JOHNS HOPKINS
CAREY BUSINESS SCHOOL

Lecture 2

BU.330.740 Large Scale Computing on the Cloud

Minghong Xu, PhD.
Associate Professor

AWS Academy Courses

- AWS Academy invitations has been sent to you
- Use “@jh.edu”, no “@jhu.edu”
- **AWS academy learner lab [104575]**
  - $50 credit per student
  - Limited AWS services, some advanced features/instances are blocked
- Cloud foundations course [104574]
  - Overall introduction to all AWS services and cloud architecture
  - Voucher policy: https://www.awsacademy.com/forums/s/article/How-Learners-Can-Secure-Exam-Vouchers-through-the-AWS-Emerging-Talent-Community

Today's Agenda and Reminders

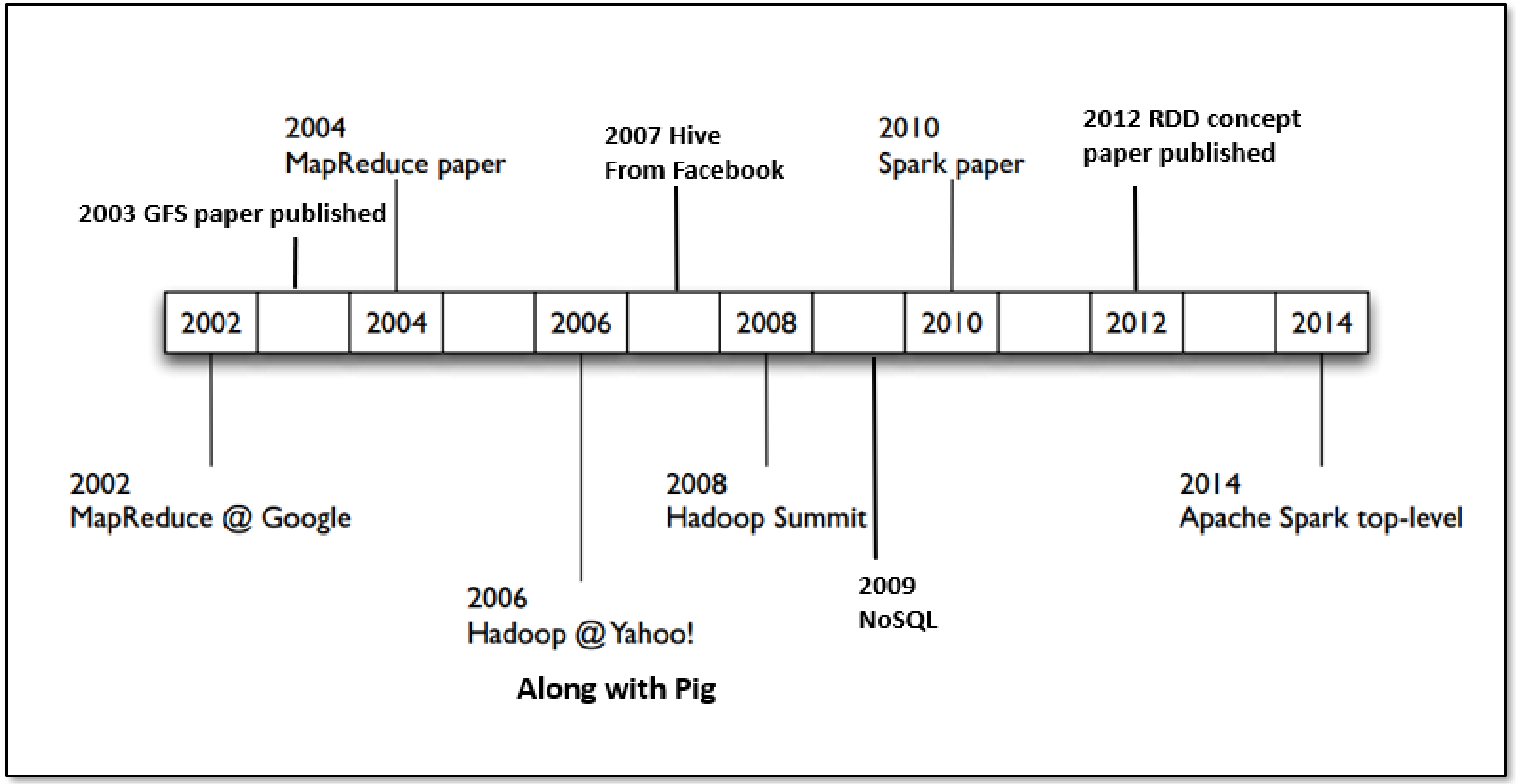
- MapReduce Framework
- Frequent Itemset Mining
- Lab1: AWS S3, EC2 and EMR
  - On-premise computing vs cloud computing, different mode, different experience
  - **It is normal to feel lost at first time**

- Submit your team roster if you have not done so

JOHNS HOPKINS
CAREY BUSINESS SCHOOL

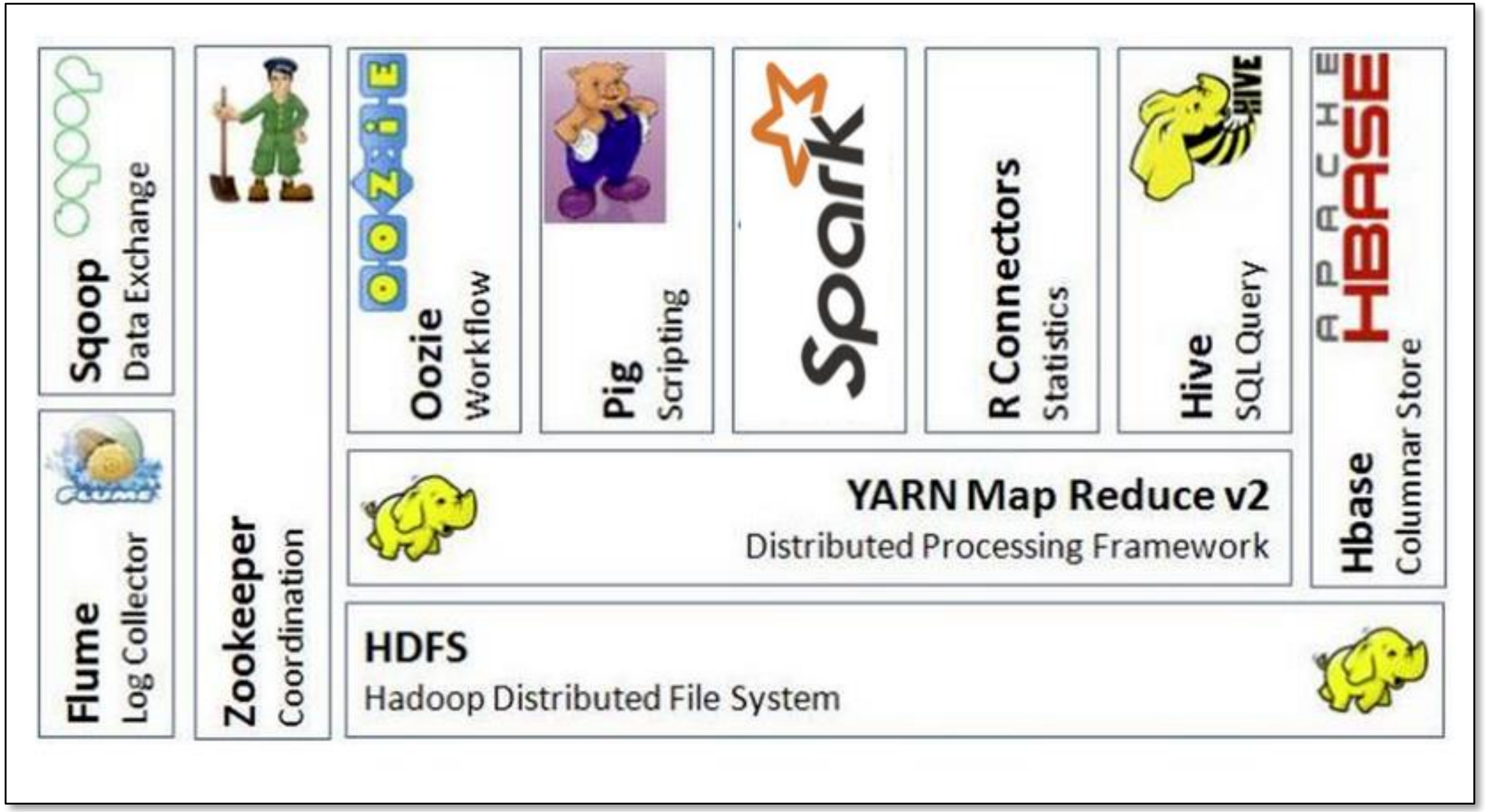
MapReduce Framework

Big Data Timeline



From BU.520.710 AI Essentials for Business

Hadoop Ecosystem



From BU.520.710 AI Essentials for Business

Distributed Processing: Parallelization

- Basic idea: Divide and Conquer
- One machine is not enough to process the data
  - Any company can have more than TBs of data
- Need to distribute the computing over several machines
  - E.g. AlphaZero uses 5000 nodes in parallel
- Basic questions: *How to divide? How to aggregate?*

Map and Reduce

- Map: the **divide** step
- Function to do map is called mapper

- Reduce: the **aggregate** step
- Function to do reduce is caller reducer

- Data processed by mapper and reducer is in the (key, value) format

Key Value Pair

- Recall: Python dictionary

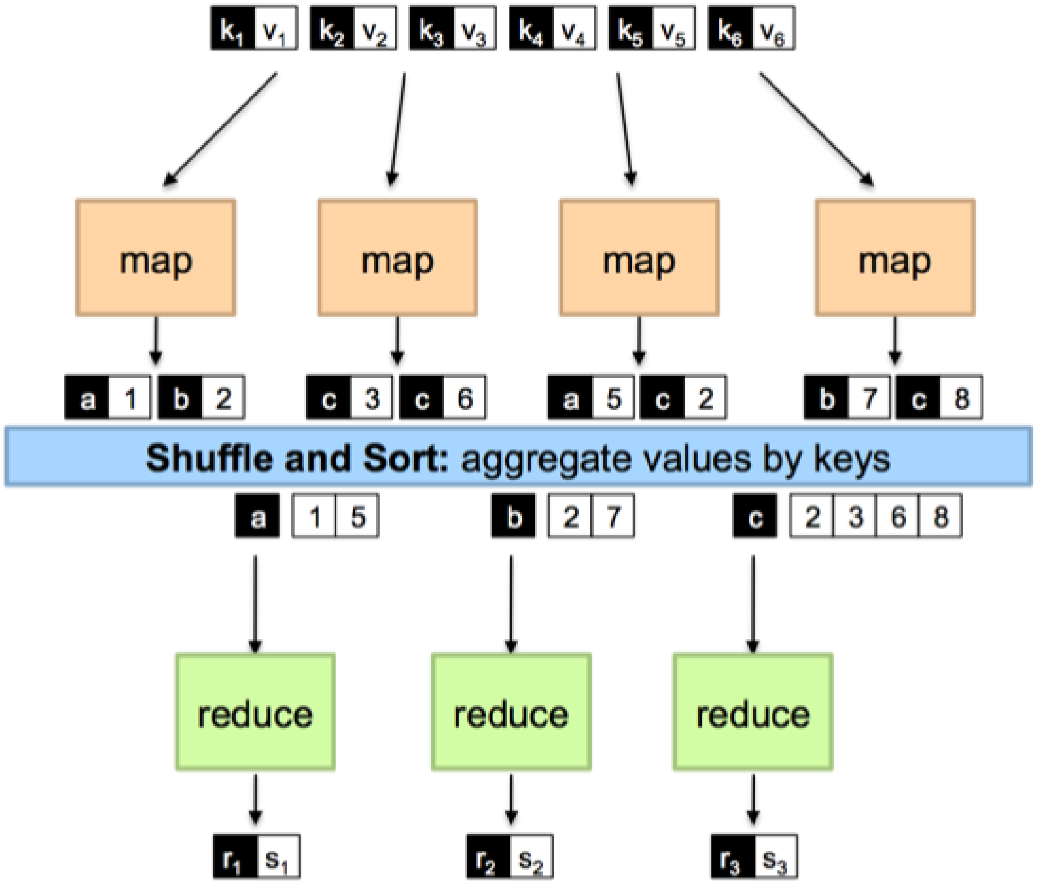
```
scoreBook={}
scoreBook["John"]=80
scoreBook["Jack"]=75

print(scoreBook)
print("Jack receives %s from my course" % convert2letter(scoreBook["Jack"]))
```

```
{'John': 80, 'Jack': 75}
Jack receives C from my course
```

- **Everything** in MapReduce is expressed using key value pair

Illustration



- Programmers define the key and value
- Mapper output a collection of key-value pairs which are input for the reducer
- Output from mapper is shuffled such that all same-key records go to same reducer
- Each reducer group records based on keys, then process each group of values and generate output
- Note: each reducer may receive multiple key sets

Word Count Example: Mapper

- **For example, count the occurrence of each word**
- Each mapper takes a line as input and breaks it into **words**
- Input: each line of the file
  - Key: line #
  - Value: the content of the line
  - Example: "I am a student at Carey Business School and I love Carey "
- Output: a set of key-value pairs
  - Key: one word
  - Value: 1
  - Example: (I, 1), (am, 1), (a, 1), (student, 1), (at, 1), (Carey, 1), (Business, 1), (School, 1), (and, 1), (I, 1), (love, 1), (Carey, 1)

Word Count Example: Reducer

- Each reducer sums the counts for each word
- Input example
  - (I, 1), (am, 1), (a, 1), (student, 1), (at, 1), (Carey, 1), (Business, 1), (School, 1), (and, 1), (I, 1), (love, 1), (Carey, 1)
- Output
  - Key: one word
  - Value: count of word
  - Example: (I, 2), (am, 1), (a, 1), (student, 1), (at, 1), (Carey, 2), (Business, 1), (School, 1), (and, 1), (love, 1)

Exercise

- Input: Final scores from different courses
- Output: Average score for each student

- You know how to achieve it using SQL
- Design it in a MapReduce style

Course ID	Student ID	Score
330740	A	100
330740	B	90
330740	C	80
330760	B	70
330760	C	60
330760	D	50
…	…	…

Exercise (Cont.)

- Mapper
  - Key:
  - Value:

- Reducer
  - Key:
  - Value:
  - Operation:

More About MapReduce

- *What is Google's incentive to design MapReduce?*
  - Refer to Appendix for PageRank and other algorithms built on MapReduce

- Chaining jobs: many problems can be solved with MapReduce by writing **several MapReduce jobs** which run in series to accomplish a goal
  - Each iteration can use the previous iteration's output as input
  - map1->reduce1->map2->reduce2->...

JOHNS HOPKINS
CAREY BUSINESS SCHOOL

Frequent Itemset Mining

Items Purchased Together

- *How companies like Amazon and Walmart know what products are frequently bought together?*
- Previous study reveals most frequent pair of products in store purchase

  - If customer purchase diaper, it's very likely he will also want beer
  - So let's make recommendation based on it
  - Or simply move them closer

https://bigdatabigworld.wordpress.com/2014/11/25/beer-and-nappies/

Frequent Itemset Mining

- Association rule learning/Market basket model/Apriori algorithm
- An unsupervised approach

- Item and basket (also called transaction)
  - Each basket contains a set of items (itemset)
- Assumptions:
  - The number of items in a basket is much smaller than the total number of items
  - The number of baskets are very large

An example

Transaction 1: {milk, bread, cereal}

Transaction 2: {milk, sugar, bread, eggs}

Transaction 3: {milk, bread, butter}

Transaction 4: {sugar, eggs}

Transaction 5: {strawberry, yogurt, cereal, blueberry}

Transaction 6: {strawberry, milk}

Support

- A set of items that appears in many baskets is said to be "frequent"

- Assume there is a number *s*, called the *support threshold*
- If *I* is a set of items, the *support* for *I* is the number of baskets for which *I* is a subset
- We say *I* is frequent if its support is *s* or more

Frequent Singletons

- Among the singleton sets, obviously {milk} and {bread} are quite frequent.
  - {milk} support is 4
  - {bread} support is 3
  - {cereal} support is 2
  - {sugar} support is 2
  - …
  - If we set threshold *s* = 3
    - Two frequent singleton itemsets: {milk}, {bread}

Transaction 1: {milk, bread, cereal}
Transaction 2: {milk, sugar, bread, eggs}
Transaction 3: {milk, bread, butter}
Transaction 4: {sugar, eggs}
Transaction 5: {strawberry, yogurt, cereal, blueberry}
Transaction 6: {strawberry, milk}

Frequent Doubletons

- {milk, bread} support=?
- {milk, cereal} support=?
- {sugar, eggs} support=?
- …
- If we set threshold *s* = 3?

Transaction 1: {milk, bread, cereal}
Transaction 2: {milk, sugar, bread, eggs}
Transaction 3: {milk, bread, butter}
Transaction 4: {sugar, eggs}
Transaction 5: {strawberry, yogurt, cereal, blueberry}
Transaction 6: {strawberry, milk}

Frequent Triples

- {milk, bread, cereal} support=?
- {milk, sugar, eggs} support=?
- {sugar, yogurt, eggs} support=?
- …

Transaction 1: {milk, bread, cereal}
Transaction 2: {milk, sugar, bread, eggs}
Transaction 3: {milk, bread, butter}
Transaction 4: {sugar, eggs}
Transaction 5: {strawberry, yogurt, cereal, blueberry}
Transaction 6: {strawberry, milk}

MapReduce Design - Singletons

- Mapper
  - Key:
  - Value:

- Reducer
  - Key:
  - Value:
  - Operation:

MapReduce Design - Doubletons

- Pseudocode and Python code will be given to you in lab 1
- You need to describe it in Assignment 1 Question 1
  - Hint: you need to get a pair of items

- Triples not required for this course
  - Reference: https://github.com/gautamdasika/Aprioiri-frequent-3-itemsets-with-Hadoop-MapReduce

Plagiarism Check

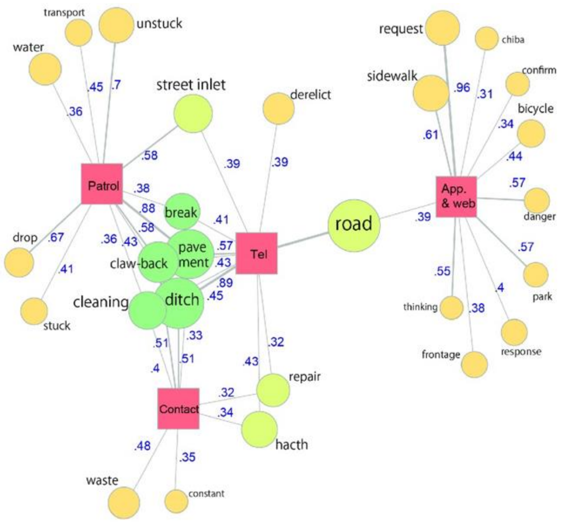
- Suppose we have

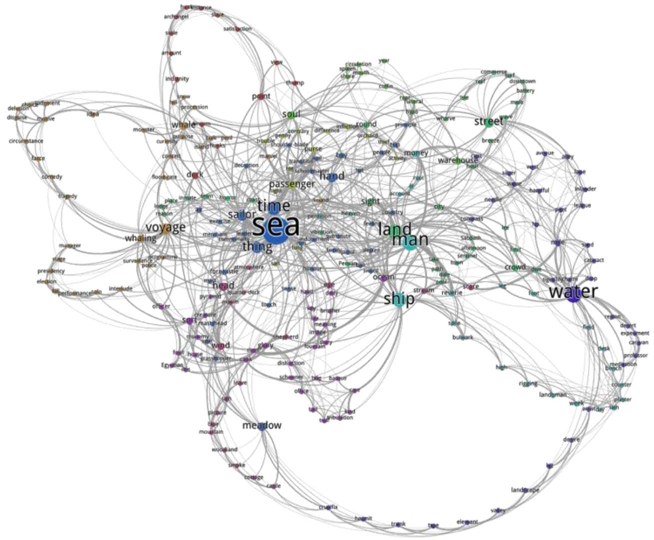
```
Article 1: sentence 1, sentence 2, sentence 3, …
Article 2: sentence 4, sentence 5, sentence 6, …
Article 3: sentence 7, sentence 8, sentence 9, …
Article 4: sentence 1, sentence 5, sentence 8, …
```

- We want to find out two articles having highest matching content
- *What is the transaction/basket? What is the item?*
- Assignment 1 Question 2
  - Hint: you will need to chain two MapReduce jobs

Association Rule Use Cases

- Co-occurrence analysis in text mining
- *How to figure out Elon Musk and Tesla are related?*
  - Humans may know it from our experience
  - How can machine learn it?
- Count number of articles that have different named entities co-occur





Discussion Time

- *Brainstorming other use cases of association rules*

JOHNS HOPKINS
CAREY BUSINESS SCHOOL

Lab 1: AWS S3, EC2 and EMR

AWS Services

- S3: Simple Storage Service
  - Cloud storage service by Amazon
  - https://aws.amazon.com/s3/
- EC2: Elastic Computer Cloud
  - Cloud computing service by Amazon
  - Choice of processor, storage, networking, operating system, etc.
  - https://aws.amazon.com/ec2/
- EMR: Elastic MapReduce
  - Cloud big data platform
  - https://aws.amazon.com/emr/
- Price table: https://aws.amazon.com/pricing/services/

Amazon S3

- Object-level storage
  - If you want to change a part of a file, you must make the change and then reupload the entire modified file
- Stores data as objects within resources called **buckets**
- Bucket name:
  - Universal and unique across all existing bucket names in Amazon S3
  - All lowercase with letters, numbers, dashes; symbols not allowed

EC2: Instance Type

- The instance type that you choose determines
  - Memory
  - Processing power (CPU)
  - Disk space and disk type (storage)
  - Network performance (bandwidth)
- https://aws.amazon.com/ec2/instance-types/
- https://aws.amazon.com/ec2/capacityblocks/pricing/

Instance Type (Cont.)

Instance type naming

- Example: t3.large
- T is the family name
- 3 is the generation number
- Large is the size

	General Purpose	Compute Optimized	Memory Optimized	Accelerated Computing	Storage Optimized
Instance Types	**a1, m4, m5, t2, t3**	**c4, c5**	**r4, r5, x1, z1**	**f1, g3, g4, p2, p3**	**d2, h1, i3**
Use Case	**Broad**	**High performance**	**In-memory databases**	**Machine learning**	**Distributed file systems**

EC2: Key Pair (Optional)

- **Note**: create your own key pair on EC2 if you register your own account on AWS
- We will use learner account key
- A key pair consists of
  - A public key that AWS stores
  - A private key that you store
- It enables secure connections to the instance
- Download if you create a new key pair
  - Save in a safe location
  - Only opportunity for you to save the private key file

MapReduce on AWS

- High level work flow:
  1. Store the data you want to process on AWS
     - Service to use: S3
  2. Create a computing cluster on AWS
     - Services to use: EC2, EMR
  3. Run your large scale computing program on the cluster
- It takes long for AWS to provision, thus we will switch step 1 and step 2 in lab

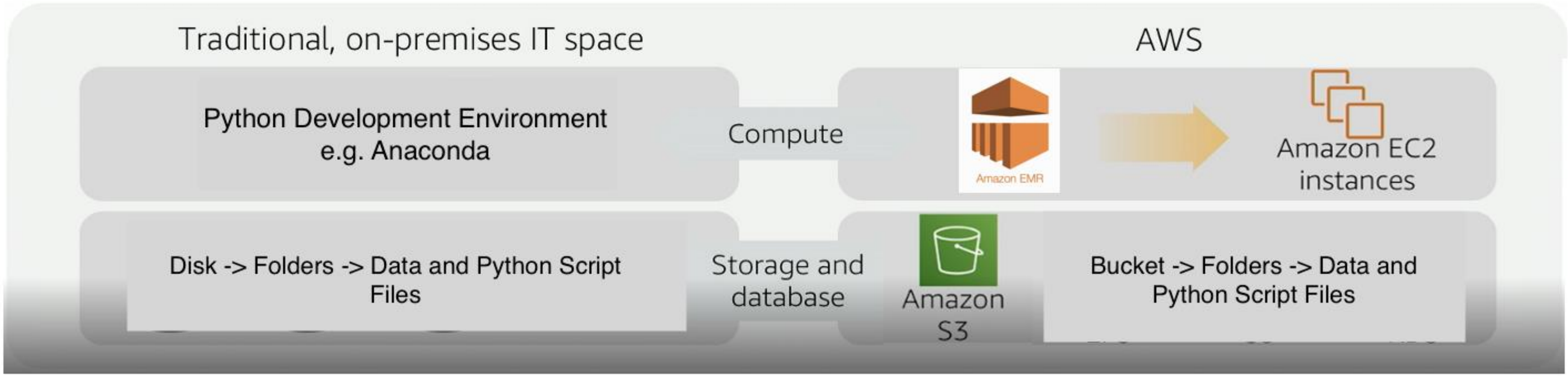
How to: Storage

- Create a S3 bucket
  - As you find an available space on your local disk
- Create a folder for input data to store your data file
  - As you create a new folder on the available disk, then move data files into it
- Create a folder for program scripts to store your script files
  - Same thing as step 2, but for .py files

How to: Compute

- Set up an EMR cluster
  - As you open a Python IDLE
  - Request some instances (machines/nodes), of certain types
- Add a step in cluster
  - As you run a Python program, but here MapReduce job

AWS Steps vs On-Premise Steps



- **Do not use blank space in any file/folder names on AWS**
- Blank/spaces cause troubles in programming

Hadoop Streaming

- A utility that comes with the Hadoop distribution
- It is a generic API which allows writing MapReduce in any language besides Java
  - **Python**
  - Perl
  - R
  - PHP
  - C++

Other Tips

- **Browsers Recommended: Chrome on iOS, or Edge on Windows**
- No coding, but only deploying
  - AWS uses Linux, Python written in Windows has issues
- Lab 1 cost: ~$1
- **On-premise computing vs cloud computing**
  - **Different mode, different experience**
  - **It is normal to feel lost at first time**

Learner Lab Restrictions on EC2 and EMR

- Supported Instance types: nano, micro, small, medium, and large
- Maximum of 9 concurrently running EC2 instances
- Maximum of 32 vCPU used by concurrently running instances
  - Caution: Any attempt to have 20 or more concurrently running instances (regardless of size), will result in immediate deactivation of the AWS account
- Example EMR cluster configuration details:
  - Cluster configuration: Instance groups, Primary: **m4.large**, Core: **m4.large**, Task -1: **m4.large**
  - Provisioning configuration: Core: 1 instance and Task -1: 1 instance
  - Security configuration: EC2 key pair: **vockey**
  - IAM roles: Amazon EMR service role: choose existing: **EMR_DefaultRole**, EC2 instance profile for Amazon EMR: choose existing: **EMR_EC2_DefaultRole**

Next Week

- Advanced Big Data Framework: Spark
- pySpark exercises on Google Colab

JOHNS HOPKINS
CAREY BUSINESS SCHOOL

Appendix: Search Engine and PageRank Algorithm

Search Engine

- Ranking System: first wave of transformative AI

- **<u>Number Game (take a guess):</u>**
- *How many webpages does Google sort through per day?*
- *How many different web pages on internet are there?*

- https://www.worldwidewebsize.com
- *How are search results determined?*

Yahoo! and Early Engine

Early Search Engines

- yahoo! : searchable directory
  - Originally entries were entered and categorized manually
  - Automated some gathering and classification process
  - Descriptive information about indexed sites
  - *Issue?*

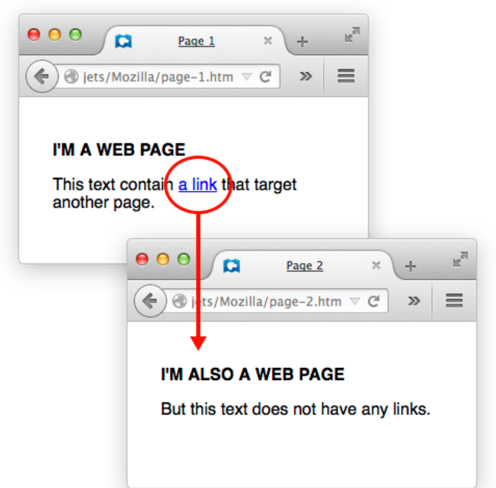
- Crawler based search engine
  - Crawling across the entire text of a web page
  - Listing the terms
  - *Issue?*

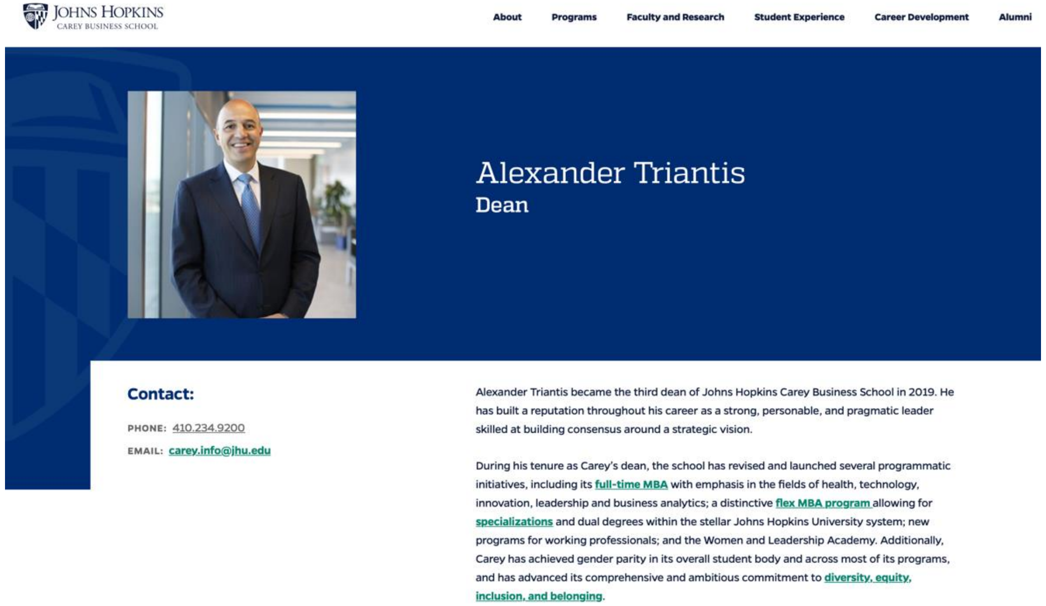
Term Spam

- Lead people to a web site by fooling search engine
  - if you are operating a site selling shirts
  - Add a term such as “movie” to your page, thousands of times
  - Same color as your site background
  - Make small fonts
  - Incorporate a lot of popular terms
- Result? Make search engines useless
- *Then what shall we do?*

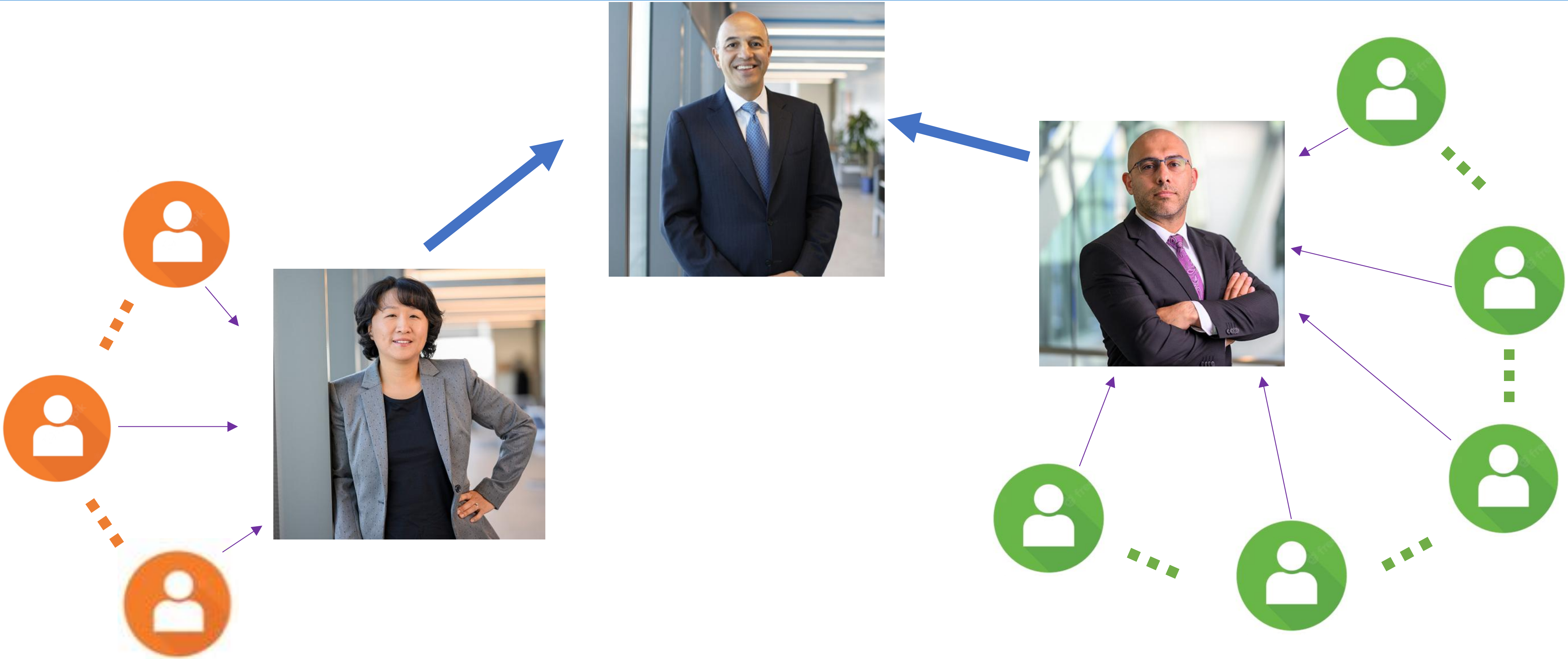
Link Analysis

- Simple Basic Idea: What other pages say about him, is more important than what he says about himself
- Links on the web pages as votes
- Pages “voted” by more pages are more important
  - ”vote with their feet”: users tend to place good/useful links on their own pages, rather than bad/useless ones
- *Can we still fool this type of search engine?*



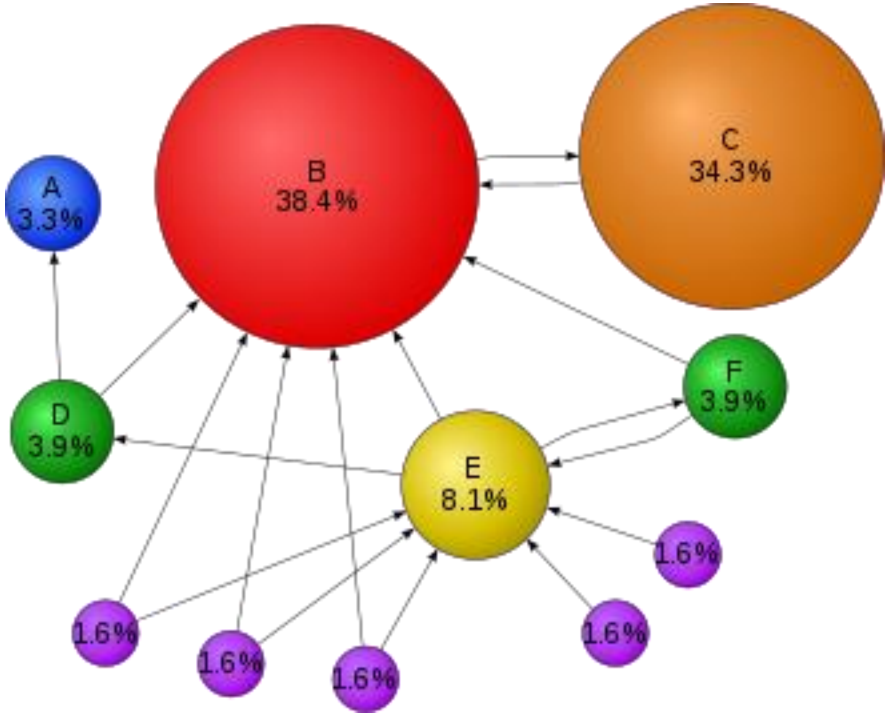


A Social Example



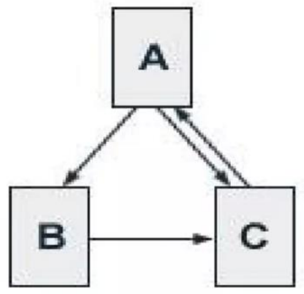
Example (Cont.)

- Insight: “quality” of vote is also important, besides of numbers



- Link passes 1 pt of ranking power
- Start with initial PageRank
- Compute ranking power changes
- Until equilibrium/convergence is reached

Illustration



Consider an imaginary web of 3 web pages.
And the inbound and outbound link structure is as shown in the figure. The calculations can be done by following method :

PR(A) = 0.5 + 0.5 PR(C)
= 0.5 + (0.5*1)
= 1

PR(B) = 0.5 + 0.5 (PR(A) / 2)
= 0.5 + 0.5 (1/2)
= 0.5 + (0.5 * 0.5)
= 0.5 + 0.25
= 0.75

PR(C) = 0.5 + 0.5 ((PR(A) / 2)+ PR (B))
= 0.5 + 0.5 (1/2 + 0.75)
= 0.5 + 0.5 (1.25)
= 0.5 + 0.625
= 1.125

PR (A) = (1-d) + d (PR (T1) / C (T1) + ... + PR (Tn) / C (Tn))

Where,
PR(A) is the PageRank of page A
PR(Ti) is the PageRank of pages Ti which link to page A
C(Ti) is the number of outbound links on page Ti and
d is a damping factor which can be set between 0 and 1

Iteration	PR(A)	PR(B)	PR(C)
0	1	1	1
1	1	0.75	1.125
2	1.0625	0.765625	1.1484375
3	1.07421875	0.76855469	1.15283203
4	1.07641602	0.76910400	1.15365601
5	1.07682800	0.76920700	1.15381050
6	1.07690525	0.76922631	1.15383947
7	1.07691973	0.76922993	1.15384490
8	1.07692245	0.76923061	1.15384592
9	1.07692296	0.76923074	1.15384611
10	1.07692305	0.76923076	1.15384615
11	1.07692307	0.76923077	1.15384615
12	1.07692308	0.76923077	1.15384615

Google and PageRank

- Developed in 1996
- By founders of Google

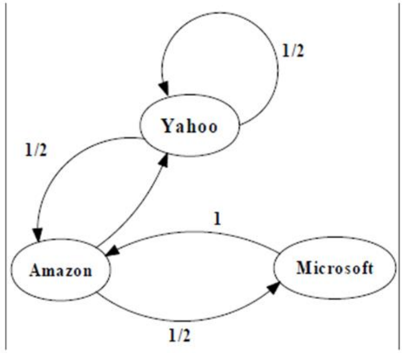
Larry Page and Sergey Brin in 2003

- PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites

Computing PageRank

- Perform matrix-vector multiplication a lot of times
- Until the vector is close to unchanged at one iteration

- Matrix-vector multiplication can be solved using MapReduce



$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

Other Algorithms using MapReduce

- Matrix-vector multiplication
  - Original purpose for Google to implement MapReduce
  - Map use index as the key
  - Reduce sums all the values at the same index
- Matrix-matrix multiplication
  - Similar to matrix-vector, but key is in 2-dimensional
- Relational-algebra operations
  - Selection, projection, union, intersection, difference, natural join, grouping and aggregation
  - Hive and Apache Pig use similar syntax as SQL, but speed up using MapReduce

Optional MapReduce Functions

- Programmers can also include the following two functions to optimize performance:
  - combine (k, v) → <k, v'>
    - Mini-reducers that run in memory after the map phase
    - Used as an optimization to reduce network traffic
  - partition (k', number of partitions) → partition for k'
    - Often a simple hash of the key, e.g., *hash(k') mod n*
    - Divides up key space for parallel reduce operations

References

- Kunpeng Zhang's notes (2019)