inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 18 – MapReduce

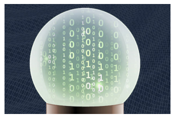
2014-10-13

Senior Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Top 10 Tech Trends ⇒

(1) Computing everywhere (2) Internet of Things (3) 3D printing (4) Analytics everywhere (5) Context rich systems (6) smart machines (7) cloud computing (8) software applications (9) web-scale IT (10) Security. Agree?



www.computerworld.com/article/2692619/gartner-lays-out-its-top-10-tech-trends-for-2015.html

Review of Last Lecture

- Warehouse Scale Computing
  - Example of parallel processing in the post-PC era
  - Servers on a rack, rack part of cluster
  - Issues to handle include load balancing, failures, power usage (sensitive to cost & energy efficiency)
  - PUE = Total building power / IT equipment power

2

Great Idea #4: Parallelism

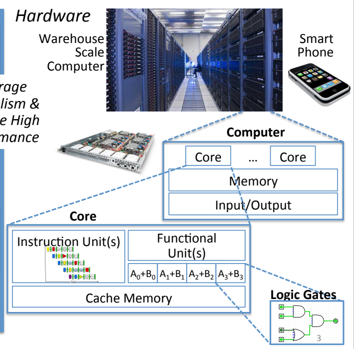
Today's Lecture

Software

- Parallel Requests
  Assigned to computer
  e.g. Search "Garcia"
- Parallel Threads
  Assigned to core
  e.g. Lookup, Ads
- Parallel Instructions
  > 1 instruction @ one time
  e.g. 5 pipelined instructions
- Parallel Data
  > 1 data item @ one time
  e.g. add of 4 pairs of words
- Hardware descriptions
  All gates functioning in parallel at same time

Leverage Parallelism & Achieve High Performance

Hardware



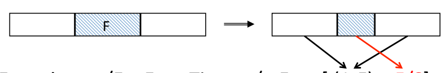
3

Amdahl's (Heartbreaking) Law

- Speedup due to enhancement E:

$$\text{Speedup w/E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/E}}$$

- **Example:** Suppose that enhancement E accelerates a fraction F (F<1) of the task by a factor S (S>1) and the remainder of the task is unaffected



- Exec time w/E = Exec Time w/o E × [(1-F) + F/S]
  Speedup w/E = 1 / [(1-F) + F/S]

4

Amdahl's Law

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

Non-sped-up part $(1 - F)$; Sped-up part $\dfrac{F}{S}$

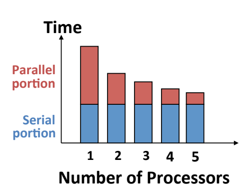
- **Example:** the execution time of 1/5 of the program can be accelerated by a factor of 10. What is the program speed-up overall?

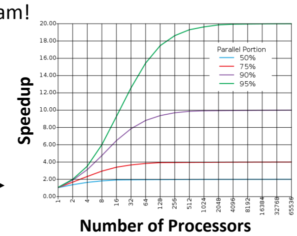
$$\frac{1}{0.8 + \frac{0.2}{10}} = \frac{1}{0.8 + 0.02} = 1.22$$

5

Consequence of Amdahl's Law

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



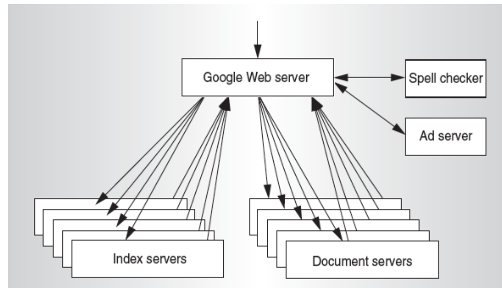


6

Request-Level Parallelism (RLP)

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

Google Query-Serving Architecture



Data-Level Parallelism (DLP)

- Two kinds:
  1) Lots of data in memory that can be operated on in parallel (e.g. adding together 2 arrays)
  2) Lots of data on many disks that can be operated on in parallel (e.g. searching for documents)

1) SIMD does Data-Level Parallelism (DLP) in memory
2) Today's lecture, Lab 6, Proj. 3 do DLP across many servers and disks using MapReduce

Administrivia … The Midterm

- Average around 10/20
  - Despite lots of partial credit
  - Regrades being processed
  - Have perspective – it's only 20 / 300 points.
  - Don't panic. Do lots of practice problems in a team. Do NOT study alone.
- Part 2 will be easier to compensate
- You can clobber Part 1 with Part 2

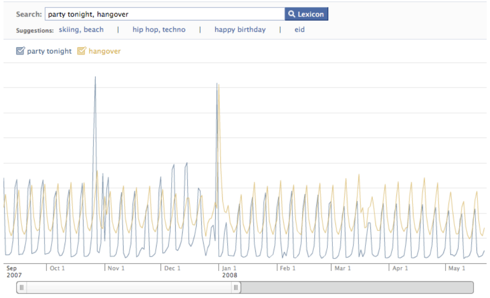
What is MapReduce?

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

What is MapReduce used for?

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

Example: Facebook Lexicon



www.facebook.com/lexicon(no longer available)

13

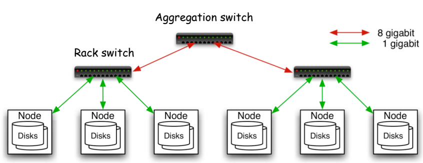
MapReduce Design Goals

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

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

14

Typical Hadoop Cluster



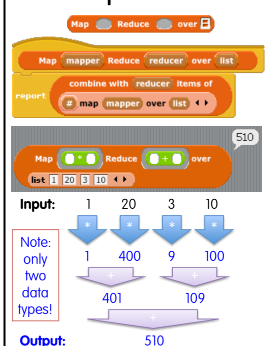
- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth within rack, 8 Gbps out of rack
- Node specs (Yahoo terasort):
  8 x 2GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

15

Image from http://wiki.apache.org/hadoop-data/attachments/HadoopPresentations/attachments/YahooHadoopIntro-apachecon-us-2008.pdf

en.wikipedia.org/wiki/MapReduce

MapReduce in CS10 & CS61A{,S}




```
> (reduce +
   (map square '(1 20 3 10))
510
```

```
>>> from functools import reduce
>>> def plus(x,y): return x+y
>>> def square(x): return x*x
>>> reduce(plus,
       map(square, (1,20,3,10)))
510
```

MapReduce Programming Model

Input & Output: each a set of key/value pairs

Programmer specifies two functions:

```
map (in_key, in_value) →
     list(interm_key, interm_value)
```

– Processes input key/value pair
– Slices data into "shards" or "splits"; distributed to workers
– Produces set of intermediate pairs

```
reduce (interm_key, list(interm_value)) →
     list(out_value)
```

– Combines all intermediate values for a particular key
– Produces a set of merged output values (usu just one)

code.google.com/edu/parallel/mapreduce-tutorial.html

MapReduce WordCount Example

- "Mapper" nodes are responsible for the **map** function

```
// "I do I learn" → ("I",1), ("do",1), ("I",1), ("learn",1)
map(String input_key,
    String input_value):
    // input_key  : document name (or line of text)
    // input_value: document contents
    for each word w in input_value:
        EmitIntermediate(w, "1");
```

- "Reducer" nodes are responsible for the **reduce** function

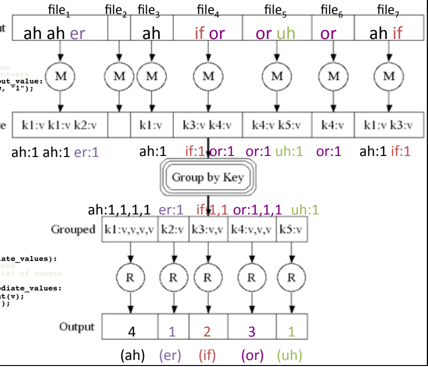
```
// ("I",[1,1]) → ("I",2)
reduce(String output_key,
       Iterator intermediate_values):
    // output_key   : a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
    Emit(AsString(result));
```

- Data on a distributed file system (DFS)

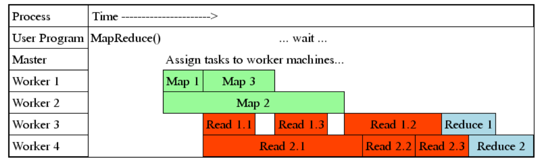
MapReduce WordCount Diagram

```
map(String input_key,
    String input_value):
    // input_key  : doc name
    // input_value: doc 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));
```



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"

20

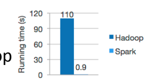
MapReduce WordCount Java code

```
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(WCMap.class);
    conf.setCombinerClass(WCReduce.class);
    conf.setReducerClass(WCReduce.class);
    conf.setInputPath(new Path(args[0]));
    conf.setOutputPath(new Path(args[1]));
    JobClient.runJob(conf);
}

public class WCMap extends MapReduceBase implements Mapper {
    private static final IntWritable ONE = new IntWritable(1);
    public void map(WritableComparable key, Writable value,
                    OutputCollector output,
                    Reporter reporter) throws IOException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            output.collect(new Text(itr.next()), ONE);
        }
    }
}

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

spark.apache.org

Spark

- **Apache Spark™** is a fast and general engine for large-scale data processing.
- **Speed**
  - Run programs up to 100x faster than Hadoop in memory, or 10x faster on disk.
  - Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



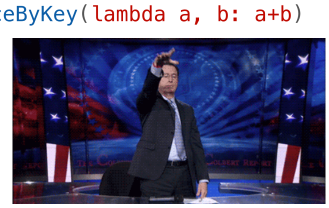
- **Ease of Use**
  - Write applications quickly in Java, Scala or Python.
  - Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala and Python shells.

22

Word Count in Spark's Python API

```
file = spark.textFile("hdfs://...")

file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Cf Java:



23

Peer Instruction

1. **Writing & managing SETI@Home is relatively straightforward; just hand out & gather data**
2. **Most parallel programs that, when run on N (N big) identical supercomputer processors will yield close to N x performance increase**
3. **The majority of the world's computing power lives in supercomputer centers and warehouses**

	123
A:	FFF
B:	FFT
B:	FTF
C:	FTT
C:	TFF
D:	TFT
D:	TTF
E:	TTT