Processing Large Datasets
Topics

• Parallel memory hierarchy
  – extend to include disk storage

• Google web search
  – Large parallel application
  – Distributed over a large cluster

• Programming models for large data collections
  – MapReduce
  – Spark
A. Extending the parallel memory hierarchy

- Incorporating disk storage
  - Parallel transfers to disks
  - Global access to all data
B. Google web search – 15 years ago

- **web statistics (2002)**
  - 3+ Billion static web pages
    » doubles every 8 months (2012: 1 Trillion pages)
  - 30% duplication

- **Google usage statistics (2002)**
  - 260 million users
    » 80% do searches
  - 150 million searches/day (2017: 3.5 billion searches/day)
    » ~2000 queries/sec average
    » ~1000 queries/sec minimum
  - query response time
    » less than 0.25 secs typical
    » target 0.5 secs max
  - uptime
    » target 100%

Sources:  
Google Linux Cluster (2002)

- **Overview**
  - 15,000+ PC cluster
    - 5 PB disk storage
      - $5 \times 10^{15}$ bytes = $50,000 \times 100$GB disks
  - node
    - 100 Mb Ethernet
    - 1-4 100 GB disks
    - mid-range processor (P III)
    - 256 MB - 2GB memory
    - runs Linux
  - rack
    - 100 to 200 nodes
    - Ethernet switch
  - router/switch
    - serves ~100 racks
    - distributes search requests
Google query processing steps (simplified)

1. secret sauce to map query to search terms
   - detect query language + fix spelling errors

2. locate search terms in dictionary
   - over 100 M words in dictionary per language

3. for each search term in dictionary
   - use inverted index to locate web pages containing term
   - ordered by page number

4. compute and order pages satisfying full query
   - explicit rules
     » conjunction, disjunction, etc. of terms
     » document language
   - implicit rules
     » search term proximity in documents
     » location of search terms in document structure
     » quality of page – PAGE RANK

5. Construct synopsis reports from documents in order
   - extract page from cache and highlight search terms in context
   - 10 results returned per query
Challenges

• **Query processing**
  – how to distribute data structures?
    » dictionary
    » inverted index
    » web pages
  – how to implement query processing algorithms?

• **Fault tolerance**
  – component count is very large
    » 10,000 servers with 3 year MTBF, expect to lose ten a day
    » 50,000 disks with 10% failing per year is a disk failure every couple of hours
    » \(10^{-15}\) undetected bit error rate on I/O is ~50 incorrect bits in 5PB copy

• **Scaling**
  – how can the system be designed to scale with
    » increasing number of queries
    » increasing size of web (number of pages and total text size)
    » increasing component failures (as a consequence of scaling up)
C. Processing large data sets

- **Process data distributed across thousands of disks**
  - Large datasets pose an I/O bottleneck
    - Attach disks to all nodes
    - Stripe data across disks
    - How to manage this?

- **MapReduce provides**
  - Parallel disk bandwidth
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates
Map/Reduce

- Map/Reduce
  - parallel programming schema
  - name inspired by functional language view of the schema

- Many problems can be phrased this way

- Easy to distribute across nodes

- Simple failure/retry semantics
Map in Lisp (Scheme)

- \( (\text{map } f \ list \ [list_2 \ list_3 \ …]) \)

- \( (\text{map square ‘(1 2 3 4))} \)
  \( \text{(1 4 9 16)} \)

- \( (\text{reduce + ‘(1 4 9 16))} \)
  \( (+ 16 (+ 9 (+ 4 1) )) \)
  \( = 30 \)
Map/Reduce a la Google

- **An input file contains a large list of items**
  - Each item is a (key, val) pair
  - The file is distributed across disks on p nodes

- **map(key, val) is run on each item in the list**
  - emits new-key / new-val pairs
  - map: \((k_1, v_1) \rightarrow \text{list}(k_2, v_2)\)

- **reduce(key, vals) is run for each unique key emitted by map()**
  - reduce: \((k_2, \text{list}(v_2)) \rightarrow \text{list}(v_2)\)
  - the result is written to a file distributed across disks attached to the nodes
Example 1: count words in docs

- Input consists of (url, contents) pairs

- map(key=url, val=contents):
  » For each word w in contents, emit (w, “1”)

- reduce(key=word, values=uniq_counts):
  » Sum all “1”s in values list
  » Emit result “(word, sum)”
map(key=url, val=contents):
    For each word w in contents, emit (w, “1”)

reduce(key=word, values=uniq_counts):
    Sum all “1”s in values list
    Emit result “(word, sum)"

see bob throw
see spot run

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>see</td>
<td>1</td>
</tr>
<tr>
<td>bob</td>
<td>1</td>
</tr>
<tr>
<td>run</td>
<td>1</td>
</tr>
<tr>
<td>see</td>
<td>1</td>
</tr>
<tr>
<td>spot</td>
<td>1</td>
</tr>
<tr>
<td>throw</td>
<td>1</td>
</tr>
</tbody>
</table>

bob 1
run 1
see 2
spot 1
throw 1
Execution

Input

Intermediate

Group by Key

Grouped

Output
Parallel Execution
Experience (10-15 years ago)

- **Rewrote Google's production indexing system using MapReduce**
  - Set of 10, 14, 17, 21, 24 MapReduce operations
  - New code is simpler, easier to understand
    - 3800 lines C++ → 700
  - MapReduce handles failures, slow machines
  - Easy to make indexing faster
    - add more machines

- **Redux**
  - MapReduce proved to be useful abstraction
  - MapReduce has an open-source implementation
    - Hadoop
  - Extensively used with large datasets
    - E.g. bioinformatics
    - focus on problem
    - let library deal w/ messy details
Improving MapReduce

• **Problem:** All computation is disk to disk
  – No notion of locality

• **Solution:** Spark
  – System for expressing computations on objects distributed on disks
  – Computations are moved to data instead of vice-versa
    » In-memory data flow model
    » Decreases number of map/reduce steps
Wrap-up

• Final exam
  – Saturday at noon, in this room
  – You may use a computer to access your own notes and all class materials
  – No communication, search or access of other materials

• Graded homeworks
  – All completed grading will be available before the exam
  – The final exam and any outstanding assignments will be available in my office Thu Dec 14 10-11:45 am and 2 – 4 pm

• Closing thoughts