Distributed Systems

COS 418: *Distributed Systems*
Lecture 1

Mike Freedman

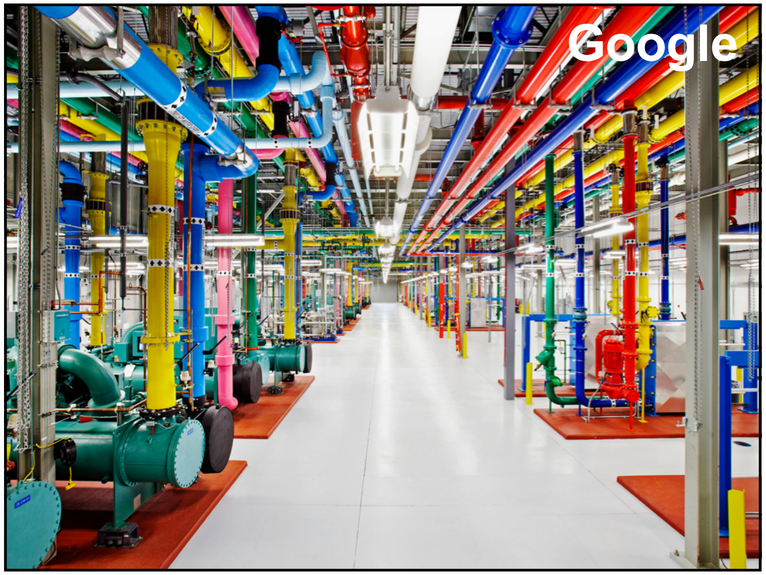
Backrub (Google) 1997

2

"The Cloud" is not amorphous

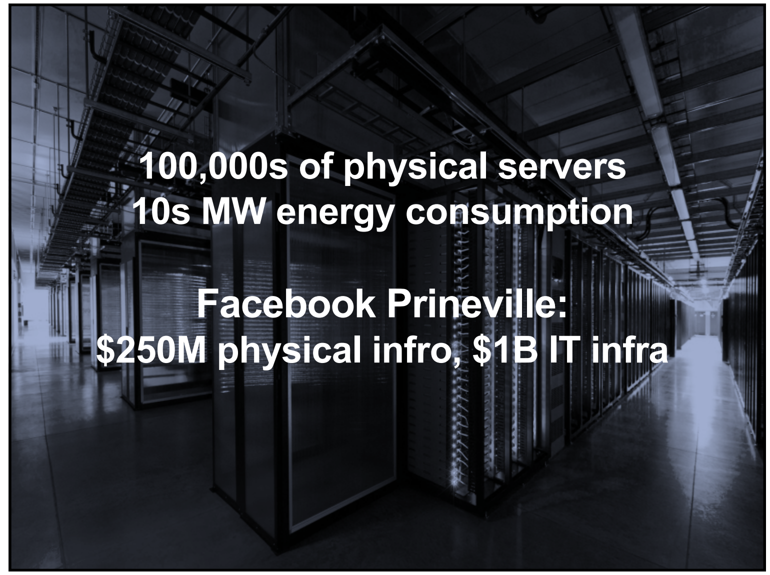
4


Microsoft
5

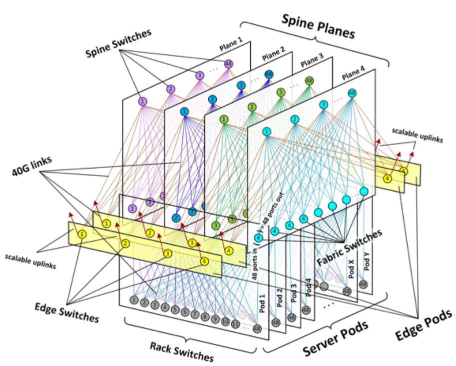

Google


Facebook


8



Everything changes at scale

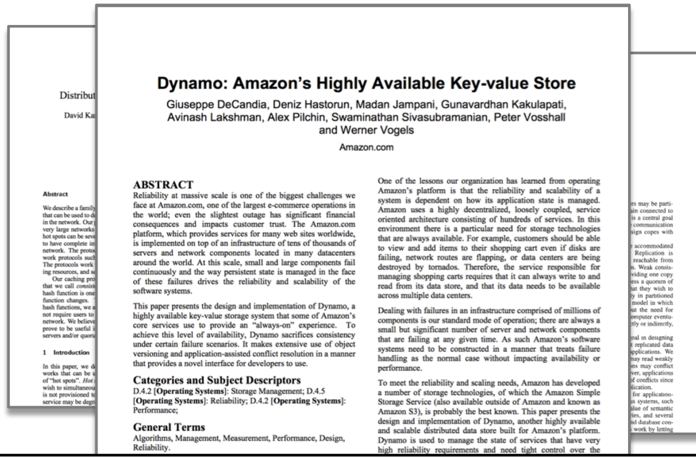


"Pods provide 7.68Tbps to backplane"

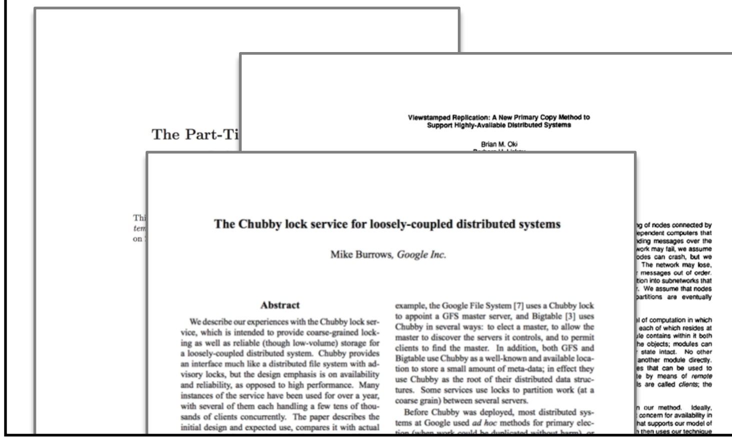
The goal of "distributed systems"

- Service with higher-level abstractions/interface
  - e.g., file system, database, key-value store, programming model, RESTful web service, …
- Hide complexity
  - Scalable (scale-out)
  - Reliable (fault-tolerant)
  - Well-defined semantics (consistent)
  - Security
- Do "heavy lifting" so app developer doesn't need to

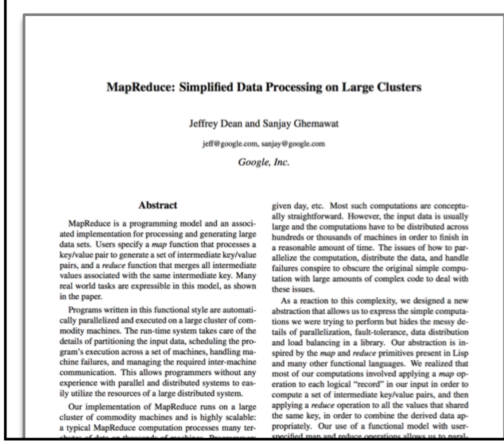
Research results matter: NoSQL



Research results matter: Paxos



Research results matter: MapReduce



hadoop
Spark
kafka
Flink
S4 distributed stream computing platform
STORM

Course Organization

Learning the material: People

- Lecture
  - Professors Mike Freedman, Kyle Jamieson
  - Slides available on course website
  - Office hours immediately after lecture
- Precept:
  - TAs Themis Melissaris, Daniel Suo
- Main Q&A forum: www.piazza.com
  - Graded on class participation: so ask & answer!
  - No anonymous posts or questions
  - Can send private messages to instructors

17

Learning the Material: Books

- Lecture notes!
- No required textbooks.
  - Programming reference:
    - *The Go Programming Language,* Alan Donovan and Brian Kernighan (www.gopl.io, $17 Amazon!)
  - Topic reference:
    - *Distributed Systems: Principles and Paradigms.* Andrew S. Tanenbaum and Maaten Van Steen
    - *Guide to Reliable Distributed Systems.* Kenneth Birman

18

Grading

- Five assignments (5% for first, then 10% each)
  - 90% 24 hours late, 80% 2 days late, 50% >5 days late
  - **THREE** free late days (we'll figure which one is best)
  - Only failing grades I've given are for students who don't (try to) do assignments
- Two exams (50% total)
  - Midterm exam before spring break (25%)
  - Final exam during exam period (25%)
- Class participation (5%)
  - In lecture, precept, and Piazza

19

Policies: Write Your Own Code

Programming is an individual creative process. At first, discussions with friends is fine. When writing code, however, the program must be your own work.

Do not copy another person's programs, comments, README description, or any part of submitted assignment. This includes character-by-character transliteration but also derivative works. Cannot use another's code, etc. even while "citing" them.

Writing code for use by another or using another's code is academic fraud in context of coursework.

Do not publish your code e.g., on github, during/after course!

20

Assignment 1

- Learn how to program in Go
  - Implement “sequential” Map/Reduce
  - Instructions on assignment web page
  - Due September 28 (two weeks)

meClickers®: Quick Surveys

- Why are you here?
  A. Needed to satisfy course requirements
  B. Want to learn Go or distributed programming
  C. Interested in concepts behind distributed systems
  D. Thought this course was “Bridges”

Case Study: MapReduce

(Data-parallel programming at scale)

Application: Word Count

```
SELECT count(word) FROM data
  GROUP BY word

cat data.txt
  | tr -s '[[:punct:][:space:]]' '\n'
  | sort | uniq -c
```

Using partial aggregation

1. Compute word counts from individual files
2. Then merge intermediate output
3. Compute word count on merged outputs

25

Using partial aggregation

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect result, wait until all finished
2. Then merge intermediate output
3. Compute word count on merged intermediates

26

MapReduce: Programming Interface

```
map(key, value) -> list(<k', v'>)
```

- Apply function to (key, value) pair and produces set of intermediate pairs

```
reduce(key, list<value>) -> <k', v'>
```

- Applies aggregation function to values
- Outputs result

27

MapReduce: Programming Interface

```
map(key, value):
  for each word w in value:
    EmitIntermediate(w, "1");


reduce(key, list(values):
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

28

MapReduce: Optimizations

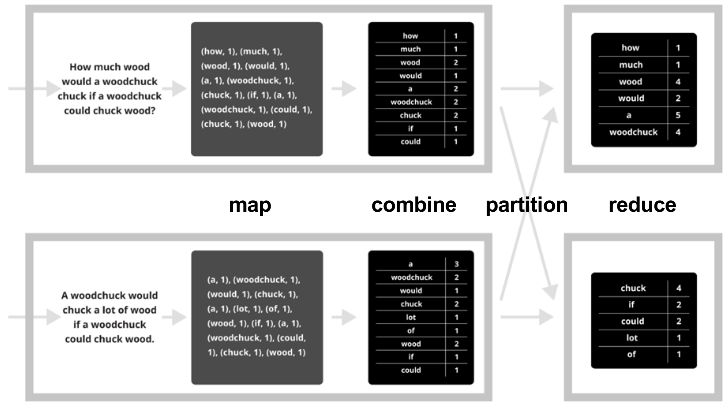
`combine(list<key, value>) -> list<k,v>`

- Perform partial aggregation on mapper node:
  - <the, 1>, <the, 1>, <the, 1> → <the, 3>
- reduce() should be commutative and associative

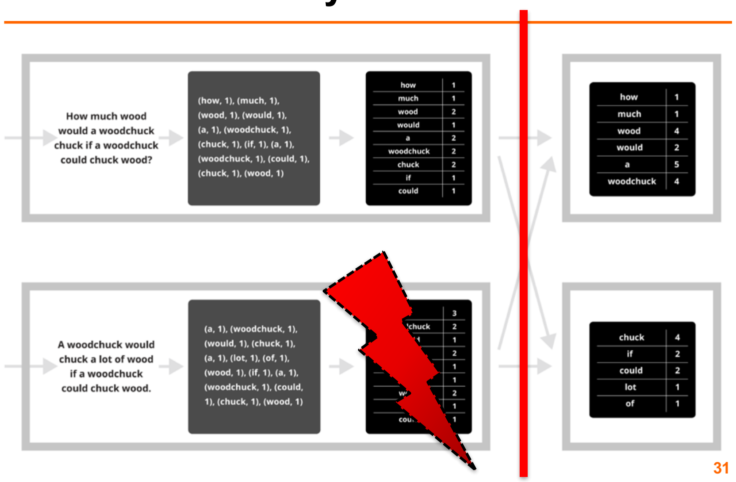
`partition(key, int) -> int`

- Need to aggregate intermediate vals with same key
- Given n partitions, map key to partition $0 \le i < n$
- Typically via hash(key) mod n

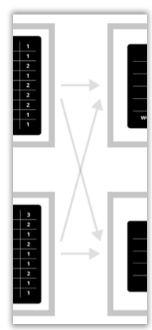
Putting it together…



Synchronization Barrier



Fault Tolerance in MapReduce



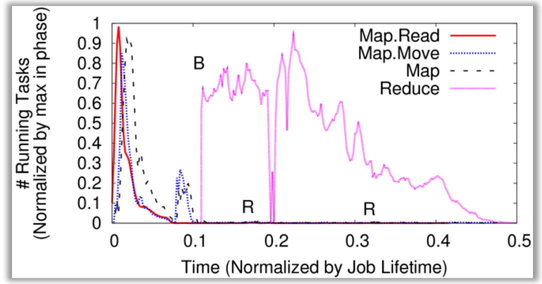
- Map worker writes intermediate output to local disk, separated by partitioning. Once completed, tells master node.
- Reduce worker told of location of map task outputs, pulls their partition's data from each mapper, execute function across data
- Note:
  - "All-to-all" shuffle b/w mappers and reducers
  - Written to disk ("materialized") b/w *each* stage

Fault Tolerance in MapReduce

- Master node monitors state of system
  - If master failures, job aborts and client notified
- Map worker failure
  - Both in-progress/completed tasks marked as idle
  - Reduce workers notified when map task is re-executed on another map worker
- Reducer worker failure
  - In-progress tasks are reset to idle (and re-executed)
  - Completed tasks had been written to global file system

33

Straggler Mitigation in MapReduce



- Tail latency means some workers finish late
- For slow map tasks, execute in parallel on second map worker as "backup", race to complete task

34

You'll build (simplified) MapReduce!

- Assignment 1: Sequential Map/Reduce
  - Learn to program in Go!
  - Due September 28 (two weeks)
- Assignment 2: Distributed Map/Reduce
  - Learn Go's concurrency, network I/O, and RPCs
  - Due October 19

35

This Friday
"Grouped" Precept, Room CS 105

"Program your next service in Go"
Sameer Ajmani

Manages Go lang team @ Google
(Earlier: PhD w/ Barbara Liskov @ MIT)

36