \begin{bmatrix}135\\251\\220\\60\\70\end{bmatrix}$$

# Page Rank Algorithm

- One iteration of the PageRank algorithm involves taking an estimated Page-Rank vector v and computing the next estimate v′ by

$$v' = \beta M v + (1 - \beta) e/n$$

- β is a constant slightly less than 1, e is a vector of all 1's, and
- n is the number of nodes in the graph that transition matrix M represents.

# Vectors **v** and **v’** are too large for Main Memory

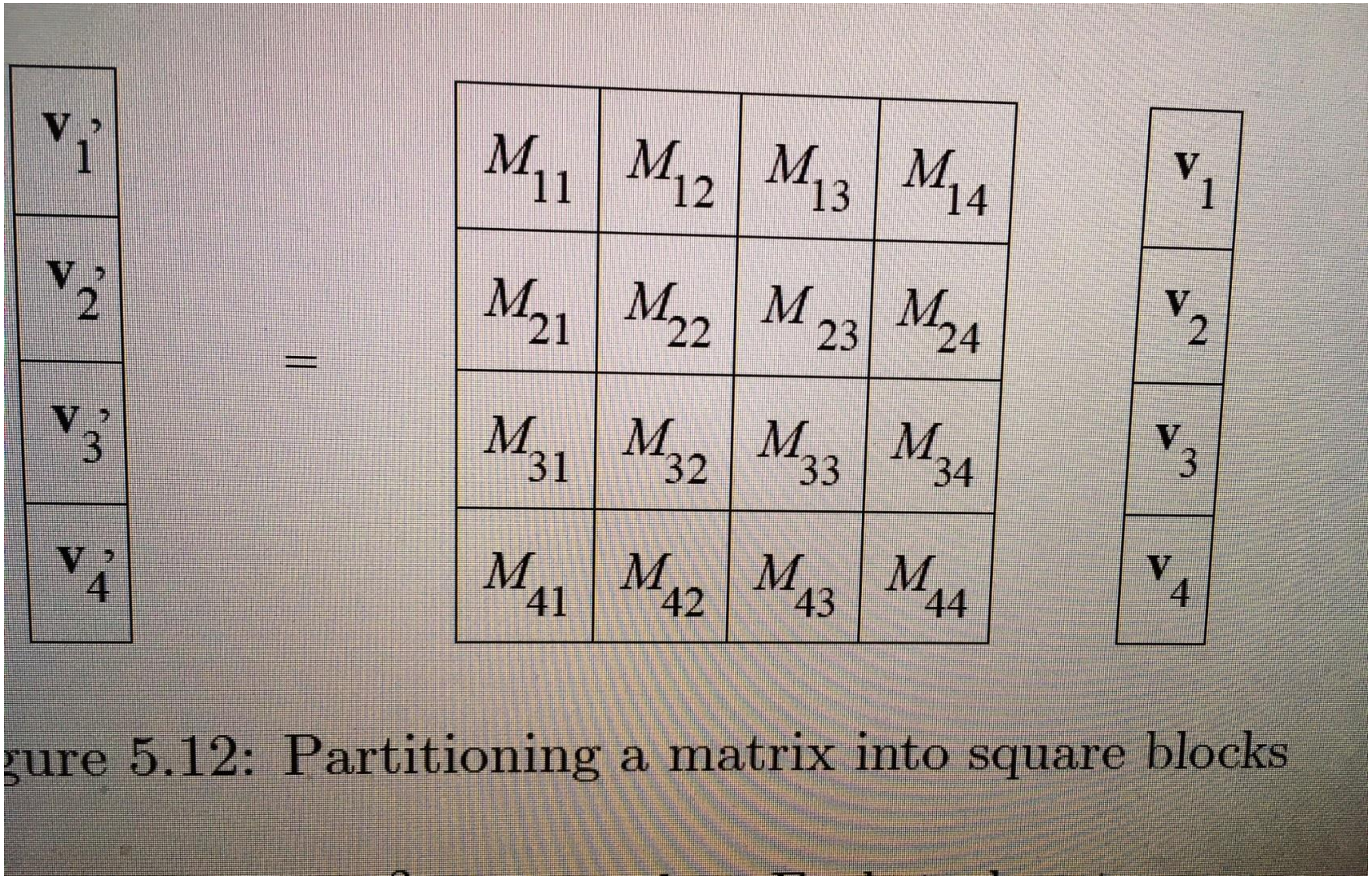
- If using the stripe method to partition a matrix and vector that do not fit in main memory, then a vertical stripe from the matrix M and a horizontal stripe from the vector **v** will contribute to all components of the result vector **v’**.
- Since that vector is the same length as **v**, it will not fit in main memory either.
- M is stored column-by-column for efficiency reasons, a column can affect any of the components of **v’**. As a result, it is unlikely that when we need to add a term to some component $\boldsymbol{v'}_{\boldsymbol{i}}$ , that component will already be in main memory.
  - Thrashing

# Square Blocks, instead of Slices

- An additional constraint is that the result vector should be partitioned in the same way as the input vector, so the output may become the input for another iteration of the matrix-vector multiplication.

- An alternative strategy is based on partitioning the matrix into $k^2$ blocks, while the vectors are still partitioned into k stripes.

# Figure from Rajaraman, Ullmann



$$\begin{array}{|c|}\hline \mathbf{v}'_1 \\ \hline \mathbf{v}'_2 \\ \hline \mathbf{v}'_3 \\ \hline \mathbf{v}'_4 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline M_{11} & M_{12} & M_{13} & M_{14} \\ \hline M_{21} & M_{22} & M_{23} & M_{24} \\ \hline M_{31} & M_{32} & M_{33} & M_{34} \\ \hline M_{41} & M_{42} & M_{43} & M_{44} \\ \hline \end{array} \quad \begin{array}{|c|}\hline \mathbf{v}_1 \\ \hline \mathbf{v}_2 \\ \hline \mathbf{v}_3 \\ \hline \mathbf{v}_4 \\ \hline \end{array}$$

gure 5.12: Partitioning a matrix into square blocks

# Iterative Computation- Square Blocks Method

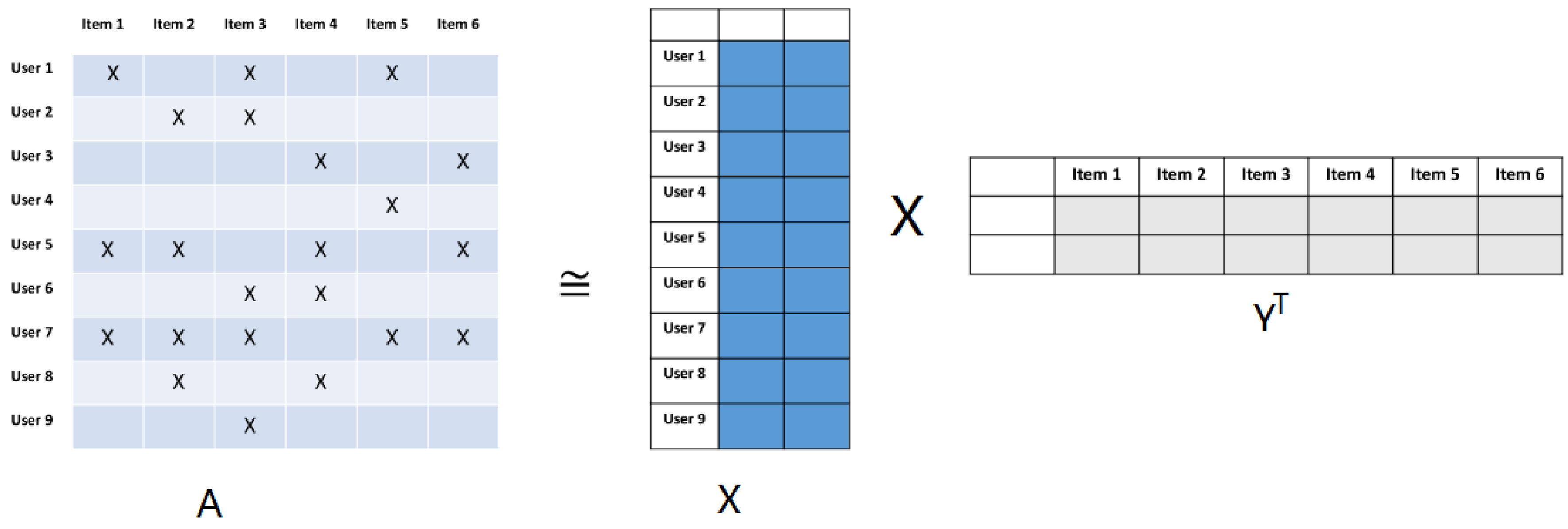
- In this method, we use $k^2$ Map tasks. Each task gets one square of the matrix M, say $M_{ij}$ , and one stripe of the vector **v**, which must be $\boldsymbol{v_j}$ . Notice that each stripe of the vector is sent to k different Map tasks; $\boldsymbol{v_j}$ is sent to the task handling $M_{ij}$ for each of the k possible values of i. Thus, **v** is transmitted over the network k times. However, each piece of the matrix is sent only once.

- The advantage of this approach is that we can keep both the jth stripe of **v** and the ith stripe of **v**' in main memory as we process $M_{ij}$. Note that all terms generated from $M_{ij}$ and $\boldsymbol{v_j}$ contribute to $\boldsymbol{v'_i}$ and no other stripe of **v'**.

# Matrix-Matrix Multiplication using MapReduce

- Like Natural Join Operation

# The Alternating Least Squares (ALS) Recommender Algorithm



| | Item 1 | Item 2 | Item 3 | Item 4 | Item 5 | Item 6 |
|---|---|---|---|---|---|---|
| User 1 | X | | X | | X | |
| User 2 | | X | X | | | |
| User 3 | | | | X | | X |
| User 4 | | | | | X | |
| User 5 | X | X | | X | | X |
| User 6 | | | X | X | | |
| User 7 | X | X | X | | X | X |
| User 8 | | X | | X | | |
| User 9 | | | X | | | |

- Matrix A represents a typical user-product-rating model
- The algorithm tries to derive a set of latent factors (like tastes) from the user-product-rating matrix from matrix A.
- So matrix X can be viewed as user-taste matrix
  And matrix Y can be viewed as a product-taste matrix

- Objective Function

$$minimize\ \frac{1}{2}||A - XY^T||^2$$

- Only check observed ratings

$$minimize\ \frac{1}{2}\sum_{(i,j)\in\Omega}(a_{i,j} - x_i{}^T y_j)^2$$

- If we fix X, the objective function becomes a convex. Then we can calculate the gradient to find Y
- If we fix Y, we can calculate the gradient to find X

# Iterative Training Algorithm

- Given Y, you can Solve X

$X_i = AY_{i-1}(Y_{i-1}^{\mathsf{T}}Y_{i-1})^{-1}$

- The algorithm works in the following way:

$Y_0 \to X_1 \to Y_1 \to X_2 \to Y_2 \to \ldots$

Till the squared difference is small enough