Lecture 8
MapReduce IV & Link Analysis I

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University

May 11, 2023

TODAY

Overview

- Understand the definition of *communication cost*
- Understand the definition of *wall clock time*
- Get to know theory and intuition of *complexity theory* for MapReduce
- PageRank: Introduction, Definition

Learning Goals: Understand these topics and get familiarized

The Communication-Cost Model: Reminder

Communication Cost & Wall-Clock Time

Definition [Communication Cost]:

- The *communication cost of a task* is the size of the input it receives
- The *communication cost of an algorithm* is the sum of the communication costs of its tasks

Definition [Wall-Clock Time]:

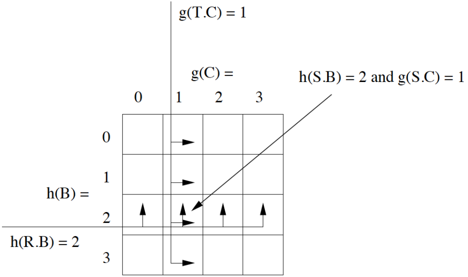
The *wall-clock time* is defined to be the time for the entire parallel algorithm to finish.

$R(A, B) \bowtie S(B, C) \bowtie T(C, D)$ in one MapReduce

Let p be the probability that an R- and an S-tuple agree on B, matching the probability for an S- and a T-tuple to agree on C.

- Hash B- and C-values, using functions h and g
- Let b and c be the number of buckets for h and g
- Let k be the number of Reducers; require that $bc = k$
  - Each reducer corresponds to a pair of buckets
  - Reducer corresponding to bucket pair (i, j) joins tuples $R(u, v), S(v, w), T(w, x)$ whenever $h(v) = i, g(w) = j$
- Hence Map tasks send R- and T-tuples to more than one reducer
  - R-tuples $R(u, v)$ go to all reducers $(h(v), y), y = 1, ..., c$
    ☞ goes to c reducers
  - T-tuples $T(w, x)$ go to all reducers $(z, g(w)), z = 1, ..., b$
    ☞ goes to b reducers

MULTIWAY JOIN: ONE MAPREDUCE II



Sixteen reducers for a 3-way join

Adopted from `mmds.org`

- $h(v) = 2, g(w) = 1$ [in Figure: $v = R.B, w = S.C$]
- S-tuple $S(v, w)$ goes to reducer for key $(2, 1)$
- R-tuple $R(u, v)$ goes to reducers for keys $(2, 0), \ldots, (2, 3)$
- T-tuple $T(w, x)$ goes to reducers for keys $(0, 1), \ldots, (3, 1)$

Multiway Join: One MapReduce III

Communication cost:

- Moving tuples to proper reducers is sum of
  - s to send tuples $S(v, w)$ to $(h(v), g(w))$
  - rc to send tuples $R(u, v)$ to $(h(v), y)$ for each of the c possible $g(w) = y$
  - bt to send tuples $T(w, x)$ to $(z, g(w))$ for each of the b possible $h(b) = z$
- Additional (constant) cost $r + s + t$ to make each tuple input to one of the Map tasks (constant)

MULTIWAY JOIN: ONE MAPREDUCE III

Communication cost:

- *Goal:* Select b and c, subject to $bc = k$, to minimize $s + cr + bt$
- Using Lagrangian multiplier λ makes solving for
  - $r - \lambda b = 0$
  - $t - \lambda c = 0$
- It follows that $rt = \lambda^2 bc$, that is $rt = \lambda^2 k$, yielding further $\lambda = \sqrt{\frac{rt}{k}}$
- So, minimum communication cost at $c = \sqrt{\frac{kt}{r}}$ and $b = \sqrt{\frac{kr}{t}}$
- Substituting into $s + cr + bt$ yields $s + 2\sqrt{krt}$
- Adding $r + s + t$ yields $r + 2s + t + 2\sqrt{krt}$, which is usually dominated by $2\sqrt{krt}$

Complexity Theory for MapReduce

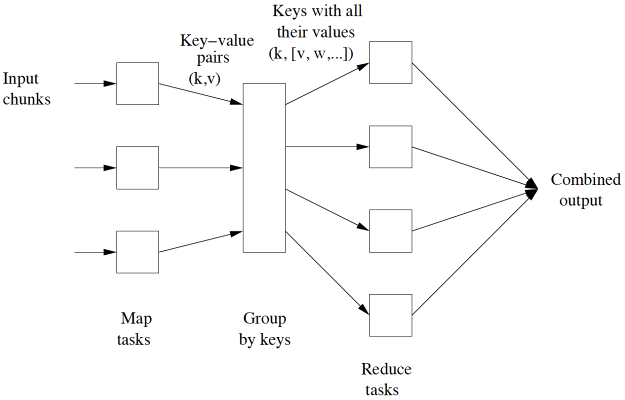
MapReduce: Complexity Theory

Idea

- *Reminder:* A "reducer" is the execution of a Reduce task on a single key and the associated value list
- *Important considerations:*
  - Keep communication cost low
  - Keep wall-clock time low
  - Execute each reducer in main memory
- *Intuition:*
  - The less communication, the less parallelism, so
  - the more wall-clock time
  - the more main memory needed
- *Goal:* Develop encompassing complexity theory

Reducer Size and Replication Rate

Reducer Size: Informal Explanation



Reducer size: maximum length of list [v,w,...] after grouping keys

Adopted from `mmds.org`

Reducer Size

Definition [Reducer Size]:
The *reducer size* q is the upper bound on the number of values to appear in the list of a single key.

Motivation

- Small reducer size forces to have many reducers
- Further creating many Reduce tasks implies high parallelism, hence small wall-clock time
- Small reducer size enables to have all data in main memory

Replication Rate

Definition [Replication Rate]:
The *replication rate* r is the number of all key-value pairs generated by Map tasks, divided by the number of inputs.

Motivating Example

- One-pass matrix multiplication algorithm:
  - Matrices involved are $n \times n$
  - *Reminder:* Key-value pairs for MN are $((i,k),(M,j,m_{ij})), j = 1,...,n$ and $((i,k),(N,j,n_{jk})), j = 1,...,n$
- Replication rate r is equal to n:
  - Inputs are all m_{ij} and n_{jk}
  - For each m_{ij}, one generates key-value pairs for $(i,k), k = 1,...,n$
  - For each n_{jk}, one generates key-value pairs for $(i,k), i = 1,...,n$
- Reducer size is $2n$: for each key (i,k) there are n values from each m_{ij} and n values from each n_{jk}

Example: Similarity Join

Situation

- Given large set X of elements
- Given similarity measure $s(x, y)$ for measuring similarity between $x, y \in X$
- Measure is symmetric: $s(x, y) = s(y, x)$
- *Output* of the algorithm: all pairs x, y where $s(x, y) \geq t$ for threshold t
- *Exemplary input:* 1 million images (i, P_i) where
  - i is ID of image
  - P_i is picture itself
  - Each picture is 1MB

EXAMPLE: SIMILARITY JOIN

MapReduce: Bad Idea

- Use keys (i, j) for pair of pictures $(i, P_i), (j, P_j)$
- *Map*: generates $((i, j), [P_i, P_j])$ as input for
- *Reduce*: computes $s(P_i, P_j)$ and decides whether $s(P_i, P_j) \geq t$
- Reducer size q is small: 2 MB; expected to fit in main memory
- *However*, each picture makes part of 999 999 key-value pairs, so

$$r = 999\,999$$

- Hence, number of bytes communicated from Map to Reduce is

$$10^6 \times 999\,999 \times 10^6 = 10^{18}$$

that is one exabyte

Example: Similarity Join

MapReduce: Better Idea

- Group images into g groups, each of $10^6/g$ pictures
- *Map:* For each (i, P_i) generate $g - 1$ key-value pairs
  - Let u be group of P_i
  - Let v be one of the other groups
  - Keys are sets $\{u, v\}$ (set, so no order!)
  - Value is (i, P_i)
  - Overall: $(\{u, v\}, (i, P_i))$ as key-value pair
- *Reduce:* Consider key $\{u, v\}$
  - Associated value list has $2 \times \frac{10^6}{g}$ values
  - Consider (i, P_i) and (j, P_j) when i, j are from different groups
  - Compute $s(P_i, P_j)$
  - Compute $s(P_i, P_j)$ for P_i, P_j from same group on processing keys $\{u, u + 1\}$

Example: Similarity Join

MapReduce: Better Idea

- *Replication rate* is $g - 1$
  - Each input element (i, P_i) is turned into $g - 1$ key-value pairs
- *Reducer size* is $2 \times \frac{10^6}{g}$
  - Number of values on list for reducer
  - This yields $2 \times \frac{10^6}{g} \times 10^6$ bytes stored at Reducer node

EXAMPLE: SIMILARITY JOIN

MapReduce: Better Idea

- *Example* $g = 1000$*:*
  - Input is 2 GB, fits into main memory
  - Communication cost:

$$\underbrace{(10^3 \times 999)}_{\text{number of reducers}} \times \underbrace{(2 \times 10^3 \times 10^6)}_{\text{reducer size}} \approx 10^{15} \quad (1)$$

  - 1000 times less than brute-force
  - Half a million reducers: maximum parallelism at Reduce nodes
- *Computation cost* is independent of g
  - Always all-vs-all comparison of pictures
  - Computing $s(P_i, P_j)$ for all i, j

MapReduce: Graph Model

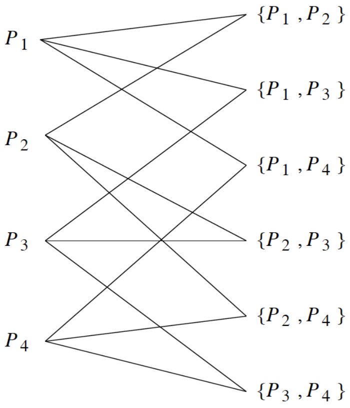
MapReduce: Graph Model

Goal: Proving lower bounds on replication rate as function of reducer size, for many problems. Therefore:

Graph Model:

- Graph describes how outputs depend on inputs
- Reducers operate independently: each output has one reducer that receives all input required to compute output
- *Model foundation:*
  - There is a set of inputs
  - There is a set of outputs
  - Outputs depend on inputs: many-to-many relationship

MapReduce: Graph Model Example



Graph for similarity join with four pictures

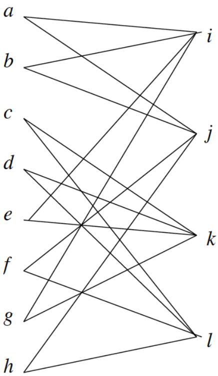
Adopted from `mmds.org`

MapReduce: Graph Model Matrix Multiplication

Graph Model Matrix Multiplication

- Multiplying $n \times n$ matrices M and N makes
  - $2n^2$ inputs $m_{ij}, n_{jk}, 1 \leq i, j, k \leq n$
  - n^2 outputs $p_{ik} := (MN)_{ik}, 1 \leq i, k \leq n$
- Each output p_{ik} needs $2n$ inputs $m_{i1}, m_{i2}, ..., m_{in}$ and $n_{1k}, n_{2k}, ..., n_{nk}$
- Each input relates to n outputs: e.g. m_{ij} to $p_{i1}, p_{i2}, ..., p_{in}$

MapReduce: Graph Model Matrix Multiplication II



$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} i & j \\ k & l \end{bmatrix}$$

Input-output relationship graph for multiplying 2x2 matrices

Adopted from `mmds.org`

MapReduce: Mapping Schemas

A *mapping schema* with a given reducer size q is an assignment of inputs to reducers such that

- No reducer receives more than q inputs
- For every output, there is a reducer that receives all inputs required to generate the output

Consideration: The existence of a mapping schema for a given q characterizes problems that can be solved in a *single* MapReduce job at reducer size q.

Mapping Schema: Example

Consider computing similarity of p pictures, divided into g groups.

- Number of outputs: $\binom{p}{2} = \frac{p(p-1)}{2} \approx \frac{p^2}{2}$
- Reducer receives $2p/g$ inputs
  ☞ necessary reducer size is $q = 2p/g$
- Replication rate is $r = g - 1 \approx g$:

$$r = 2p/q$$

  ☞ r inversely proportional to q which is common
- In a mapping schema for reducer size $q = 2p/g$:
  - Each reducer is assigned exactly $2p/g$ inputs
  - In all cases, every output is covered by some reducer

Mapping Schemas: Not all Inputs Present

Example: Natural Join $R(A, B) \bowtie S(B, C)$, where many possible tuples $R(a, b), S(b, c)$ are missing.

- Theoretically $q = |A| \cdot |C|$ because of keys $b \in B$ where
  - $(a, b) \in R$ for all $a \in A$
  - $(b, c) \in S$ for all $c \in C$
- But in practice many tuples $(a, b), (b, c)$ are missing for each b, so q possibly much smaller than $|A| \cdot |C|$

Main Consideration: One can decrease q because of the missing inputs, without that inputs do no longer fit into main memory in practice

Mapping Schemas: Lower Bounds on Replication Rate

Technique for proving lower bounds on replication rates

1. Prove upper bound $g(q)$ on how many outputs a reducer with q inputs can cover ☞ may be difficult in some cases
2. Determine total number of outputs O
3. Let there be k reducers with $q_i < q, i = 1, ..., k$ inputs
   ☞ observe that $\sum_{i=1}^{k} g(q_i)$ needs to be no less than O
4. Manipulate the inequality $\sum_{i=1}^{k} g(q_i) \geq O$ to get a lower bound on $\sum_{i=1}^{k} q_i$
5. Dividing the lower bound on $\sum_{i=1}^{k} q_i$ by number of inputs is lower bound on replication rate

Lower Bounds: Example All-Pairs Problem

- Recall that $r \leq 2p/q$ was upper bound on replication rate for all-pairs problem
- *Here:* Lower bound on r that is half the upper bound

Lower Bounds: Example All-Pairs Problem

- *Steps from slide before:*
  - Step 1: reducer with q inputs cannot cover more than $\binom{q}{2} \approx q^2/2$ outputs
  - Step 2: overall $\binom{p}{2} \approx p^2/2$ outputs must be covered
  - Step 3: So, the inequality approximately evaluates as

$$\sum_{i=1}^{k} q_i^2/2 \geq p^2/2 \qquad \Longleftrightarrow \qquad \sum_{i=1}^{k} q_i^2 \geq p^2$$

  - Step 4: From $q \geq q_i$, we obtain

$$q\sum_{i=1}^{k} q_i \geq p^2 \qquad \Longleftrightarrow \qquad \sum_{i=1}^{k} q_i \geq \frac{p^2}{q}$$

  - Step 5: Noting that $r = (\sum_{i=1}^{k} q_i)/p$, we obtain

$$r \geq \frac{p}{q}$$

which is half the size of upper bound

PageRank
Introduction

PageRank: Overview

- Motivation of PageRank definition: history of search engines
- Concept of *random surfers* foundation of PageRank's effectiveness
- *Taxation* ("recycling of random surfers") allows to deal with problematic web structures

History: Early Search Engines

- *Early search engines*
  - Crawl the (entire) web
  - List all terms encountered in an *inverted index*
  - An inverted index is a data structure that, given a term, provides pointers to all places where term occurs
- On a *search query* (a list of terms)
  - pages with those terms are extracted from the index
  - ranked according to use of terms within pages
  - E.g. the term appearing in the header renders page more important
  - or the term appearing very often

Term Spam

- *Spammers* exploited this to their advantage
- *Simple strategy:*
  - Add terms thousands of times to own webpages
  - Terms can be made hidden by using background color
  - So pages are listed in searches that do not relate to page contents
  - Example: add term "movie" 1000 times to page that advertizes shirts
- *Alternative strategy:*
  - Carry out web search on term
  - Copy-paste highest ranked page into own page
  - Upon new search on term, own page will be listed high up
- Corresponding techniques are referred to as *term spam*

PageRank's Motivation: Fighting Term Spam

Idea:

- Simulate *random web surfers*
  - They start at random pages
  - They randomly follow web links leaving the page
  - Iterate this procedure sufficiently many times
  - Eventually, they gather at "important" pages
- Judge page also by *contents of surrounding pages*
  - Difficult to add terms to pages not owned by spammer

PageRank's Motivation: Fighting Term Spam

Justification

- Ranking web pages by number of in-links does not work
  - Spammers create “spam farms” of dummy pages all linking to one page
- *But*, spammers' pages do not have in-links from elsewhere

☞ Random surfers do not wind up at spammers' pages

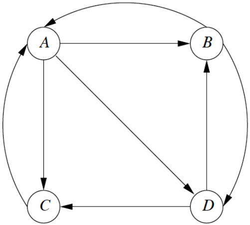
- (Non-spammer) page owners place links to pages they find helpful
- Random surfers indicate which pages are likely to visit
  ☞ Users are more likely to visit useful pages

PageRank: Definition

- PageRank is a function that assigns a real number to each (accessible) web page
- *Intuition:* The higher the PageRank, the more important the page
- There is not one fixed algorithm for computing PageRank
- There are many variations, each of which caters to particular issue

PageRank: Definition

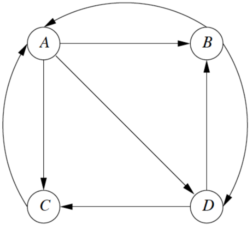
- Consider the web as a directed graph
  - Nodes are web pages
  - Directed edges are links leaving from and leading to web pages



Hypothetical web with four pages

Adopted from `mmds.org`

PageRank: Definition

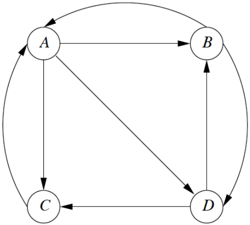


Random walking a web with four pages

Adopted from `mmds.org`

- For example, a *random surfer* starts at node A
- Walks to B, C, D each with probability $1/3$
- So has probability 0 to be at A after first step

PageRank: Definition



Random walking a web with four pages

Adopted from `mmds.org`

- *Random surfer* at B, for example, in next step
  - is at A, D each with probability $1/2$
  - is at B, C with probability 0

Web Transition Matrix: Definition

Definition [Web Transition Matrix]:

- Let n be the number of pages in the web
- The *transition matrix* $M = (m_{ij})_{1 \leq i,j \leq n} \in \mathbb{R}^{n \times n}$ has n rows and columns
- For each $(i, j) \in \{1, ..., n\} \times \{1, ..., n\}$
  - $m_{ij} = 1/k$, if page j has k arcs out, of which one leads to page i
  - $m_{ij} = 0$ otherwise

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

Transition matrix for web from slides before

Adopted from `mmds.org`

PageRank Function: Definition

Definition [PageRank Function]:

- Let n be the number of pages in the web
- Let $p_i^t, i = 1, ..., n$ be the probability that the random surfer is at page i after t steps
- The *PageRank function* for $t \geq 0$ is defined to be the vector

$$p^t = (p_1^t, p_2^t, ..., p_n^t) \in [0, 1]^n$$

PageRank Function: Interpretation

- Usually, $p^0 = (1/n, ...1/n)$ for each $i = 1, ..., n$
- So before the first iteration, the random surfer is at each page with equal probability
- The probability to be at page i in step $t + 1$ is the sum of probabilities to be at page j in step t times the probability to move from page j to i
- That is, $p_i^{t+1} = \sum_{j=1}^{n} m_{ij} p_j^t$ for all i, t, or, in other words

$$p^{t+1} = Mp^t \quad \text{for all } t \geq 0 \tag{2}$$

- So, applying the web transition matrix to a PageRank function yields another one

PageRank Function: Markov Processes

$$\boxed{p^{t+1} = Mp^t \quad \text{for all } t \geq 0}$$

- This relates to the theory of *Markov processes*
- Given that the web graph is *strongly connected*
  - That is: one can reach any node from any other node
  - In particular, there are no *dead ends*, nodes with no arcs out
- it is known that the surfer reaches a *limiting distribution* $\bar{p}$, characterized by

$$M\bar{p} = \bar{p} \tag{3}$$

PAGERANK FUNCTION: MARKOV PROCESSES

$$\boxed{M\bar{p} = \bar{p}}$$

- Further, because M is *stochastic* (= columns each add up to one)
  - $\bar{p}$ is the *principal eigenvector*, which is
  - the eigenvector associated with the largest eigenvalue, which is one
- Principal eigenvector of M expresses where surfer will end up
- $\bar{p}_i$ is the probability that the surfer is at page i after a long time
- *Reasoning:* The greater $\bar{p}_i$, the more important page i

DEFINITION [PAGERANK]:

$\bar{p}_i$ is the *PageRank* of web page i

PageRank Function: Computation

$$\boxed{M\bar{p} = \bar{p}}$$

- Consider the series

$$p^0, p^1 = Mp^0, p^2 = Mp^1 = M^2p^0, p^3 = Mp^2 = M^3p^0, ... \tag{4}$$

- It holds that

$$M^t p^0 \quad \underset{t \to \infty}{\longrightarrow} \quad \bar{p} \tag{5}$$

- So, for *computing* $\bar{p}$, apply iterative matrix-vector multiplication until (approximate) convergence

PageRank Function: Computation

$$\boxed{M\bar{p} = \bar{p}}$$

- For *computing* $\bar{p}$, apply iterative matrix-vector multiplication

$$p^0 \to Mp^0 \to M^2p^0 \to M^3p^0 \to \ldots \tag{6}$$

  until (approximate) convergence
- *Example:* Iterative application of transition matrix from above

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix}, \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix}, \ldots, \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

Convergence to limiting distribution for four-node web graph

Adopted from `mmds.org`

PageRank Function: Computation

$$\boxed{M\bar{p} = \bar{p}}$$

- It holds that

$$M^t p_0 \quad \underset{t\to\infty}{\longrightarrow} \quad \bar{p} \tag{7}$$

- So, for *computing* $\bar{p}$, apply iterative matrix-vector multiplication until (approximate) convergence
- In practice, working real web graphs
  - 50-75 iterations do just fine
  - For *efficient computation*, recall MapReduce based matrix-vector multiplication techniques

Materials / Outlook

- See *Mining of Massive Datasets*, chapters 2.4–2.5, 5.1
- As usual, see `http://www.mmds.org/` in general for further resources
- Next lecture: “Link Analysis II”
  - See *Mining of Massive Datasets* chapter 5.3–5.5