Large-scale machine learning course

MapReduce and PageRank

Chloé-Agathe Azencott
Center for Computational Biology (CBIO)
Mines ParisTech – Institut Curie – INSERM U1331
PSL Research University, Paris, France

March 7, 2025

http://cazencott.info chloe-agathe.azencott@minesparis.psl.eu @cazencott@lipn.info

Support **US researchers** facing major attacks from the Trump administration, threatening **science** & **academic freedom.**

Support **US researchers** facing major attacks from the Trump administration, threatening **science** & **academic freedom.**

In **France:**

- campaigns **against post-colonial studies** (alleged “islamo-leftism”);
- 60 out of 75 French universities were in **deficit** in 2024;
- 1 billion EUR **budget cut** to higher ed in 2025.

1. Technical challenges of big data

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day)

[1] http://www.worldwidewebsize.com/

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day)   1 PB = 10^{15} B = 1 000 TB.

[1] http://www.worldwidewebsize.com/

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day) 1 PB = 10^{15} B = 1 000 TB.
  - **reading data from disk:**

[1] http://www.worldwidewebsize.com/

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day)   1 PB = 10^{15} B = 1 000 TB.
  - **reading data from disk:**
    - **Hard disk drives** (HDD): 100-200 MB/s $\Rightarrow$ 50-100 years.

[1] http://www.worldwidewebsize.com/

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day) 1 PB = 10^{15} B = 1 000 TB.
  - **reading data from disk:**
    - **Hard disk drives** (HDD): 100-200 MB/s ⇒ 50-100 years.
    - **Solid state drives** (SSD): 500 MB/s ⇒ 20 years.

[1] http://www.worldwidewebsize.com/

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day) 1 PB = 10^{15} B = 1 000 TB.
  - **reading data from disk:**
    - **Hard disk drives** (HDD): 100-200 MB/s $\Rightarrow$ 50-100 years.
    - **Solid state drives** (SSD): 500 MB/s $\Rightarrow$ 20 years.
- **Google**

[1] http://www.worldwidewebsize.com/

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day)   1 PB = 10^{15} B = 1 000 TB.
  - **reading data from disk:**
    - **Hard disk drives** (HDD): 100-200 MB/s ⇒ 50-100 years.
    - **Solid state drives** (SSD): 500 MB/s ⇒ 20 years.
- **Google**
  - 60 billion[1] webpages indexed.

[1] http://www.worldwidewebsize.com/

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day) 1 PB = 10^{15} B = 1 000 TB.
  - **reading data from disk:**
    - **Hard disk drives** (HDD): 100-200 MB/s $\Rightarrow$ 50-100 years.
    - **Solid state drives** (SSD): 500 MB/s $\Rightarrow$ 20 years.
- **Google**
  - 60 billion[1] webpages indexed.
  - median webpage size: 2 MB.

---

[1] http://www.worldwidewebsize.com/

Motivating examples

- **Facebook:**
  - 300 PB of data stored in 2014
    (4 PB of data created each day) 1 PB = 10^{15} B = 1 000 TB.
  - **reading data from disk:**
    - **Hard disk drives** (HDD): 100-200 MB/s $\Rightarrow$ 50-100 years.
    - **Solid state drives** (SSD): 500 MB/s $\Rightarrow$ 20 years.
- **Google**
  - 60 billion[1] webpages indexed.
  - median webpage size: 2 MB.
    $\Rightarrow$ 120 PB.

[1] http://www.worldwidewebsize.com/

Motivating examples

- Data set with 10^{10} samples and 10^4 features

Motivating examples

- Data set with 10^{10} samples and 10^4 features
- **Memory usage**
  - 10^{14} numbers $\Rightarrow \mathcal{O}(10^{15})$B $=$ 1PB (with a 64-bit encoding)
  - big hard drives

Motivating examples

- Data set with 10^{10} samples and 10^4 features
- **Memory usage**
  - 10^{14} numbers $\Rightarrow \mathcal{O}(10^{15})$B $=$ 1PB (with a 64-bit encoding)
  - big hard drives $\approx$ 20 TB $\Rightarrow$ 50 of them

Motivating examples

- Data set with 10^{10} samples and 10^4 features
- **Memory usage**
  - 10^{14} numbers $\Rightarrow \mathcal{O}(10^{15})$B $=$ 1PB (with a 64-bit encoding)
  - big hard drives $\approx$ 20 TB $\Rightarrow$ 50 of them
- **Time complexity**
  - PCA, ridge regression, etc:

Motivating examples

- Data set with 10^{10} samples and 10^4 features
- **Memory usage**
  - 10^{14} numbers $\Rightarrow \mathcal{O}(10^{15})$B $=$ 1PB (with a 64-bit encoding)
  - big hard drives $\approx$ 20 TB $\Rightarrow$ 50 of them
- **Time complexity**
  - PCA, ridge regression, etc: $\mathcal{O}(np^2)$ operations = 10^{18} operations

    $p^3 \ll np^2$

Motivating examples

- Data set with 10^{10} samples and 10^4 features
- **Memory usage**
  - 10^{14} numbers $\Rightarrow \mathcal{O}(10^{15})\text{B} = 1\text{PB}$ (with a 64-bit encoding)
  - big hard drives $\approx$ 20 TB $\Rightarrow$ 50 of them
- **Time complexity**
  - PCA, ridge regression, etc: $\mathcal{O}(np^2)$ operations = 10^{18} operations

    $p^3 \ll np^2$
  - My laptop: 8 IntelCore i7-8565U CPUs

    Peak performance of one i7-8565U CPU: 115 GFLOPS $\approx 10^{11}$ FLOPS

    FLOPS = FLoating point Operations Per Second

Motivating examples

- Data set with 10^{10} samples and 10^4 features
- **Memory usage**
  - 10^{14} numbers $\Rightarrow \mathcal{O}(10^{15})$B $=$ 1PB (with a 64-bit encoding)
  - big hard drives $\approx$ 20 TB $\Rightarrow$ 50 of them
- **Time complexity**
  - PCA, ridge regression, etc: $\mathcal{O}(np^2)$ operations = 10^{18} operations

    $p^3 \ll np^2$
  - My laptop: 8 IntelCore i7-8565U CPUs
    Peak performance of one i7-8565U CPU: 115 GFLOPS $\approx 10^{11}$ FLOPS
    FLOPS = FLoating point Operations Per Second

Motivating examples

- Data set with 10^{10} samples and 10^4 features
- **Memory usage**
  - 10^{14} numbers $\Rightarrow \mathcal{O}(10^{15})$B $=$ 1PB (with a 64-bit encoding)
  - big hard drives $\approx$ 20 TB $\Rightarrow$ 50 of them
- **Time complexity**
  - PCA, ridge regression, etc: $\mathcal{O}(np^2)$ operations = 10^{18} operations

    $$p^3 \ll np^2$$
  - My laptop: 8 IntelCore i7-8565U CPUs
    Peak performance of one i7-8565U CPU: 115 GFLOPS $\approx 10^{11}$ FLOPS
    FLOPS = FLoating point Operations Per Second

$\Rightarrow$ 10^7 seconds $\approx$ 4 months

Most powerful computing system

- Number 1 of the Nov. 2024 TOP500[2]:

[2] https://www.top500.org/

Most powerful computing system

- Number 1 of the Nov. 2024 TOP500[2]: **El Capitan**
  - Nodes:
    - CPU: AMD 4th Gen EPYC 24C Genoa
    - GPU: AMD Instinct MI300A (24 CPU cores + one GPU)
  - Interconnect switch: HPE Slingshot 64-port switch (12.8 Tb/s)
- 11 039 616 cores (10% CPUs, 90% GPUs)

[2] https://www.top500.org/

Most powerful computing system

- Number 1 of the Nov. 2024 TOP500[2]: **El Capitan**
  - Nodes:
    - CPU: AMD 4th Gen EPYC 24C Genoa
    - GPU: AMD Instinct MI300A (24 CPU cores + one GPU)
  - Interconnect switch: HPE Slingshot 64-port switch (12.8 Tb/s)
- 11 039 616 cores (10% CPUs, 90% GPUs)
- **Peak performance:** 2 746 PFLOPS $\approx 3 \times 10^{18}$ FLOPS

[2] https://www.top500.org/

Most powerful computing system

- Number 1 of the Nov. 2024 TOP500[2]: **El Capitan**
  - Nodes:
    - CPU: AMD 4th Gen EPYC 24C Genoa
    - GPU: AMD Instinct MI300A (24 CPU cores + one GPU)
  - Interconnect switch: HPE Slingshot 64-port switch (12.8 Tb/s)
- 11 039 616 cores (10% CPUs, 90% GPUs)
- **Peak performance:** 2 746 PFLOPS $\approx 3 \times 10^{18}$ FLOPS

  3×10^7 times faster than my laptop with 10^6 times more cores

[2] https://www.top500.org/

Most powerful computing system

- Number 1 of the Nov. 2024 TOP500[2]: **El Capitan**
  - Nodes:
    - CPU: AMD 4th Gen EPYC 24C Genoa
    - GPU: AMD Instinct MI300A (24 CPU cores + one GPU)
  - Interconnect switch: HPE Slingshot 64-port switch (12.8 Tb/s)
- 11 039 616 cores (10% CPUs, 90% GPUs)
- **Peak performance:** 2 746 PFLOPS $\approx 3 \times 10^{18}$ FLOPS

  3×10^7 times faster than my laptop with 10^6 times more cores
- **Power:** 29 581 kW $\approx$ 250 GWh/year

  Power consumption per year and inhabitant in France $\approx$

[2] https://www.top500.org/

Most powerful computing system

- Number 1 of the Nov. 2024 TOP500[2]: **El Capitan**
  - Nodes:
    - CPU: AMD 4th Gen EPYC 24C Genoa
    - GPU: AMD Instinct MI300A (24 CPU cores + one GPU)
  - Interconnect switch: HPE Slingshot 64-port switch (12.8 Tb/s)
- 11 039 616 cores (10% CPUs, 90% GPUs)
- **Peak performance:** 2 746 PFLOPS $\approx 3 \times 10^{18}$ FLOPS

  3×10^{7} times faster than my laptop with 10^{6} times more cores
- **Power:** 29 581 kW $\approx$ 250 GWh/year

  Power consumption per year and inhabitant in France $\approx$ 2 MWh/year

[2] https://www.top500.org/

Most powerful computing system

- Number 1 of the Nov. 2024 TOP500[2]: **El Capitan**
  - Nodes:
    - CPU: AMD 4th Gen EPYC 24C Genoa
    - GPU: AMD Instinct MI300A (24 CPU cores + one GPU)
  - Interconnect switch: HPE Slingshot 64-port switch (12.8 Tb/s)
- 11 039 616 cores (10% CPUs, 90% GPUs)
- **Peak performance:** 2 746 PFLOPS $\approx 3 \times 10^{18}$ FLOPS

  3×10^7 times faster than my laptop with 10^6 times more cores
- **Power:** 29 581 kW $\approx$ 250 GWh/year

  Power consumption per year and inhabitant in France $\approx$ 2 MWh/year

  $\Rightarrow$ equivalent to a city of 250 000 inhabitants

[2] https://www.top500.org/

Disk storage vs volatile memory

- Accessing a **disk block** ≈ 100 µs (SDD) – 10 ms (HDD).
- Accessing **DRAM** ≈ 100 ns.

2. Distributing computations with MapReduce

MapReduce

- Working with large data requires **distributing:**
  - the **data;**
  - the **computations.**

Cluster architecture

- **Single node:**
  - 1–2 **CPU** (central processing unit), each containing 8–32 cores
  - shared **memory** (RAM)
- **Cluster architecture: switches** connect **racks** which contain 16–64 **nodes.**

Image source: Megcluster

Challenges

Large-scale computing for data mining / machine learning problems on **commodity hardware:**

- Distribute **data;**
- Distribute **computations;**
- **Write distributed programs** easily;
- **Robustness** to failure:

  If one node fails every 3 years and you have 1 000 nodes: 1 failure/day.

MapReduce idea

- Divide the data in **chunks;**
- **Keep computation close to the data** (chunk);
- **Redundancy:** store data multiple times;
- **MapReduce: two ingredients:**
  - Storage infrastructure: **distributed file systems**
    Google File System (GFS), Hadoop Distributed File Systems (HDFS);
  - **Programming model:** MapReduce
    Google MapReduce, Hadoop, Spark, etc.

2.1 Distributed file systems

- Goal: store data **persistently**, immune to node failure;
- When to use a DFS:
  - **Huge** files (> 100 GB;)
  - Rare data **modifications;**
  - Frequent data **reads** and **appends.**
- Examples: Google File System (GFS), Hadoop Distributed File Systems (HDFS).

Data chunks

- Data is split in contiguous **chunks;**
- Chunk size: 16-64 MB;
- **Replication:**
  - Each chunk is replicated 2-3 times;
  - Each replicate is kept in a different rack (ideally.)
- **Master node** (aka Name node);
  - stores meta-data about where chunks are stored;
  - may be replicated as well.
- **Accessing data:**
  - talk to master node to know which chunk server to address;
  - talk to chunk server to access the data.
- Keep computation close to the data: chunk server = compute server

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

    Here: split the document in chunks of text.

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

    Here: split the document in chunks of text.

  - **Map step:** extract information from chunk, return as **(key, value) pairs.**
    - All **Map workers** execute the same **Map function.**
    - **Map task:** Executing a Map function on one chunk.

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

    Here: split the document in chunks of text.

  - **Map step:** extract information from chunk, return as **(key, value) pairs.**
    - All **Map workers** execute the same **Map function.**
    - **Map task:** Executing a Map function on one chunk.

    Here: produce one (word, word_occurrence) pair per word occurring in the chunk.

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

    Here: split the document in chunks of text.

  - **Map step:** extract information from chunk, return as **(key, value) pairs.**
    - All **Map workers** execute the same **Map function.**
    - **Map task:** Executing a Map function on one chunk.

    Here: produce one (word, word_occurrence) pair per word occurring in the chunk.

  - **Group** (key, value) pairs **per key.**

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

    Here: split the document in chunks of text.

  - **Map step:** extract information from chunk, return as **(key, value) pairs.**
    - All **Map workers** execute the same **Map function.**
    - **Map task:** Executing a Map function on one chunk.

    Here: produce one (word, word_occurrence) pair per word occurring in the chunk.

  - **Group** (key, value) pairs **per key.**
  - **Reduce step:** transform the **values** for **the same key.**
    - All **Reduce workers** (also called **reducers**) execute the same **Reduce function.**
    - **Reduce task:** Executing a Reduce function on one key.

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

    Here: split the document in chunks of text.

  - **Map step:** extract information from chunk, return as **(key, value) pairs.**
    - All **Map workers** execute the same **Map function.**
    - **Map task:** Executing a Map function on one chunk.

    Here: produce one (word, word_occurrence) pair per word occurring in the chunk.

  - **Group** (key, value) pairs **per key.**

  - **Reduce step:** transform the **values** for **the same key.**
    - All **Reduce workers** (also called **reducers**) execute the same **Reduce function.**
    - **Reduce task:** Executing a Reduce function on one key.

    Here: sum the values of all counts for the same word.

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

    Here: split the document in chunks of text.

  - **Map step:** extract information from chunk, return as **(key, value) pairs.**
    - All **Map workers** execute the same **Map function.**
    - **Map task:** Executing a Map function on one chunk.

    Here: produce one (word, word_occurrence) pair per word occurring in the chunk.

  - **Group** (key, value) pairs **per key.**

  - **Reduce step:** transform the **values** for **the same key.**
    - All **Reduce workers** (also called **reducers**) execute the same **Reduce function.**
    - **Reduce task:** Executing a Reduce function on one key.

    Here: sum the values of all counts for the same word.

  - **Combine** the outputs of the different keys in a final result.

2.2 The MapReduce programming system

- **Exemple** = counting the occurrences of all words that appear in a document

  Overall principle:

  - **Split** the data in chunks.

    Here: split the document in chunks of text.

  - **Map step:** extract information from chunk, return as **(key, value) pairs.**
    - All **Map workers** execute the same **Map function.**
    - **Map task:** Executing a Map function on one chunk.

    Here: produce one (word, word_occurrence) pair per word occurring in the chunk.

  - **Group** (key, value) pairs **per key.**

  - **Reduce step:** transform the **values** for **the same key.**
    - All **Reduce workers** (also called **reducers**) execute the same **Reduce function.**
    - **Reduce task:** Executing a Reduce function on one key.

    Here: sum the values of all counts for the same word.

  - **Combine** the outputs of the different keys in a final result.

    Here: write the (word, total occurrences) to output file.

MapReduce environment

- The **MapReduce environment** takes care of:
  - **Partitioning** the input data;
  - **Scheduling** the program's execution across a set of compute nodes;
  - **Grouping** Map outputs by keys;
  - Handling **node failures;**
  - Managing the **communication** between nodes.
- The **programmer** provides:
  - The **Map** function;
  - The **Reduce** function.

Examples of problems suited for MapReduce

- **Web corpus:**
  - **Data:** each entry is a line describing a web page.
    `URL size data ...`
  - **Problems:** find information (e.g. total number of bytes, frequent words, etc.) for each host.
- **Natural language processing:**
  - **Data:** each entry is a line describing a document.
    `document text`
  - **Problems:** count occurrences of words, phrases, sentences, sequences of 3 words, etc. in the corpus.
- **Database operations:**

  Selection, union, intersection, natural join, aggregation.
- **Linear algebra.**

Matrix-vector multiplication

$M \in \mathbb{R}^{n \times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as

Matrix-vector multiplication

$M \in \mathbb{R}^{n \times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as a list of (i, j, M_{ij}).

Matrix-vector multiplication

$M \in \mathbb{R}^{n \times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as a list of (i, j, M_{ij}).

Version 1: Assume $\vec{v}$ fits in worker memory.

- **Map function:**
  - input = M_{ij}.
  - value = $M_{ij} v_j$.
  - key = i.

Matrix-vector multiplication

$M \in \mathbb{R}^{n \times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as a list of (i, j, M_{ij}).

Version 1: Assume $\vec{v}$ fits in worker memory.

- **Map function:**
  - input = M_{ij}.
  - value = $M_{ij} v_j$.
  - key = i.
- **Reduce function:**

Matrix-vector multiplication

$M \in \mathbb{R}^{n \times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as a list of (i, j, M_{ij}).

Version 1: Assume $\vec{v}$ fits in worker memory.

- **Map function:**
  - input = M_{ij}.
  - value = $M_{ij} v_j$.
  - key = i.
- **Reduce function:**
  - Sum all values for key i.
  - output = (i, x_i).

Matrix-vector multiplication

$M \in \mathbb{R}^{n \times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as a list of (i, j, M_{ij}).

Version 2: Assume $\vec{v}$ does not fit in worker memory.

Matrix-vector multiplication

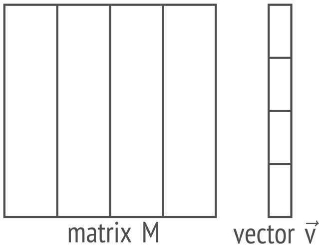
$M \in \mathbb{R}^{n \times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as a list of (i, j, M_{ij}).

Version 2: Assume $\vec{v}$ does not fit in worker memory.

- Divide M and $\vec{v}$ into K **stripes** $\mathcal{I}_k = [km \ldots (k+1)m - 1]$



Matrix-vector multiplication

$M \in \mathbb{R}^{n\times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as a list of (i, j, M_{ij}).

Version 2: Assume $\vec{v}$ does not fit in worker memory.

- Divide M and $\vec{v}$ into K **stripes** $\mathcal{I}_k = [km \ldots (k+1)m - 1]$
- Each **Map task** gets:
  - one stripe of M (fits in memory because M sparse);
  - the corresponding stripe of $\vec{v}$, in memory.
- and outputs: $(i, \sum_{j \in \mathcal{I}_k} M_{ij} v_j)$

Matrix-vector multiplication

$M \in \mathbb{R}^{n \times n}$ and $\vec{v} \in \mathbb{R}^n$.

Goal: compute $x_i = \sum_{j=1}^{n} M_{ij} v_j$ for all $i = 1, \ldots, n$.

n large but M **sparse:** store M as a list of (i, j, M_{ij}).

Version 2: Assume $\vec{v}$ does not fit in worker memory.

- Divide M and $\vec{v}$ into K **stripes** $\mathcal{I}_k = [km \ldots (k+1)m - 1]$
- Each **Map task** gets:
  - one stripe of M (fits in memory because M sparse);
  - the corresponding stripe of $\vec{v}$, in memory.
- and outputs: $(i, \sum_{j \in \mathcal{I}_k} M_{ij} v_j)$
- The **Reduce function** is the same as before (sum all values for key i).

Matrix-matrix multiplication

$M \in \mathbb{R}^{n\times m}, P \in \mathbb{R}^{m\times p}$ Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik}P_{kj}$

Version 1: one MapReduce step.

Matrix-matrix multiplication

$M \in \mathbb{R}^{n\times m}, P \in \mathbb{R}^{m\times p}$ Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik}P_{kj}$

Version 1: one MapReduce step.

- **Reduce step:** For key (i, j) :
  - Multiply the values that have the same k.
  - Sum these products and return $(i, j), Q_{ij}$.

Matrix-matrix multiplication

$M \in \mathbb{R}^{n\times m}, P \in \mathbb{R}^{m\times p}$ Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik}P_{kj}$

Version 1: one MapReduce step.

- **Map step:**
  - Input = (u, v, A_{uv}, δ) where $\delta = 0$ if $A = M$ and 1 otherwise
  - Output if $A = M$: for $j = 1, \ldots, p$, key=(u, j), value=(v, M_{uv})
  - Output if $A = P$: for $i = 1, \ldots, n$, key=(i, v), value=(u, P_{uv})
- **Reduce step:** For key (i, j) :
  - For each key (i, j), we have all M_{ik} and all P_{kj} values.
  - Multiply the values that have the same k.
  - Sum these products and return $(i, j), Q_{ij}$.

Matrix-matrix multiplication

$M \in \mathbb{R}^{n \times m}$, $P \in \mathbb{R}^{m \times p}$ Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik} P_{kj}$

Version 2: two MapReduce steps

Matrix-matrix multiplication

$M \in \mathbb{R}^{n \times m}, P \in \mathbb{R}^{m \times p}$ Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik} P_{kj}$

Version 2: two MapReduce steps

- **Map step 1:**
  - Input = (u, v, A_{uv}, δ) where $\delta = 0$ if $A = M$ and 1 otherwise.
  - Output if $A = M$: key=v, value=$(0, u, M_{uv})$.
  - Output if $A = P$: key=u, value=$(1, v, P_{uv})$.

Matrix-matrix multiplication

$M \in \mathbb{R}^{n\times m}, P \in \mathbb{R}^{m\times p}$ Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik}P_{kj}$

Version 2: two MapReduce steps

- **Map step 1:**
  - Input = (u, v, A_{uv}, δ) where $\delta = 0$ if $A = M$ and 1 otherwise.
  - Output if $A = M$: key=v, value=$(0, u, M_{uv})$.
  - Output if $A = P$: key=u, value=$(1, v, P_{uv})$.
- **Reduce step 1:** For key k :
  - Input = $(0, i, M_{ik})$ for all $1 \leq i \leq n$ and $(1, j, P_{kj})$ for all $1 \leq j \leq p$.
  - Return $((i, j), M_{ik}P_{kj})$ for all $1 \leq i \leq n$ and $1 \leq j \leq p$.

Matrix-matrix multiplication

$M \in \mathbb{R}^{n \times m}, P \in \mathbb{R}^{m \times p}$ Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik} P_{kj}$

Version 2: two MapReduce steps

- **Map step 1:**
  - Input = (u, v, A_{uv}, δ) where $\delta = 0$ if $A = M$ and 1 otherwise.
  - Output if $A = M$: key=v, value=$(0, u, M_{uv})$.
  - Output if $A = P$: key=u, value=$(1, v, P_{uv})$.
- **Reduce step 1:** For key k :
  - Input = $(0, i, M_{ik})$ for all $1 \leq i \leq n$ and $(1, j, P_{kj})$ for all $1 \leq j \leq p$.
  - Return $((i, j), M_{ik} P_{kj})$ for all $1 \leq i \leq n$ and $1 \leq j \leq p$.
- **Map step 2:** Identity.

Matrix-matrix multiplication

$M \in \mathbb{R}^{n\times m}, P \in \mathbb{R}^{m\times p}$ Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik}P_{kj}$

Version 2: two MapReduce steps

- **Map step 1:**
  - Input = (u, v, A_{uv}, δ) where $\delta = 0$ if $A = M$ and 1 otherwise.
  - Output if $A = M$: key=v, value=$(0, u, M_{uv})$.
  - Output if $A = P$: key=u, value=$(1, v, P_{uv})$.
- **Reduce step 1:** For key k :
  - Input = $(0, i, M_{ik})$ for all $1 \leq i \leq n$ and $(1, j, P_{kj})$ for all $1 \leq j \leq p$.
  - Return $((i, j), M_{ik}P_{kj})$ for all $1 \leq i \leq n$ and $1 \leq j \leq p$.
- **Map step 2:** Identity.
- **Reduce step 2:** For key (i, j), sum all values and return.

Data flow

- **Input data** and **final output** are stored on a **distributed file system (DFS)**.

  The **scheduler** manages to schedule Map tasks near the physical location of the input data they need.

- **Intermediate results** are stored on **local file systems** of Map and Reduce workers.

- MapReduce operations can be **stacked:** the Reduce output becomes the input to a new Map task (as in our second matrix-matrix multiplication example).

The master node

- The **master node** takes care of coordination:
  - Tracking **task status:** idle / in progress / completed.
  - Scheduling **idle tasks** when workers become available.
  - Upon **task completion:** receiving (from worker) the location and size of the intermediate files (to be sent to reducers).
  - **Pushing info** to reducers:

    Grouping tasks per key.
    Hash keys into R buckets $\Rightarrow$ create R files.
  - Detecting **failure.**

Dealing with failures

- Master node periodically **pings** workers to detect whether they're still up.
- **Map worker failure:**
  - Outputs of Map tasks already executed by this worker are unavailable.
  - Map tasks **completed/in progress** at worker are reset to idle.
- **Reduce worker failure:**

  Reduce tasks **in progress are reset to idle.**
- **Master failure:**

  The whole MapReduce operation is aborted with notification.

Number of Map and Reduce tasks

- M **Map tasks:**
  - M much larger than the number of nodes in the cluster.
  - One DFS chunk per Map task.
    Better dynamic load balancing and recovery from worker failures.
- R **Reduce tasks:**
  - If R is large:
    - More final output files.
    - **Skew:** different processing times for different Reduce tasks.
      Due to differences in **value list length.**
      Running several Reduce tasks at the same node **averages out** these differences.
  - If R is too small:
    - The amount of data that each reducer must process may become too large.

Backup tasks

- **Problem:** **slow workers** significantly increase job completion time.
  - Other jobs running on the same machine.
  - Bad disks.
  - Weird things.
- **Solution:** spawn **backup copies** of tasks.
  - Near end of **phase.**
  - Once one of the copies finishes, stop all of them.

Combiners

- When Map tasks produce many key-value pairs **for the same key**

  E.g. word counting.

- **Combiner:** pre-aggregates values in the mapper.
  - **Combine** $\{(k, v_1), (k, v_2), \ldots, (k, v_m)\}$ in a unique (k, v).
  - Combiner is usually the same as the **Reduce** function.

    E.g. word counting: combiner sums values.
  - The Reduce function must be **commutative** and **associative.**
  - Save **communication cost**: less data needs to be copied and shuffled.

2.3 Algorithmic costs

- **Communication cost:** total input/output of all processes:
  - input file (= input to all Map tasks).
  - 2 x sum of all files communicated from Map processes to Reduce processes (= output of all Map tasks + input to all Reduce tasks).
  - sum of all Reduce output files (= output of all Reduce tasks).
- **Elapsed communication cost:** maximum input/output along any path **= wall-clock** time.
  - largest input+output files for any Map process.
  - largest input+output files for any Reduce process.
- **Computation cost:** total runtime of all processes.
- **Elapsed computation cost:** maximum runtime.
- Typically, one of the **communication** and the **computation** costs dominates.

2.4 Complexity Theory

- Studying the balance between **communication** and **computation** costs.

- **Reducer size** q : Upper bound on the number of values that can be associated with a single key.

  Small reducer size:

  - many reducers;
  - high degree of parallelism;
  - low **computation cost,**
    especially if the Reduce tasks can be executed in main memory.

- **Replication rate** r : Number of key-value pairs per **input.**
  Average **communication cost** from Map tasks to Reduce tasks.

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$.   Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik}P_{kj}$.

- **Map step:** $M_{ik} \mapsto \{((i,j),(k,M_{ik}))\}_{j=1,2,\ldots,n}$.
  $P_{kj} \mapsto \{((i,j),(k,P_{kj}))\}_{i=1,2,\ldots,n}$.
- **Reduce step:** $(i,j) \mapsto$ sum of pairwise products.
- **Reducer size** q : Upper bound on the number of values that can be associated with a single key.
- **Replication rate** r : Number of key-value pairs per input.

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n \times n}, P \in \mathbb{R}^{n \times n}$.   Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik} P_{kj}$.

- **Map step:** $M_{ik} \mapsto \{((i,j),(k,M_{ik}))\}_{j=1,2,\dots,n}$.
  $P_{kj} \mapsto \{((i,j),(k,P_{kj}))\}_{i=1,2,\dots,n}$.
- **Reduce step:** $(i,j) \mapsto$ sum of pairwise products.
- **Reducer size** q : Upper bound on the number of values that can be associated with a single key.
- **Replication rate** r : Number of key-value pairs per input.

  Here: n key-value pairs are created per input element.

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$. Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik}P_{kj}$.

- **Map step:** $M_{ik} \mapsto \{((i,j),(k,M_{ik}))\}_{j=1,2,\ldots,n}$.
  $P_{kj} \mapsto \{((i,j),(k,P_{kj}))\}_{i=1,2,\ldots,n}$.

- **Reduce step:** $(i,j) \mapsto$ sum of pairwise products.

- **Reducer size** q : Upper bound on the number of values that can be associated with a single key.

  Here: Each key (i,j) is associated with $2n$ values.

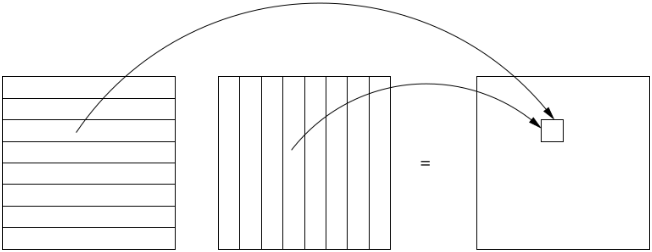
- **Replication rate** r : Number of key-value pairs per input.

  Here: n key-value pairs are created per input element.

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$. Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik}P_{kj}$.

Idea: Reduce communication by **grouping inputs:**
divide M in G bands of $\frac{n}{G}$ rows each; P in G bands of $\frac{n}{G}$ columns each.



- **Map step:** $M_{ik} \mapsto (g_M(i), g_P) : (i, k, M_{ik})$ for $g_P = 1, 2, \ldots, G$.
  $P_{kj} \mapsto (g_M, g_P(j)) : (k, j, P_{kj})$ for $g_M = 1, 2, \ldots, G$.
- **Reduce step:** $(g_M, g_P) \mapsto (i, j) : \sum_{k=1}^{m} M_{ik}P_{kj}$ for all $i \in g_M$ and $j \in g_P$.

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$.   Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik}P_{kj}$.

Idea: Reduce communication by grouping inputs: divide M in G bands of $\frac{n}{G}$ rows each; P in G bands of $\frac{n}{G}$ columns each.

- **Map step:** $M_{ik} \mapsto (g_M(i), g_P) : (i, k, M_{ik})$ for $g_P = 1, 2, \ldots, G$.
  $P_{kj} \mapsto (g_M, g_P(j)) : (k, j, P_{kj})$ for $g_M = 1, 2, \ldots, G$.
- **Reduce step:** $(g_M, g_P) \mapsto (i, j) : \sum_{k=1}^{m} M_{ik}P_{kj}$ for all $i \in g_M$ and $j \in g_P$.

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$. Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik}P_{kj}$.

Idea: Reduce communication by grouping inputs: divide M in G bands of $\frac{n}{G}$ rows each; P in G bands of $\frac{n}{G}$ columns each.

- **Map step:** $M_{ik} \mapsto (g_M(i), g_P) : (i, k, M_{ik})$ for $g_P = 1, 2, \ldots, G$.
  $P_{kj} \mapsto (g_M, g_P(j)) : (k, j, P_{kj})$ for $g_M = 1, 2, \ldots, G$.
- **Reduce step:** $(g_M, g_P) \mapsto (i, j) : \sum_{k=1}^{m} M_{ik}P_{kj}$ for all $i \in g_M$ and $j \in g_P$.
- **Reducer size:**

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$. Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik}P_{kj}$.

Idea: Reduce communication by grouping inputs: divide M in G bands of $\frac{n}{G}$ rows each; P in G bands of $\frac{n}{G}$ columns each.

- **Map step:** $M_{ik} \mapsto (g_M(i), g_P) : (i, k, M_{ik})$ for $g_P = 1, 2, \ldots, G$.
  $P_{kj} \mapsto (g_M, g_P(j)) : (k, j, P_{kj})$ for $g_M = 1, 2, \ldots, G$.
- **Reduce step:** $(g_M, g_P) \mapsto (i, j) : \sum_{k=1}^{m} M_{ik}P_{kj}$ for all $i \in g_M$ and $j \in g_P$.
- **Reducer size:**

$$q = \underbrace{\left(\overbrace{n}^{\text{cols}} \times \overbrace{\frac{n}{G}}^{\text{rows}}\right)}_{\text{from M}} + \underbrace{\left(n \times \frac{n}{G}\right)}_{\text{from P}} = \frac{2n^2}{G}.$$

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$. Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik} P_{kj}$.

Idea: Reduce communication by grouping inputs: divide M in G bands of $\frac{n}{G}$ rows each; P in G bands of $\frac{n}{G}$ columns each.

- **Map step:** $M_{ik} \mapsto (g_M(i), g_P) : (i, k, M_{ik})$ for $g_P = 1, 2, \ldots, G$.
  $P_{kj} \mapsto (g_M, g_P(j)) : (k, j, P_{kj})$ for $g_M = 1, 2, \ldots, G$.
- **Reduce step:** $(g_M, g_P) \mapsto (i, j) : \sum_{k=1}^{m} M_{ik} P_{kj}$ for all $i \in g_M$ and $j \in g_P$.
- **Reducer size:**

$$q = \underbrace{\left(\overbrace{n}^{\text{cols}} \times \overbrace{\frac{n}{G}}^{\text{rows}} \right)}_{\text{from M}} + \underbrace{\left(n \times \frac{n}{G} \right)}_{\text{from P}} = \frac{2n^2}{G}.$$

- **Replication rate:**

Example: One pass matrix-matrix multiplication

$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$. Goal: compute $Q_{ij} = \sum_{k=1}^{n} M_{ik}P_{kj}$.

Idea: Reduce communication by grouping inputs: divide M in G bands of $\frac{n}{G}$ rows each; P in G bands of $\frac{n}{G}$ columns each.

- **Map step:** $M_{ik} \mapsto (g_M(i), g_P) : (i, k, M_{ik})$ for $g_P = 1, 2, \ldots, G$.
  $P_{kj} \mapsto (g_M, g_P(j)) : (k, j, P_{kj})$ for $g_M = 1, 2, \ldots, G$.
- **Reduce step:** $(g_M, g_P) \mapsto (i, j) : \sum_{k=1}^{m} M_{ik}P_{kj}$ for all $i \in g_M$ and $j \in g_P$.
- **Reducer size:**

$$q = \underbrace{\left(\overbrace{n}^{\text{cols}} \times \overbrace{\frac{n}{G}}^{\text{rows}}\right)}_{\text{from M}} + \underbrace{\left(n \times \frac{n}{G}\right)}_{\text{from P}} = \frac{2n^2}{G}.$$

- **Replication rate:** $r = G$.

Example: One pass matrix-matrix multiplication

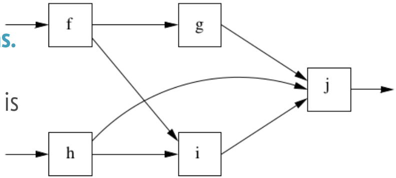
$M \in \mathbb{R}^{n\times n}, P \in \mathbb{R}^{n\times n}$. Goal: compute $Q_{ij} = \sum_{k=1}^{m} M_{ik}P_{kj}$.

Idea: Reduce communication by grouping inputs: divide M in G bands of $\frac{n}{G}$ rows each; P in G bands of $\frac{n}{G}$ columns each.

- **Reducer size:** $q = \frac{2n^2}{G}$.
- **Replication rate:** $r = G$.
- Consistent with previous results when $G = n$.
- $r = \frac{2n^2}{q}$: the replication rate varies inversely with the reducer size.
- Two-pass matrix-matrix multiplication: see Chapter 2.6.7 of Mining of Massive Datasets.

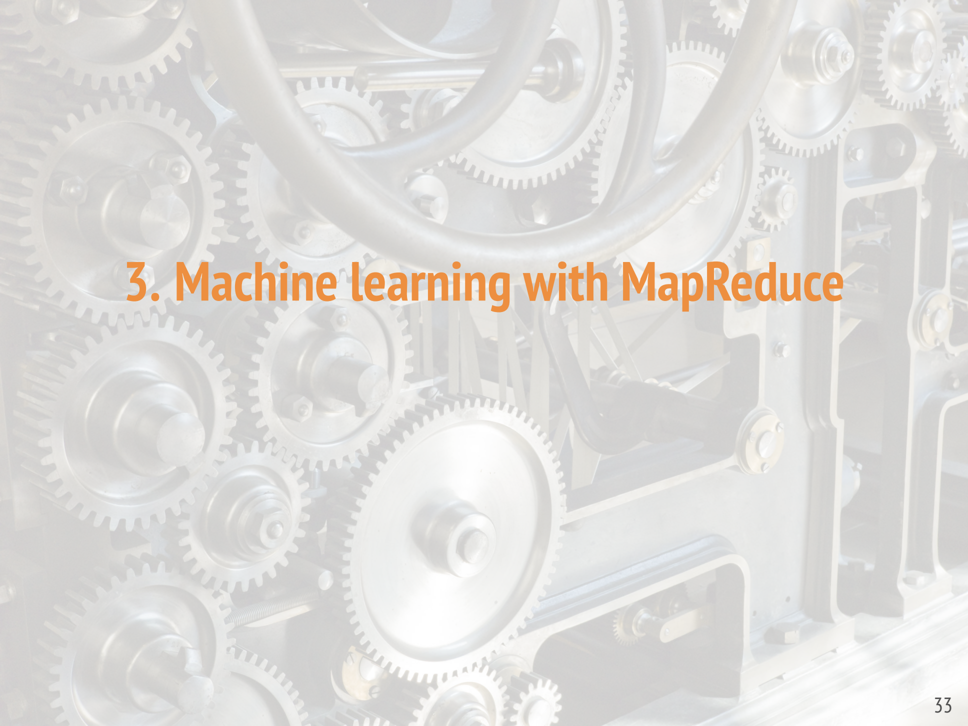
2.5 Extensions to MapReduce

- **Workflow Systems:**
  - Generalize the concept of a **cascade of functions.**
  - MapReduce: two-step workflow.
  - **Flow graph:** arc a $\rightarrow$ b indicates the input of b is the output of a.



- **Spark:**
  - copes more efficiently with failures.
  - groups tasks more efficiently.
  - integrates loops and function libraries.

- **TensorFlow:**
  - Similar to Spark.
  - Data is arranged in **tensors** = multi-dimensional matrices.
  - Specifically designed for machine learning.



3. Machine learning with MapReduce

Overall principle

- Many machine learning algorithms rely heavily on:

Overall principle

- Many machine learning algorithms rely heavily on:
  - **matrix-vector multiplication;**
  - **matrix-matrix multiplications.**

Overall principle

- Many machine learning algorithms rely heavily on:
  - **matrix-vector multiplication;**
  - **matrix-matrix multiplications.**

- Many machine learning algorithms can be written in **summation form:** computations can be expressed as sums over data points.

Linear regression

n observations in p dimensions: $X \in \mathbb{R}^{n \times p}$.   n labels: $\vec{y} \in \mathbb{R}^n$.

Suppose n large and p small.

- **Linear regression:** Solve

Linear regression

n observations in p dimensions: $X \in \mathbb{R}^{n \times p}$. n labels: $\vec{y} \in \mathbb{R}^n$.

Suppose n large and p small.

- **Linear regression:** Solve

$$\arg\min_{\vec{w} \in \mathbb{R}^p} \sum_{i=1}^{n} \left(\vec{w}^\top \vec{x}_i - y_i \right)^2 .$$

Linear regression

n observations in p dimensions: $X \in \mathbb{R}^{n \times p}$. n labels: $\vec{y} \in \mathbb{R}^n$.

Suppose n large and p small.

- **Linear regression:** Solve
$$\arg\min_{\vec{w} \in \mathbb{R}^p} \sum_{i=1}^{n} \left(\vec{w}^\top \vec{x}_i - y_i \right)^2 .$$

- **Exact solution:**

Linear regression

n observations in p dimensions: $X \in \mathbb{R}^{n \times p}$.   n labels: $\vec{y} \in \mathbb{R}^{n}$.

Suppose n large and p small.

- **Linear regression:** Solve
$$\arg\min_{\vec{w} \in \mathbb{R}^p} \sum_{i=1}^{n} \left(\vec{w}^\top \vec{x}_i - y_i\right)^2 .$$
- **Exact solution:** $\vec{w}^* = (X^\top X)^{-1} X^\top \vec{y}$.

Linear regression

n observations in p dimensions: $X \in \mathbb{R}^{n \times p}$. n labels: $\vec{y} \in \mathbb{R}^n$.

Suppose n large and p small.

- **Linear regression:** Solve

$$\arg\min_{\vec{w} \in \mathbb{R}^p} \sum_{i=1}^{n} \left(\vec{w}^\top \vec{x}_i - y_i \right)^2 .$$

- **Exact solution:** $\vec{w}^* = (X^\top X)^{-1} X^\top \vec{y}$.
- Computationally **costly** operations:

Linear regression

n observations in p dimensions: $X \in \mathbb{R}^{n\times p}$. n labels: $\vec{y} \in \mathbb{R}^n$.

Suppose n large and p small.

- **Linear regression:** Solve

$$\arg\min_{\vec{w}\in\mathbb{R}^p} \sum_{i=1}^{n} \left(\vec{w}^\top \vec{x}_i - y_i\right)^2 .$$

- **Exact solution:** $\vec{w}^* = (X^\top X)^{-1} X^\top \vec{y}$.
- Computationally **costly** operations:
  - Computing $X^\top X$ **summation form:** $\sum_{i=1}^n \vec{x}_i \vec{x}_i^\top$.
  - Computing $X^\top y$ **summation form:** $\sum_{i=1}^n \vec{x}_i y_i$.
  - The matrix inversion is not a problem because p is small.

Ridge regularization

$$J(\vec{w}) = \ell(y_i f_{\vec{w}}(x_i)) + \lambda \vec{w}^\top \vec{w}.$$

- Examples of loss ℓ:
  - **Logistic regression:** $\ell(u) = \log(1 + \exp(-u))$.
  - **Ridge regression:** $\ell(u) = (1 - u)^2$.
  - **Support vector machines:** $\ell(u) = \max(1 - u, 0)$.
- **Newton-Raphson:** $\vec{w} \leftarrow \vec{w} - \left[\nabla^2_{\vec{w}} J(\vec{w})\right]^{-1} \nabla_{\vec{w}} J(\vec{w})$.
- The gradients and Hessian can be written in summation form.

Other algorithms

- **PCA**
  - Compute the principal eigenvectors of the covariance matrix
$$\Sigma = \frac{1}{n}(X - \vec{\mu})^\top (X - \vec{\mu}).$$
  - $\vec{\mu} = \frac{1}{n}\sum_{i=1}^{n} \vec{x}_i \vec{x}_i^\top$.
  - Σ has small dimension.
- **k-means**
  - Computing cluster assignment: $\arg\min_{c \in \{1,\ldots,k\}} ||\vec{x}_i - \vec{\mu}_c||$.
  - Computing centroids: $\vec{\mu}_c = \frac{1}{|C_c|}\sum_{j:C_j=i} \vec{x}_j$.

4. Link analysis with PageRank

Link analysis: searching the web

- The web = **directed graph** of **nodes** (webpages) connected by **directed edges** (hyperlinks).
- Other examples:
  - **Media networks:**
    Connections between political blogs, Facebook communities, newspaper articles.
  - **Information networks:**
    Citation networks, Internet.
  - **Technical networks:**
    Highways, seven bridges of Königsberg.
  - **Biological networks:**
    Systems biology organizes knowledge about biomolecules (genes, proteins, etc.) in networks.
- **How to organize the web?**
  - Web **directories.**
  - Web search.

Web search

- **Challenges:**
  - **Trust:** the web contains many sources of information – including spam.
  - What is the **best** answer to the web query “machine learning”?

    No single right answer

- **Idea:** Good pages about machine learning might all be pointing to many relevant pages, and conversely.

Web search

- **Challenges:**
  - **Trust:** the web contains many sources of information – including spam.
  - What is the **best** answer to the web query "machine learning"?

    No single right answer

- **Idea:** Good pages about machine learning might all be pointing to many relevant pages, and conversely.

  $\Rightarrow$ Use **link structure** to rank pages.

- Use links as **votes**: a page is more important if it has more **incoming** links.

- Are all links equally important?

Web search

- **Challenges:**
  - **Trust:** the web contains many sources of information – including spam.
  - What is the **best** answer to the web query "machine learning"?

    No single right answer

- **Idea:** Good pages about machine learning might all be pointing to many relevant pages, and conversely.

  $\Rightarrow$ Use **link structure** to rank pages.

- Use links as **votes**: a page is more important if it has more **incoming** links.

- Are all links equally important?

  Links from important pages count more.

  The question is **recursive.**

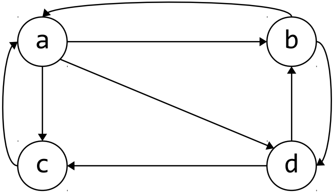
PageRank

- **PageRank** is one of the many ingredients used by the Google **search engine** to rank webpages.
  Trivia: It is named after Larry Page, and not webpages.
- **Recursive formulation:**
  - Each link's vote is proportional to the **importance** of its source page:
  - If page j with importance r_j has d_j out-links, each link gets r_j/d_j votes
  - **Importance** of page i :

$$r_i = \sum_{j:j\to i\in\mathcal{E}} \frac{r_j}{d_j}.$$

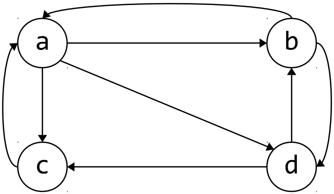
PageRank example

$$r_i = \sum_{j:j \to i \in \mathcal{E}} \frac{r_j}{d_j}$$



PageRank example

$$r_i = \sum_{j: j \to i \in \mathcal{E}} \frac{r_j}{d_j}$$

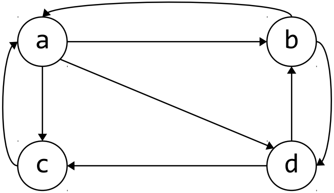


Equations:

- $r_a = \frac{r_b}{2} + r_c$.
- $r_b = \frac{r_a}{3} + \frac{r_d}{2}$.
- $r_c = \frac{r_a}{3} + \frac{r_d}{2}$.
- $r_d = \frac{r_a}{3} + \frac{r_b}{2}$.

PageRank example

$$r_i = \sum_{j:j \to i \in \mathcal{E}} \frac{r_j}{d_j}$$



Equations:

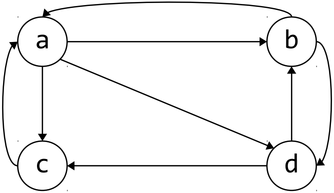
- $r_a = \frac{r_b}{2} + r_c.$
- $r_b = \frac{r_a}{3} + \frac{r_d}{2}.$
- $r_c = \frac{r_a}{3} + \frac{r_d}{2}.$
- $r_d = \frac{r_a}{3} + \frac{r_b}{2}.$

- 4 equations, 4 unknowns, no constants.

  No **unique solution:** all solutions are equivalent modulo a scale factor.

PageRank example

$$r_i = \sum_{j: j \to i \in \mathcal{E}} \frac{r_j}{d_j}$$



Equations:

- $r_a = \frac{r_b}{2} + r_c.$
- $r_b = \frac{r_a}{3} + \frac{r_d}{2}.$
- $r_c = \frac{r_a}{3} + \frac{r_d}{2}.$
- $r_d = \frac{r_a}{3} + \frac{r_b}{2}.$

- 4 equations, 4 unknowns, no constants.

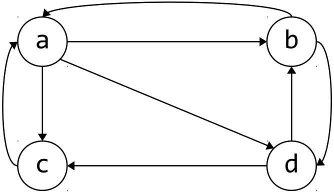
  No **unique solution:** all solutions are equivalent modulo a scale factor.

- Additional **constraint** for uniqueness:

$$\sum_i r_i = 1.$$

PageRank example

$$r_i = \sum_{j:j\to i\in\mathcal{E}} \frac{r_j}{d_j}$$



Equations:

- $r_a = \frac{r_b}{2} + r_c$.
- $r_b = \frac{r_a}{3} + \frac{r_d}{2}$.
- $r_c = \frac{r_a}{3} + \frac{r_d}{2}$.
- $r_d = \frac{r_a}{3} + \frac{r_b}{2}$.

- 4 equations, 4 unknowns, no constants.

  No **unique solution:** all solutions are equivalent modulo a scale factor.

- Additional **constraint** for uniqueness:

$$\sum_i r_i = 1.$$

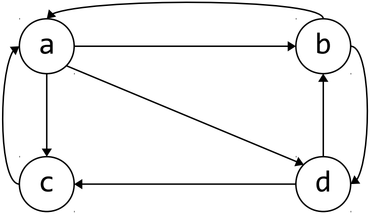
- **Solution** by **Gaussian elimination:**
  - $r_a = \frac{1}{3}$.
  - $r_b = r_c = r_d = \frac{2}{9}$.

Random walkers

- For **large graphs,** solving linear systems of equations is intractable.
- **Random surfers:** Where do you end if you follow links at random?

Random walkers

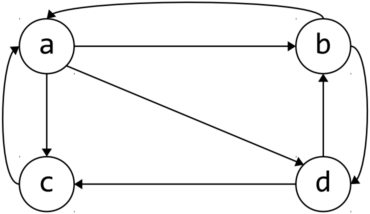
- For **large graphs,** solving linear systems of equations is intractable.
- **Random surfers:** Where do you end if you follow links at random?



Start at node a: after one step, end up in b, c, or d with probability $\frac{1}{3}$.

Random walkers

- For **large graphs,** solving linear systems of equations is intractable.
- **Random surfers:** Where do you end if you follow links at random?

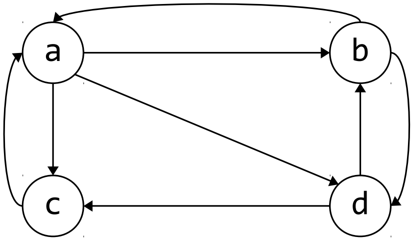


Start at node a: after one step, end up in b, c, or d with probability $\frac{1}{3}$.

- **Transition matrix:** $M_{ij} = \frac{1}{d_j}$ if $j \rightarrow i \in \mathcal{E}$ and 0 otherwise.

  The transition matrix is **column-stochastic:** columns sum to 1.

Random walkers: Transition matrix example



- **Transition matrix:**

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

PageRank with random walkers

- Start random surfers **at all pages** with **equal probability** $\frac{1}{n}$

$$\vec{v}_0 = [1/n, 1/n, \ldots, 1/n] \, .$$

- **After one step,** the distribution will be

PageRank with random walkers

- Start random surfers **at all pages** with **equal probability** $\frac{1}{n}$

$$\vec{v}_0 = [1/n, 1/n, \ldots, 1/n] .$$

- **After one step,** the distribution will be

$$\vec{v}_1 = M\vec{v}_0 .$$

PageRank with random walkers

- Start random surfers **at all pages** with **equal probability** $\frac{1}{n}$

$$\vec{v}_0 = [1/n, 1/n, \ldots, 1/n] \, .$$

- **After one step,** the distribution will be

$$\vec{v}_1 = M\vec{v}_0 .$$

- **After k steps:**

$$\vec{v}_k = M^k\vec{v}_0 .$$

PageRank with random walkers

- Start random surfers **at all pages** with **equal probability** $\frac{1}{n}$

$$\vec{v}_0 = [1/n, 1/n, \ldots, 1/n] \,.$$

- **After one step,** the distribution will be

$$\vec{v}_1 = M\vec{v}_0.$$

- **After** k **steps:**

$$\vec{v}_k = M^k\vec{v}_0.$$

- **Markov process:** The distribution approaches a limiting distribution $\vec{v}$ such that $\vec{v} = M\vec{v}$ if
  - The graph is **strongly connected:** can get from a node to any other node.
  - No **dead ends:** nodes that have no out-links.

PageRank with random walkers

$\vec{v} = M\vec{v}$.

- Surfers are **stationary.**
- The more important a page, and the more likely it is to have a surfer.
- $\vec{v}$ is … of M.

PageRank with random walkers

$\vec{v} = M\vec{v}$.

- Surfers are **stationary.**
- The more important a page, and the more likely it is to have a surfer.
- $\vec{v}$ is **the principal eigenvector** of M.

PageRank with random walkers

$\vec{v} = M\vec{v}$.

- Surfers are **stationary.**
- The more important a page, and the more likely it is to have a surfer.
- $\vec{v}$ is **the principal eigenvector** of M. (M stochastic has largest eigenval 1.)

PageRank with random walkers

$\vec{v} = M\vec{v}.$

- Surfers are **stationary.**
- The more important a page, and the more likely it is to have a surfer.
- $\vec{v}$ is **the principal eigenvector** of M. (M stochastic has largest eigenval 1.)
- **Power iteration:** compute $\vec{v}$ by iterative **matrix-vector multiplications.**
  - Stop when $||\vec{v}_t - \vec{v}_{t-1}|| \leq \epsilon$.
  - How eigenvectors are computed in large dimensions (eg. Lanczos method.)
  - Amenable to

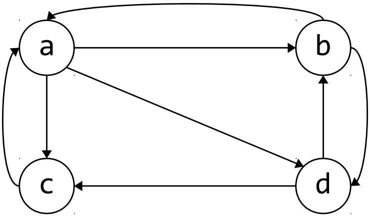
PageRank with random walkers

$\vec{v} = M\vec{v}.$

- Surfers are **stationary.**
- The more important a page, and the more likely it is to have a surfer.
- $\vec{v}$ is **the principal eigenvector** of M. (M stochastic has largest eigenval 1.)
- **Power iteration:** compute $\vec{v}$ by iterative **matrix-vector multiplications.**
  - Stop when $||\vec{v}_t - \vec{v}_{t-1}|| \leq \epsilon$.
  - How eigenvectors are computed in large dimensions (eg. Lanczos method.)
  - Amenable to **MapReduce** parallelization.
- Equivalent to previous PageRank formulation:

$$v_i = \sum_{j:j \to i \in \mathcal{E}} \frac{v_j}{d_j}$$

Example

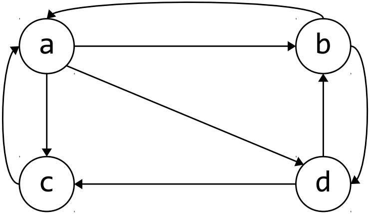


Transition matrix:

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

- **Initialization:** $\vec{v}_0 = [1/4,\ 1/4,\ 1/4,\ 1/4]$.

Example

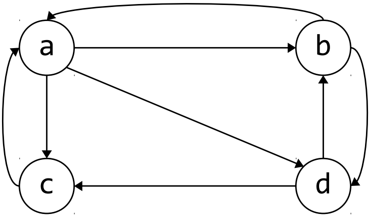


Transition matrix:

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

- **Initialization:** $\vec{v}_0 = [1/4,\ 1/4,\ 1/4,\ 1/4]\,.$
- **After one step:** $\vec{v}_1 = [9/24,\ 5/24,\ 5/24,\ 5/24]\,.$

Example

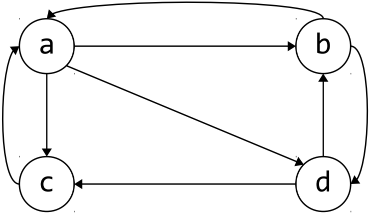


Transition matrix:

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

- **Initialization:** $\vec{v}_0 = [1/4, 1/4, 1/4, 1/4]$.
- **After one step:** $\vec{v}_1 = [9/24, 5/24, 5/24, 5/24]$.
- **After two steps:** $\vec{v}_2 = [15/48, 11/48, 11/48, 11/48]$.

Example



Transition matrix:

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

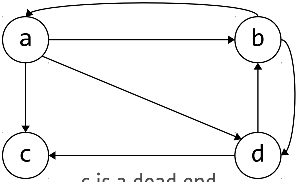
- **Initialization:** $\vec{v}_0 = [1/4,\ 1/4,\ 1/4,\ 1/4]$.
- **After one step:** $\vec{v}_1 = [9/24,\ 5/24,\ 5/24,\ 5/24]$.
- **After two steps:** $\vec{v}_2 = [15/48,\ 11/48,\ 11/48,\ 11/48]$.

  ...

- **Converges to:** $\vec{v} = [1/3,\ 2/9,\ 2/9,\ 2/9]$.

Dead ends

- **Dead ends:** nodes that have no out-links.



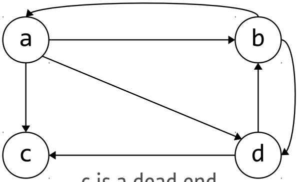
c is a dead end

Transition matrix:

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

Dead ends

- **Dead ends:** nodes that have no out-links.



c is a dead end

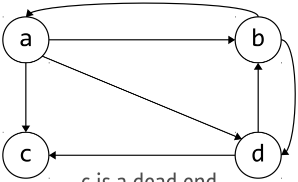
Transition matrix:

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

- The **transition matrix** does not have full rank.
- It cannot be **inverted**, i.e. our linear system of equations has **no solution.**
- The **power method** converges to

Dead ends

- **Dead ends:** nodes that have no out-links.



c is a dead end

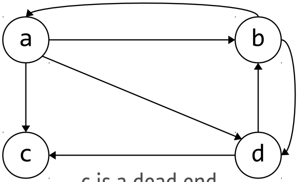
Transition matrix:

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

- The **transition matrix** does not have full rank.
- It cannot be **inverted**, i.e. our linear system of equations has **no solution.**
- The **power method** converges to $\vec{v} = \vec{0}$.

Dead ends

- **Dead ends:** nodes that have no out-links.



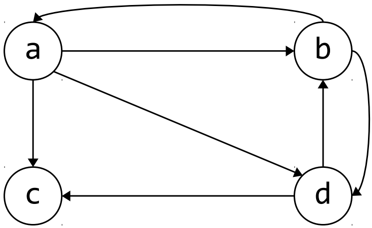
c is a dead end

Transition matrix:

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

- The **transition matrix** does not have full rank.
- It cannot be **inverted**, i.e. our linear system of equations has **no solution.**
- The **power method** converges to $\vec{v} = \vec{0}$.
- **Solutions:**
  - Recursively **remove** dead ends and their incoming links.
  - When at a dead end, **teleport** (with equal probability) to another node.

Example



Transition matrix:

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

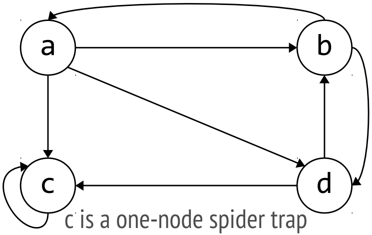
- New **transition matrix:**

$$\begin{bmatrix} 0 & 1/2 & \mathbf{1/4} & 0 \\ 1/3 & 0 & \mathbf{1/4} & 1/2 \\ 1/3 & 0 & \mathbf{1/4} & 1/2 \\ 1/3 & 1/2 & \mathbf{1/4} & 0 \end{bmatrix}$$

- Eventually, $\vec{v} = [1/5,\ 4/15,\ 4/15,\ 4/15]$.

Spider traps

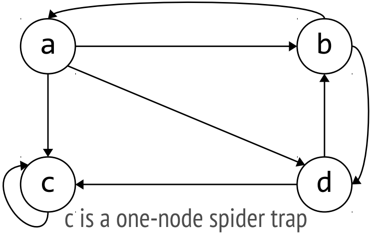
- **Spider trap:** set of nodes with no dead ends but no links out.
- **Problem:**



c is a one-node spider trap

Spider traps

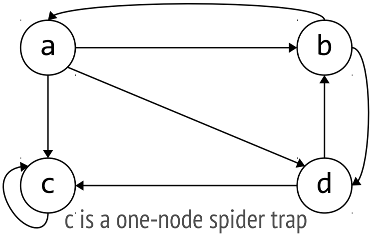
- **Spider trap:** set of nodes with no dead ends but no links out.
- **Problem:**
  - All random surfers end up in the spider trap.



c is a one-node spider trap

Spider traps

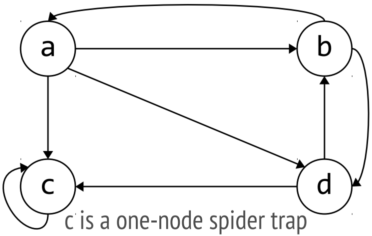
- **Spider trap:** set of nodes with no dead ends but no links out.
- **Problem:**
  - All random surfers end up in the spider trap.
- **Transition matrix:**



c is a one-node spider trap

Spider traps

- **Spider trap:** set of nodes with no dead ends but no links out.
- **Problem:**
  - All random surfers end up in the spider trap.



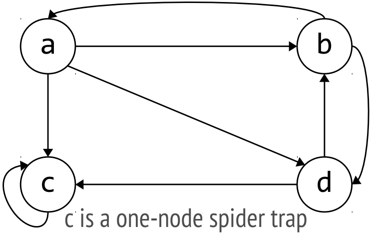
c is a one-node spider trap

- **Transition matrix:**

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

Spider traps

- **Spider trap:** set of nodes with no dead ends but no links out.



- **Problem:**
  - All random surfers end up in the spider trap.

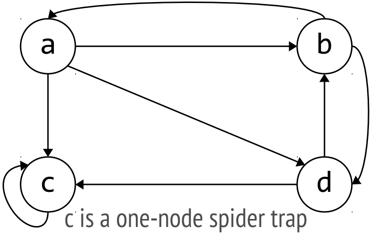
- **Transition matrix:**

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

- $\vec{v}$ **converges to**

Spider traps

- **Spider trap:** set of nodes with no dead ends but no links out.



- **Problem:**
  - All random surfers end up in the spider trap.
- **Transition matrix:**

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

- $\vec{v}$ **converges to** $\vec{v} = [0,\ 0,\ 1,\ 0]$.

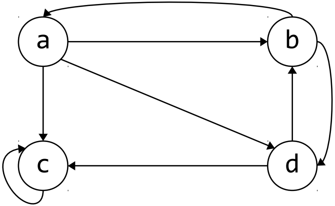
Taxation

- How to get out of **spider traps?**
  - A random surfer can **leave the graph** at any moment.
  - **New surfers** can be started at any page at any moment.
- **Taxation:** Allow each random surfer a probability $1 - \beta$ of **teleporting** to a random page

$$\vec{v} = \beta M \vec{v} + \frac{(1-\beta)}{n} \vec{1}.$$

Typically, $\beta \in [0.8 - 0.9]$.

Example



Transition matrix:

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

$$\vec{v} = \beta M \vec{v} + \frac{(1-\beta)}{n}\vec{1}$$

- $\beta = 0.8 = 4/5$

$$\vec{v} = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \vec{v} + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}, \quad \vec{v}_0 = \left[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right].$$

- **Solution:** $\vec{v} = \left[\frac{15}{148}, \frac{19}{148}, \frac{95}{148}, \frac{19}{148}\right].$

Summary

- **Large-scale data** poses new technical problems for:
  - **storage** $\Rightarrow$ distributed file systems.
  - **computations** $\Rightarrow$ MapReduce programming model.
    - Split the data in chunks.
    - Map workers all execute the same operation on a chunk and return a key-val pair.
    - Reduce workers process all key-val pairs with the same key at once.
- **Algorithmic costs** of MapReduce:
  - **Communication costs** vs. **computation costs.**
  - **Reducer size** and **replication rate.**
- Extensions of MapReduce: **Spark** and **TensorFlow.**
- MapReduce for **machine learning.**
- **Link analysis** with **PageRank.**

References & Acknowledgements

- These slides are mostly inspired from:
  - The book and slides of J. Leskovec, A. Rajaraman, and J. Ullman, **Mining of Massive Datasets**. http://www.mmds.org
  - C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Y. Yu, G. Bradski, A. Y. Ng and K. Olukotun, **Map-reduce for machine learning on multicore.** NeurIPS 2007.
- To learn more you can refer to (clickable links!):
  - J. Dean and S. Ghemawat. **MapReduce: Simplified data processing on large clusters.**
  - S. Ghemawat, H. Gobioff, and S.-T. Leung. **The Google File System.**
  - The Hadoop Wiki.
  - http://www.worldwidewebsize.com/
  - Scaling the Facebook data warehouse to 300 PB.