

Map-Reduce (Part II)

Yijun Lin

yijunlin@usc.edu

Spatial Sciences Institute

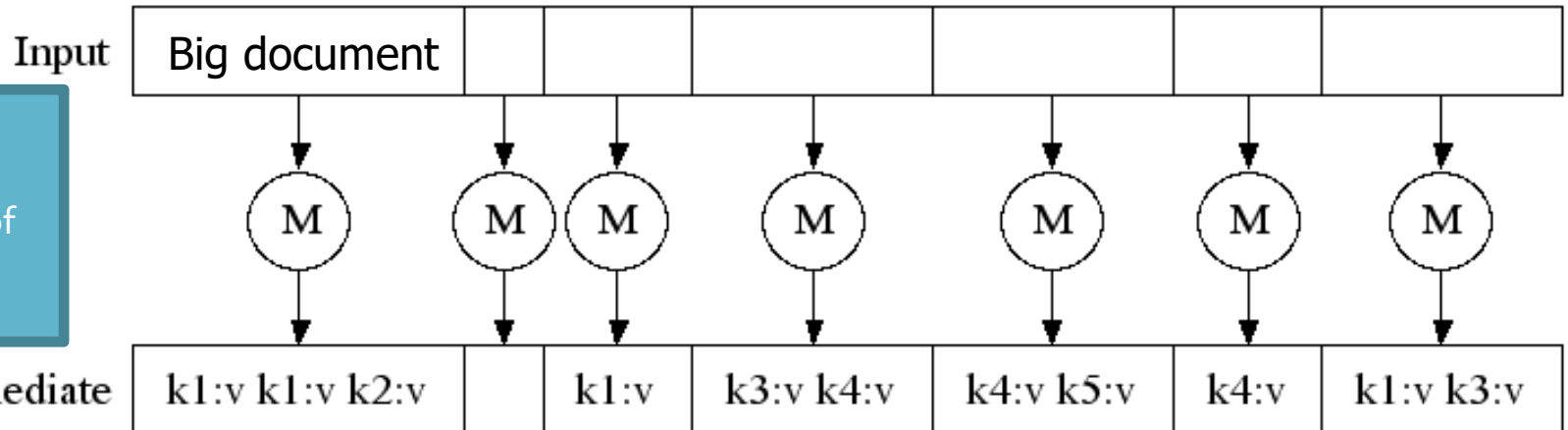
Office: AHF B56A

Thanks for source slides and material to: J. Leskovec, A. Rajaraman,
J. Ullman: Mining of Massive Datasets (<http://www.mmds.org>)

Map-Reduce: A diagram

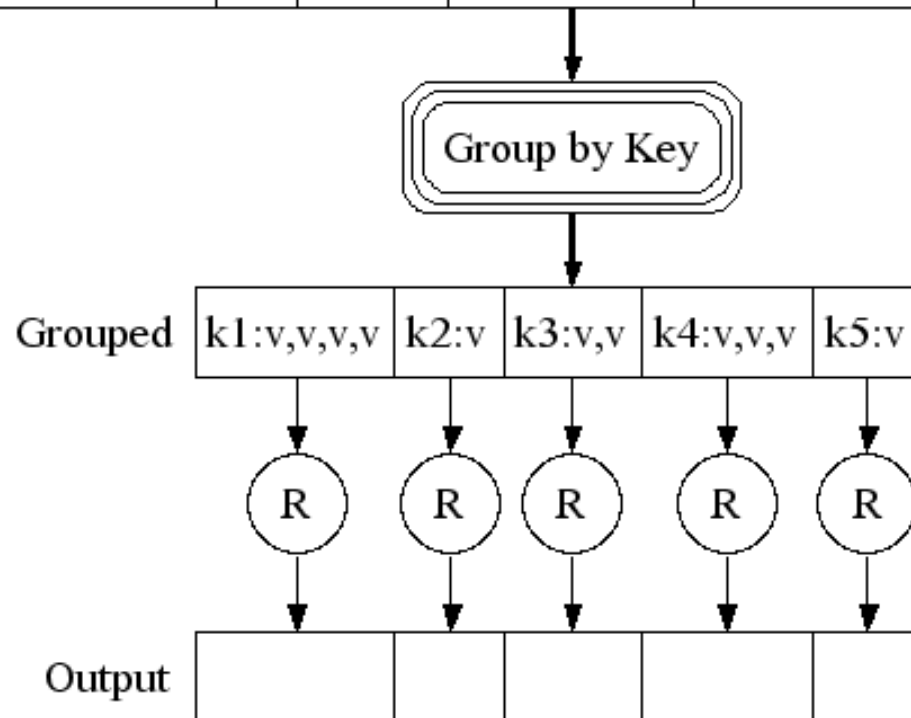
MAP:

Read input and produces a set of key-value pairs



Group by key:

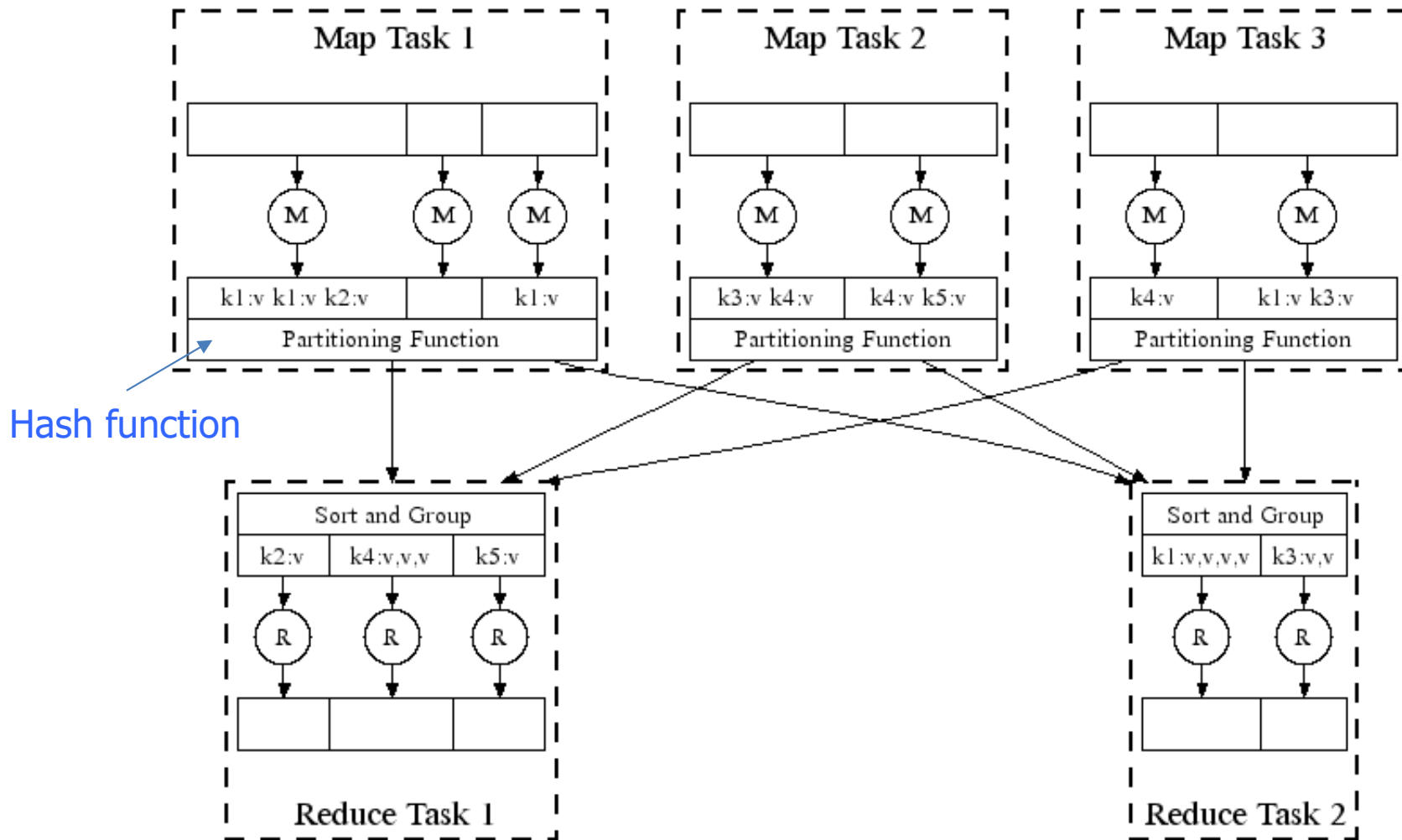
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)



Reduce:

Collect all values belonging to the key and output

Map-Reduce: In Parallel

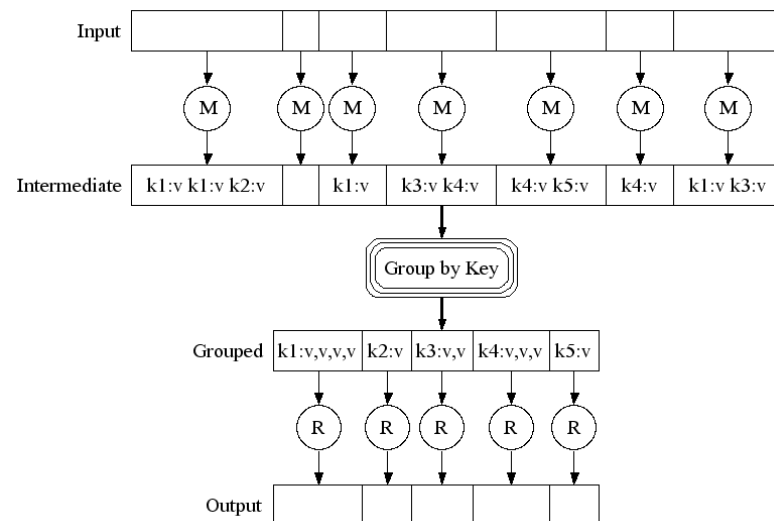


All phases are distributed with many tasks doing the work

Map-Reduce: Environment

MapReduce environment takes care of:

- **Partitioning** the input data
- **Scheduling** the program's execution across a set of nodes
- Performing the **group by key** step
- Handling machine **failures**
- Managing required inter-machine **communication**



Data Flow

- Input and final output are stored **on a distributed file system (DFS)**
 - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored **on local file system of Map workers**
e.g., output of the map step
- Output is often input to another Map-Reduce task

Coordination: Master

- Master node takes care of coordination:
 - **Task status:** idle, in-progress, completed
 - **Idle tasks** get scheduled as workers become available
 - When a map task completes, it sends the **master** the **location** and **sizes** of its R intermediate files, one for each reducer (R = number of reducers)
 - **Master pushes this info to reducers**
- Master pings workers **periodically** to detect failures

Dealing with Failures

- **Map worker failure**
 - Map tasks **completed or in-process** at worker are reset to idle
 - Idle tasks eventually rescheduled on other worker(s)

Dealing with Failures

- **Map worker failure**
 - Map tasks **completed or in-process** at worker are reset to idle
 - Idle tasks eventually rescheduled on other worker(s)
- **Reduce worker failure**
 - Only **in-process tasks** are reset to idle
 - Idle Reduce tasks restarted on other worker(s)

Dealing with Failures

- **Map worker failure**
 - Map tasks **completed or in-process** at worker are reset to idle
 - Idle tasks eventually rescheduled on other worker(s)
- **Reduce worker failure**
 - Only **in-process tasks** are reset to idle
 - Idle Reduce tasks restarted on other worker(s)
- **Master failure**
 - Map-reduce task is aborted and client is notified

How many map and reduce jobs?

- **M map tasks, R reduce tasks**

How many map and reduce jobs?

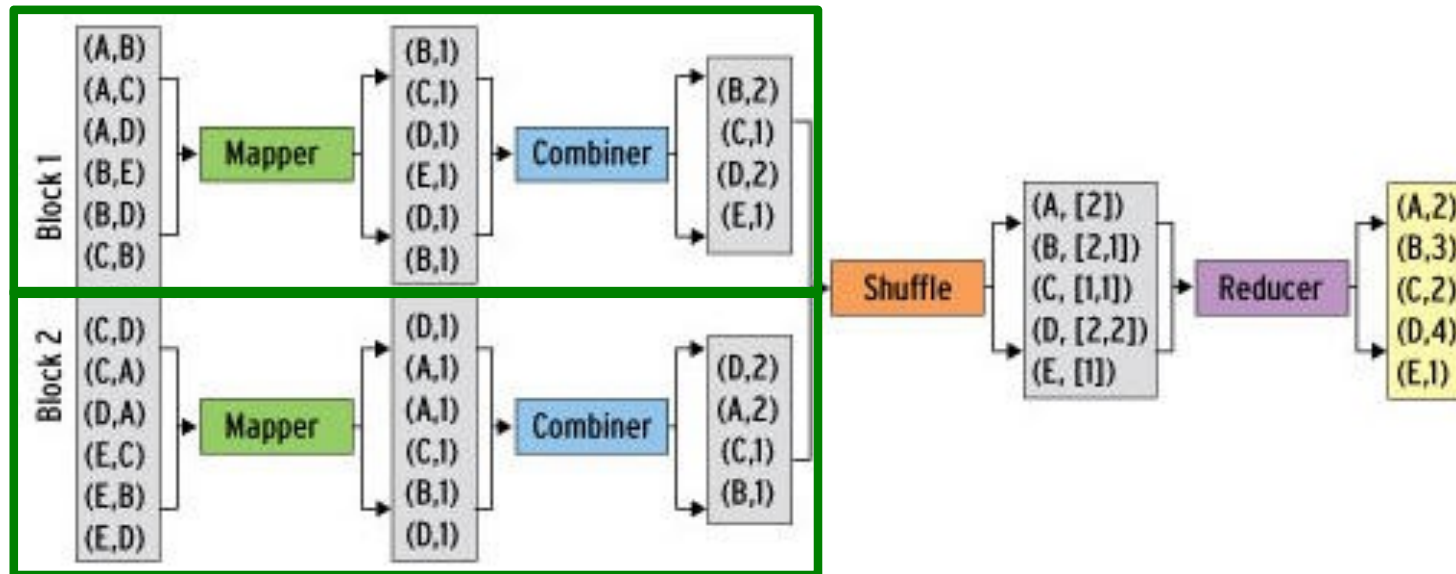
- M map tasks, R reduce tasks
- **Rule of a thumb:**
 - Make M **much larger** than the number of nodes in the cluster
 - One DFS chunk per map task is common
 - Improves dynamic load balancing and speeds up recovery from worker failures

How many map and reduce jobs?

- **M map tasks, R reduce tasks**
- **Rule of a thumb:**
 - Make M **much larger** than the number of nodes in the cluster
 - One DFS chunk per map task is common
 - Improves dynamic load balancing and speeds up recovery from worker failures
- **Usually R is smaller than M**
 - Output is spread across R files
 - Google example: Often use 200,000 map tasks, 5000 reduce tasks on 2000 machines

Refinement: Combiners

- **Combiner** combines the values of all keys of a single mapper (single node)



Much less data needs to be copied and shuffled!

Works if reduce function is commutative and associative

Refinement: Partition Function

- **Control how keys get partitioned**
 - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function:
 - **$\text{hash}(\text{key}) \bmod R$**
- Sometimes useful to override the hash function:
 - E.g., **$\text{hash}(\text{hostname}(\text{URL})) \bmod R$** ensures URLs from a host to end up in the same output file

Implementations

- **Google's MapReduce**
 - Not available outside Google
- **Hadoop**
 - Open-source implementation in Java
 - Uses HDFS for stable storage
 - Download: <http://hadoop.apache.org/releases.html>
- **Spark**

Example: Relational Join

Employee

Name	SSN
Sue	999999999
Tony	777777777

Assigned Departments

EmpSSN	DepName
999999999	Accounts
777777777	Sales
777777777	Marketing

Employee ⋈ Assigned Departments

Name	SSN	EmpSSN	DepName
Sue	999999999	999999999	Accounts
Tony	777777777	777777777	Sales
Tony	777777777	777777777	Marketing

Example: Relational Join

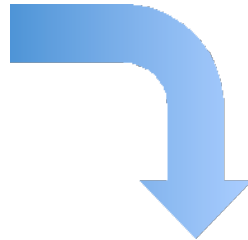
- Map Task: Emit (key, value) pair
 - **Key is key used for join**
 - **Value is a tuple** with all fields from table (including the table name)

Employee

Name	SSN
Sue	999999999
Tony	777777777

Assigned Departments

EmpSSN	DepName
999999999	Accounts
777777777	Sales
777777777	Marketing



key=999999999, value=(Employee, Sue, 999999999)
key=777777777, value=(Employee, Tony, 777777777)
key=999999999, value=(Department, 999999999, Accounts)
key=777777777, value=(Department, 777777777, Sales)
key=777777777, value=(Department, 777777777, Marketing)

Example: Relational Join

- **Group by Key:** groups together all values (tuples) associated with each key
- **Reduce task:** emit joined values (without table names)

key=999999999, values=[(Employee, Sue, 999999999),
(Department, 999999999, Accounts)]



Sue, 999999999, 999999999, Accounts

key=777777777, values=[(Employee, Tony, 777777777),
(Department, 777777777, Sales),
(Department, 777777777, Marketing)]



Tony, 777777777, 777777777, Sales
Tony, 777777777, 777777777, Marketing

Example: Distributed Sort

- Goal: Sort a very large list of (firstName, lastName) pairs by lastName followed by firstName
- Map task:
- Reduce task:

Example: Distributed Sort

- Map task
 - Emit (lastName, firstName)
- Group by keys:
 - Group together entries with same last name
 - **Divide into non-overlapping alphabetical ranges (sorting)**
 - Keys are sorted in alphabetical order
- Reduce task
 - Processes one key at a time
 - For each (lastName, list(firstName)), emit (lastName, firstName) in alphabetical order (**sorting**)
 - Merge output from all Reduce tasks (e.g., write)

Example: Matrix Multiplication

- Assume two matrices A and B, and $AB = C$
- A_{ij} is the element in **row i** and **column j** of matrix A
 - Similarly for B and C
- **$C_{ik} = \sum_j A_{ij} \times B_{jk}$**
 - C_{ik} depends on the i^{th} row of A, that is A_{ij} for all j , and the k^{th} column of B, that is B_{jk} for all j

$$\begin{matrix} \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} & = & \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix} & \text{e.g., } C_{11} = 1 \times 1 + 3 \times 0 \\ A & B & & C & + 2 \times 5 = 11 \end{matrix}$$

Matrix Multiplication

Map-Reduce (One phase)

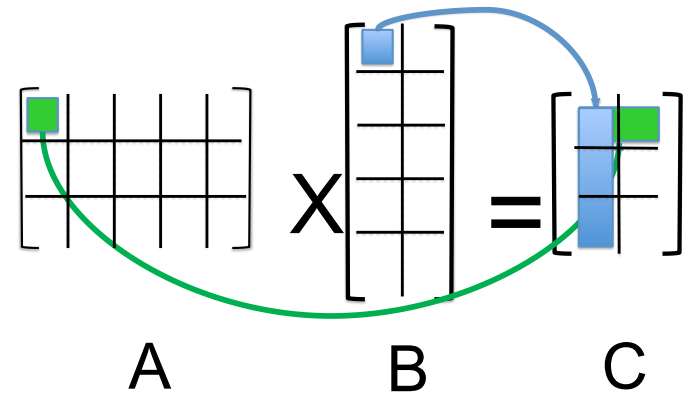
$$C = A \times B$$

A has dimensions $L \times M$

B has dimensions $M \times N$

C has dimensions $L \times N$

Matrix Multiplication: $C[i, k] = \text{SUM}_j (A[i, j] \times B[j, k])$



Map task:

Reduce task:

Matrix Multiplication

Map-Reduce (One phase)

$$C = A \times B$$

A has dimensions $L \times M$

B has dimensions $M \times N$

C has dimensions $L \times N$

Matrix Multiplication: $C[i, k] = \text{SUM}_j (A[i, j] \times B[j, k])$

$$\begin{matrix} \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} & = & \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix} \\ A & B & & C \end{matrix}$$

Map task:

for each element (i,j) of A, emit **$((i,k), A[i,j])$** for k in $1..N$

e.g., For A[1, 1] emit $((1, 1), 1), ((1, 2), 1)$

For A[1, 2] emit $((1, 1), 3), ((1, 2), 3)$

For A[2, 1] emit $((2, 1), 4), ((2, 2), 4)$

for each element (j,k) of B, emit **$((i,k), B[j,k])$** for i in $1..L$

e.g., For B[1, 1] emit $((1, 1), 1), ((2, 1), 1)$

For B[2, 1] emit $((1, 1), 0), ((2, 1), 0)$

For B[1, 2] emit $((1, 2), 3), ((2, 2), 3)$

Matrix Multiplication

Map-Reduce (One phase)

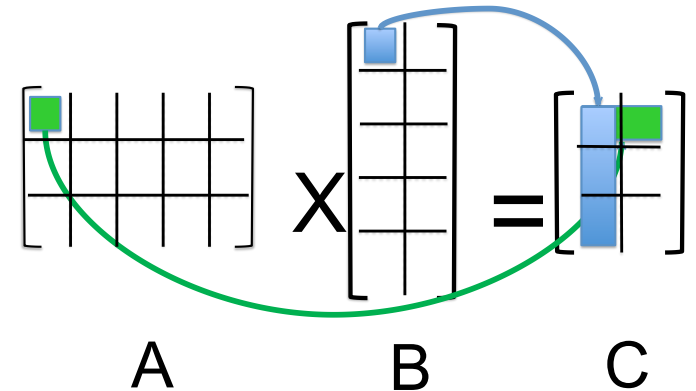
$$C = A \times B$$

A has dimensions $L \times M$

B has dimensions $M \times N$

C has dimensions $L \times N$

Matrix Multiplication: $C[i, k] = \text{SUM}_j (A[i, j] \times B[j, k])$



Map task:

for each element (i,j) of A, emit $((i,k), A[i,j])$ for k in $1..N$

Better: emit $((i,k), ('A', i, j, A[i,j]))$ for k in $1..N$

Or just emit $((i,k), ('A', j, A[i,j]))$ for k in $1..N$

for each element (j,k) of B, emit $((i,k), B[j,k])$ for i in $1..L$

Better: emit $((i,k), ('B', j, k, B[j,k]))$ for i in $1..L$

Or just emit $((i,k), ('B', j, B[j,k]))$ for i in $1..L$

Matrix Multiplication

Map-Reduce (One phase)

$C = A \times B$

A has dimensions $L \times M$

B has dimensions $M \times N$

C has dimensions $L \times N$

Matrix Multiplication: $C[i, k] = \text{SUM}_j (A[i, j] \times B[j, k])$

$$\begin{matrix} \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} & = & \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix} \\ A & B & & C \end{matrix}$$

Map task:

for each element (i,j) of A, emit **((i,k), ('A', i, j, A[i,j]))** for k in 1..N

e.g., For A[1, 1] emit ((1, 1), ('A', 1, 1, 1)), ((1, 2), ('A', 1, 1, 1))

For A[1, 2] emit ((1, 1), ('A', 1, 2, 3)), ((1, 2), ('A', 1, 2, 3))

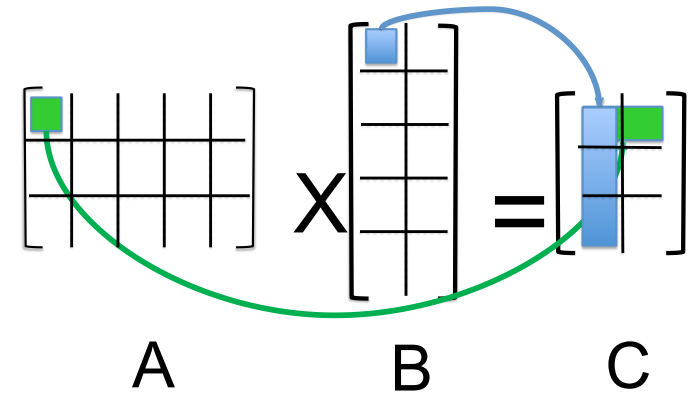
For A[2, 1] emit ((2, 1), ('A', 2, 1, 4)), ((2, 2), ('A', 2, 1, 4))

for each element (j,k) of B, emit **((i,k), ('B', j, k, B[j,k]))** for i in 1..L

e.g., For B[1, 1] emit ((1, 1), ('B', 1, 1, 1)), ((2, 1), ('B', 1, 1, 1))

For B[2, 1] emit ((1, 1), ('B', 2, 1, 0)), ((2, 1), ('B', 2, 1, 0))

For B[1, 2] emit ((1, 2), ('B', 1, 2, 3)), ((2, 2), ('B', 1, 2, 3))



$C[i,k] = \text{Sum}_j (A[i,j] \times B[j,k])$, C is $L \times N$

In the map phase:

- for each element (i,j) of A , emit $((i,k), ('A', i, j, A[i,j]))$ for k in $1..N$
- for each element (j,k) of B , emit $((i,k), ('B', j, k, B[j,k]))$ for i in $1..L$

e.g.,

$$C[1,1] = A[1,1] * B[1,1] + \mathbf{A[1,2]} * B[2,1] + A[1,3] * \mathbf{B[3,1]} + A[1,4] * B[4,1] + A[1,5] * B[5,1]$$

$$C[1,2] = A[1,1] * B[1,2] + \mathbf{A[1,2]} * B[2,2] + A[1,3] * B[3,2] + A[1,4] * B[4,2] + A[1,5] * B[5,2]$$

$$C[2,1] = A[2,1] * B[1,1] + A[2,2] * B[2,1] + A[2,3] * \mathbf{B[3,1]} + A[2,4] * B[4,1] + A[2,5] * B[5,1]$$

$$C[3,1] = A[3,1] * B[1,1] + A[3,2] * B[2,1] + A[3,3] * \mathbf{B[3,1]} + A[3,4] * B[4,1] + A[3,5] * B[5,1]$$

Map phase: For $\mathbf{A[1,2]}$, emit $((1, k), ('A', 1, 2, A[1,2]))$ for k in $1..2$

emit $((1,1)('A', 1, 2, A[1,2])) ((1,2)('A', 1, 2, A[1,2]))$

For $\mathbf{B[3,1]}$, emit $((i, 1), ('B', 3, 1, B[3,1]))$ for i in $1..3$

emit $((1,1), ('B', 3, 1, B[3,1])), ((2,1)('B', 3, 1, B[3,1])), ((3,1)('B', 3, 1, B[3,1]))$

Matrix Multiplication

Map-Reduce (Two phase)

Idea: 1, Multiply the appropriate values in 1st MapReduce phase
2, Add up in 2nd MapReduce phase

Try this tonight!

General Characteristic of Good Problem for Map-Reduce

Data set is truly “big”

- Terabytes, not tens of gigabytes
- Hadoop/MapReduce designed for terabyte/petabyte scale computation
- Most real-world problems process less than 100 GB of input
 - Microsoft, Yahoo: median job under 14 GB
 - Facebook: 90% of jobs under 100 GB

General Characteristic of Good Problem for Map-Reduce

Don't need fast response time

- When submitting jobs, Hadoop latency can be **1 min**
- Not well-suited for problems that require **faster response time**
 - online purchases, transaction processing
- A good **pre-computation engine**
 - E.g., pre-compute related items for every item in inventory

General Characteristic of Good Problem for Map-Reduce

- Good for applications that work in **batch mode**
- **Runs over entire data set**
 - Takes time to initiate, run;
 - Shuffle step can be time-consuming;
- Does not provide good support for **random access to datasets**
 - Extensions: Hive, Dremel, Shark, Amplab

General Characteristic of Good Problem for Map-Reduce

- Best suited for data that can be expressed as **key-value pairs** without losing context, dependencies
 - **Graph data is hard to process using Map-Reduce**
 - Implicit relationships: edges, sub-trees, child/parent relationships, weights, etc.
 - Graph algorithms need information about the entire graph for each iteration
 - Hard to break into independent chunks for Map tasks
 - Alternatives: Google's Pregel, Apache Giraph

General Characteristic of Good Problem for Map-Reduce

Other problems/data **NOT** suited for MapReduce

- Tasks that need results of intermediate steps to compute results of current step
 - Interdependencies among tasks
 - Map tasks must be independent
- Some machine learning algorithms
 - Gradient-based learning

General Characteristic of Good Problem for Map-Reduce

Summary: Good candidates for Map-Reduce:

- Jobs that process huge quantities of data and either summarize or transform the content
- Collected data has elements that can easily be captured with an identifier (key) and corresponding value