Lecture 5
Finding Similar Items IV / Map Reduce I

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University

April 26, 2023

Learning Goals Today

- Understand the theory supporting *Locality Sensitive Hashing (LSH)*
- Understand the technical challenges of parallelism / multi-node computation
- Understand the *MapReduce* paradigm

Locality Sensitive Hashing

–

Reminder

BANDING TECHNIQUE: THE S-CURVE

DEFINITION: [S-CURVE]
For given b and r, the *S-curve* is defined by the prescription

$$s \mapsto 1 - (1 - s^r)^b \tag{1}$$

s	$1 - (1 - s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

Table: Values for S-curve with $b = 20$ and $r = 5$

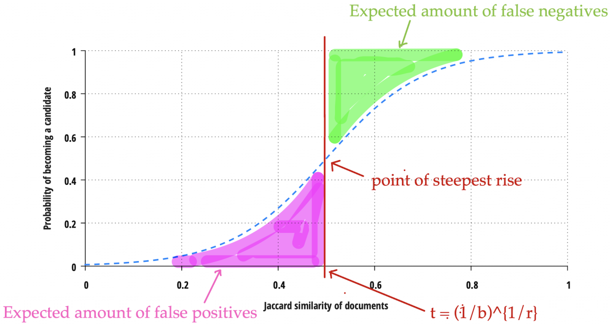
Locality Sensitive Hashing: Guidelines

- One needs to determine b, r where $br = n$
- One needs to determine threshold t:
  - $s \geq t$: candidate pair
  - $s < t$: no candidate pair
- t corresponds with point of steepest rise on S-curve: approximately $(1/b)^{(1/r)}$

Motivation:

- *False Positive:* dissimilar pair hashing to the same bucket
- *False Negative:* similar pair never hashing to the same bucket
- *Motivation:* limit both false positives and negatives

LSH: False Negatives / Positives



Exemplary S-curve

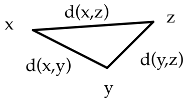
- Pick threshold t, number of bands b and rows r
- Avoiding false negatives: have $t \approx (1/b)^{1/r}$ large (not low!)
- Avoiding false positives, or enhancing speed: have $t \approx (1/b)^{1/r}$ low (not large!)

Distance Measures

DISTANCE MEASURE: DEFINITION

DEFINITION: [DISTANCE MEASURE]
Consider a set of objects. A *distance measure* is a function $d(x, y)$ that maps two objects x, y to a number such that

1. $d(x, y) \geq 0$ [d is *non-negative*]
2. $d(x, y) = 0$ implies $x = y$ [only if two objects are identical, the distance is zero; strictly positive otherwise]
3. $d(x, y) = d(y, x)$ [distance is *symmetric*]
4. $d(x, z) \leq d(x, y) + d(y, z)$ [*triangle inequality*]



Distance Measures: Examples

- In *n-dimensional Euclidean space*: points = real-valued vectors of length n
- The L_r-distance, defined to be

$$d([x_1, ..., x_n], [y_1, ..., y_n]) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{1/r} \tag{2}$$

  is a distance measure
- A particular example is the Euclidean distance, defined as the L_2-distance
- *Cosine:* Let $||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$ be the L_2-*norm* of a point in Euclidean space. The *cosine similarity* for two points $[x_1, ..., x_n], [y_1, ..., y_n]$ is defined to be

$$\frac{\sum_{i=1}^{n} x_i y_i}{||x||_2 ||y||_2} \tag{3}$$

  - Measures the *angle* between two vectors x and y
  - Gives rise to distance measure between lines that pass through origin

Distance Measures: Examples

- Let $\mathrm{SIM}(x, y)$ be the Jaccard similarity between two sets x, y. The quantity
$$1 - \mathrm{SIM}(x, y) \tag{4}$$
can be proven to be a distance measure.
- *Edit distance:* Objects are strings. The edit distance between two strings $x = x_1 ... x_m, y = y_1 ... y_n$ is the smallest number of insertions and deletions of single characters to be applied to turn x into y.
- *Hamming Distance:* For $[x_1, ..., x_n], [y_1, ..., y_n]$, the Hamming distance is the number of positions $i \in [1, ..., n]$ where $x_i \neq y_i$

Edit / Hamming Distance: Example

Edit Distance D_E:

Consider $x =$ "*abcde*", $y =$ "*acfdeg*". Claim: $D_E(x, y) = 3$.

- For proving $D_E(x, y) \leq 3$, consider edit sequence
  1. Delete b
  2. Insert f after c
  3. Insert g after e
- For $D_E(x, y) \geq 3$, consider that x contains b, which y does not, which holds vice versa for f, g. This implies that 3 edit operations are necessary at least.

Hamming Distance D_H:

Consider $x = 10101, y = 11110$:

$$D_H(x, y) = 3$$

because disagreeing in 3 positions (of five overall).

Locality Sensitive Functions

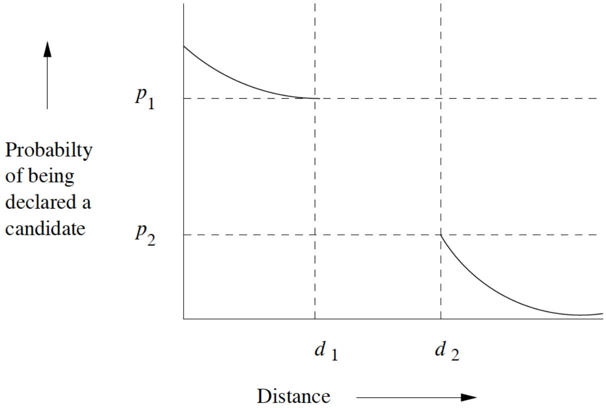
Locality Sensitive Family of Functions: Definition

- Consider functions f that hash items. The notation $f(x) = f(y)$ means that x and y form a candidate pair.
- A collection $\mathcal{F}$ of functions f of this form is called a *family of functions*
- Unless stated otherwise, $d(x, y) = 1 - \text{SIM}(x, y)$ is the Jaccard distance

Definition: [Locality Sensitive (LS) Family of Functions]
A family $\mathcal{F}$ of functions is said to be (d_1, d_2, p_1, p_2)-*sensitive* if for each $f \in \mathcal{F}$:

1. $d(x, y) \leq d_1$ implies that the probability that $f(x) = f(y)$ is at least p_1
2. $d(x, y) \geq d_2$ implies that the probability that $f(x) = f(y)$ is at most p_2

LS Family of Function: Illustration



Behaviour of any member of a (d_1, d_2, p_1, p_2)-sensitive family of function

From `mmds.org`

LS Family of Functions: Example

Consider minhash functions.

Reminder: Minhash functions map a column in the characteristic matrix to the minimum value the rows, in which there are 1's in the column, get hashed to.

Example: LS Family of Minhash Functions

- Consider $d(x, y) = 1 - \text{SIM}(x, y)$ to measure the distance between two sets x, y.
- Then it holds that the family of minhash functions is a $(d_1, d_2, 1 - d_1, 1 - d_2)$-sensitive family for any $0 \leq d_1 < d_2 \leq 1$.

Proof: By definition, $d(x, y) \leq d_1$ implies $\text{SIM}(x, y) = 1 - d(x, y) \geq 1 - d_1$. If, on the other hand, $d(x, y) \geq d_2$, we obtain $\text{SIM}(x, y) = 1 - d(x, y) \leq 1 - d_2$

AMPLIFYING LS FAMILIES OF FUNCTIONS: AND-CONSTRUCTION

Consider a (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{F}$. We construct a new family $\mathcal{F}_{r,AND}$ by the following principle:

- Each single member of $f \in \mathcal{F}_{r,AND}$ is based on r members $f_1, ..., f_r$ of $\mathcal{F}$.
-
$$f(x) = f(y) \quad \Leftrightarrow \quad f_i(x) = f_i(y) \text{ for all } i = 1, ..., r \tag{5}$$

Example: Consider the members of one band of size r when applying the banding technique.

Fact: It is easy to show (consider yourself!) that $\mathcal{F}_{r,AND}$ is a $(d_1, d_2, (p_1)^r, (p_2)^r)$-sensitive family of functions

Amplifying LS Families of Functions: OR-Construction

Consider a (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{F}$. We construct a new family $\mathcal{F}_{b,OR}$ by the following principle:

- Each single member of $f \in \mathcal{F}_{b,OR}$ is based on b members $f_1, ..., f_b$ of $\mathcal{F}$.
-
$$f(x) = f(y) \quad \Leftrightarrow \quad f_i(x) = f_i(y) \text{ for one } i = 1, ..., r \tag{6}$$

Example: The OR-construction reflects the effect of combining several bands when applying the banding technique.

Fact: It is easy to show (consider yourself again!) that $\mathcal{F}_{b,OR}$ is a $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensitive family of functions.

Amplifying LS Families of Functions: Locality Sensitive Hashing

Example: Applying the OR-construction to $\mathcal{F}_{r,AND}$, yielding $(\mathcal{F}_{r,AND})_{b,OR}$ reflects applying the banding technique altogether.

Fact: $(\mathcal{F}_{r,AND})_{b,OR}$ is a $(d_1, d_2, 1-(1-p_1^r)^b, 1-(1-p_2^r)^b)$-sensitive family of functions. Varying p_1, p_2 reflects reproducing the S-curve.

This justifies to study LS families of functions as a useful thing to do. For example:

- How does behaviour change when varying r and b?
  ☞ S-curve
- What happens when exhanging AND and OR?

Amplifying LS Families of Functions: Locality Sensitive Hashing

p	$1-(1-p^4)^4$
0.2	0.0064
0.3	0.0320
0.4	0.0985
0.5	0.2275
0.6	0.4260
0.7	0.6666
0.8	0.8785
0.9	0.9860

p	$\left(1-(1-p)^4\right)^4$
0.1	0.0140
0.2	0.1215
0.3	0.3334
0.4	0.5740
0.5	0.7725
0.6	0.9015
0.7	0.9680
0.8	0.9936

Original family $\mathcal{F}$ is $(0.2, 0.6, 0.8, 0.4)$-sensitive.

Left: Applying first the AND- and then the OR-construction, reflecting locality sensitive hashing, yields a $(0.2, 0.6, 0.8785, 0.0985)$-sensitive family.

Right: Applying first the OR- and then the AND-construction, yields a $(0.2, 0.6, 0.9936, 0.5740)$-sensitive family.

LS Families for Other Distance Measures

LS Families for Hamming Distance

LS Families for Hamming Distance

- Assume we have a d-dimensional vector space V
- Let $h(x, y)$ be the Hamming distance between vectors $x = (x_1, ..., x_d), y = (y_1, ..., y_d) \in V$
- Let $f_i(x) := x_i$ be the entry of x at the i-th position
- So $f_i(x) = f_i(y)$ if and only if $x_i = y_i$
- For randomly chosen x, y, the probability that $f_i(x) = f_i(y)$ is

$$\frac{d - h(x, y)}{d} = 1 - \frac{h(x, y)}{d}$$

  the fraction of positions in which x and y agree
- Thus, the family $\mathcal{F}$ of $\{f_1, ..., f_d\}$ is

$$(d_1, d_2, 1 - \frac{d_1}{d}, 1 - \frac{d_2}{d}) - \text{sensitive}$$

  for any $d_1 < d_2$

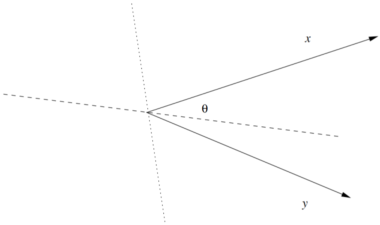
LS Families for Hamming Distance

- Let $h(x, y)$ be the Hamming distance between vectors $x = (x_1, ..., x_d), y = (y_1, ..., y_d) \in V$
- So $f_i(x) = f_i(y)$ if and only if $x_i = y_i$
- The family $\mathcal{F}$ of $\{f_1, ..., f_d\}$ is $(d_1, d_2, 1 - \frac{d_1}{d}, 1 - \frac{d_2}{d})$ − sensitive for any $d_1 < d_2$

Differences

- Jaccard distance runs from 0 to 1, Hamming distance from 0 to d: need to scale with $1/d$
- There is an unlimited number of minhash functions, but size of $\mathcal{F}$ is only d
- The limited size of $\mathcal{F}$ puts limits to AND/OR constructions

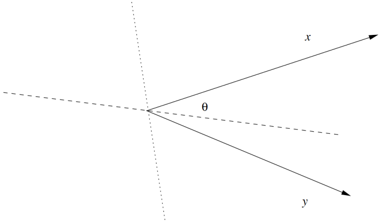
LS Families for Cosine Distance



Two vectors making an angle θ
From mmds.org

- Cosine distance for $x, y \in V$ corresponds with the angle $\theta(x, y) \in [0, 180]$ between x and y
- Whatever the dimension $d = \dim V$, two vectors x, y span a plane $V(x, y)$ (so $\dim V(x, y) = 2$)
- Angle θ is measured in that plane $V(x, y)$

LS Families for Cosine Distance: Random Hyperplanes

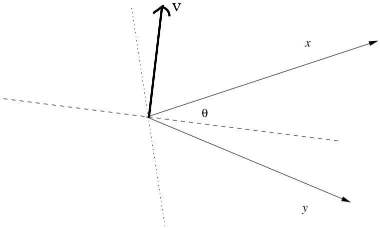


Two vectors making an angle θ
From mmds.org

- Any hyperplane (dimension $\dim V - 1$) intersects $V(x, y)$ in a line
- Figure: two hyperplanes, indicated by dotted and dashed line
- Determine hyperplanes U by picking normal vectors v
- That is

$$U = \{u \in V \mid \langle u, v \rangle = 0\}$$

LS Families for Cosine Distance: Random Hyperplanes



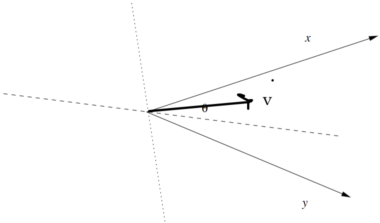
Two vectors making an angle θ
From mmds.org

- Consider dashed line hyperplane U: x and y on different sides
- Let v be normal vector of U:

$$\operatorname{sgn}\langle x, v\rangle \neq \operatorname{sgn}\langle y, v\rangle$$

so one scalar product is positive and the other one is negative

LS Families for Cosine Distance: Random Hyperplanes



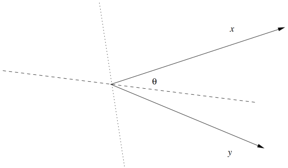
Two vectors making an angle θ
From mmds.org

- Consider dotted line hyperplane U: x and y on the same side
- Let v be normal vector of U:

$$\mathrm{sgn}\langle x, v\rangle = \mathrm{sgn}\langle y, v\rangle$$

so both scalar products positive or both negative

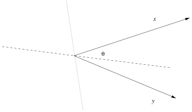
LS Families for Cosine Distance: Random Hyperplanes



Two vectors making an angle θ
From mmds.org

- Choose x, y at an angle $\theta(x, y)$
- Probability that
  - hyperplane like dashed line: $\theta(x, y)/180$
  - hyperplane like dotted line: $(180 - \theta(x, y))/180$
- Consider hash functions f corresponding to randomly picked normal vectors v_f of hyperplanes

LS Families for Cosine Distance: Random Hyperplanes



Two vectors making an angle θ
From mmds.org

- Consider family $\mathcal{F}$ of hash functions f corresponding to randomly picked hyperplanes, represented by their normal vectors v_f
- For $x, y \in V$, let

$$f(x) = f(y) \quad \text{if and only if} \quad \operatorname{sgn}\langle v_f, x\rangle = \operatorname{sgn}\langle v_f, y\rangle$$

- $\mathcal{F}$ is $(d_1, d_2, (180 - d_1)/180, (180 - d_2)/180)$-sensitive
- One can amplify the family as desired
  - Apart from rescaling by 180, $\mathcal{F}$ is just like minhash family

Sampling Random Normal Vectors: Sketches

- When determining normal vectors of random hyperplanes, it can be shown that it suffices to pick random vectors with entries either -1 or $+1$
- Let $v_1, ..., v_n$ be such random vectors
- For a vector x, the array

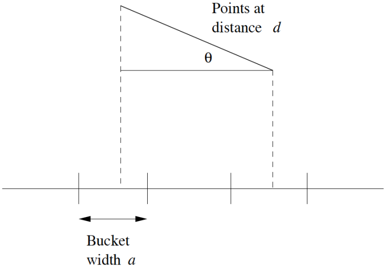
$$[\text{sgn}\langle v_1, x\rangle, ..., \text{sgn}\langle v_n, x\rangle] \in [-1, +1]^n \tag{7}$$

is said to be the *sketch* of x

Sketches: Example

- Let $x = [3, 4, 5, 6], y = [4, 3, 2, 1]$
- Let $v_1 = [+1, -1, +1, +1], v_2 = [-1, +1, -1, +1], v_3 = [+1, +1, -1, -1]$
- Then
  - Sketch of x is $[+1, +1, -1]$
  - Sketch of y is $[+1, -1, +1]$
  - Sketches of x, y agree in 1 out of 3 positions: we estimate $\widehat{\theta(x, y)} = 120$
  - However true $\theta(x, y) = 38$
- There are 16 different vectors with $+1, -1$ (cardinality of $\{-1, +1\}^4$ is 16)
- Computing sketches based on all of them yields estimate $\widehat{\theta(x, y)} = 45$

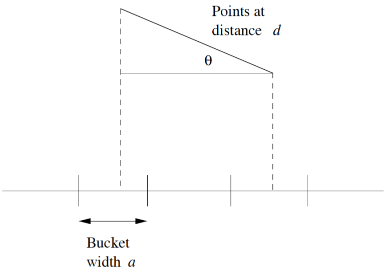
LS Families for Euclidean Distance



Two points at distance $d >> a$ are hashed to identical bucket with small probability
From mmds.org

- Let us consider 2-dimensional space V
- Each member f of family $\mathcal{F}$ is associated with line in V
- Line is divided into buckets (segments) of length a
- Points $x, y \in V$ are "hashed" to buckets
- $f(x) = f(y)$ when hashed to the same segment

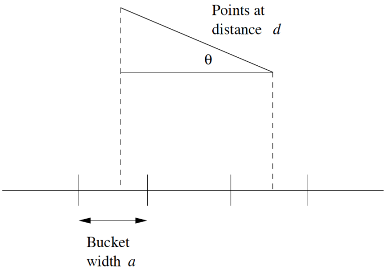
LS FAMILIES FOR EUCLIDEAN DISTANCE



Two points at distance $d >> a$ are hashed to identical bucket with small probability
From mmds.org

- If Euclidean distance $d(x,y) \leq a/2$, then probability to hash x,y to same segment is at least $1/2$
  - Distance between x,y after projecting is $d(x,y)\cos\theta \leq d(x,y) \leq a/2$

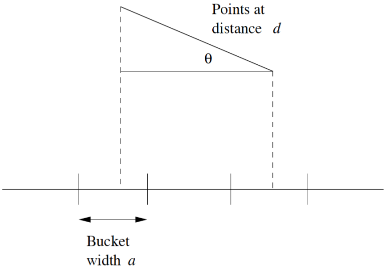
LS Families for Euclidean Distance



Two points at distance $d >> a$ are hashed to identical bucket with small probability
From mmds.org

- If distance between x, y after projecting is greater than a, they will be hashed to different buckets
- So, if $d(x, y) \geq 2a$, we have that $d(x, y) \cos \theta > a$ for $\theta \in [0, 60]$
- It holds that $\theta \in [0, 60]$ with probability $2/3$ (note: here $\theta \in [0, 90]$)

LS Families for Euclidean Distance



Two points at distance $d >> a$ are hashed to identical bucket with small probability
From mmds.org

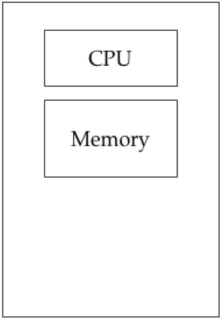
- In conclusion, the family described has been

$$(a/2, 2a, 1/2, 1/3) - \text{sensitive}$$

- Family can be amplified as desired
- If families for arbitrary $d_1 < d_2$ (and not just $d_1 = a/2, d_2 = 2a$), and also for arbitrary-dimensional vector spaces are desired, special efforts are required

Map Reduce: Introduction

MapReduce: Motivation I

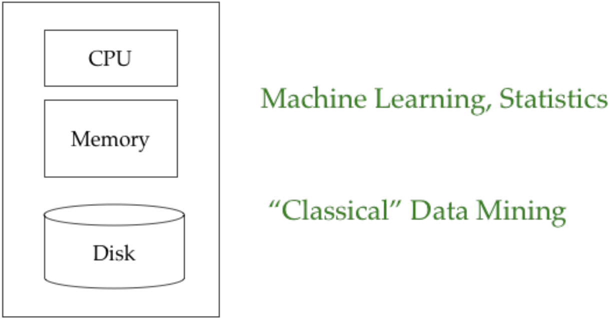


Machine Learning, Statistics

Adopted from `mmds.org`

- *Machine Learning, Statistics:* all data fits in main memory
- *Classical Data Mining:* data too big to fit in main memory

MapReduce: Motivation I



Adopted from `mmds.org`

- *Machine Learning, Statistics:* all data fits in main memory
- *Classical Data Mining:* data too big to fit in main memory

MapReduce: Motivation II

- Need to manage massive amounts of data quickly
- Within one particular application, data is massive
  - For example (web searches), even with high performance disk read bandwidth, just reading 10 billion web pages requires several days
- But operations can be very regular (do the same thing to each web page) ☞ exploit the parallelism
  - Many operations on databases (as supported by SQL, for example) can and need to be parallelized
  - Ranking web pages (*"PageRank"*) requires iterated multiplication of matrices with dimensions in the billions
  - Searching for "friend networks" in social networks require operations on graphs with billions of nodes and edges

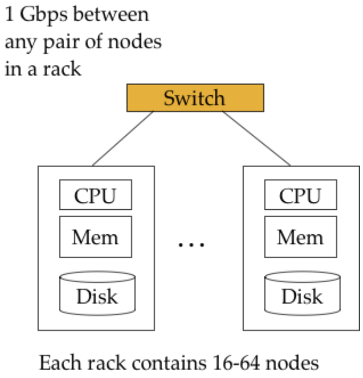
MapReduce: Motivation II

- New software stack: get parallelism not from single supercomputer, but from computing clusters
  - *First*, need to deal with storing data
    ☞ Distributed file systems (hardware based issues/solutions)
  - *Second*, new higher-level programming systems required
    ☞ *MapReduce*
  - *Third*, MapReduce reflects early attempts:
    ☞ More sophisticated *workflow systems*
- Here, we will deal predominantly with MapReduce first
- We will also consider most advanced workflow systems
- *Reminder:* it's about *analytics* in this course

MapReduce: Motivation III

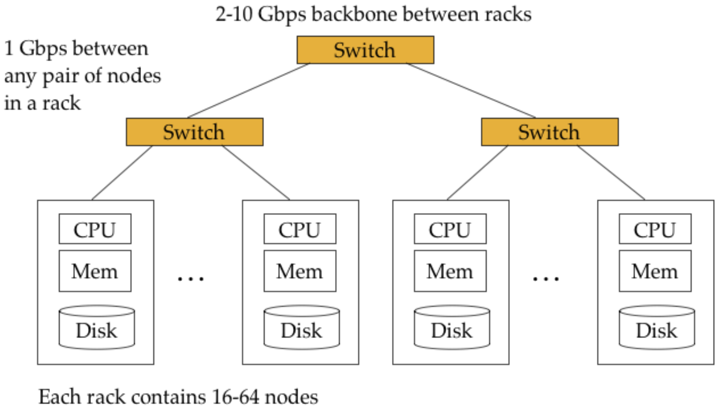
- MapReduce enables convenient execution of parallelizable operations on compute clusters and clouds
- MapReduce executes such operations in a *fault-tolerant* manner
- MapReduce is the origin of more general ideas
  - Systems supporting *acyclic workflows* in general
  - Systems supporting *recursive operations*

MapReduce: Motivation III



Adopted from `mmds.org`

MapReduce: Motivation III



Adopted from `mmds.org`

Distributed File Systems

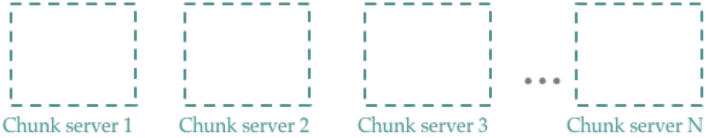
Distributed File Systems: Challenges and Characteristics

- *Node Failure:* Single nodes fail (e.g. by disk crash) or entire racks can fail (e.g. by network failure)
  ☞ no starting over every time: back up data
- *File Size:* can be huge
  ☞ how to distribute them?
- *Computation Time:* should not be dominated by input/output
  ☞ data should be as close as possible to compute nodes
- *Data:* does not change, new data only makes small appends
  ☞ otherwise DFS not suitable

Distributed File Systems: Summary

- Data is divided into *chunks* (usually of size 64 MB)
- Chunks are replicated (3 times is common)
- Chunk copies are distributed across the nodes
- A file called *master node* keeps track of where chunks went
- A *client library* provides file access; talks to master and connects to individual servers
- *Examples of DFS Implementations:*
  - *Google File System (GFS):* the original
  - *Hadoop Distributed File System (HDFS):* open source, used with Hadoop, a MapReduce implementation
  - *Colossus:* supposed to be an improvement over GFS; little has been published

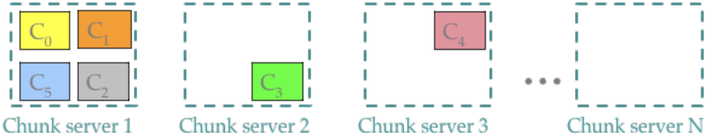
Distributed File Systems: Mode of Operation



Adopted from `mmds.org`

- Chunk servers correspond to nodes in racks

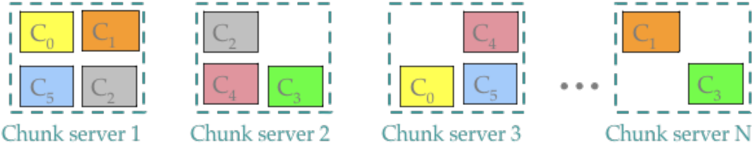
Distributed File Systems: Mode of Operation



Adopted from `mmds.org`

- One file ("File C") in 6 chunks, C0, C1, C2, C3, C4, C5

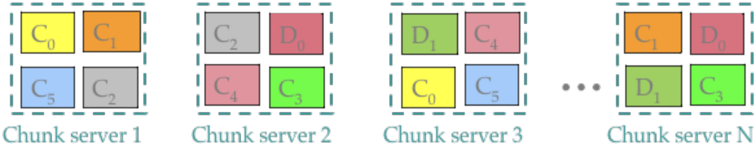
Distributed File Systems: Mode of Operation



Adopted from `mmds.org`

- Replicating each chunk twice and putting copies to different nodes prevents damage due to failure

Distributed File Systems: Mode of Operation



Adopted from `mmds.org`

- Fill servers up; computations are carried out immediately by chunk servers

Map Reduce: Workflow

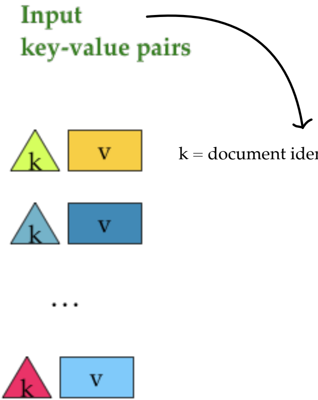
MapReduce: Workflow

1. Chunks are assigned to Map tasks, which turn each chunk into sequence of *key-value* pairs.
   - Key-value pair generation is specified by user
2. *Master controller* (automatic):
   - Key-value pairs are collected
   - Key-value pairs are sorted
   - Keys are divided among Reduce tasks
3. Reduce tasks combine values into final output
   - Reduce tasks are specified by user
   - Reduce tasks work on one key at a time

MapReduce: Running Example

- *Input:* One, or several huge documents
- *Desired Output:* Counts of all words appearing in the documents
- *Applications:*
  - Detecting plagiarism
  - Determining words characterizing documents for web searches
- Important: In the example, distinguish between
  - *Input key-value pairs* that reflect id-file pairs
  - *Intermediate key-value pairs* that reflect key-value pairs from Map tasks, as seen in the slide before
  - The latter ones are important for MapReduce

MapReduce: Map

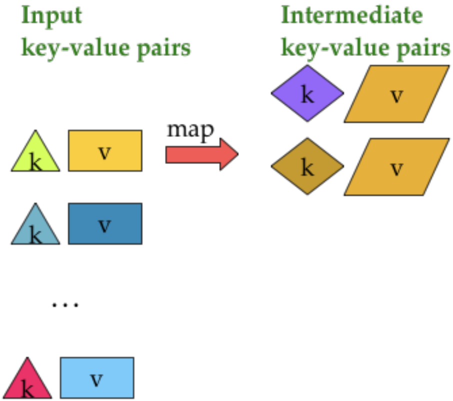


k = document identifier, v = document itself

Here, input key-value pairs refer to id-file (id-document) pairs

Adopted from `mmds.org`

MapReduce: Map



< k = “apple”, v = 1>

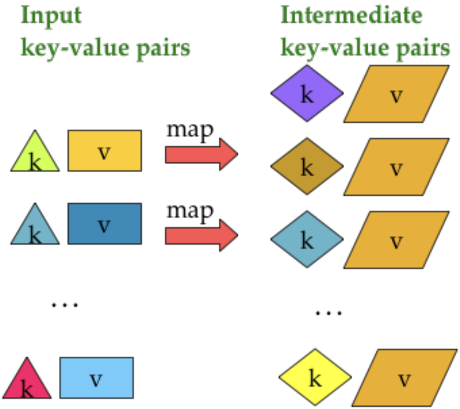
< “the”, 1>

< “apple”, 1>

Intermediate key-value pairs are the ones to be generated by a Map task

Adopted from `mmds.org`

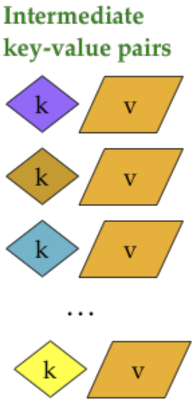
MapReduce: Map



Here: intermediate key-value pairs correspond to <'word',1> tuples

Adopted from `mmds.org`

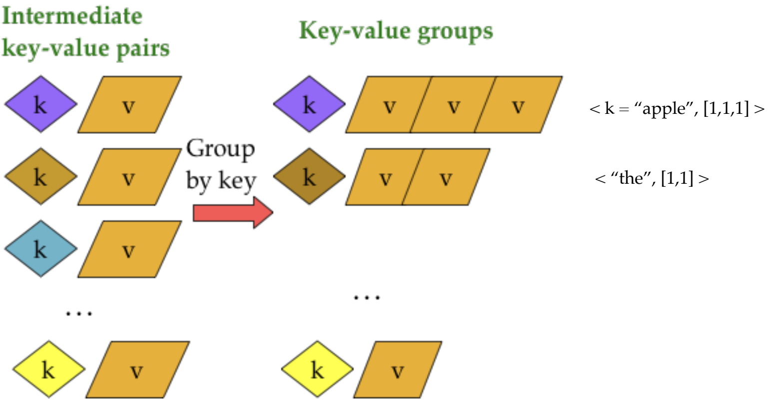
MapReduce: Reduce



Intermediate key-value pairs (<'word',1> tuples) generated by Map

Adopted from `mmds.org`

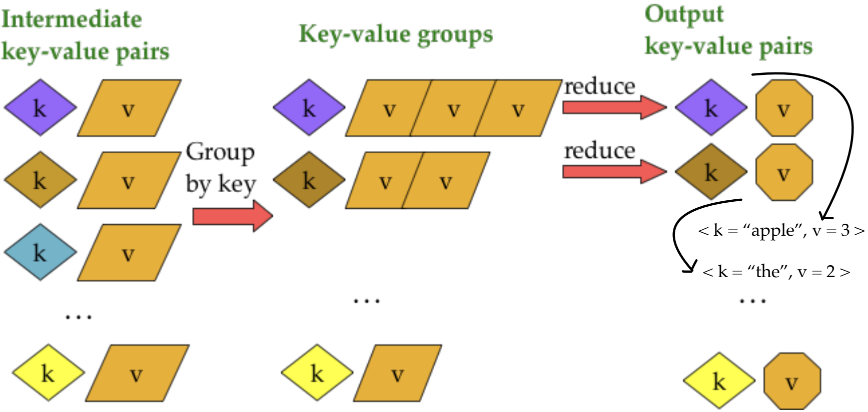
MapReduce: Reduce



Intermediate key-value pairs generated by Map

Adopted from `mmds.org`

MapReduce: Reduce



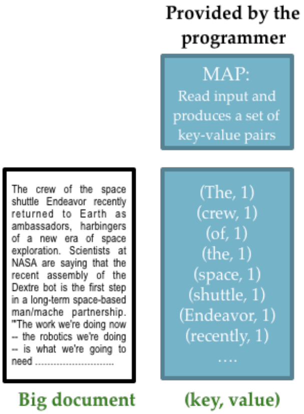
Output key-value pairs generated by Reduce: here <’word’,count> tuples

Adopted from `mmds.org`

MapReduce: Formal Summary

- *Input:* A set of (key, value)-pairs $< k, v >$
  - $< k, v >$ usually correspond to file (v) and id (k) of the file
- *To be provided by programmer:*
  - $Map(< k, v >) \rightarrow < k', v' >^*$
    - Maps input pair $< k, v >$ to multi-set of key-value pairs $< k', v' >$
    - $< k', v' >$ is intermediate key-value in schematic on slides before
    - One Map call for each input key-value pair $< k, v >$
  - $Reduce(< k', v' >^*) \rightarrow < k', v'' >^*$
    - For each key k' all key-value pairs $< k', v' >$ are reduced together
    - One Reduce call for each unique key k'

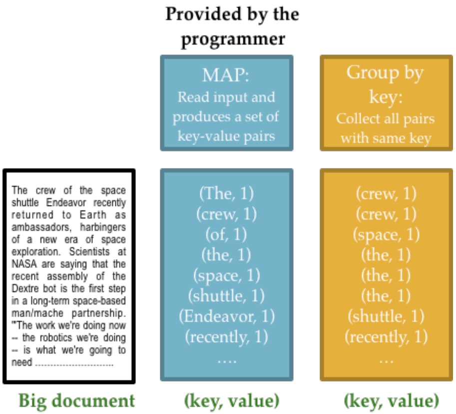
MapReduce Example: Word Counting



Intermediate key-value pairs correspond to <'word',1> tuples

Adopted from `mmds.org`

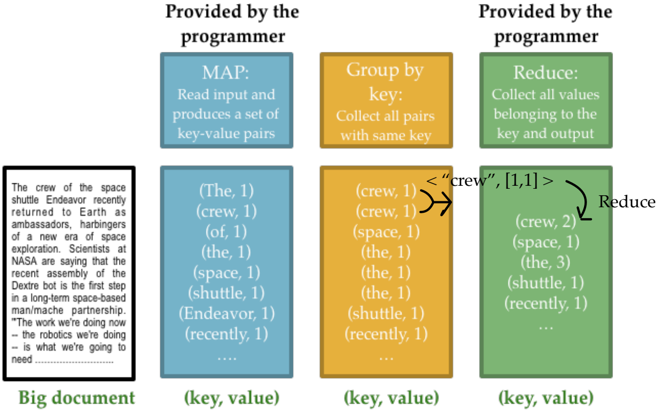
MapReduce Example: Word Counting



Intermediate key-value pairs are sorted and hashed by key (automatic)

Adopted from `mmds.org`

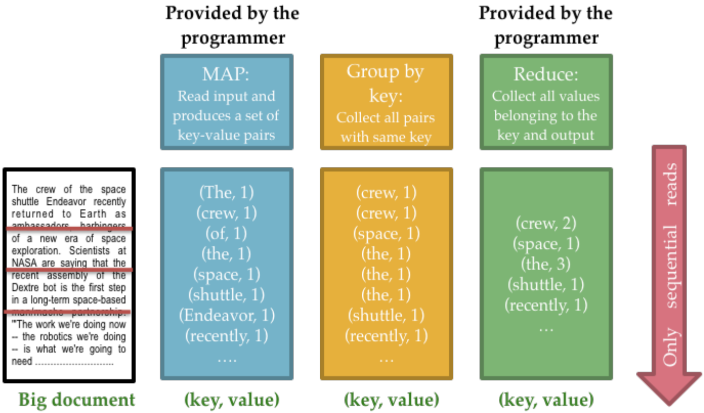
MapReduce Example: Word Counting



Reduce sums up all values for each key

Adopted from `mmds.org`

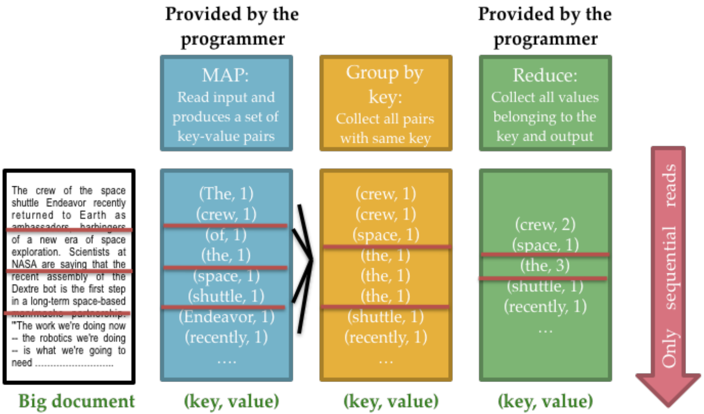
MapReduce Example: Word Counting



Map tasks are parallelized across nodes: one Map per chunk

Adopted from `mmds.org`

MAPREDUCE EXAMPLE: WORD COUNTING



Reduce tasks are parallelized across nodes: one Reduce for a subset of keys

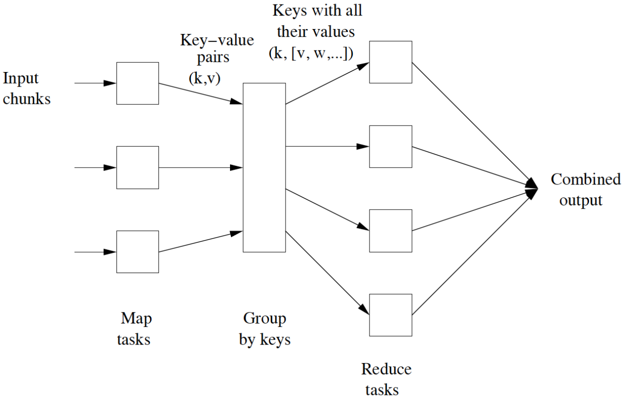
Adopted from `mmds.org`

Example: Word Counting Code

```
map(key, value)
// key:  document name, value:  text of document
  foreach word w in value:
    emit(w,1)

reduce(key, values)
// key:  a word, values:  an iterator over counts
  result = 0
  foreach count v in values:
    result += v
  emit(key, result)
```

MapReduce: Workflow Summary



Summary

Here $< k, v >$ refers to intermediate key-value pair earlier

Upon sorting key-value pairs are hashed

Adopted from `mmds.org`

Materials / Outlook

- See *Mining of Massive Datasets*, chapter 3.5–3.7, chapter 2
- See `http://www.mmds.org/` for further resources
- Next lecture: “MapReduce II”
  - See *Mining of Massive Datasets*, chapter 2

Example / Illustration