Internet Measurement and Data Analysis (14)

Kenjiro Cho

2012-01-11

review of previous class

Class 13 Data mining

- pattern extraction
- classification
- clustering
- exercise: clustering

today's topics

Class 14 Scalable measurement and analysis

- distributed parallel processing
- cloud technology

measurement, data analysis and scalability

measurement methods

- network bandwidth, data volume, processing power on measurement machines

data collection

- collecting data from multiple sources
- network bandwidth, data volume, processing power on collecting machines

data analysis

- analysis of huge data sets
- repetition of relatively simple jobs
- complex data processing by data mining methods
- data volume, processing power of analyzing machines
  - communication power for distributed processing

computational complexity

metrics for the efficiency of an algorithm

- time complexity
- space complexity

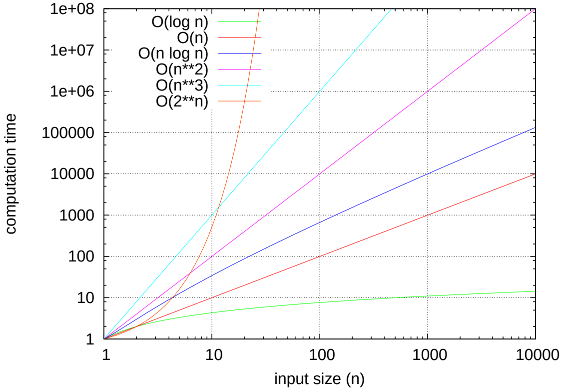
- average-case complexity
- worst-case complexity

big O notation

- describe algorithms simply by the growth order of execution time as input size n increases
  - example: $O(n), O(n^2), O(n \log n)$
- more precisely, "$f(n)$ is order $g(n)$" means:
  for function $f(n)$ and function $g(n)$, $f(n) = O(g(n)) \Leftrightarrow$ there exist constants C and n_0 such that $|f(n)| \leq C|g(n)| \, (\forall n \geq n_0)$

computational complexity

- logarithmic time
- polynomial time
- exponential time



example of computational complexity

search algorithms

- linear search: $O(n)$
- binary search: $O(\log_2 n)$

sort algorithms

- selection sort: $O(n^2)$
- quick sort: $O(n \log_2 n)$ on average, $O(n^2)$ for worst case

in general,

- linear algorithms (e.g., loop): $O(n)$
- binary trees: $O(\log n)$
- double loops for a variable: $O(n^2)$
- triple loops for a variable: $O(n^3)$
- combination of variables (e.g., shortest path): $O(c^n)$

distributed algorithms

parallel or concurrent algorithms

- split a job and process them by multiple computers
- issues of communication cost and synchronization

distributed algorithms

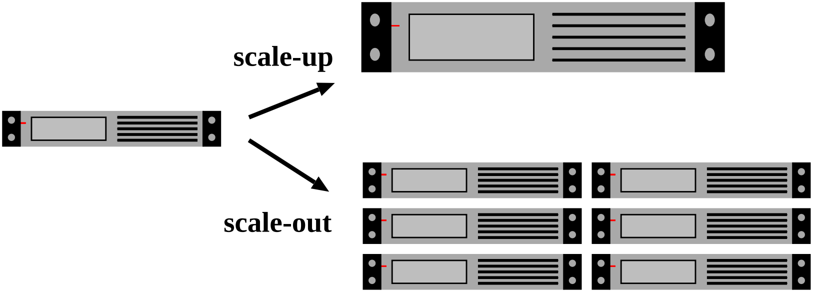
- assume that communications are message passing among independent computers
- failures of computers and message losses

merits

- scalability
    - improvement is only linear at best
- fault tolerance

scale-up and scale-out

- scale-up
  - strengthen or extend a single node
  - without issues of parallel processing
- scale-out
  - extend a system by increasing the number of nodes
  - cost performance, fault-tolerance (use of cheap off-the-shelf computers)



cloud computing

cloud computing: various definitions

- broadly, computer resources behind a wide-area network

background

- market needs:
  - outsourcing IT resources, management and services
    - no initial investment, no need to predict future demands
  - cost reduction as a result
- as well as risk management and energy saving, especially after the Japan Earthquake
- providers: economy of scale, walled garden

various clouds

- public/private/hybrid
  - public cloud: public services over the Internet
  - private cloud: internal services for a single organization
  - personal cloud, cloud federation
- service classification: SaaS/PaaS/IaaS
  - SaaS (Software as a Service)
    - provides applications (e.g., Google Apps, Microsoft Online Services)
  - PaaS (Platform as a Service)
    - provides a platform for applications (e.g., Google App Engine, Microsoft Windows Azure)
  - IaaS (Infrastructure as a Service)
    - provides (hardware) infrastructures such as virtualized servers or shared storage (e.g., Amazon EC2, Amazon S3)

  - IaaS provider - IaaS user (utility computing)
  - IaaS user = SaaS provider - SaaS user (web applications)
  - PaaS: a framework to make SaaS development open for third party
- scale-out cloud/server cloud

key technologies

- virtualization: OS level, I/O level, network level
- utility computing
- energy saving
- data center networking
- management and monitoring technologies
- automatic scaling and load balancing
- large-scale distributed data processing

- related research fields: networking, OS, distributed systems, database, grid computing
  - led by commercial services

MapReduce

MapReduce: a parallel programming model developed by Google

Dean, Jeff and Ghemawat, Sanjay.
MapReduce: Simplified Data Processing on Large Clusters.
OSDI'04. San Francisco, CA. December 2004.
`http://labs.google.com/papers/mapreduce.html`

the slides are taken from the above materials

motivation: large scale data processing

- want to use hundreds or thousands of CPUs for large data processing
- make it easy to use the system without understanding the details of the hardware infrastructures

MapReduce provides

- automatic parallelization and distribution
- fault-tolerance
- I/O scheduling
- status and monitoring

MapReduce programming model

Map/Reduce

- idea from Lisp or other functional programming languages
- generic: for a wide range of applications
- suitable for distributed processing
- able to re-execute after a failure

Map/Reduce in Lisp
(map square '(1 2 3 4)) $\rightarrow$ (1 4 9 16)
(reduce + '(1 4 9 16)) $\rightarrow$ 30

Map/Reduce in MapReduce

map(in_key, in_value) → list(out_key, intermediate_value)

- key/value pairs as input, produce another set of key/value pairs

reduce(out_key, list(intermediate_value)) → list(out_value)

- using the results of map(), produce a set of merged output values for a particular key

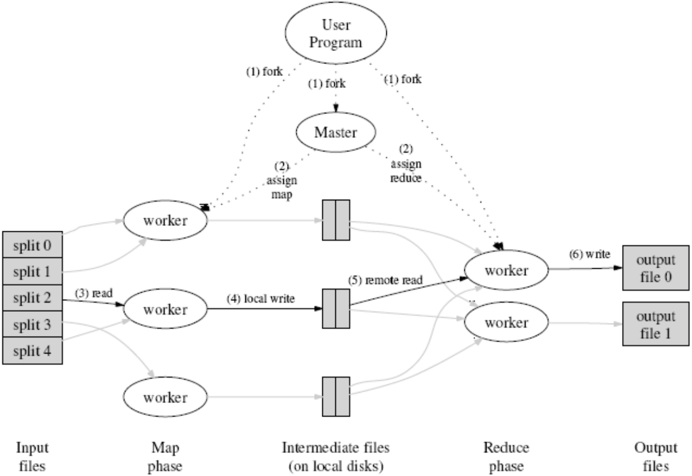
example: count word occurrences

```
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");

reduce(String output_key, Iterator intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
  Emit(AsString(result));
```

other applications

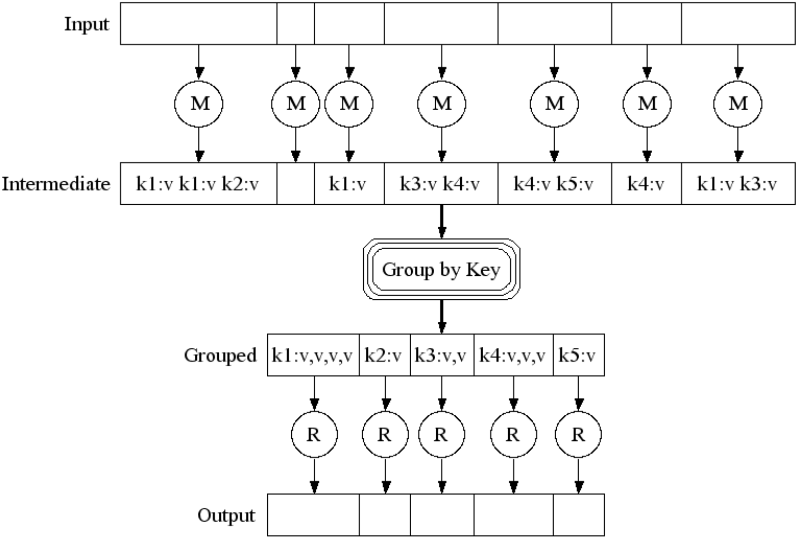
- distributed grep
  - map: output lines matching a supplied pattern
  - reduce: nothing
- count of URL access frequency
  - map: reading web access log, and outputs $< URL, 1 >$
  - reduce: adds together all values for the same URL, and emits $< URL, count >$
- reverse web-link graph
  - map: outputs $< target, source >$ pairs for each link in web pages
  - reduce: concatenates the list of all source URLs associated with a given target URL and emits the pair $< target, list(source) >$
- inverted index
  - map: emits $< word, docID >$ from each document
  - reduce: emits the list of $< word, list(docID) >$

MapReduce Execution Overview



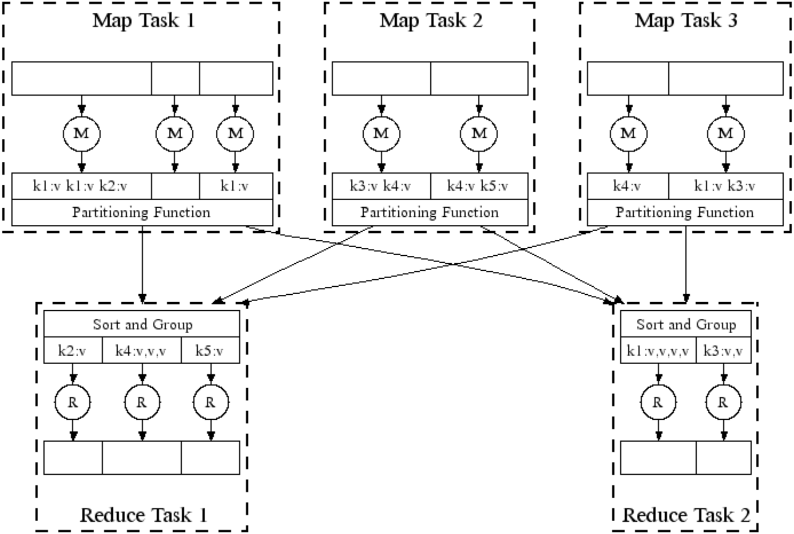
source: MapReduce: Simplified Data Processing on Large Clusters

MapReduce Execution



source: MapReduce: Simplified Data Processing on Large Clusters

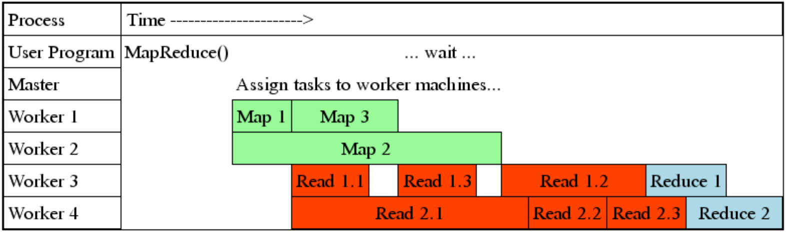
MapReduce Parallel Execution



source: MapReduce: Simplified Data Processing on Large Clusters

Task Granularity and Pipelining

- tasks are fine-grained: the number of Map tasks >> number of machines
  - minimizes time for fault recovery
  - can pipeline shuffling with map execution
  - better dynamic load balancing
- often use 2,000 map/5,000 reduce tasks w/ 2,000 machines



source: MapReduce: Simplified Data Processing on Large Clusters

fault tolerance: handled via re-execution

on worker failure

- detect failure via periodic heartbeats
- re-execute completed and in-progress map tasks
  - need to re-execute completed tasks as results are stored on local disks
- re-execute in progress reduce tasks
- task completion committed through master

robust: lost 1600 of 1800 machines once, but finished fine

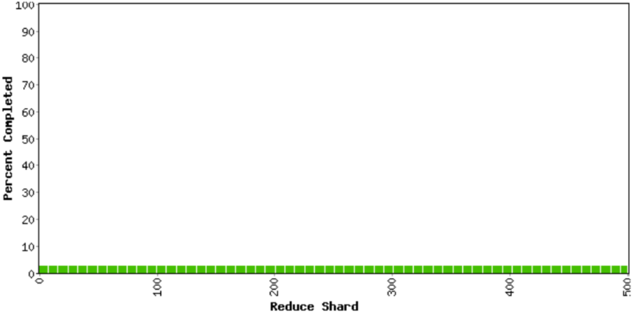
MapReduce status

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	Minute
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

source: MapReduce: Simplified Data Processing on Large Clusters

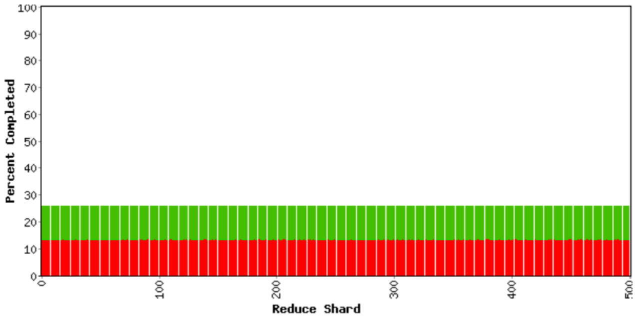
MapReduce status

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	Minute
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

source: MapReduce: Simplified Data Processing on Large Clusters

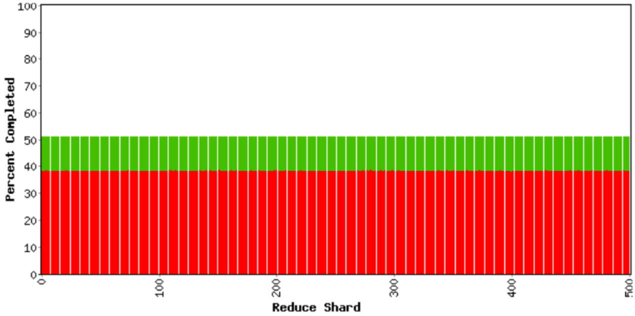
MapReduce status

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	Minute
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

source: MapReduce: Simplified Data Processing on Large Clusters

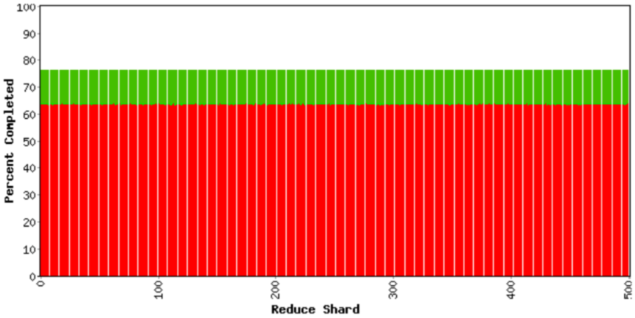
MapReduce status

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	Minute
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

source: MapReduce: Simplified Data Processing on Large Clusters

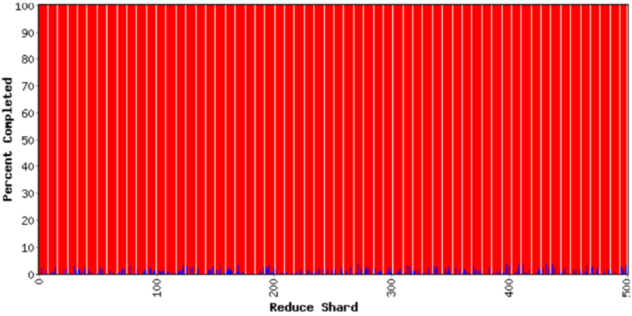
MapReduce status

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	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	105
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

source: MapReduce: Simplified Data Processing on Large Clusters

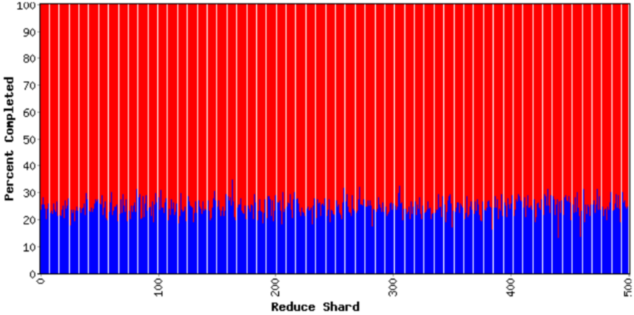
MapReduce status

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

source: MapReduce: Simplified Data Processing on Large Clusters

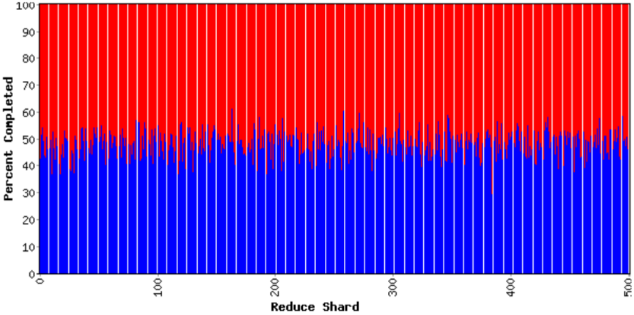
MapReduce status

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	Minute
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

source: MapReduce: Simplified Data Processing on Large Clusters

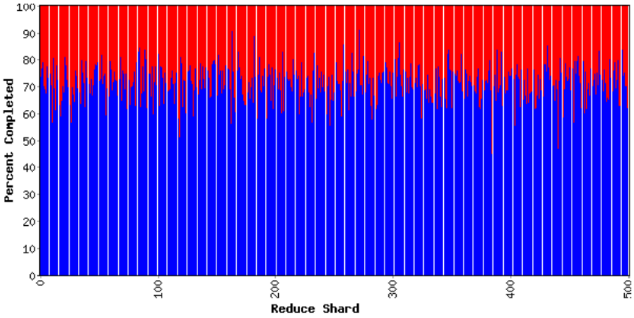
MapReduce status

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	Minute
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

source: MapReduce: Simplified Data Processing on Large Clusters

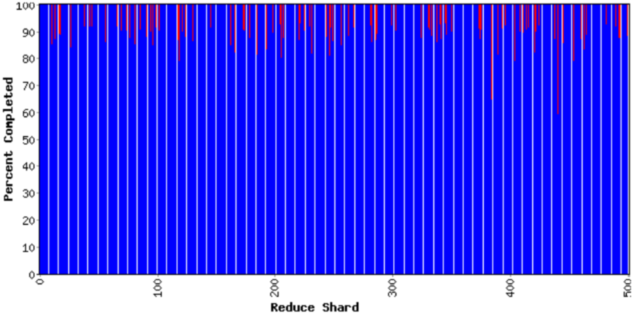
MapReduce status

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	Minute	
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	

source: MapReduce: Simplified Data Processing on Large Clusters

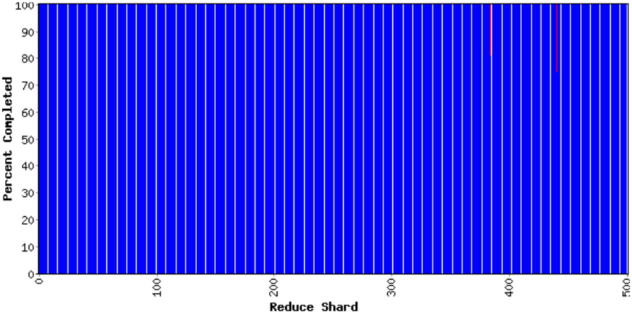
MapReduce status

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	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	105
docs-indexed	0	3
dups-in-index-merge	0	
mr-merge-calls	394792	3
mr-merge-outputs	394792	3

source: MapReduce: Simplified Data Processing on Large Clusters

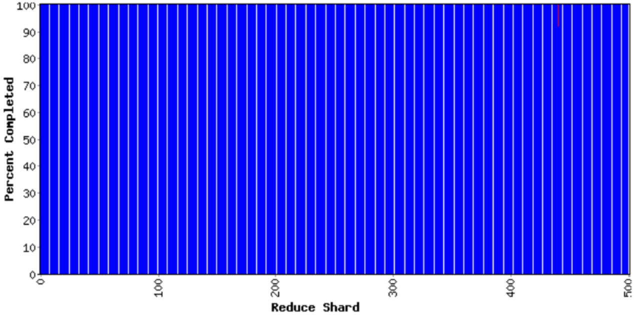
MapReduce status

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	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	1
dups-in-index-merge	0	
mr-merge-calls	73442	1
mr-merge-outputs	73442	1

source: MapReduce: Simplified Data Processing on Large Clusters

refinement: 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: drastically shortens completion time

refinement: locality optimization

master scheduling policy

- asks GFS for locations of replicas of input file blocks
- map tasks typically split into 64MB (== GFS block size)
- map tasks scheduled so GFS input block replicas are on same machine or same rack

effect: thousands of machines read input at local disk speed

- without this, rack switches limit read rate

refinement: skipping bad records

Map/Reduce functions sometimes fail for particular inputs

- best solution is to debug and fix, but not always possible
- on Segmentation Fault
  - send UDP packet to master from signal handler
  - include sequence number of record being processed
- if master sees two failures for same record,
  - next worker is told to skip the record

effect: can work around bugs in third party libraries

other refinement

- sorting guarantees within each reduce partition
- compression of intermediate data
- Combiner: useful for saving network bandwidth
- local execution for debugging/testing
- user-defined counters

performance

test run on cluster of 1800 machines

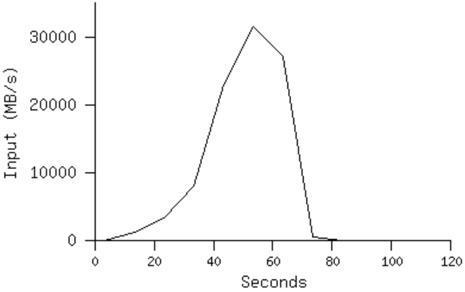
- 4GB of memory
- Dual-processor 2GHz Xeons with Hyperthreading
- Dual 160GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100Gbps

2 benchmarks:

- MR_Grep: scan 10^{10} 100-byte records to extract records matching a rare pattern (92K matching records)
- MR_Sort: sort 10^{10} 100-byte records (modeled after TeraSort benchmark)

MR_Grep

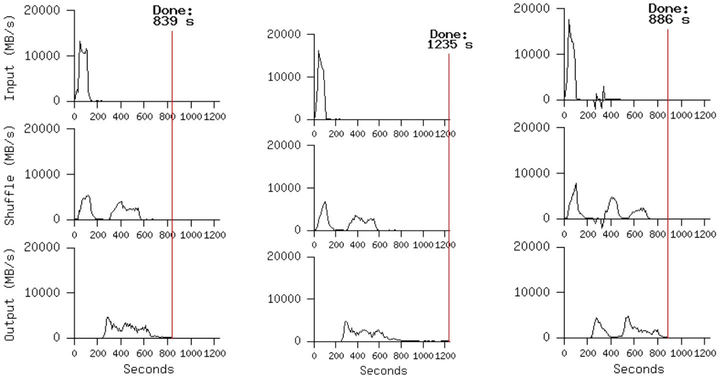
- locality optimization helps
  - 1800 machines read 1TB of data at peak of 31GB/s
  - without this, rack switches would limit to 10GB/s
- startup overhead is significant for short jobs



source: MapReduce: Simplified Data Processing on Large Clusters

MR_Sort

- backup tasks reduce job completion time significantly
- system deals well with failures



Normal(left) No backup tasks(middle) 200 processes killed(right)

source: MapReduce: Simplified Data Processing on Large Clusters

MapReduce summary

- MapReduce: abstract model for distributed parallel processing
- considerably simplify large-scale data processing
- easy to use, fun!
  - the system takes care of details of parallel processing
  - programmers can concentrate on solving a problem
- various applications inside Google including search index creation

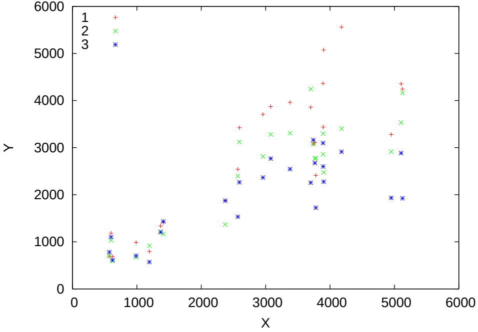
additional note

- Google does not publish the implementation of MapReduce
- Hadoop: open source MapReduce implementation by Apache Project

previous exercise: k-means clustering

- data: hourly traffic for Monday vs. Wednesday/Friday/Sunday

```
% cat km-1.txt km-2.txt km-3.txt | ruby k-means.rb | \
  sort -k3,3 -s -n > km-results.txt
```



k-means code (1/2)

```
k = 3 # k clusters
re = /^(\d+)\s+(\d+)/
INFINITY = 0x7fffffff

# read data
nodes = Array.new # array of array for data points: [x, y, cluster_index]
centroids = Array.new # array of array for centroids: [x, y]
ARGF.each_line do |line|
  if re.match(line)
    c = rand(k) # randomly assign initial cluster
    nodes.push [$1.to_i, $2.to_i, c]
  end
end

round = 0
begin
  updated = false

  # assignment step: assign each node to the closest centroid
  if round != 0  # skip assignment for the 1st round
    nodes.each do |node|
      dist2 = INFINITY # square of dsistance to the closest centroid
      cluster = 0 # closest cluster index
      for i in (0 .. k - 1)
        d2 = (node[0] - centroids[i][0])**2 + (node[1] - centroids[i][1])**2
        if d2 < dist2
          dist2 = d2
          cluster = i
        end
      end
      node[2] = cluster
    end
  end
```

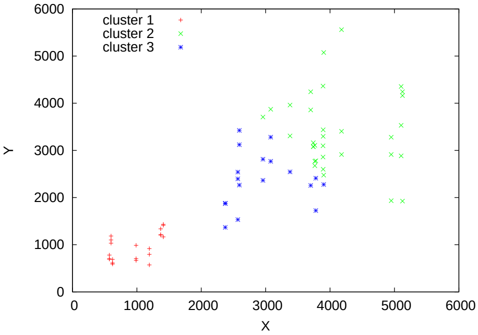
k-means code (2/2)

```
  # update step: compute new centroids
  sums = Array.new(k)
  clsize = Array.new(k)
  for i in (0 .. k - 1)
    sums[i] = [0, 0]
    clsize[i] = 0
  end
  nodes.each do |node|
    i = node[2]
    sums[i][0] += node[0]
    sums[i][1] += node[1]
    clsize[i] += 1
  end

  for i in (0 .. k - 1)
    newcenter = [Float(sums[i][0]) / clsize[i], Float(sums[i][1]) / clsize[i]]
    if round == 0 || newcenter[0] != centroids[i][0] || newcenter[1] != centroids[i][1]
      centroids[i] = newcenter
      updated = true
    end
  end

  round += 1

end while updated == true

# print the results
nodes.each do |node|
  puts "#{node[0]}\t#{node[1]}\t#{node[2]}"
end
```

k-means clustering results

- different results with different initial values

```
set key left
set xrange [0:6000]
set yrange [0:6000]
set xlabel "X"
set ylabel "Y"
plot "km-c1.txt" using 1:2 title "cluster 1" with points, \
"km-c2.txt" using 1:2 title "cluster 2" with points, \
"km-c3.txt" using 1:2 title "cluster 3" with points
```



final report

- select A or B
  - A. web access log analysis
  - B. free topic
- up to 8 pages in the PDF format
- submission via SFC-SFS by 2012-01-25 (Wed) 23:59

final report (cont'd)

A. web access log analysis

- data: apache log (combined log format) used in Class 3
- from a JAIST server, access log for 24 hours
  `http://www.iijlab.net/~kjc/classes/sfc2011f-measurement/sample_access_log.bz2`
- write a script to extract the access count of each unique content, and plot the distribution in a log-log plot
  - X-axis:request count, Y-axis:CCDF for the number of URLs
- optionally, do other analysis
- the report should include (1) your script to extract the access counts, (2) a plot of the access count distribution, and (3) your analysis of the results

B. free topic

- select a topic by yourself
- the topic is not necessarily on networking
- but the report should include some form of data analysis and discussion about data and results

summary

Class 14 Scalable measurement and analysis

- distributed parallel processing
- cloud technology

next class

Class 15 Summary (1/18)

- summary of the class
- Internet measurement and privacy issues