


inst.eecs.berkeley.edu/~cs61c



UCB CS61C : Machine Structures

Guest Lecturer
Sagar Karandikar

Lecture 18 – RLP, MapReduce

03-05-2014

3/05/2014 Spring 2014 – Lecture #18 1

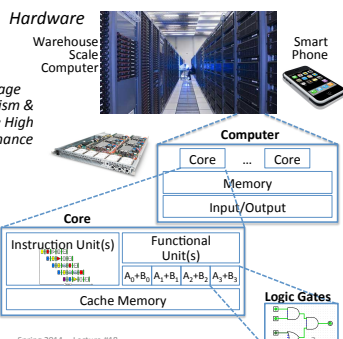
Review of Last Lecture

- Warehouse Scale Computing
 - Example of parallel processing in the post-PC era
 - Servers on a rack, rack part of cluster
 - Issues to handle include **load balancing, failures, power usage** (sensitive to cost & energy efficiency)
 - PUE** = Total building power / IT equipment power
 - [ECS PUE Stats Demo](#) (B: 165 Cory, G: 288 Soda)

3/05/2014 Spring 2014 – Lecture #18 2

Great Idea #4: Parallelism

Today's Lecture

Software	Hardware
<ul style="list-style-type: none"> Parallel Requests Assigned to computer e.g. Search "Garcia" Parallel Threads Assigned to core e.g. Lookup, Ads Parallel Instructions > 1 instruction @ one time e.g. 5 pipelined instructions Parallel Data > 1 data item @ one time e.g. add of 4 pairs of words Hardware descriptions All gates functioning in parallel at same time 	<p style="text-align: center;">Leverage Parallelism & Achieve High Performance</p> 

3/05/2014 Spring 2014 – Lecture #18

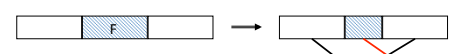
Agenda

- Amdahl's Law**
- Request Level Parallelism
- MapReduce (Data Level Parallelism)
 - Background
 - Design
 - Theory
- Administrivia
- More MapReduce
 - The Combiner + Example 1: Word Count
 - Execution Walkthrough
 - (Bonus) Example 2: PageRank (aka How Google Search Works)

3/05/2014 Spring 2014 – Lecture #18 4

Amdahl's (Heartbreaking) Law

- Speedup due to enhancement E:

$$\text{Speedup } w/E = \frac{\text{Exec time } w/o E}{\text{Exec time } w/E}$$
- Example:** Suppose that enhancement E accelerates a fraction F (F<1) of the task by a factor S (S>1) and the remainder of the task is unaffected
 
- Exec time w/E = Exec Time w/o E × [(1-F) + F/S]
Speedup w/E = 1 / [(1-F) + F/S]

3/05/2014 Spring 2014 – Lecture #18 5

Amdahl's Law

- Speedup = $\frac{1}{(1-F) + \frac{F}{S}}$

Non-spiced-up part ← (1 - F) ← F/S → Spiced-up part
- Example:** the execution time of half of the program can be accelerated by a factor of 2. What is the program speed-up overall?

$$\frac{1}{0.5 + \frac{0.5}{2}} = \frac{1}{0.5 + 0.25} = 1.33$$

3/05/2014 Spring 2014 – Lecture #18 6

Consequence of Amdahl's Law

- The amount of speedup that can be achieved through parallelism is limited by the non-parallel portion of your program!

3/05/2014 Spring 2014 -- Lecture #18 7

Agenda

- Amdahl's Law
- Request Level Parallelism**
- MapReduce (Data Level Parallelism)
 - Background
 - Design
 - Theory
- Administrivia
- More MapReduce
 - The Combiner + Example 1: Word Count
 - Execution Walkthrough
 - (Bonus) Example 2: PageRank (aka How Google Search Works)

3/05/2014 Spring 2014 -- Lecture #18 8

Request-Level Parallelism (RLP)

- Hundreds or thousands of requests per sec
 - Not your laptop or cell-phone, but popular Internet services like web search, social networking, ...
 - Such requests are largely independent
 - Often involve read-mostly databases
 - Rarely involve strict read-write data sharing or synchronization across requests
- Computation easily partitioned within a request and across different requests

3/05/2014 Spring 2014 -- Lecture #18 9

Google Query-Serving Architecture

3/05/2014 Spring 2014 -- Lecture #18 10

Anatomy of a Web Search

- Google "Dan Garcia"

3/05/2014 Spring 2014 -- Lecture #18 11

Anatomy of a Web Search (1 of 3)

- Google "Dan Garcia"
 - Direct request to "closest" Google Warehouse Scale Computer
 - Front-end load balancer directs request to one of many arrays (cluster of servers) within WSC
 - Within array, select one of many Google Web Servers (GWS) to handle the request and compose the response pages
 - GWS communicates with Index Servers to find documents that contain the search words, "Dan", "Garcia", uses location of search as well
 - Return document list with associated relevance score

3/05/2014 Spring 2014 -- Lecture #18 12

Anatomy of a Web Search (2 of 3)

- In parallel,
 - Ad system: run ad auction for bidders on search terms
 - Get images of various Dan Garcias
- Use docids (document IDs) to access indexed documents
- Compose the page
 - Result document extracts (with keyword in context) ordered by relevance score
 - Sponsored links (along the top) and advertisements (along the sides)

3/05/2014

Spring 2014 -- Lecture #18

13

Anatomy of a Web Search (3 of 3)

- Implementation strategy
 - Randomly distribute the entries
 - Make many copies of data (a.k.a. “replicas”)
 - Load balance requests across replicas
- Redundant copies of indices and documents
 - Breaks up search hot spots, e.g. “WhatsApp”
 - Increases opportunities for request-level parallelism
 - Makes the system more tolerant of failures

3/05/2014

Spring 2014 -- Lecture #18

14

Agenda

- Amdahl’s Law
- Request Level Parallelism
- **MapReduce (Data Level Parallelism)**
 - Background
 - Design
 - Theory
- Administrivia
- More MapReduce
 - The Combiner + Example 1: Word Count
 - Execution Walkthrough
 - (Bonus) Example 2: PageRank (aka How Google Search Works)

3/05/2014

Spring 2014 -- Lecture #18

15

Data-Level Parallelism (DLP)

- Two kinds:
 - 1) Lots of **data in memory** that can be operated on in parallel (e.g. adding together 2 arrays)
 - 2) Lots of **data on many disks** that can be operated on in parallel (e.g. searching for documents)
- 1) SIMD does Data-Level Parallelism (DLP) in memory
- 2) Today’s lecture, Lab 6, Proj. 3 do DLP across many servers and disks using **MapReduce**

3/05/2014

Spring 2014 -- Lecture #18

16

What is MapReduce?

- Simple data-parallel programming model designed for scalability and fault-tolerance
- Pioneered by Google
 - Processes > 25 petabytes of data per day
- Popularized by open-source Hadoop project
 - Used at Yahoo!, Facebook, Amazon, ...



3/05/2014

Spring 2014 -- Lecture #18

17

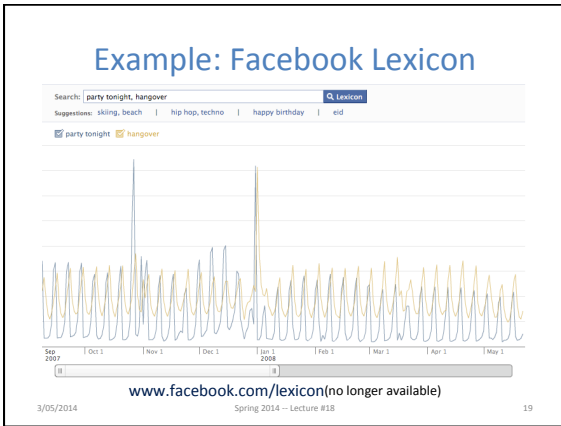
What is MapReduce used for?

- At Google:
 - Index construction for Google Search
 - Article clustering for Google News
 - Statistical machine translation
 - For computing multi-layer street maps
- At Yahoo!:
 - “Web map” powering Yahoo! Search
 - Spam detection for Yahoo! Mail
- At Facebook:
 - Data mining
 - Ad optimization
 - Spam detection

3/05/2014

Spring 2014 -- Lecture #18

18



MapReduce Design Goals

1. Scalability to large data volumes:
 - 1000's of machines, 10,000's of disks
2. Cost-efficiency:
 - Commodity machines (cheap, but unreliable)
 - Commodity network
 - Automatic fault-tolerance (fewer administrators)
 - Easy to use (fewer programmers)

Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," 6th USENIX Symposium on Operating Systems Design and Implementation, 2004. (optional reading, linked on course homepage – a digestible CS paper at the 61C level)

3/05/2014 Spring 2014 -- Lecture #18 20

MapReduce Processing: "Divide and Conquer" (1/3)

- Apply **Map** function to user supplied record of key/value pairs
 - Slice data into "shards" or "splits" and distribute to workers
 - Compute set of intermediate key/value pairs

```
map(in_key, in_val) :
    // DO WORK HERE
    emit(inter_key, inter_val)
```

3/05/2014 Spring 2014 -- Lecture #18 21

MapReduce Processing: "Divide and Conquer" (2/3)

- Apply **Reduce** operation to all values that share same key in order to combine derived data properly
 - Combines all intermediate values for a particular key
 - Produces a set of merged output values

```
reduce(inter_key, list(inter_val)) :
    // DO WORK HERE
    emit(out_key, out_val)
```

3/05/2014 Spring 2014 -- Lecture #18 22

MapReduce Processing: "Divide and Conquer" (3/3)

- User supplies Map and Reduce operations in functional model
 - Focus on problem, let MapReduce library deal with messy details
 - Parallelization handled by framework/library
 - Fault tolerance via re-execution
 - Fun to use!

3/05/2014 Spring 2014 -- Lecture #18 23

Typical Hadoop Cluster

- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth within rack, 8 Gbps out of rack
- Node specs (Yahoo terasort): 8 x 2GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

3/05/2014 Spring 2014 -- Lecture #18 24

Agenda

- Amdahl's Law
- Request Level Parallelism
- MapReduce (Data Level Parallelism)
 - Background
 - Design
 - Theory
- **Administrivia**
- **More MapReduce**
 - The Combiner + Example 1: Word Count
 - Execution Walkthrough
 - (Bonus) Example 2: PageRank (aka How Google Search Works)

3/05/2014

Spring 2014 -- Lecture #18

25

Administrivia

3/05/2014

Spring 2014 -- Lecture #18

26

Agenda

- Amdahl's Law
- Request Level Parallelism
- MapReduce (Data Level Parallelism)
 - Background
 - Design
 - Theory
- **Administrivia**
- **More MapReduce**
 - **The Combiner + Example 1: Word Count**
 - Execution Walkthrough
 - (Bonus) Example 2: PageRank (aka How Google Search Works)

3/05/2014

Spring 2014 -- Lecture #18

27

The Combiner (Optional)

- One missing piece for our first example:
 - Many times, the output of a single mapper can be “compressed” to save on bandwidth and to distribute work (usually more map tasks than reduce tasks)
 - To implement this, we have the combiner:
- ```
combiner(inter_key, list(inter_val)):
 // DO WORK (usually like reducer)
 emit(inter_key2, inter_val2)
```

3/05/2014

Spring 2014 -- Lecture #18

28

## Our Final Execution Sequence

- **Map** – Apply operations to all input key, val
- **Combine** – Apply reducer operation, but distributed across map tasks
- **Reduce** – Combine all values of a key to produce desired output

3/05/2014

Spring 2014 -- Lecture #18

29

## MapReduce Processing Example: Count Word Occurrences (1/2)

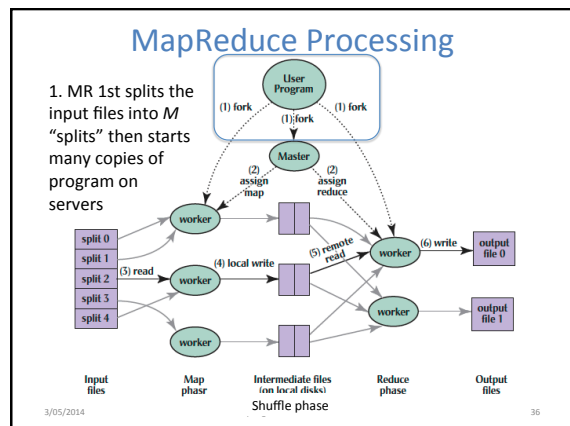
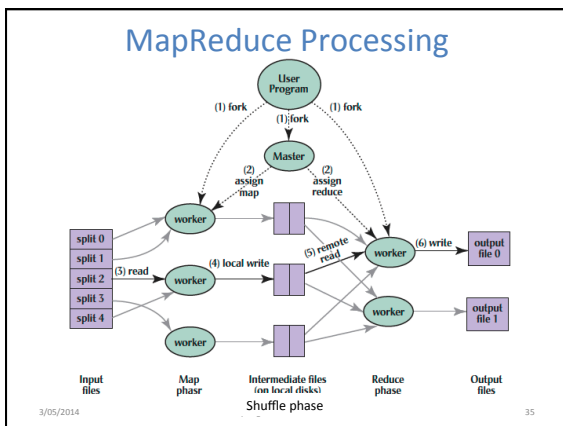
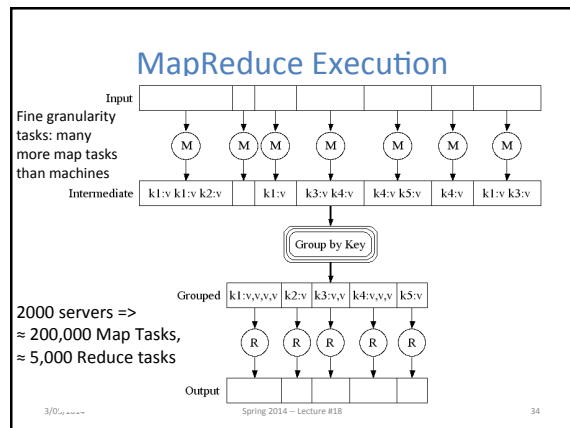
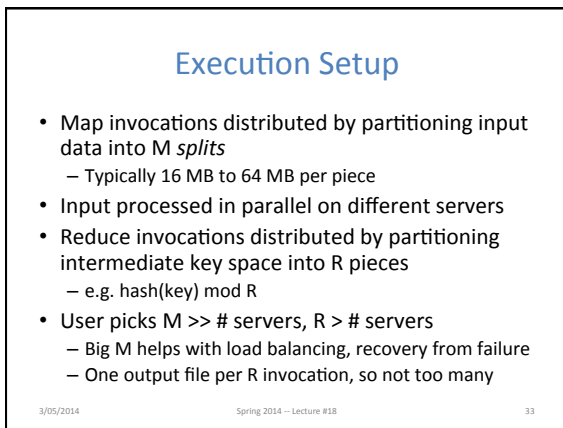
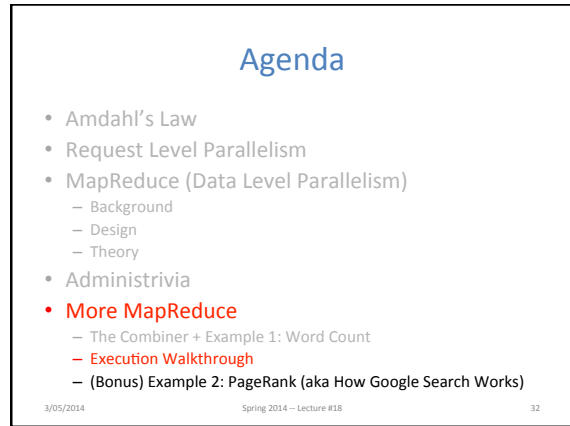
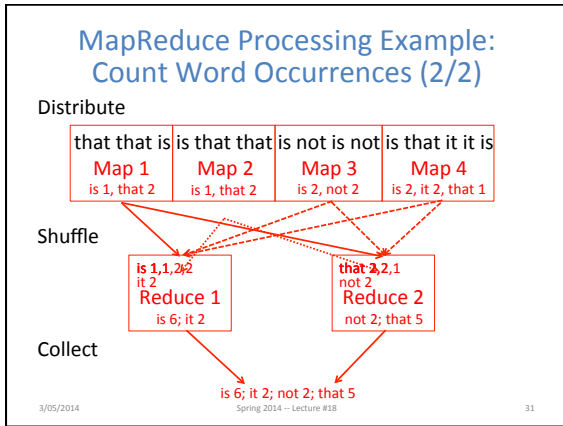
- Pseudo Code: for each word in input, generate <key=word, value=1>
- Reduce sums all counts emitted for a particular word across all mappers

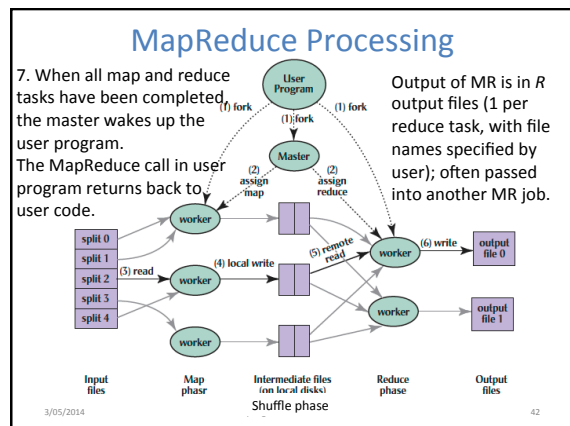
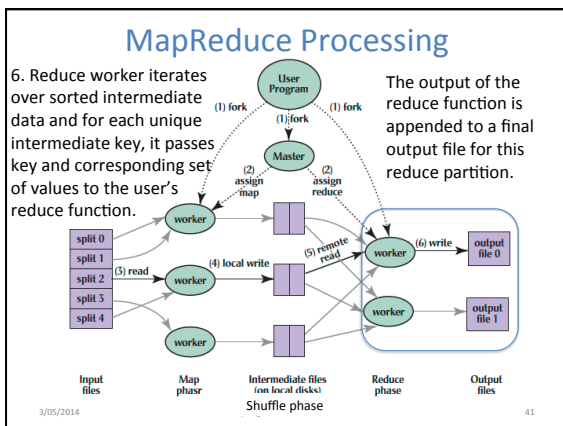
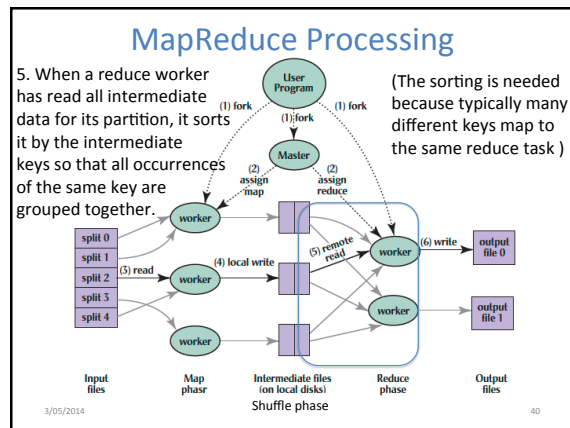
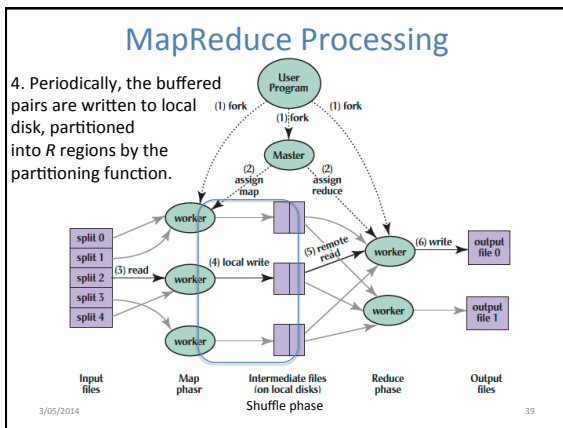
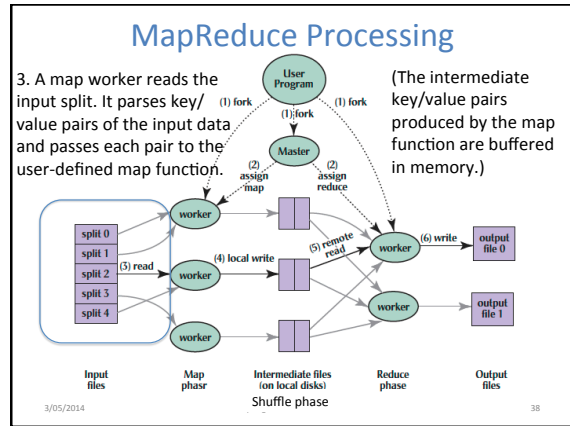
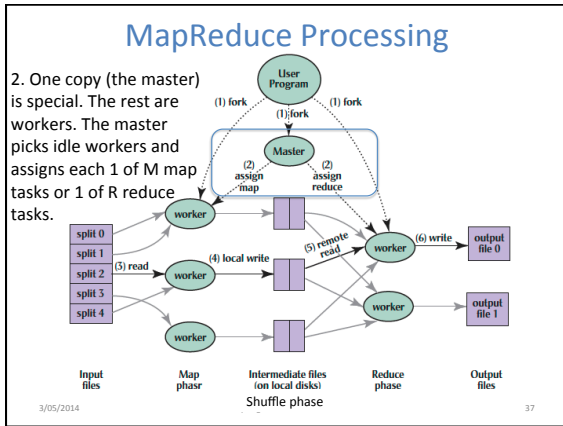
```
map(String input_key, String input_value):
 // input_key: document name
 // input_value: document contents
 for each word w in input_value:
 EmitIntermediate(w, "1"); // Produce count of words
combiner: (same as below reducer)
reduce(String output_key, Iterator intermediate_values):
 // output_key: a word
 // intermediate_values: a list of counts
 int result = 0;
 for each v in intermediate_values:
 result += ParseInt(v); // get integer from key-value
 Emit(output_key, result);
```

3/05/2014

Spring 2014 -- Lecture #18

30





## What Does the Master Do?

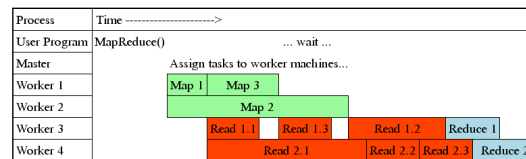
- For each map task and reduce task, keep track:
  - State: idle, in-progress, or completed
  - Identity of worker server (if not idle)
- For each completed map task
  - Stores location and size of R intermediate files
  - Updates files and size as corresponding map tasks complete
- Location and size are pushed incrementally to workers that have in-progress reduce tasks

3/05/2014

Spring 2014 -- Lecture #18

43

## MapReduce Processing Time Line



- Master assigns map + reduce tasks to “worker” servers
- As soon as a map task finishes, worker server can be assigned a new map or reduce task
- Data shuffle begins as soon as a given Map finishes
- Reduce task begins as soon as all data shuffles finish
- To tolerate faults, reassign task if a worker server “dies”

3/05/2014

Spring 2014 -- Lecture #18

44

## MapReduce Failure Handling

- On worker failure:
  - Detect failure via periodic heartbeats
  - Re-execute completed and in-progress map tasks
  - Re-execute in progress reduce tasks
  - Task completion committed through master
- Master failure:
  - Protocols exist to handle (master failure unlikely)
- Robust: lost 1600 of 1800 machines once, but finished fine

3/05/2014

Spring 2014 -- Lecture #18

45

## MapReduce Redundant Execution

- Slow workers significantly lengthen completion time
  - Other jobs consuming resources on machine
  - Bad disks with soft errors transfer data very slowly
  - Weird things: processor caches disabled (!!)
- Solution: Near end of phase, spawn backup copies of tasks
  - Whichever one finishes first “wins”
- Effect: Dramatically shortens job completion time
  - 3% more resources, large tasks 30% faster

3/05/2014

Spring 2014 -- Lecture #18

46

## Summary

- MapReduce Data Parallelism
  - Divide large data set into pieces for independent parallel processing
  - Combine and process intermediate results to obtain final result
- Simple to Understand
  - But we can still build complicated software
  - Chaining lets us use the MapReduce paradigm for many common graph and mathematical tasks
- MapReduce is a “Real-World” Tool
  - Worker restart, monitoring to handle failures
  - Google PageRank, Facebook Analytics

3/05/2014

Spring 2014 -- Lecture #18

47

Bonus!

3/05/2014

Spring 2014 -- Lecture #18

48



## Agenda

- Amdahl's Law
- Request Level Parallelism
- MapReduce (Data Level Parallelism)
  - Background
  - Design
  - Theory
- Administrivia
- **More MapReduce**
  - The Combiner + Example 1: Word Count
  - Execution Walkthrough
  - **(Bonus) Example 2: PageRank (aka How Google Search Works)**

3/05/2014

Spring 2014 -- Lecture #18

49

## PageRank: How Google Search Works

- Last time: RLP – how Google handles searching its huge index
- Now: How does Google generate that index?
- PageRank is the famous algorithm behind the “quality” of Google’s results
  - Uses link structure to rank pages, instead of matching only against content (keyword)

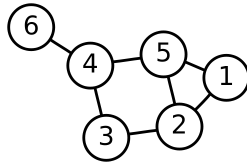
3/05/2014

Spring 2014 -- Lecture #18

50

## A Quick Detour to CS Theory: Graphs

- **Def:** A set of objects connected by links
- The “objects” are called Nodes
- The “links” are called Edges
- Nodes: {1, 2, 3, 4, 5, 6}
- Edges: {(6, 4), (4, 5), (4, 3), (3, 2), (5, 2), (5, 1), (1, 2)}



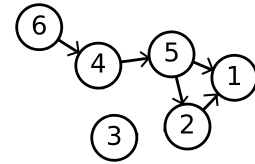
3/05/2014

Spring 2014 -- Lecture #18

51

## Directed Graphs

- Previously assumed that all edges in the graph were two-way
- Now we have one-way edges:
- Nodes: Same as before
- Edges: (order matters)
  - {(6, 4), (4, 5), (5, 1), (5, 2), (2, 1)}



3/05/2014

Spring 2014 -- Lecture #18

52

## The Theory Behind PageRank

- The Internet is really a directed graph:
  - Nodes: webpages
  - Edges: links between webpages
- Terms (Suppose we have a page A that links to page B):
  - Page A has a forward-link to page B
  - Page B has a back-link from page A

3/05/2014

Spring 2014 -- Lecture #18

53

## The Magic Formula

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

3/05/2014

Spring 2014 -- Lecture #18

54

### The Magic Formula

Node  $u$  is the vertex (webpage) we're interested in computing the PageRank of

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

3/05/2014

Spring 2014 -- Lecture #18

55

### The Magic Formula

$R'(u)$  is the PageRank of Node  $u$

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

3/05/2014

Spring 2014 -- Lecture #18

56

### The Magic Formula

$c$  is a normalization factor that we can ignore for our purposes

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

3/05/2014

Spring 2014 -- Lecture #18

57

### The Magic Formula

$E(u)$  is a "personalization" factor that we can ignore for our purposes

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

3/05/2014

Spring 2014 -- Lecture #18

58

### The Magic Formula

We sum over all backlinks of  $u$ : the PageRank of the website  $v$  linking to  $u$  divided by the number of forward-links that  $v$  has

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u)$$

3/05/2014

Spring 2014 -- Lecture #18

59

### But wait! This is Recursive!

- Uh oh! We have a recursive formula with no base-case
- We rely on convergence
  - Choose some initial PageRank value for each site
  - Simultaneously compute/update PageRanks
  - When our Delta is small between iterations:
    - Stop, call it "good enough"

3/05/2014

Spring 2014 -- Lecture #18

60

### Sounds Easy. Why MapReduce?

- Assume in the best case that we've crawled and captured the internet as a series of (url, outgoing links) pairs
- We need about 50 iterations of the PageRank algorithm for it to converge
- We quickly see that running it on one machine is not viable

3/05/2014 Spring 2014 -- Lecture #18 61

### Building a Web Index using PageRank

- Scrape Webpages
- Strip out content, keep only links (input is key = url, value = links on page at url)
  - This step is actually pushed into the MapReduce
- Feed into PageRank Mapreduce
- Sort Documents by PageRank
- Post-process to build the indices that our Google RLP example used

3/05/2014 Spring 2014 -- Lecture #18 62

### Using MapReduce to Compute PageRank, Step 1

|                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• <b>Map:</b> <ul style="list-style-type: none"> <li>– Input:                             <ul style="list-style-type: none"> <li>• key = URL of website</li> <li>• val = source of website</li> </ul> </li> <li>– Output for each outgoing link:                             <ul style="list-style-type: none"> <li>• key = URL of website</li> <li>• val = outgoing link url</li> </ul> </li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• <b>Reduce:</b> <ul style="list-style-type: none"> <li>– Input:                             <ul style="list-style-type: none"> <li>• key = URL of website</li> <li>• values = Iterable of all outgoing links from that website</li> </ul> </li> <li>– Output:                             <ul style="list-style-type: none"> <li>• key = URL of website</li> <li>• value = Starting PageRank, Outgoing links from that website</li> </ul> </li> </ul> </li> </ul> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

3/05/2014 Spring 2014 -- Lecture #18 63

### Using MapReduce to Compute PageRank, Step 2

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• <b>Map:</b> <ul style="list-style-type: none"> <li>– Input:                             <ul style="list-style-type: none"> <li>• key = URL of website</li> <li>• val = PageRank, Outgoing links from that website</li> </ul> </li> <li>– Output for each outgoing link:                             <ul style="list-style-type: none"> <li>• key = Outgoing Link URL</li> <li>• val = Original Website URL, PageRank, # Outgoing links</li> </ul> </li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• <b>Reduce:</b> <ul style="list-style-type: none"> <li>– Input:                             <ul style="list-style-type: none"> <li>• key = Outgoing Link URL</li> <li>• values = Iterable of all links to Outgoing Link URL</li> </ul> </li> <li>– Output:                             <ul style="list-style-type: none"> <li>• key = Outgoing Link URL</li> <li>• value = Newly computed PageRank (using the formula), Outgoing links from document @ Outgoing Link URL</li> </ul> </li> </ul> </li> </ul> <p style="color: red; font-size: small;">Repeat this step until PageRank converges – chained MapReduce!</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

3/05/2014 Spring 2014 -- Lecture #18 64

### Using MapReduce to Compute PageRank, Step 3

|                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• Finally, we want to sort by PageRank to get a useful index</li> <li>• MapReduce's built in sorting makes this easy!</li> </ul> | <ul style="list-style-type: none"> <li>• <b>Map:</b> <ul style="list-style-type: none"> <li>– Input:                             <ul style="list-style-type: none"> <li>• key = Website URL</li> <li>• value = PageRank, Outgoing Links</li> </ul> </li> <li>– Output:                             <ul style="list-style-type: none"> <li>• key = PageRank</li> <li>• value = Website URL</li> </ul> </li> </ul> </li> </ul> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

3/05/2014 Spring 2014 -- Lecture #18 65

### Using MapReduce to Compute PageRank, Step 3

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• <b>Reduce:</b> <ul style="list-style-type: none"> <li>– In case we have duplicate PageRanks</li> <li>– Input:                             <ul style="list-style-type: none"> <li>• key = PageRank</li> <li>• value = Iterable of URLs with that PageRank</li> </ul> </li> <li>– Output (for each URL in the Iterable):                             <ul style="list-style-type: none"> <li>• key = PageRank</li> <li>• value = Website URL</li> </ul> </li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• Since MapReduce automatically sorts the output from the reducer and joins it together:</li> <li>• We're done!</li> </ul> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|

3/05/2014 Spring 2014 -- Lecture #18 66

## Using the PageRanked Index

- Do our usual keyword search with RLP implemented
- Take our results, sort by our pre-generated PageRank values
- Send results to user!
- PageRank is still the basis for Google Search
  - (of course, there are many proprietary enhancements in addition)

3/05/2014

Spring 2014 -- Lecture #18

67

## Further Reading (Optional)

- Some PageRank slides adapted from <http://www.cs.toronto.edu/~jasper/PageRankForMapReduceSmall.pdf>
- PageRank Paper:
  - [Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd. \*The PageRank Citation Ranking: Bringing Order to the Web.\*](#)

3/05/2014

Spring 2014 -- Lecture #18

68