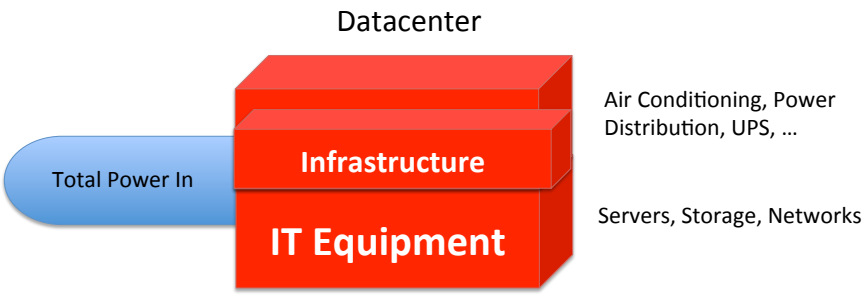
CS 61C:
Great Ideas in Computer Architecture (Machine Structures)
Map Reduce

Instructors
Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa13

PUE Revisited

- Power Utilization Efficiency



PUE = Total Power/IT Power

PUE = 2.5

Energy Proportionality

"The Case for Energy-Proportional Computing,"
Luiz André Barroso, Urs Hölzle,
IEEE Computer
December 2007

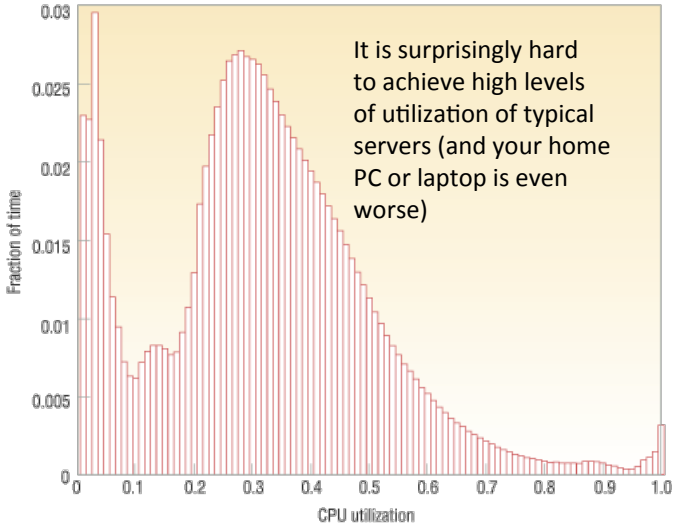
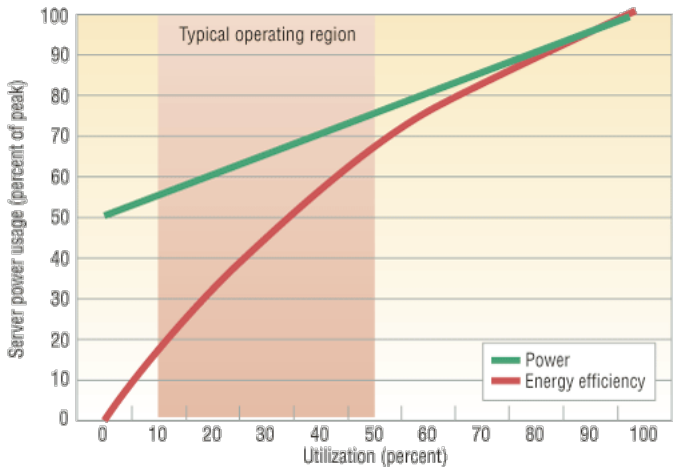


Figure 1. Average CPU utilization of more than 5,000 servers during a six-month period. Servers are rarely completely idle and seldom operate near their maximum utilization, instead operating most of the time at between 10 and 50 percent of their maximum

3

Energy Proportional Computing

"The Case for Energy-Proportional Computing,"
Luiz André Barroso, Urs Hölzle,
IEEE Computer
December 2007



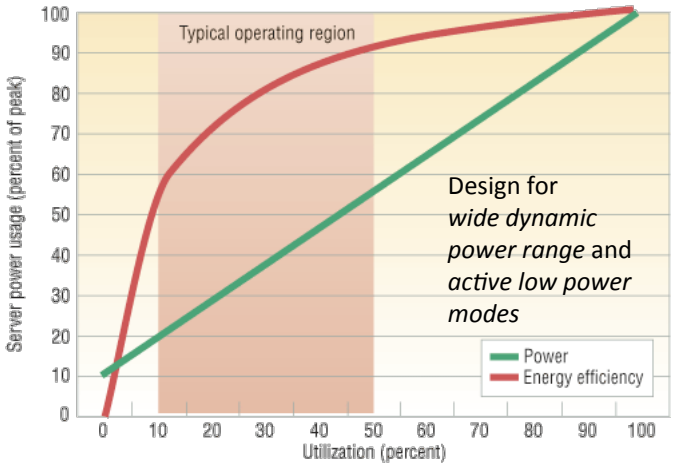
Energy Efficiency = Utilization/Power

Figure 2. Server power usage and energy efficiency at varying utilization levels, from idle to peak performance. Even an energy-efficient server still consumes about half its full power when doing virtually no work.

4

Energy Proportionality

"The Case for Energy-Proportional Computing," Luiz André Barroso, Urs Hölzle, *IEEE Computer* December 2007



Design for *wide dynamic power range* and *active low power modes*

Energy Efficiency = Utilization/Power

Figure 4. Power usage and energy efficiency in a more energy-proportional server. This server has a power efficiency of more than 80 percent of its peak value for utilizations of 30 percent and above, with efficiency remaining above 50 percent for utilization levels as low as 10 percent.

5

Which statements are NOT true about Warehouse Scale Computing?

- ☐ Servers, IT equipment represent less than half of WSC power budget
- ☐ The Internet supplies the communication for SaaS
- ☐ Power Usage Effectiveness (PUE) also measures efficiency of the individual servers
- ☐ The goal of energy proportionality is energy usage should track equipment utilization

New-School Machine Structures (It's a bit more complicated!)

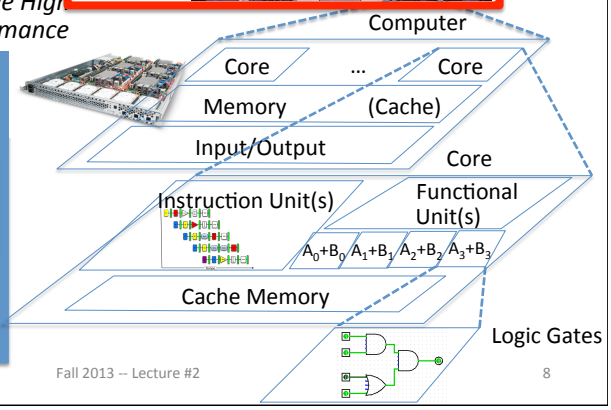
Today's Lecture

Software

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- 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 @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance

Hardware



Agenda

- Request Level Parallelism
- MapReduce Examples
- Administrivia + 61C in the News +
  The secret to getting good grades at Berkeley
- MapReduce Execution
- Costs in Warehouse Scale Computer

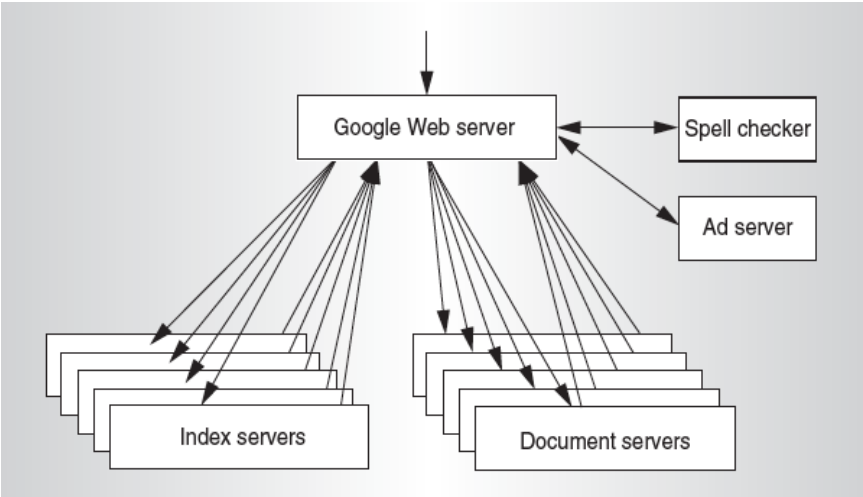
Agenda

- Request Level Parallelism
- MapReduce Examples
- Administrivia + 61C in the News +
  The secret to getting good grades at Berkeley
- MapReduce Execution
- Costs in Warehouse Scale Computer

Request-Level Parallelism (RLP)

- Hundreds or thousands of requests per second
  - Not your laptop or cell-phone, but popular Internet services like Google search
  - Such requests are largely independent
    - Mostly involve read-only databases
    - Little read-write (aka "producer-consumer") sharing
    - Rarely involve read–write data sharing or synchronization across requests
- Computation easily partitioned within a request and across different requests

Google Query-Serving Architecture

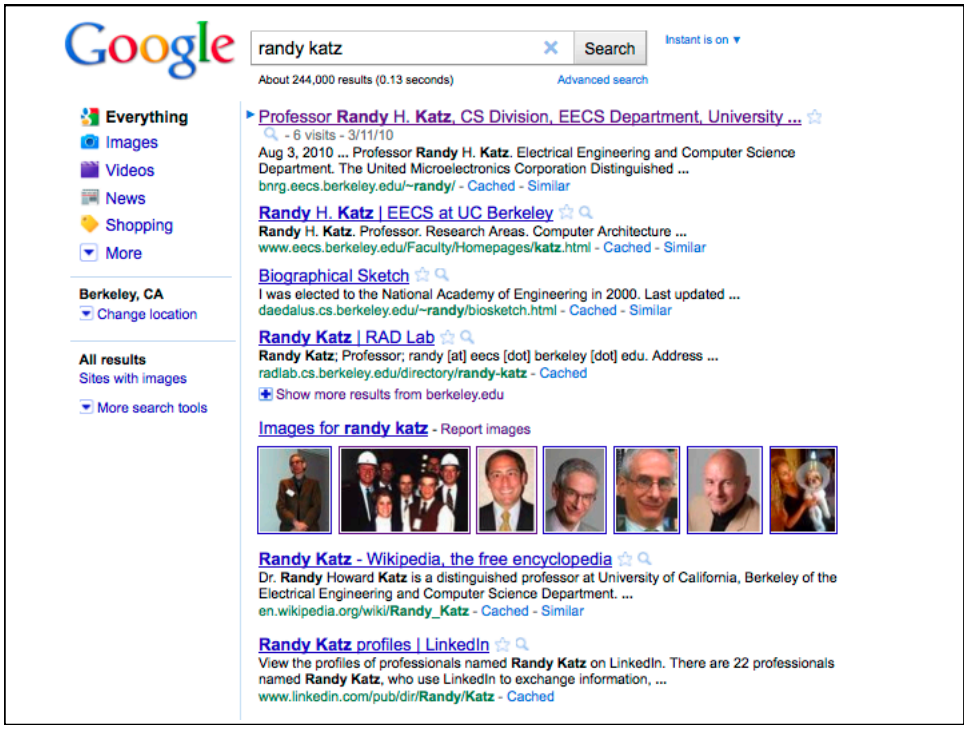


Anatomy of a Web Search

- Google "Randy H. Katz"
  1. Direct request to "closest" Google Warehouse Scale Computer
  2. Front-end load balancer directs request to one of many clusters of servers within WSC
  3. Within cluster, select one of many Google Web Servers (GWS) to handle the request and compose the response pages
  4. GWS communicates with Index Servers to find documents that contain the search words, "Randy", "Katz", uses location of search as well
  5. Return document list with associated relevance score

Anatomy of a Web Search

- In parallel,
  - Ad system: books by Katz at Amazon.com
  - Images of Randy Katz
- 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)



Anatomy of a Web Search

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

Question: Which statements are NOT TRUE about about Request Level Parallelism?

- ☐ RLP runs naturally independent requests in parallel
- ☐ RLP also runs independent tasks within a request
- ☐ RLP typically uses equal number of reads and writes
- ☐ Search uses redundant copies of indices and data to deliver parallelism

Agenda

- Request Level Parallelism
- MapReduce Examples
- Administrivia + 61C in the News +
  The secret to getting good grades at Berkeley
- MapReduce Execution
- Costs in Warehouse Scale Computer

Data-Level Parallelism (DLP)

- Two kinds
  - Lots of data in memory that can be operated on in parallel (e.g., adding together two arrays)
  - Lots of data on many disks that can be operated on in parallel (e.g., searching for documents)
- October 10 lecture and 3rd project does Data Level Parallelism (DLP) in memory
- Today's lecture and 1st project does DLP across 1000s of servers and disks using MapReduce

Problem Trying To Solve

- How process large amounts of raw data (crawled documents, request logs, …) every day to compute derived data (inverted indices, page popularity, …) when computation conceptually simple but input data large and distributed across 100s to 1000s of servers so that finish in reasonable time?
- Challenge: Parallelize computation, distribute data, tolerate faults without obscuring simple computation with complex code to deal with issues
- Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, Jan 2008.

MapReduce Solution

- Apply Map function to user supplied record of key/value pairs
- Compute set of intermediate key/value pairs
- Apply Reduce operation to all values that share same key to combine derived data properly
  - Often produces smaller set of values
  - Typically 0 or 1 output value per Reduce invocation
- User supplies Map and Reduce operations in functional model so can parallelize, re-execute for fault tolerance

Data-Parallel "Divide and Conquer" (MapReduce Processing)

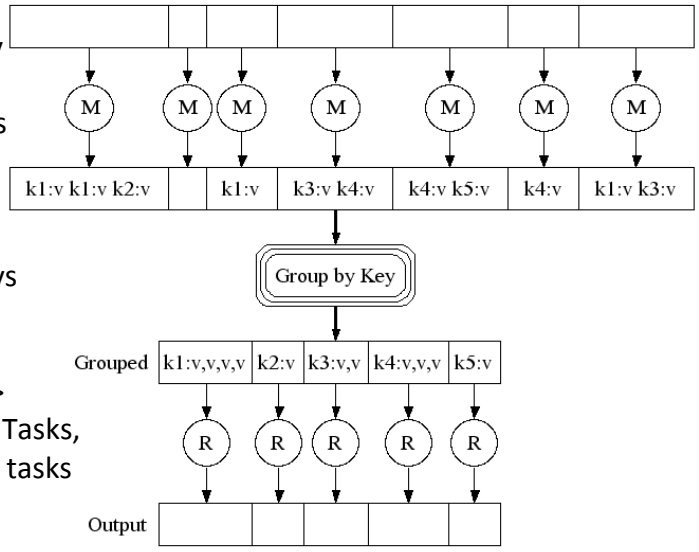
- Map:
  - Slice data into "shards" or "splits", distribute these to workers, compute sub-problem solutions
  - `map(in_key,in_value)->list(out_key,intermediate value)`
    - Processes input key/value pair
    - Produces set of intermediate pairs
- Reduce:
  - Collect and combine sub-problem solutions
  - `reduce(out_key,list(intermediate_value))->list(out_value)`
    - Combines all intermediate values for a particular key
    - Produces a set of merged output values (usually just one)
- Fun to use: focus on problem, let MapReduce library deal with messy details

MapReduce Execution

Fine granularity tasks: many more map tasks than machines

Bucket sort to get same keys together

2000 servers => ≈ 200,000 Map Tasks, ≈ 5,000 Reduce tasks



Google Uses MapReduce For …

- Web crawl: Find outgoing links from HTML documents, aggregate by target document
- Google Search: Generating inverted index files using a compression scheme
- Google Earth: Stitching overlapping satellite images to remove seams and to select high-quality imagery
- Google Maps: Processing all road segments on Earth and render map tile images that display segments
- More than 10,000 MR programs at Google in 4 years, run 100,000 MR jobs per day (2008)

MapReduce Popularity at Google

	Aug-04	Mar-06	Sep-07	Sep-09
Number of MapReduce jobs	29,000	171,000	2,217,000	3,467,000
Average completion time (secs)	634	874	395	475
Server years used	217	2,002	11,081	25,562
Input data read (TB)	3,288	52,254	403,152	544,130
Intermediate data (TB)	758	6,743	34,774	90,120
Output data written (TB)	193	2,970	14,018	57,520
Average number servers / job	157	268	394	488

What if Ran Google Workload on EC2?

	Aug-04	Mar-06	Sep-07	Sep-09
Number of MapReduce jobs	29,000	171,000	2,217,000	3,467,000
Average completion time (secs)	634	874	395	475
Server years used	217	2,002	11,081	25,562
Input data read (TB)	3,288	52,254	403,152	544,130
Intermediate data (TB)	758	6,743	34,774	90,120
Output data written (TB)	193	2,970	14,018	57,520
Average number of servers per job	157	268	394	488
Average Cost/job EC2	$17	$39	$26	$38
Annual Cost if on EC2	$0.5M	$6.7M	$57.4M	$133.1M

Question: Which statements are NOT TRUE about about MapReduce?

- ☐ Users express computation as two functions, Map and Reduce, and supply code for them
- ☐ MapReduce works well for tasks like Search and Matrix Multiply
- ☐ There are typically many more Map Tasks than Reduce Tasks (e.g., 40:1)
- ☐ MapReduce hides details of parallelization, fault tolerance, locality optimization, and load balancing

Agenda

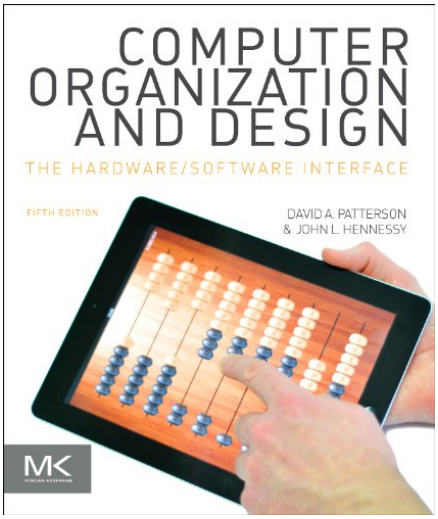
- MapReduce Examples
- Administrivia + 61C in the News +
  The secret to getting good grades at Berkeley
- MapReduce Execution
- Costs in Warehouse Scale Computer

Administrivia

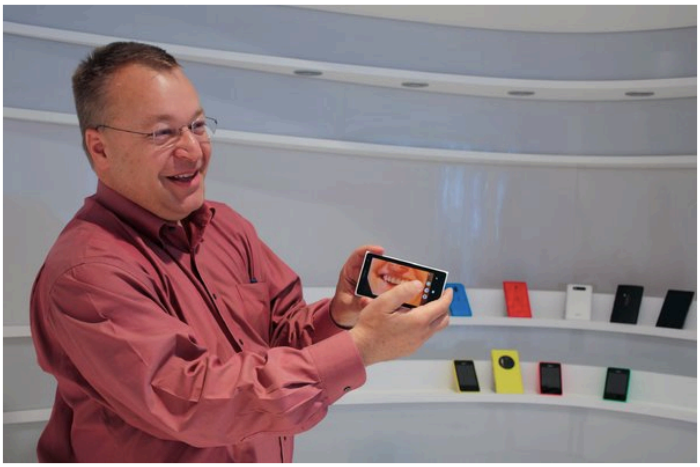
- HW #1, Lab #1 posted
  - HW #1 due Sunday before midnight
  - Labs checked off in lab or in TA office hours *before your next lab*
- Your professor respectfully asks:
  - Leave lecture early? *Sit near the aisles please*
  - Use computer in class? *Sit at back of auditorium …*

The Secret to Getting Good Grades

- It's easy!
- Do assigned reading the night before the lecture, to get more value from lecture



Microsoft Acquires Nokia Units, and Leader



Ritsuko Ando/Reuters

Stephen Elop, the chief executive of Nokia, will rejoin Microsoft, setting him up as a potential successor for Steven A. Ballmer.

By NICK WINGFIELD
Published: September 3, 2013

SEATTLE — Microsoft said it had reached an agreement to acquire the handset and services business of Nokia for about $7.2 billion, in an audacious effort to transform Microsoft's business for a mobile era that has largely passed it by.

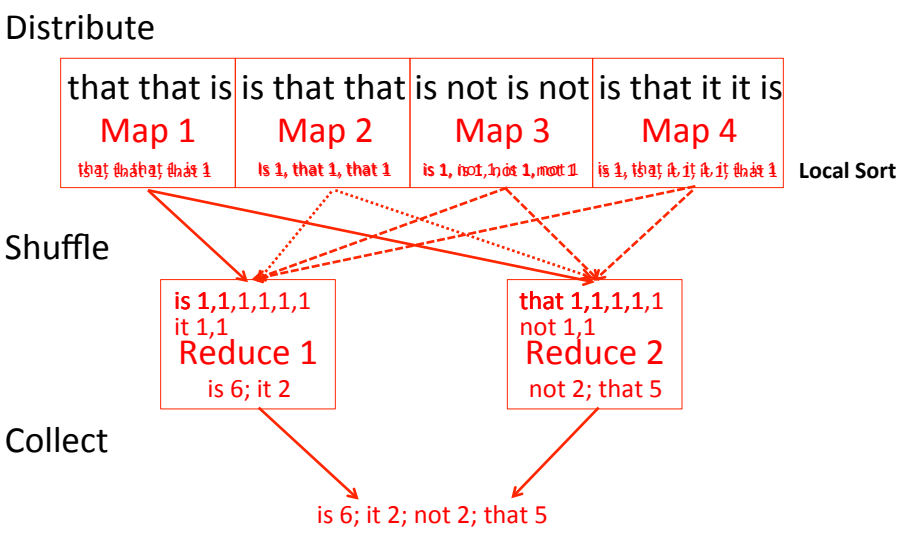
MapReduce Processing Example: Count Word Occurrences

- 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

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(AsString(result));
```


Another Example: Word Index (How Often Does a Word Appear?)



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(interm_key,list(interm_val))
:
    // DO WORK (usually like reducer)
    emit(interm_key2, interm_val2)
```


Our Final Execution Sequence

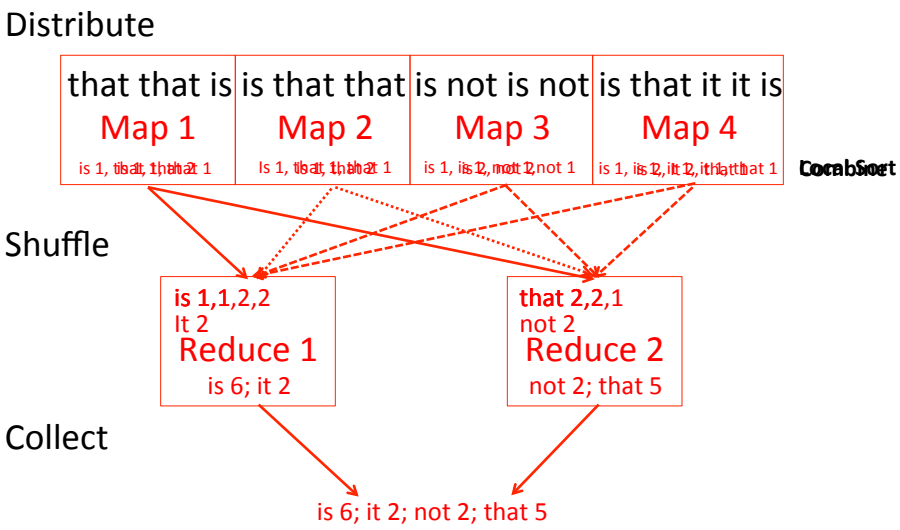
- Map – Apply operations to all input key, val
- Combine – Apply reducer operation, but distributed across map tasks
- Reduce – Combine all values of a key to produce desired output

MapReduce Processing Example: Count Word Occurrences

- 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);
```


Another Example: Word Index (How Often Does a Word Appear?)



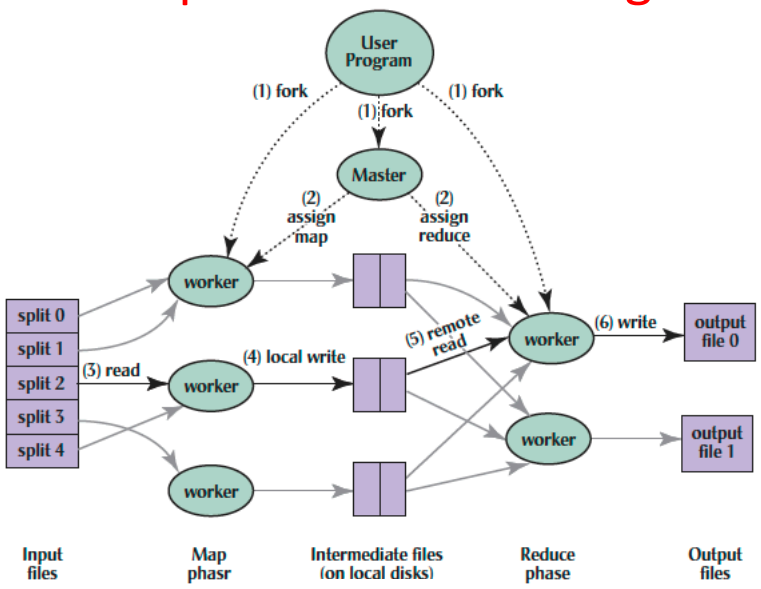
Types

- map (k1,v1) ➔ list(k2,v2)
- reduce (k2,list(v2)) ➔ list(v2)
- Input keys and values from *different* domain than output keys and values
- Intermediate keys and values from *same* domain as output keys and values

Execution Setup

- Map invocations distributed by partitioning input data into M *splits*
  - Typically 16 MB to 64 MB per piece
- Input processed in parallel on different servers
- Reduce invocations distributed by partitioning intermediate key space into R pieces
  - E.g., hash(key) mod R
- User picks M >> # servers, R > # servers
  - Big M helps with load balancing, recovery from failure
  - One output file per R invocation, so not too many

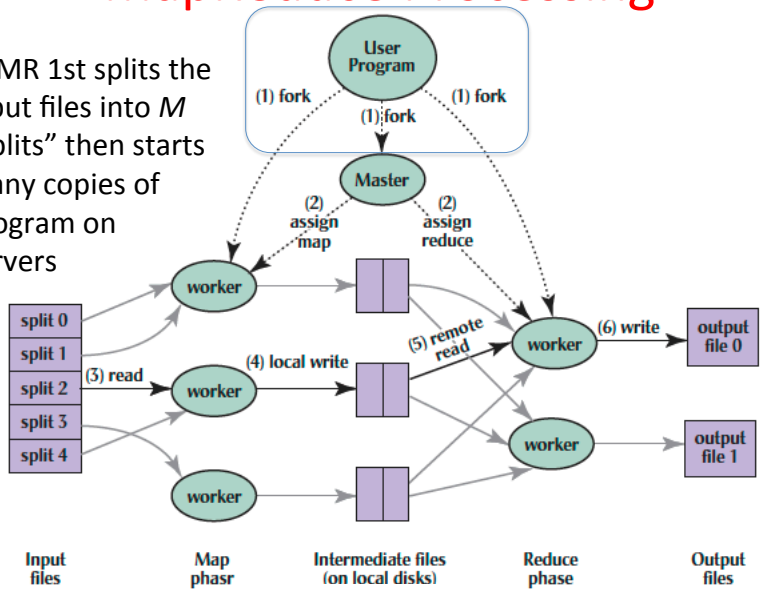
MapReduce Processing



Shuffle phase

MapReduce Processing

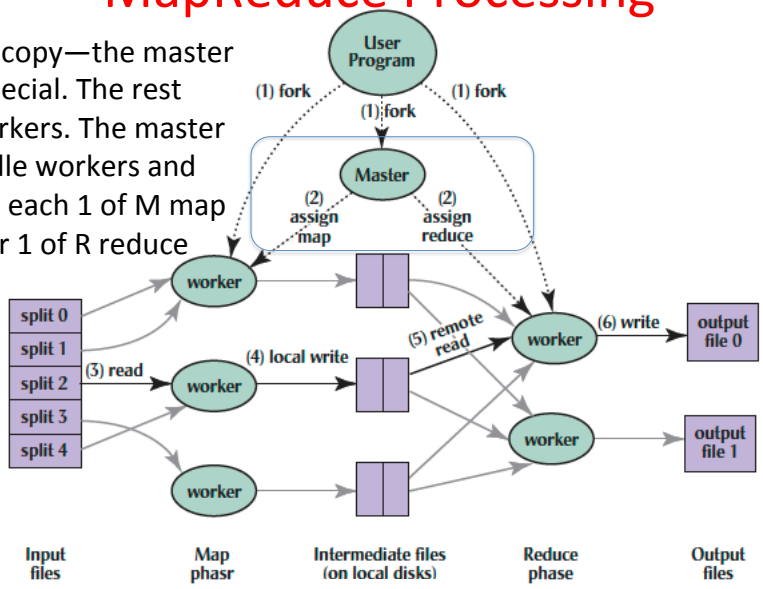
1. MR 1st splits the input files into *M* "splits" then starts many copies of program on servers



Shuffle phase

MapReduce Processing

2. One copy—the master — is special. The rest are workers. The master picks idle workers and assigns each 1 of M map tasks or 1 of R reduce tasks.

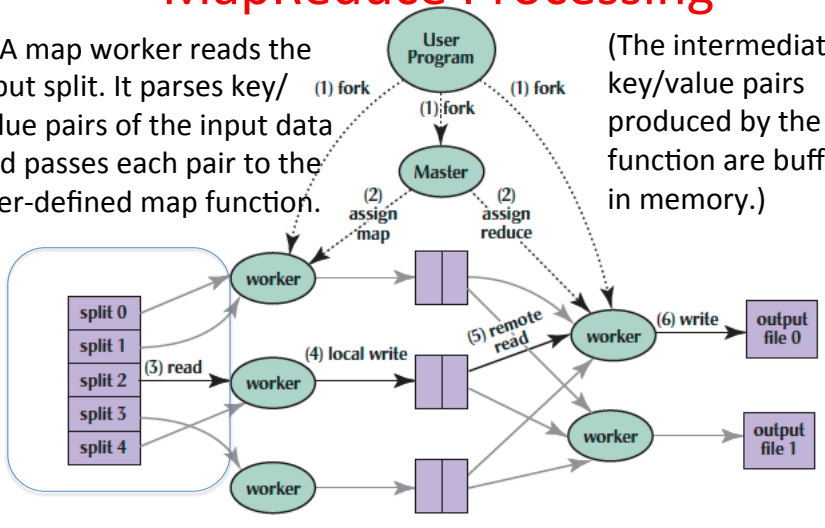


Shuffle phase

MapReduce Processing

3. A map worker reads the input split. It parses key/ value pairs of the input data and passes each pair to the user-defined map function.

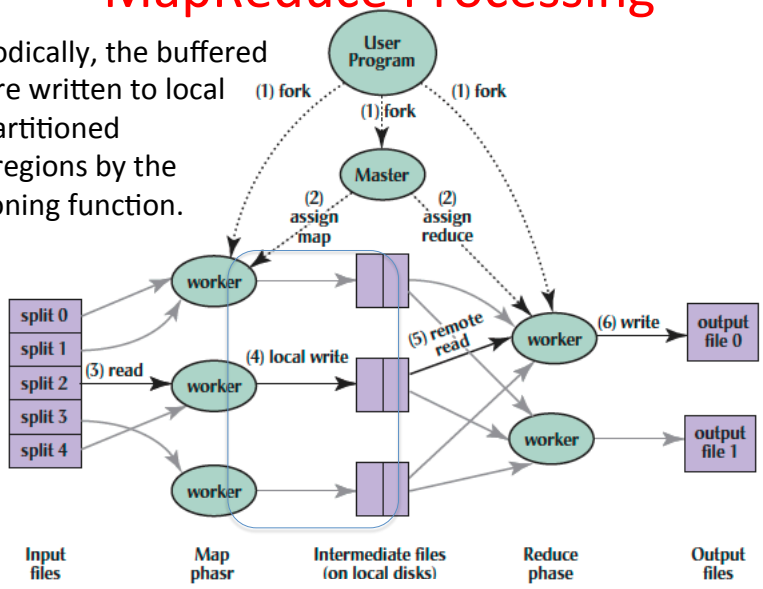
(The intermediate key/value pairs produced by the map function are buffered in memory.)



Shuffle phase

MapReduce Processing

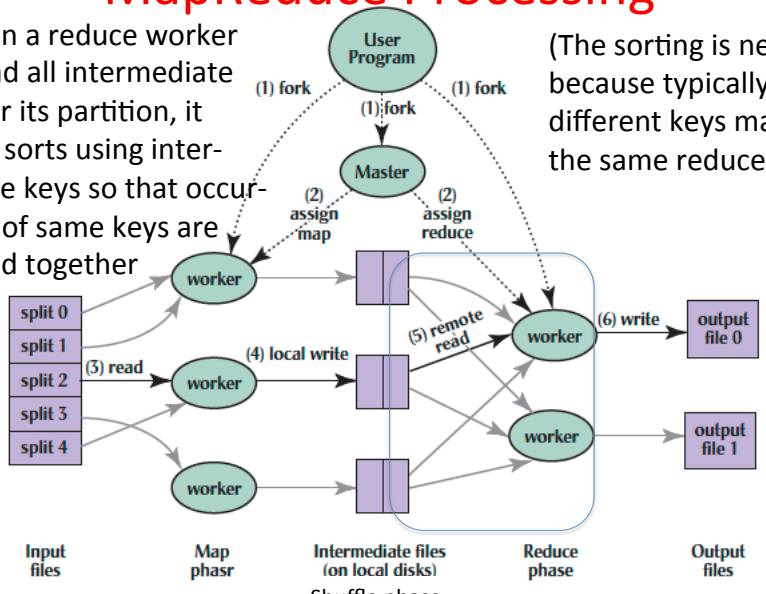
4. Periodically, the buffered pairs are written to local disk, partitioned into *R* regions by the partitioning function.



MapReduce Processing

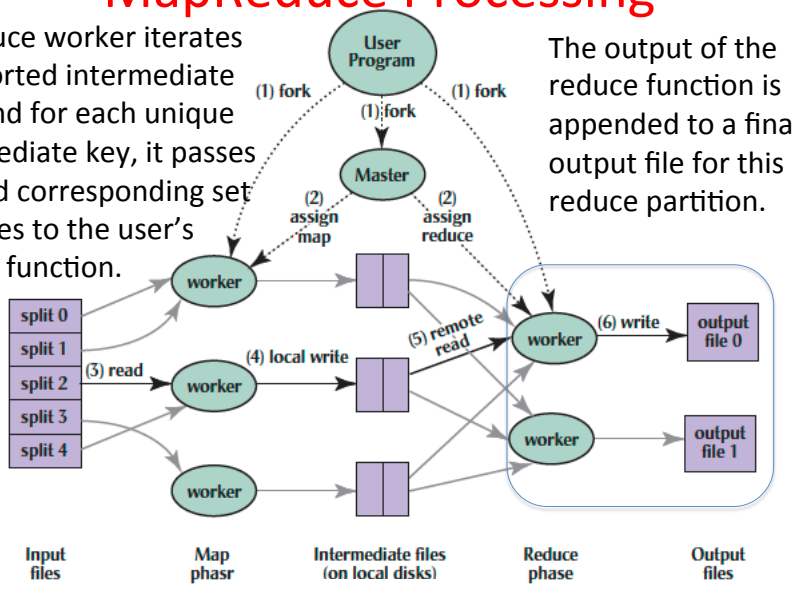
5. When a reduce worker has read all intermediate data for its partition, it bucket sorts using inter-mediate keys so that occur-rences of same keys are grouped together

(The sorting is needed because typically many different keys map to the same reduce task)



MapReduce Processing

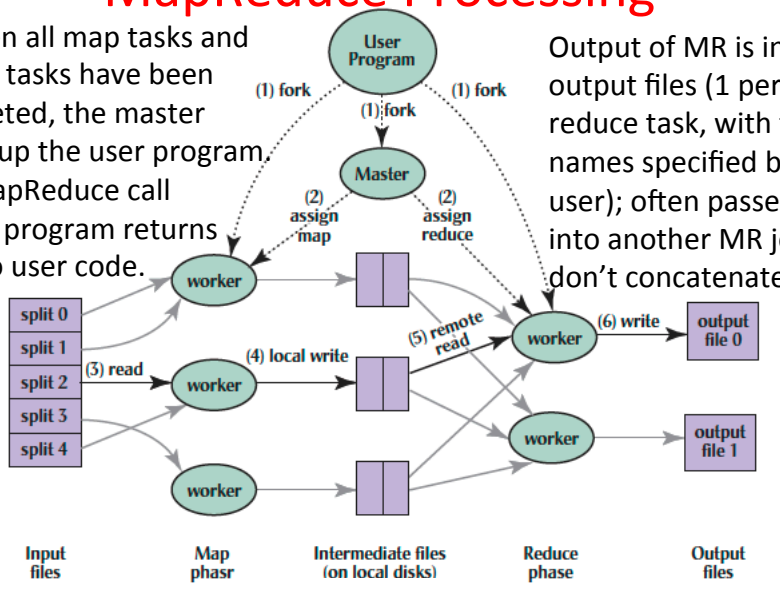
6. Reduce worker iterates over sorted intermediate data and for each unique intermediate key, it passes key and corresponding set of values to the user's reduce function.



The output of the reduce function is appended to a final output file for this reduce partition.

MapReduce Processing

7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. The MapReduce call in user program returns back to user code.



Output of MR is in *R* output files (1 per reduce task, with file names specified by user); often passed into another MR job so don't concatenate

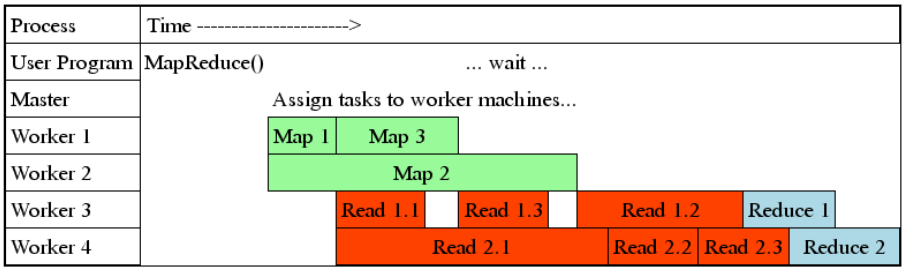
Master Data Structures

- For each map task and reduce task
  - State: idle, in-progress, or completed
  - Identify 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

Agenda

- MapReduce Examples
- Administrivia + 61C in the News + The secret to getting good grades at Berkeley
- MapReduce Execution
- Costs in Warehouse Scale Computer

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"

Show MapReduce Job Running

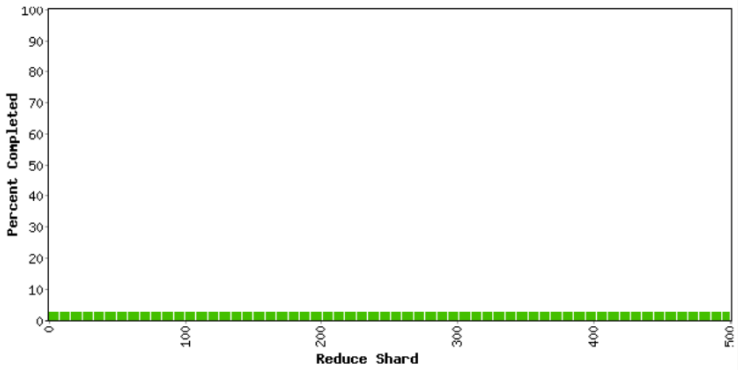
- ~41 minutes total
  - ~29 minutes for Map tasks & Shuffle tasks
  - ~12 minutes for Reduce tasks
  - 1707 worker servers used
- Map (Green) tasks read 0.8 TB, write 0.5 TB
- Shuffle (Red) tasks read 0.5 TB, write 0.5 TB
- Reduce (Blue) tasks read 0.5 TB, write 0.5 TB

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

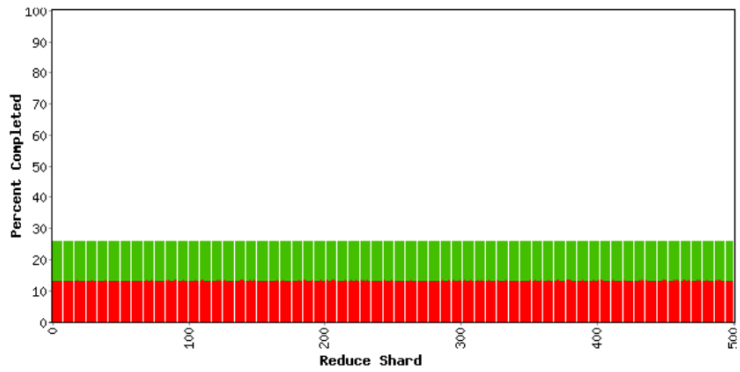
Variable	
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

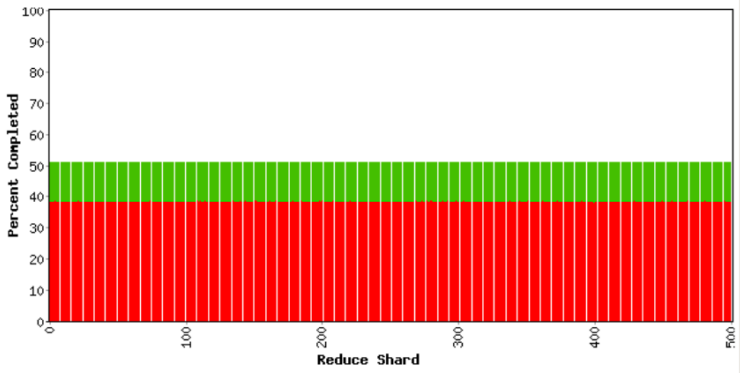
Variable	
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

Variable	
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0

Counters

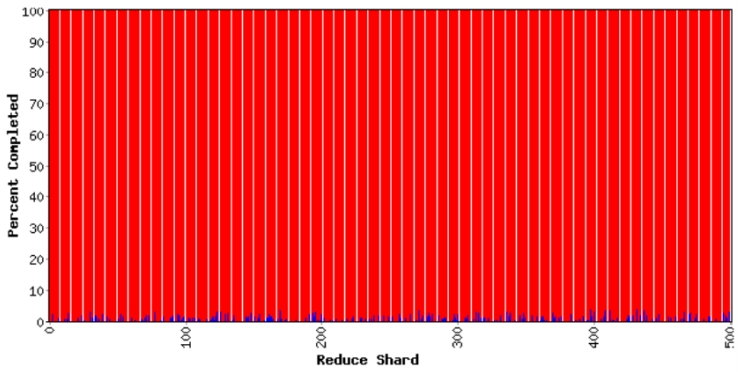
Variable	
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

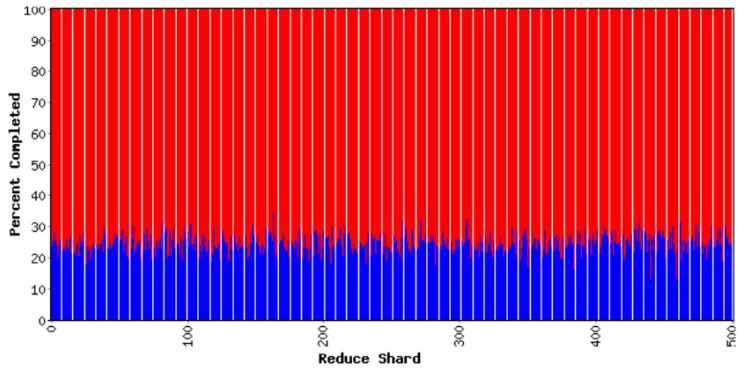
Variable		
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	10
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

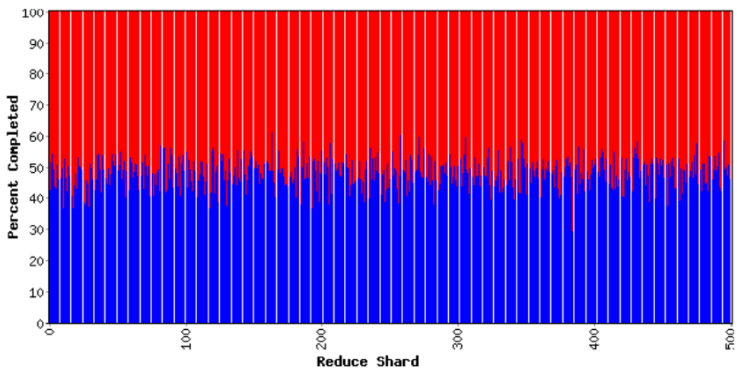
Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.1	
Output (MB/s)	1238.8	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51738599	
mr-merge-outputs	51738599	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

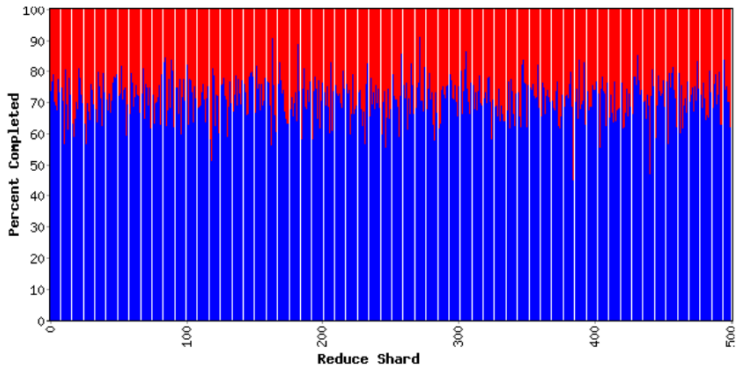
Variable	
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1225.1
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51842100
mr-merge-outputs	51842100

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

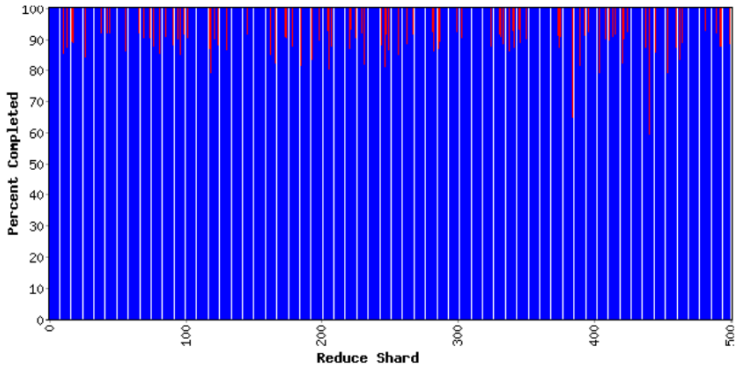
Variable	
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1222.0
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51640600
mr-merge-outputs	51640600

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

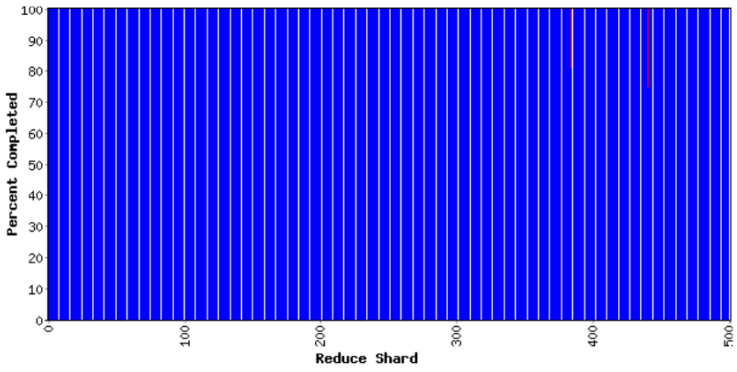
Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	849.5	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	35083350	
mr-merge-outputs	35083350	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

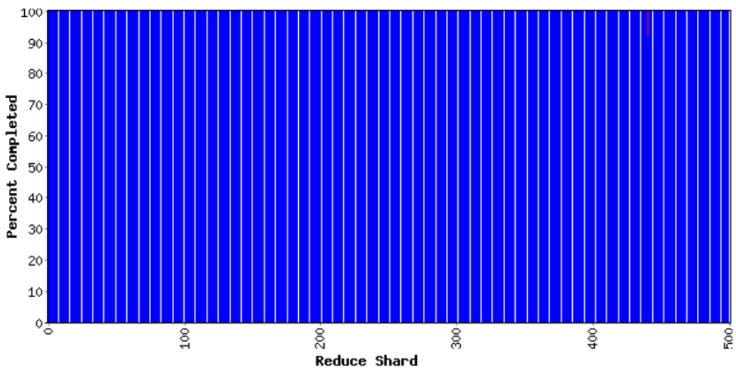
Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	105(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	394792	
mr-merge-outputs	394792	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

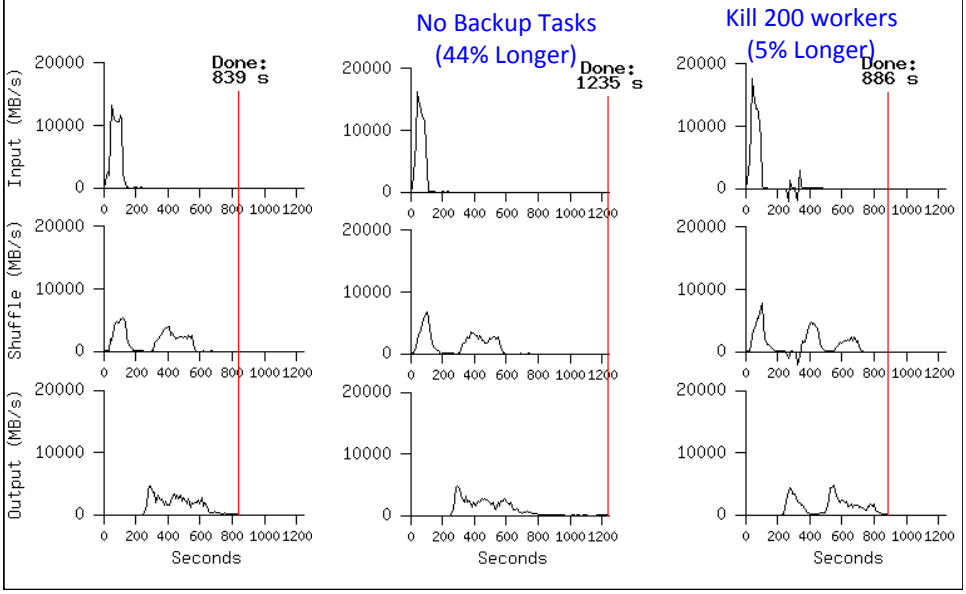
MapReduce Failure Handling

- 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:
  - Could handle, but don't yet (master failure unlikely)
- Robust: lost 1600 of 1800 machines once, but finished fine

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

Impact on Execution of Restart, Failure for 10B record Sort using 1800 servers



MapReduce Locality Optimization during Scheduling

- Master scheduling policy:
  - Asks GFS (Google File System) for locations of replicas of input file blocks
  - Map tasks typically split into 64MB (== GFS block size)
  - Map tasks scheduled so GFS input block replica are on same machine or same rack
- Effect: Thousands of machines read input at local disk speed
- Without this, rack switches limit read rate

Question: Which statements are NOT TRUE about about MapReduce?

- ☐ MapReduce divides computers into 1 master and N-1 workers; masters assigns MR tasks
- ☐ Towards the end, the master assigns uncompleted tasks again; 1st to finish wins
- ☐ Reducers can start reducing as soon as they start to receive Map data
- ☐ Reduce worker sorts by intermediate keys to group all occurrences of same key

69

Agenda

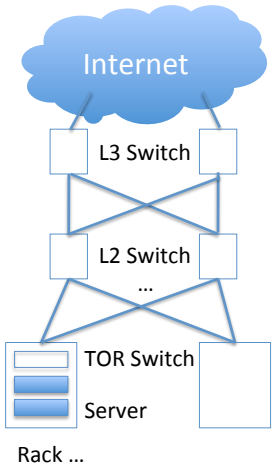
- MapReduce Examples
- Administrivia + 61C in the News + The secret to getting good grades at Berkeley
- MapReduce Execution
- Costs in Warehouse Scale Computer

Design Goals of a WSC

- Unique to Warehouse-scale
  - *Ample parallelism*:
    - Batch apps: large number independent data sets with independent processing. Also known as *Data-Level Parallelism*
  - *Scale and its Opportunities/Problems*
    - Relatively small number of these make design cost expensive and difficult to amortize
    - But price breaks are possible from purchases of very large numbers of commodity servers
    - Must also prepare for high component failures
  - *Operational Costs Count*:
    - Cost of equipment purchases << cost of ownership

WSC Case Study
Server Provisioning

WSC Power Capacity		8.00 MW
Power Usage Effectiveness (PUE)	1.45	
IT Equipment Power Share	0.67	5.36 MW
Power/Cooling Infrastructure	0.33	2.64 MW
IT Equipment Measured Peak (W)	145.00	
Assume Average Pwr @ 0.8 Peak	116.00	
# of Servers		46207
# of Servers		46000
# of Servers per Rack	40.00	
# of Racks		1150
Top of Rack Switches		1150
# of TOR Switch per L2 Switch	16.00	
# of L2 Switches		72
# of L2 Switches per L3 Switch	24.00	
# of L3 Switches		3



Cost of WSC

- US account practice separates purchase price and operational costs
- Capital Expenditure (CAPEX) is cost to buy equipment (e.g.. buy servers)
- Operational Expenditure (OPEX) is cost to run equipment (e.g., pay for electricity used)

WSC Case Study
Capital Expenditure (Capex)

- Facility cost and total IT cost look about the same

Facility Cost	$88,000,000
Total Server Cost	$66,700,000
Total Network Cost	$12,810,000
Total Cost	$167,510,000

- However, replace servers every 3 years, networking gear every 4 years, and facility every 10 years

Cost of WSC

- US account practice allow converting Capital Expenditure (CAPEX) into Operational Expenditure (OPEX) by amortizing costs over time period
  - Servers 3 years
  - Networking gear 4 years
  - Facility 10 years

WSC Case Study
Operational Expense (Opex)

		Years Amortization		Monthly Cost	
Amortized Capital Expense	Server	3	$66,700,000	$2,000,000	55%
	Network	4	$12,530,000	$295,000	8%
	Facility		$88,000,000		
	Pwr&Cooling	10	$72,160,000	$625,000	17%
	Other	10	$15,840,000	$140,000	4%
Operational Expense	Amortized Cost			$3,060,000	
	Power (8MW)		$0.07	$475,000	13%
	People (3)			$85,000	2%
	Total Monthly			$3,620,000	100%

$/kWh

- Monthly Power costs
  - $475k for electricity
  - $625k + $140k to amortize facility power distribution and cooling
  - 60% is amortized power distribution and cooling

How much does a watt cost in a WSC?

- 8 MW facility
- Amortized facility, including power distribution and cooling is $625k + $140k = $765k
- Monthly Power Usage = $475k
- Watt-Year = ($765k+$475k)*12/8M = $1.86 or about $2 per year
- To save a watt, if spend more than $2 a year, lose money

Which statement is TRUE about Warehouse Scale Computer economics?

- ☐ The dominant operational monthly cost is server replacement.
- ☐ The dominant operational monthly cost is the electric bill.
- ☐ The dominant operational monthly cost is facility replacement.
- ☐ The dominant operational monthly cost is operator salaries.

79

WSC Case Study
Operational Expense (Opex)

		Years Amortization		Monthly Cost	
Amortized Capital Expense	Server	3	$66,700,000	$2,000,000	55%
	Network	4	$12,530,000	$295,000	8%
	Facility		$88,000,000		
	Pwr&Cooling	10	$72,160,000	$625,000	17%
	Other	10	$15,840,000	$140,000	4%
Operational Expense	Amortized Cost			$3,060,000	
	Power (8MW)		$0.07	$475,000	13%
	People (3)			$85,000	2%
	Total Monthly			$3,620,000	100%

$/kWh

- $3.8M/46000 servers = ~$80 per month per server in revenue to break even
- ~$80/720 hours per month = $0.11 per hour
- So how does Amazon EC2 make money???

January 2013 AWS Instances & Prices

Instance	Per Hour	Ratio to Small	Compute Units	Virtual Cores	Compute Unit/ Core	Memory (GB)	Disk (GB)	Address
Standard Small	$0.085	1.0	1.0	1	1.00	1.7	160	32 bit
Standard Large	$0.340	4.0	4.0	2	2.00	7.5	850	64 bit
Standard Extra Large	$0.680	8.0	8.0	4	2.00	15.0	1690	64 bit
High-Memory Extra Large	$0.500	5.9	6.5	2	3.25	17.1	420	64 bit
High-Memory Double Extra Large	$1.200	14.1	13.0	4	3.25	34.2	850	64 bit
High-Memory Quadruple Extra Large	$2.400	28.2	26.0	8	3.25	68.4	1690	64 bit
High-CPU Medium	$0.170	2.0	5.0	2	2.50	1.7	350	32 bit
High-CPU Extra Large	$0.680	8.0	20.0	8	2.50	7.0	1690	64 bit
Cluster Quadruple Extra Large	$1.300	15.3	33.5	16	2.09	23.0	1690	64 bit

- Closest computer in WSC example is Standard Extra Large
- @$0.11/hr, Amazon EC2 can make money!
  - even if used only 50% of time

August 2013 AWS Instances & Prices

Instance	Per Hour	Ratio to Small	Compute Units	Virtual Cores	Compute Unit/ Core	Memory (GB)	Disk (GB)	Address
Standard Small	$0.065	1.0	1.0	1	1.00	1.7	160	32 bit
Standard Large	$0.260	4.0	4.0	2	2.00	7.5	840	64 bit
Standard Extra Large	$0.520	8.0	8.0	4	2.00	15.0	1680	64 bit
High-Memory Extra Large	$0.460	7.1	6.5	2	3.25	17.1	420	64 bit
High-Memory Double Extra Large	$0.920	14.2	13.0	4	3.25	34.2	850	64 bit
High-Memory Quadruple Extra Large	$1.840	28.3	26.0	8	3.25	68.4	1690	64 bit
High-CPU Medium	$0.165	2.5	5.0	2	2.50	1.7	350	32 bit
High-CPU Extra Large	$0.660	10.2	20.0	8	2.50	7.0	1690	64 bit
XXXXXXXXXXXXXX	$X	15.3	33.5	16	2.09	23.0	1690	64 bit

- Closest computer in WSC example is Standard Extra Large
- @$0.11/hr, Amazon EC2 can make money!
  - even if used only 50% of time
  - See http://aws.amazon.com/ec2/pricing and http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html

And in Conclusion, …

- Request-Level Parallelism
  - High request volume, each largely independent of other
  - Use replication for better request throughput, availability
- MapReduce Data Parallelism
  - Map: Divide large data set into pieces for independent parallel processing
  - Reduce: Combine and process intermediate results to obtain final result
- WSC CapEx vs. OpEx
  - Economies of scale mean WSC can sell computing as a utility
  - Servers dominate cost
  - Spend more on power distribution and cooling infrastructure than on monthly electricity costs