Internet Measurement and Data Analysis (13)

Kenjiro Cho

2012-12-26

review of previous class

Class 12 Search and Ranking (12/19)

- Search systems
- PageRank
- exercise: PageRank algorithm
- **on final report**

today's topics

Class 13 Scalable measurement and analysis

- Distributed parallel processing
- Cloud computing technology
- MapReduce
- exercise: MapReduce algorithm

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

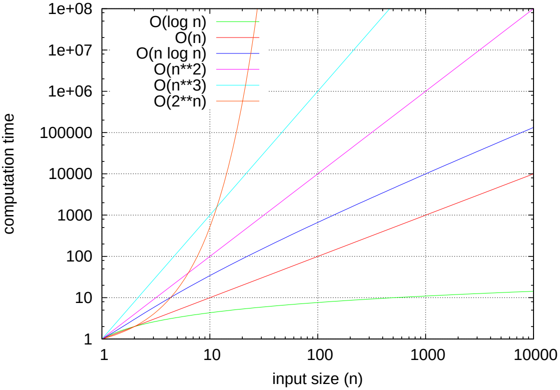
- average-case complexity
- worst-case complexity

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

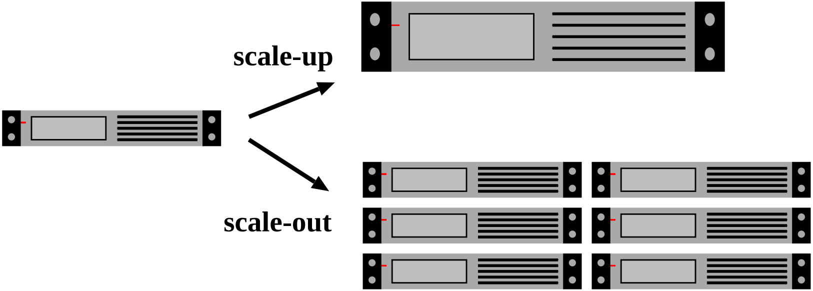
- assume that communications are message passing among independent computers
- failures of computers and message losses

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

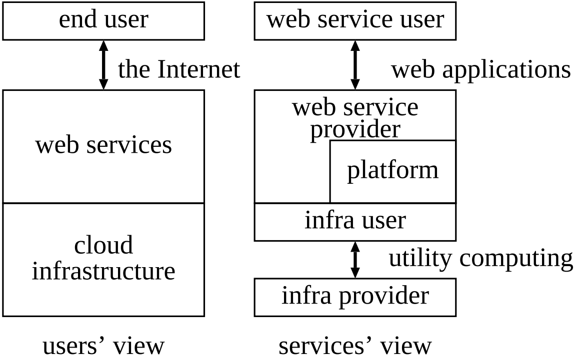
- broadly, computer resources behind a wide-area network

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
  - efficient use of resource pool

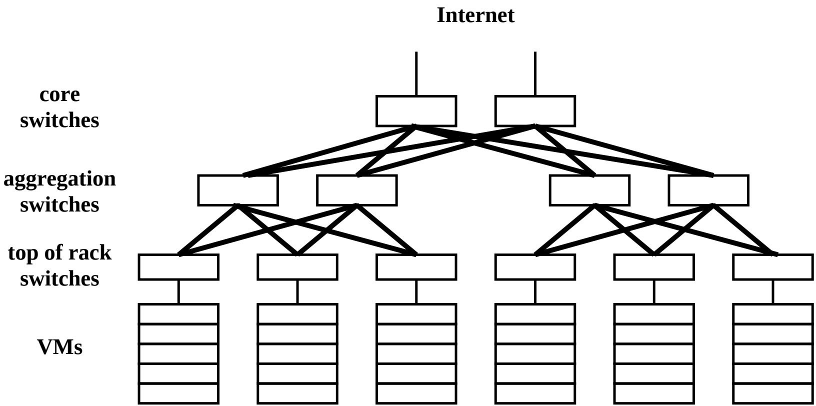
various clouds

- public/private/hybrid
- service classification: SaaS/PaaS/IaaS



physical clouds

typical cloud network topology



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

economics of cloud

- economies of scale (purchase cost, operation cost, statistical multiplexing)
- commodity hardware
- economical locations (including airconditioning, electricity, networking)

Will Japanese clouds be competitive in the global market?
(The bigger, the better?)

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

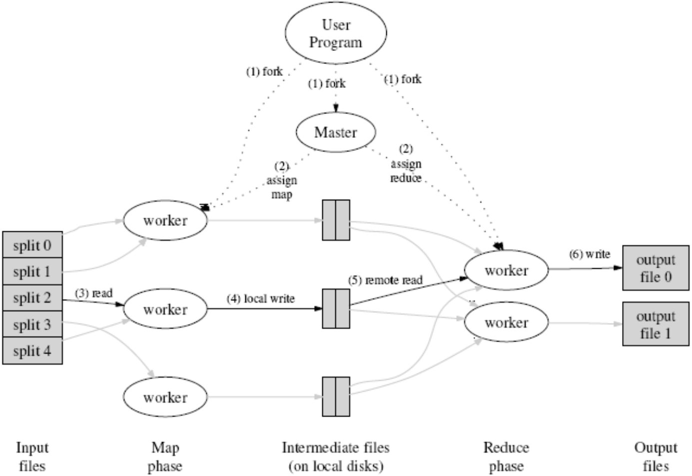
```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");

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

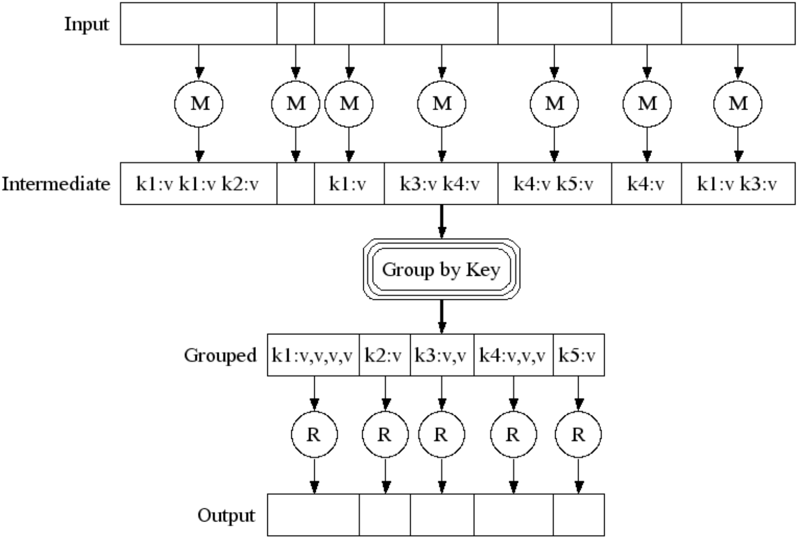
- distributed grep
  - map: output lines matching a supplied pattern
  - reduce: nothing
- count of URL access frequency
  - map: reading web access log, and outputs $< URL, 1 >$
  - reduce: adds together all values for the same URL, and emits $< URL, count >$
- reverse web-link graph
  - map: outputs $< target, source >$ 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



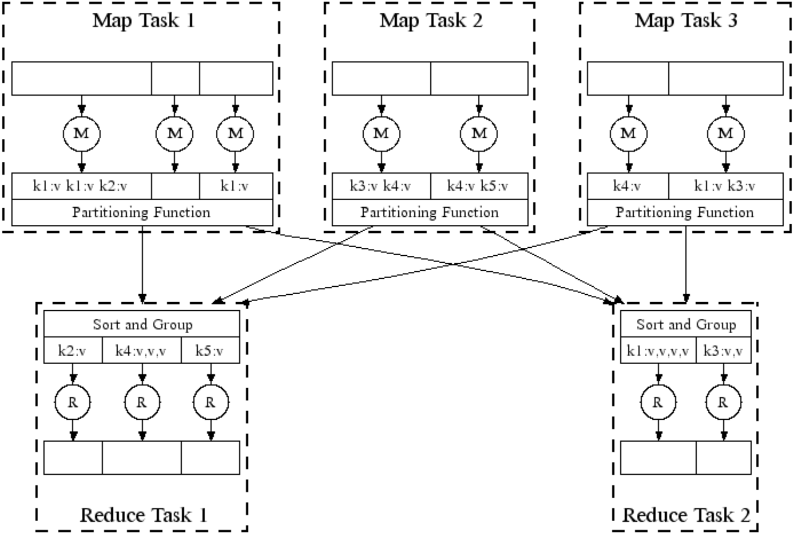
source: MapReduce: Simplified Data Processing on Large Clusters

MapReduce Execution



source: MapReduce: Simplified Data Processing on Large Clusters

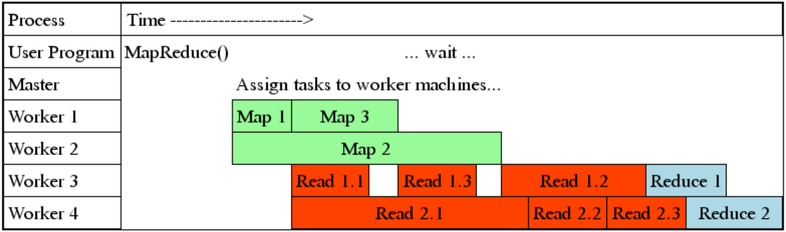
MapReduce Parallel Execution



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

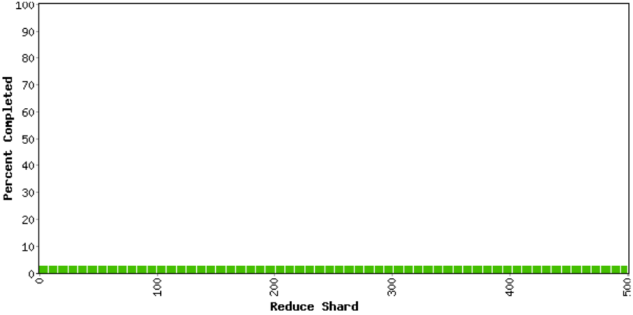
MapReduce status

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

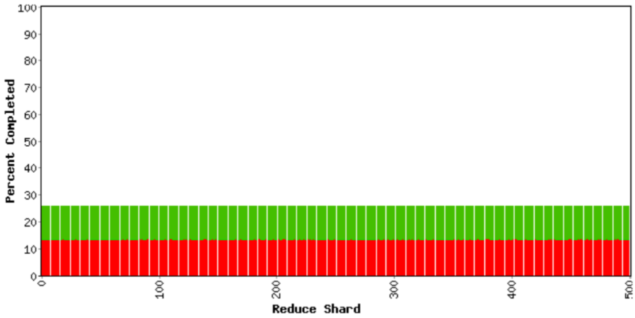
MapReduce status

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

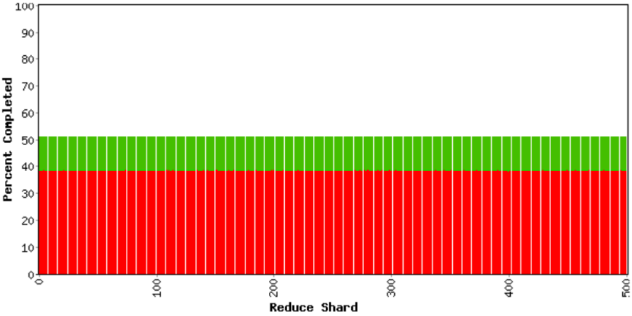
MapReduce status

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

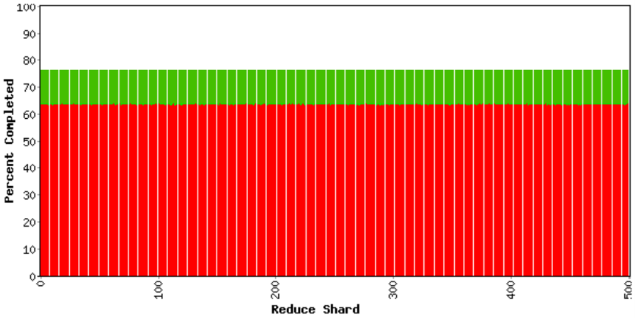
MapReduce status

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

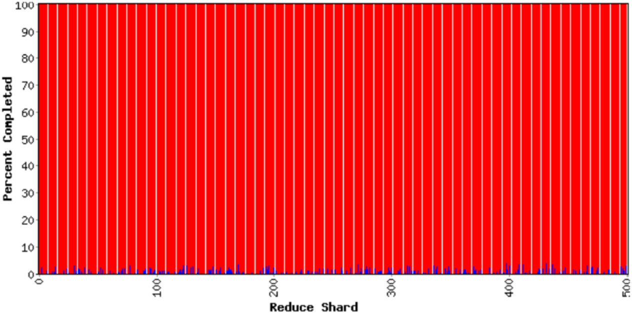
MapReduce status

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

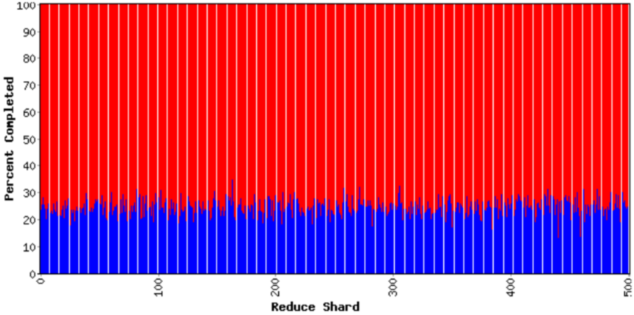
MapReduce status

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	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51738599	
mr-merge-outputs	51738599	

source: MapReduce: Simplified Data Processing on Large Clusters

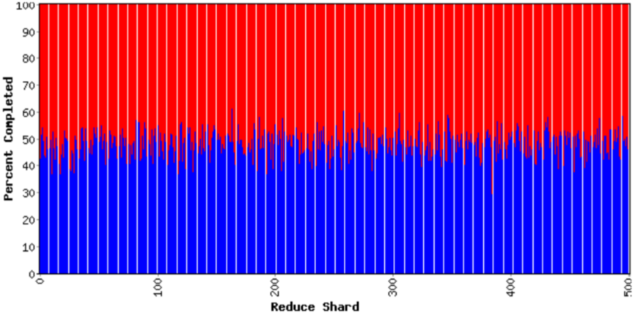
MapReduce status

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

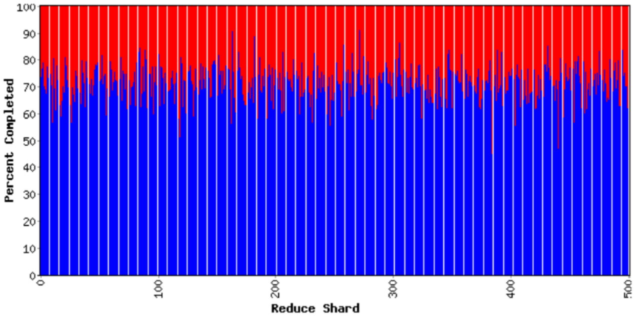
MapReduce status

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

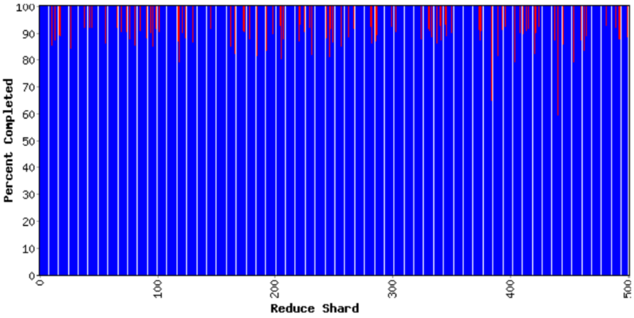
MapReduce status

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
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

source: MapReduce: Simplified Data Processing on Large Clusters

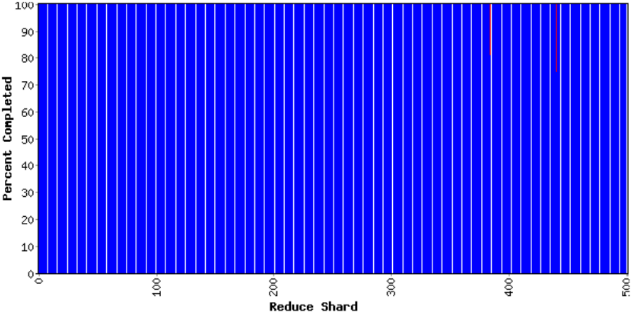
MapReduce status

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	1056
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

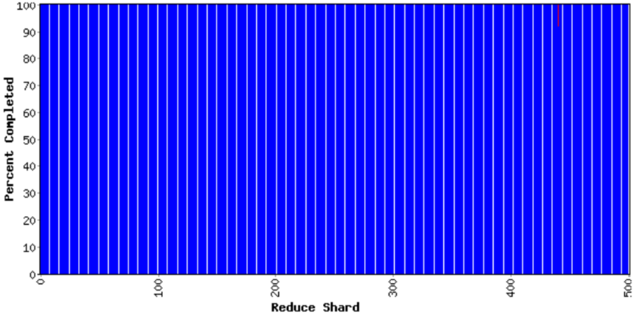
MapReduce status

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	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

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

- sorted order is guaranteed 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

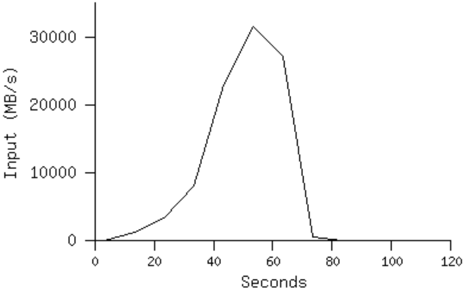
- 4GB of memory
- Dual-processor 2GHz Xeons with Hyperthreading
- Dual 160GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100Gbps

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

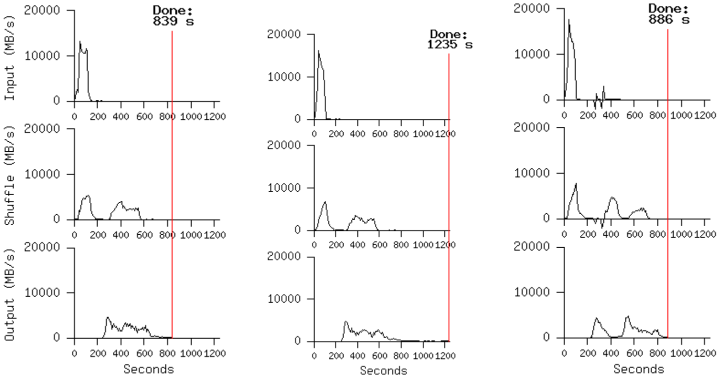
- locality optimization helps
  - 1800 machines read 1TB of data at peak of 31GB/s
  - without this, rack switches would limit to 10GB/s
- startup overhead is significant for short jobs



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

Hadoop MapReduce

- Hadoop
  - open source software by the Apache Project
  - Java software framework
  - implemention of Google's GFS and Mapreduce
  - widely used for large-scale data analysis platform
- Hadoop MapReduce
  - Java implementation
  - servers and libraries for MapReduce processing
  - Master/Slave architecture

WordCount in Hadoop MapReduce (1/3)

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

   public static class Map extends MapReduceBase implements Mapper<LongWritable,
         Text, Text, IntWritable> {
     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();

     public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
           Reporter reporter) throws IOException {
       String line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
         word.set(tokenizer.nextToken());
         output.collect(word, one);
       }
     }
   }
```

WordCount in Hadoop MapReduce (2/3)

```
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable,
      Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>
        output, Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

WordCount in Hadoop MapReduce (3/3)

```
  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }
}
```

today's exercise: WordCount in Ruby

MapReduce-style programming in Ruby

```
% cat wc-data.txt
Hello World Bye World
Hello Hadoop Goodbye Hadoop
% cat wc-data.txt | ruby wc-map.rb | sort | ruby wc-reduce.rb
bye     1
goodbye 1
hadoop  2
hello   2
world   2
```

WordCount in Ruby: Map

```
#!/usr/bin/env ruby
#
# word-count map task: input <text>, output a list of <word, 1>

ARGF.each_line do |line|
  words = line.split(/\W+/)
  words.each do |word|
    if word.length < 20 && word.length > 2
      printf "%s\t1\n", word.downcase
    end
  end
end
```

WordCount in Ruby: Reduce

```
#!/usr/bin/env ruby
#
# word-count reduce task: input a list of <word, count>, output <word, count>
# assuming the input is sorted by key
current_word = nil
current_count = 0
word = nil

ARGF.each_line do |line|
  word, count = line.split

  if current_word == word
    current_count += count.to_i
  else
    if current_word != nil
      printf "%s\t%d\n", current_word, current_count
    end
    current_word = word
    current_count = count.to_i
  end
end
if current_word == word
  printf "%s\t%d\n", current_word, current_count
end
```

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: PageRank

```
% cat sample-links.txt
# PageID: OutLinks
1:      2       3       4       5       7
2:      1
3:      1       2
4:      2       3       5
5:      1       3       4       6
6:      1       5
7:      5
% ruby pagerank.rb -f 1.0 sample-links.txt
reading input...
initializing... 7 pages dampingfactor:1.00 thresh:0.000001
iteration:1  diff_sum:0.661905 rank_sum: 1.000000
iteration:2  diff_sum:0.383333 rank_sum: 1.000000
...
iteration:20  diff_sum:0.000002 rank_sum: 1.000000
iteration:21  diff_sum:0.000001 rank_sum: 1.000000
[1] 1 0.303514
[2] 5 0.178914
[3] 2 0.166134
[4] 3 0.140575
[5] 4 0.105431
[6] 7 0.060703
[7] 6 0.044728
```

PageRank code (1/4)

```
require 'optparse'

d = 0.85   # damping factor (recommended value: 0.85)
thresh = 0.000001 # convergence threshold

OptionParser.new {|opt|
  opt.on('-f VAL', Float) {|v| d = v}
  opt.on('-t VAL', Float) {|v| thresh = v}
  opt.parse!(ARGV)
}

outdegree = Hash.new # outdegree[id]: outdegree of each page
inlinks = Hash.new # inlinks[id][src0, src1, ...]: inlinks of each page
rank = Hash.new # rank[id]: pagerank of each page
last_rank = Hash.new # last_rank[id]: pagerank at the last stage
dangling_nodes = Array.new # dangling pages: pages without outgoing link

# read a page-link file: each line is "src_id  dst_id_1 dst_id_2 ..."
ARGF.each_line do |line|
  pages = line.split(/\D+/)  # extract list of numbers
  next if line[0] == ?# || pages.empty?

  src = pages.shift.to_i  # the first column is the src
  outdegree[src] = pages.length
  if outdegree[src] == 0
    dangling_nodes.push src
  end
  pages.each do |pg|
    dst = pg.to_i
    inlinks[dst] ||= []
    inlinks[dst].push src
  end
end
```

PageRank code (2/4)

```
# initialize
# sanity check: if dst node isn't defined as src, create one as a dangling node
inlinks.each_key do |j|
  if !outdegree.has_key?(j)
    # create the corresponding src as a dangling node
    outdegree[j] = 0
    dangling_nodes.push j
  end
end

n = outdegree.length # total number of nodes
# initialize the pagerank of each page with 1/n
outdegree.each_key do |i|  # loop through all pages
  rank[i] = 1.0 / n
end
$stderr.printf " %d pages dampingfactor:%.2f thresh:%f\n", n, d, thresh
```

PageRank code (3/4)

```
# compute pagerank by power method
k = 0 # iteration number
begin
  rank_sum = 0.0  # sum of pagerank of all pages: should be 1.0
  diff_sum = 0.0  # sum of differences from the last round
  last_rank = rank.clone  # copy the entire hash of pagerank

  # compute dangling ranks
  danglingranks = 0.0
  dangling_nodes.each do |i| # loop through dangling pages
    danglingranks += last_rank[i]
  end

  # compute page rank
  outdegree.each_key do |i|  # loop through all pages
    inranks = 0.0
    # for all incoming links for i, compute
    #    inranks = sum (rank[j]/outdegree[j])
    if inlinks[i] != nil
      inlinks[i].each do |j|
        inranks += last_rank[j] / outdegree[j]
      end
    end

    rank[i] = d * (inranks + danglingranks / n) + (1.0 - d) / n
    rank_sum += rank[i]

    diff = last_rank[i] - rank[i]
    diff_sum += diff.abs
  end

  k += 1
  $stderr.printf "iteration:%d  diff_sum:%f rank_sum: %f\n", k, diff_sum, rank_sum
end while diff_sum > thresh
```

PageRank code (4/4)

```
# print pagerank in the decreasing order of the rank
# format: [position]  id  pagerank
i = 0
rank.sort_by{|k, v| -v}.each do |k, v|
  i += 1
  printf "[%d] %d %f\n", i, k, v
end
```

on the final report

- select A or B
  - A. PageRank computation of Wikipedia
  - B. free topic
- up to 8 pages in the PDF format
- submission via SFC-SFS by 2013-01-25 (Fri) 23:59

final report topics

A. PageRank computation of Wikipedia

- data: link data within Wikipedia English version (5.7M pages)
- A-1 investigate the distribution of pages
  - A-1-1 plot CDF and CCDF of the outdegree of pages
  - A-1-2 discussion on the outdegree distribution of Wikipedia pages
- A-2 PageRank computation
  - A-2-1 compute PageRank, and show the top 30 of the results
  - A-2-2 other analysis (optional)
  - A-2-3 discussion on 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

note: you may work with a classmate on programming. but, if you work with someone, make it clear in the report. still, you must write discussions by yourself.

A. PageRank computation of Wikipedia

data: link data of Wikipedia English version (5.7M pages)

- created by Henry Haselgrove (http://haselgrove.id.au/wikipedia.htm)
  - a local copy is avaialble from the class web page
  - a test data set (a subset of 100K pages)
- links-simple-sorted.zip: link data (323MB compressed, 1GB uncompressed)
  - each page has an unique integer ID
  - format: $from : to_1, to_2, ...to_n$
- titles-sorted.zip: title data (28MB compressed, 106MB uncompresed)
  - $n-$th line: the title of page ID n (1 origin)

```
% head -3 links-simple-sorted.txt
1: 1664968
2: 3 747213 1664968 1691047 4095634 5535664
3: 9 77935 79583 84707 564578 594898 681805 681886 835470 ...
%
% sed -n '2713439p' titles-sorted.txt
Keio-Gijuku_University
```

A-1 investigate the distribution of pages

A-1 investigate the distribution of pages

- A-1-1 plot CDF and CCDF of the outdegree of pages
  - include pages with outdegree 0
- A-1-2 discussion on the outdegree distribution of Wikipedia pages
  - optional other analysis
  - hint: you may compare low-degree pages and high-degree pages

A-2 PageRank computation

A-2 PageRank computation

- A-2-1 compute PageRank, and show top 30 of the results
  - format: rank PageRank_value page_ID page_title
  - you may use the script for the exercise
    - use damping factor:0.85 thresh:0.000001
  - takes 5 hours with iMac with 8GB memory (requiring at least 4GB memory)
- A-2-2 other analysis (optional)
  - examples:
  - how to reduce the processing time
  - implement an improved verion of the PageRank algorithm
- A-2-3 discussion on the results

summary

Class 13 Scalable measurement and analysis

- Distributed parallel processing
- Cloud computing technology
- MapReduce
- exercise: MapReduce algorithm

next class

Class 14 Privacy Issues (1/9)

- Internet data analysis and privacy issues
- Summary of the class