Frequently asked questions from the previous class survey

- BitTorrent
  - Why download the rarest chunk first?
  - Is the partial torrent returned to different peers different?
  - Is the data structure need by tracker stored in memory?
  - Who maintains the tracker servers?
  - Popcorn Time
    - Sequential downloading
  - UTP or TCP: µTorrent
  - Pastry assignment
  - IDs for content and nodes: Should be specifiable

Outsourcing allows smaller services to benefit from mega services

- Automate the routine
  - Harness economies-of-scale

- Companies outsource payroll, insurance, web presence, and e-mail
  - Universities have tied up with Google for e-mail for instance

Outsourcing works under certain conditions ...

- Should be a service business
  - And computing should be CENTRAL
    - To operating and supporting the customer

- Application should be nearly identical across companies
  - Payroll, E-mail
  - Exception not the rule
Distributed computing does not have an outsourcing or business model

- Designed for computer-to-computer interactions
  - No eyeballs involved
- Need new business models to make profit
  - Enter the notion of leasing in modern Cloud systems

Baseline hardware parameters

- 2 GHz CPU with 8 GB RAM = $2000
- 200 GB disk = $200
- 100 access/sec
- 50 MB/sec transfer speed
- 1 Gbps Ethernet port-pair = $200
- 1 Mbps WAN link = $50/month

Note: Numbers are circa 2003

1 dollar buys you

- 1 GB transfer over WAN
- 1 day of CPU time ($1000/3)
- 1 GB disk space for 3 years
- 4 GB RAM for a day
- 10 M database accesses
- 10 TB of sequential disk access
- 10 TB of LAN Bandwidth (bulk)
- 10 KWhrs = 4 days of computer time

Caveats

- Beowulf clusters have different networking economics
  - Networking costs comparable to disk bandwidth
    - 10,000 times cheaper than price of Internet transports
  - Do not confuse with Internet-scale computations
- If telecom costs drop faster than Moore’s law … analysis fails
  - Over past 40 years telecom costs have fallen the slowest

The right abstraction level for Internet Distributed Computing

- Disk Block? No
- File? No
- Database? No
- Applications? Yes
  - BLAST search
  - Google search
  - Send/GET e-mail

Computing on-demand enable mobile applications

- Tasks are mobile
- Computing is dynamically provisioned
- Write-once-run-anywhere (WORA)
  - Java
  - COBOL
A computation task has 4 demands that must be met

① Networking
   Questions & Answers

② Computation
   Transform data/info into new information

③ Database/File Access
   Access to reference information

④ Database/File Storage
   Long term storage

Ratios of demands and the relative costs is pivotal

① OK to send GB of data if it saves years of computation

② NOT OK to send KB of data over network
   If computation can be performed locally

Ideal mobile computation task

① Stateless
   No disk access

② Tiny network input or output

③ Huge computational demand

④ Examples:
   - Cryptographic search
     (encrypted text, clear text, key search range)
   - Monte Carlo simulation
   - SETI@HOME

Why SETI@HOME is a good deal

① Sends out 10^9 jobs: each is 300 KB

② Network costs
   - 1 GB = $1
   - 1 MB = 10^-3 $ 
   - 100 KB = 10^-4 $

③ Compute Cost = 0.5$

④ Compute Cost/Network Cost = 0.5/(3*10^-4)
   Approx: 1600:1

How do you move a Terabyte?

<table>
<thead>
<tr>
<th>Speed</th>
<th>Rent/month</th>
<th>$/Mbps</th>
<th>$/TB</th>
<th>Time/TB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.04</td>
<td>40</td>
<td>1,000</td>
<td>3,086</td>
<td>6 years</td>
</tr>
</tbody>
</table>

Source: TeraScale SneakerNet, Microsoft Research, Jim Gray, Chang, Tom Bailey; Alex Szalay; Jan Vandenberg

Consequences

① The cheapest & fastest way to move Terabytes cross country is sneakernet
   - 24 hours = 4 MB/s
   - $50 shipping vs $1000 WAN cost

② Sending 10PB CERN data via network is silly:
   ① Buy disk bricks in Geneva
   ② Fill them
   ③ Ship them

TeraScale SneakerNet: Using Inexpensive Disks for Backup, Archiving, and Data Exchange
Jim Gray, Wyman Chong, Tom Bailey; Alex Szalay; Jan Vandenberg
Web Data processing systems

- Network or State intensive
- 100 MB FTP task = 10 cents
- 99% network cost
- HTML webpage access
  - 10^-6 dollars, 88% network cost
- Hotmail
  - 10^-3 dollars; some balance in CPU and network costs

Why Napster was a good deal

- 5 MB song
  - Network cost = 5 x 10^-2 $ ≈ ½ a penny
  - Both sender and receiver could afford it
- Yahoo! Serving web pages
  - 10^-2 $ in advertising revenue per page
  - 10^-4 $ total cost in serving web page
  - ROI: 100:1

Computations that are not economically viable

- Data loading and data scanning tasks
  - CPU-intensive, but also data intensive.
  - Therefore not economically viable as mobile applications.

Break even point for mobile computation tasks

- 10 Tops & 1 GB of networking both cost $1
- Break-even point
  - 10,000 instructions per byte of network traffic
  - 1 minute of processing per MB of network traffic
- Outsourcing becomes attractive when the cost-benefit ratio involves
  - 30,000 instructions per byte

The type of network also matters

- LAN is 10,000 cheaper than WAN
- Computational Fluid Dynamics
  - Simulate crack propagation in an Object
  - 100 MB input, 10 GB output, 7 CPU years
  - 10^6 instructions per byte : so good for WAN
  - But needs to executed in a tightly connected cluster
  - Cluster networking is free when compared to WAN networking

Toy Story 2

- A 200 MB image takes several CPU hours to render
- Instruction density
  - 200-600 x 10^3 instructions per byte
- Send 50 MB task; compute for 10 hours;
  - Return 200 MB image
Bioinformatics systems

- BLAST, FASTA and Smith-Waterman
  - Algorithms for matching DNA sequences against a database (GenBank or SwissProt).
  - Database sizes 50 GB
- Does it make sense to send SwissProt (40GB) to a server if processing (7220 hrs) is free?
  - Yes

Do not provision databases, provision the searches instead

- Does NOT make sense to provision databases on demand
- Set up dedicated servers instead
  - Use inexpensive servers and processors
  - Provision searches!
  - 40 GB server costs $20K
    - Can deliver complex 1-hour searches for $1

What does this imply?

- Put the computations near the data
  - Instruction density must exceed 10^5 per byte
- Combining data from multiple sites
  - PUSH processing to data sources
    - Filter the data early

MapReduce: Topics that we will cover

- Why?
- What it is and what it is not?
- The core framework and original Google paper
- Development of simple programs using Hadoop
  - The dominant MapReduce implementation

MapReduce

- It’s a framework for processing data residing on a large number of computers
- Very powerful framework
  - Superb for some problems
  - Challenging or not applicable in other classes of problems
What is MapReduce

- More a framework than a tool
- You are required to fit (some folks shoehorn it) your solution into the MapReduce framework
- MapReduce is not a feature, but rather a constraint

What does this constraint mean?

- It makes problem solving easier and harder
- Clear boundaries for what you can and cannot do
  - You actually need consider fewer options than what you are used to
  - But solving problems with constraints requires planning and a change in your thinking

But what does this get us?

- Tradeoff of being confined to the MapReduce framework?
  - Ability to process data on a large number of computers
  - But, more importantly, without having to worry about concurrency, scale, fault tolerance, and robustness

A challenge in writing MapReduce programs

- Design!
  - Good programmers can produce bad software due to poor design
  - Good programmers can produce bad MapReduce algorithms
  - Only in this case your mistakes will be amplified
    - Your job may be distributed on 100s or 1000s of machines and operating on a Petabyte of data

MapReduce: Origins of the design

- Process crawled data and logs of web requests
- Several computations work on this raw data to compute derived data
  - Inverted indices
  - Representation of graph structure of web documents
  - Pages crawled per host
  - Most frequent queries in a day …

Most computations are conceptually straightforward

- But data is large
- Computation must be scalable
  - Distributed across thousands of machines
  - To complete in a reasonable amount of time
Complexity of managing distributed computations can …

- Obscure simplicity of original computation
- Contributing factors:
  1. How to parallelize computation
  2. Distribute the data
  3. Handle failures

MapReduce was developed to cope with this complexity

- Express simple computations
- Hide messy details of
  - Parallelization
  - Data distribution
  - Fault tolerance
  - Load balancing

MapReduce

- Programming model
- Associated implementation for
  - Processing & Generating large data sets

Programming model

- Computation takes a set of input key/value pairs
- Produces a set of output key/value pairs
- Express the computation as two functions:
  - Map
  - Reduce

Map

- Takes an input pair
- Produces a set of intermediate key/value pairs

MapReduce library

- Groups all intermediate values with the same intermediate key
- Passes them to the Reduce function
Reduce function

- Accepts intermediate key I and
  - Set of values for that key
- Merge these values together to get
  - Smaller set of values

Counting number occurrences of each word in a large collection of documents

map (String key, String value)
  //key: document name
  //value: document contents
  for each word w in value
  EmitIntermediate(w, “1”)

reduce (String key, Iterator values)
  //key: a word
  //value: a list of counts
  int result = 0;
  for each v in values
    result += ParseInt(v);
  Emit(AsString(result));
  Sums together all counts emitted for a particular word

MapReduce specification object contains

- Names of
  - Input
  - Output
- Tuning parameters

Map and reduce functions have associated types drawn from different domains

map(k1, v1) → list(k2, v2)
reduce(k2, list(v2)) → list(v2)

What’s passed to-and-from user-defined functions

- Strings
- User code converts between
  - String
  - Appropriate types
Programs expressed as MapReduce computations: Distributed Grep

- **Map**
  - Emit line if it matches specified pattern
- **Reduce**
  - Just copy intermediate data to the output

**Term-Vector per Host**

- Summarizes important terms that occur in a set of documents `<word, frequency>`
  - **Map**
    - Emit `<hostname, term vector>`
    - For each input document
  - **Reduce function**
    - Has all per-document vectors for a given host
    - Add term vectors; discard away infrequent terms

**Implementation**

- Machines are commodity machines
- GFS is used to manage the data stored on the disks

**IMPLEMENTATION OF THE RUNTIME**

**Execution Overview – Part I**

- Maps distributed across multiple machines
- Automatic partitioning of data into M splits
- Splits processed concurrently on different machines

**Execution Overview – Part II**

- Partition intermediate key space into R pieces
  - E.g. `hash(key) mod R`
- User specified parameters
  - Partitioning function
  - Number of partitions (R)
Execution Overview

Execution Overview: Step I
The MapReduce library
- Splits input files into M pieces
  - 16-64 MB per piece
- Starts up copies of the program on a cluster of machines

Execution Overview: Step II
Program copies
- One of the copies is a Master
- There are M map tasks and R reduce tasks to assign
  - Master
    - Picks idle workers
    - Assigns each worker a map or reduce task

Execution Overview: Step III
Workers that are assigned a map task
- Read contents of their input split
- Parses <key, value> pairs out of input data
  - Pass each pair to user-defined Map function
  - Intermediate <key, value> pairs from Maps
    - Buffered in Memory

Execution Overview: Step IV
Writing to disk
- Periodically, buffered pairs are written to disk
  - These writes are partitioned
    - By the partitioning function
  - Locations of buffered pairs on local disk
    - Reported to back to Master
    - Master forwards these locations to reduce workers

Execution Overview: Step V
Reading Intermediate data
- Master notifies Reduce worker about locations
- Reduce worker reads buffered data from the local disks of Maps
  - Read all intermediate data; sort by intermediate key
    - All occurrences of same key grouped together
    - Many different keys map to the same Reduce task
Execution Overview: Step VI
Processing data at the Reduce worker

- Iterate over sorted intermediate data
- For each unique key pass
  - Key + set of intermediate values to Reduce function
- Output of Reduce function is appended
  - To output file of reduce partition

Execution Overview: Step VII
Waking up the user

- After all Map & Reduce tasks have been completed
- Control returns to the user code

Master Data Structures

- For each Map and Reduce task
  - State: (idle, in-progress, completed)
  - Worker machine identity
- For each completed Map task store
  - Location and sizes of R intermediate file regions
- Information pushed incrementally to in-progress Reduce tasks

The contents of this slide-set are based on the following references

- **JEFFREY DEAN** and **SANJAY GHEMAWAT**: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004: 137-150