Cloud Computing: A Few More Basics + MapReduce
April 17, 2014
Today

• Wrap up cloud computing basics
  • Virtualization
  • 10 obstacles/opportunities
• Programming the cloud: MapReduce
  • Basics
  • Graph algorithms in MapReduce (!)
Examples of cloud applications

• Application hosting
• Backup and Storage
• Content delivery
• E-commerce
• High-performance computing
• Media hosting
• On-demand workforce
• Search engines
• Web hosting
What is virtualization?

• Suppose Alice has a machine with 4 CPUs and 8 GB of memory, and three customers:
  • Bob wants a machine with 1 CPU and 3GB of memory
  • Charlie wants 2 CPUs and 1GB of memory
  • Daniel wants 1 CPU and 4GB of memory

• What should Alice do?
What is virtualization?

- Alice can sell each customer a virtual machine (VM) with the requested resources
  - From each customer's perspective, it appears as if they had a physical machine all by themselves (isolation)
How does it work?

- Resources (CPU, memory, ...) are virtualized
  - VMM ("Hypervisor") has translation tables that map requests for virtual resources to physical resources
  - Example: VM 1 accesses memory cell #323; VMM maps this to memory cell 123.
Benefit: Migration

- What if the machine needs to be shut down?
  - e.g., for maintenance, consolidation, ...
  - Alice can migrate the VMs to different physical machines without any customers noticing
Benefit: Time sharing

• What if Alice gets another customer?
  • Multiple VMs can time-share the existing resources
  • Result: Alice has more virtual CPUs and virtual memory than physical resources (but not all can be active at the same time)
Benefit and challenge: Isolation

- **Good:** Emil can't access Charlie's data
- **Bad:** What if the load suddenly increases?
  - Example: Emil's VM shares CPUs with Charlie's VM, and Charlie suddenly starts a large compute job
  - Emil's performance may decrease as a result
  - VMM can move Emil's software to a different CPU, or migrate it to a different machine
Recap: Virtualization in the cloud

• Gives cloud provider a lot of flexibility
  • Can produce VMs with different capabilities
  • Can migrate VMs if necessary (e.g., for maintenance)
  • Can increase load by overcommitting resources

• Provides security and isolation
  • Programs in one VM cannot influence programs in another

• Convenient for users
  • Complete control over the virtual 'hardware' (can install own operating system, own applications, ...)

• But: Performance may be hard to predict
  • Load changes in other VMs on the same physical machine may affect the performance seen by the customer
10 obstacles and opportunities

1. Availability
   • What happens to my business if there is an outage in the cloud?

2. Data lock-in
   • How do I move my data from one cloud to another?

3. Data confidentiality and auditability
   • How do I make sure that the cloud doesn't leak my confidential data?
   • Can I comply with regulations like HIPAA and Sarbanes/Oxley?

CACM “A View of Cloud Computing”
4. Data transfer bottlenecks
   • How do I copy large amounts of data from/to the cloud?
   • Example: 10 TB from UC Berkeley to Amazon in Seattle, WA
     • ~45 days versus FedExing 10 1TB disks overnight
   • Motivated Import/Export feature on AWS

5. Performance unpredictability
   • Example: VMs sharing the same disk → I/O interference
   • Example: HPC tasks that require coordinated scheduling

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Mean perf.</th>
<th>Std dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory bandwidth</td>
<td>1.3GB/s</td>
<td>0.05GB/s  (4%)</td>
</tr>
<tr>
<td>Disk bandwidth</td>
<td>55MB/s</td>
<td>9MB/s (16%)</td>
</tr>
</tbody>
</table>

Performance of 75 EC2 instances in benchmarks

CACM “A View of Cloud Computing”
10 obstacles and opportunities

6. Scalable storage
   • Cloud model (short-term usage, no up-front cost, infinite capacity on demand) does not fit persistent storage well

7. Bugs in large distributed systems
   • Many errors cannot be reproduced in smaller configurations

8. Scaling quickly
   • Problem: Boot time; idle power
   • Fine-grain accounting?
10 obstacles and opportunities

9. Reputation fate sharing
   • One customer's bad behavior can affect the reputation of others using the same cloud
   • Example: Spam blacklisting, FBI raid after criminal activity

10. Software licensing
   • What if licenses are for specific computers?
     • Example: Microsoft Windows
   • How to scale number of licenses up/down?
     • Need pay-as-you-go model as well
The rest of the story …

- Practicalities: Amazon Web Services (Last week: Hancheng)
- Foundations: Bloom filters (Last time)
- MapReduce (today)
- Cloud Storage
MapReduce
How do we do “big computation” on a data center-sized computer?
MapReduce

• A famous distributed programming model

• In many circles, considered the key building block for much of Google’s data analysis

• Other similar languages: Yahoo’s Pig Latin and Pig; Microsoft’s Dryad

• Count the number of times each distinct word appears in this document
Word Count - Case 1: Small Single File

- Read file
- Use a dictionary to track the number of times a word appears.
Word Count – Case 2:
A Google Crawl of 50 billion web pages

• Distributed / parallel computing
The Hope: Divide and Conquer

"Work"

$w_1$  $w_2$  $w_3$

"worker"

$r_1$  $r_2$  $r_3$

"Result"
Distributed Computing Challenges - Scheduling

• How do we assign work units to workers?
• What if we have more work units than workers?
Distributed Computing Challenges - Synchronization

• What if workers need to share partial results?
• How do we aggregate partial results?
• How do we know all the workers have finished?
Distributed Computing Challenges – Handling Network Failure
Current Tools

- Programming models
  - Shared memory (pthreads)
  - Message passing (MPI)
- Design Patterns
  - Master-slaves
  - Producer-consumer flows
  - Shared work queues
Where the rubber meets the road

• Concurrency is difficult to reason about
• Concurrency is even more difficult to reason about
  – At the scale of datacenters (even across datacenters)
  – In the presence of failures
  – In terms of multiple interacting services
• Not to mention debugging...
MapReduce to the rescue
Filter+Stack Worker

CountStack Worker

blue: 4k

green: 4k

cyan: 3k

gray: 1k

orange: 4k
Two kinds of workers

- There are two kinds of workers:
  - Those that take input data items and produce output items for the “stacks”
  - Those that take the stacks and aggregate the results to produce outputs on a per-stack basis

- We’ll call these:
  - **map**: takes (item_key, value), produces one or more (stack_key, value’) pairs
  - **reduce**: takes (stack_key, {set of value’}), produces one or more output results – typically (stack_key, agg_value)
The MapReduce programming model

- Simple distributed functional programming primitives
- Modeled after Lisp primitives:
  - **map** (apply function to all items in a collection) and
  - **reduce** (apply function to set of items with a common key)
- We start with:
  - A user-defined function to be applied to all data,
    - **map**: (key, value) —> (key, value)
  - Another user-specified operation
    - **reduce**: (key, {set of values}) —> result
  - A set of n nodes, each with data
- All nodes run map on all of their data, producing new data with keys
  - This data is collected by key, then shuffled, and finally reduced
  - Dataflow is through temp files on GFS
What do you do?

• Define two functions:
  – Map(k,v) → <k’, v’>*
  – Reduce(k’,<v’>*) → <k’, v’’>*

• All v’ with same k’ are reduced together and processed in v’ order
What MapReduce Does?

• Handles scheduling
  – Assigns workers to map and reduce tasks
• Handles “data distribution”
  – Moves processes to data
• Handles synchronization
  – Gathers, sorts, and shuffles intermediate data
• Handles errors and faults
  – Detects worker failures and restarts
• Everything happens on top of a distributed FS
  – Ex: Google’s GFS and Hadoop’s HDFS
MapReduce: Word counting

• Program specifies two primary methods
  – Map(k,v) → <k’, v’>*
  – Reduce(k’,<v’>*) → <k’, v’’>*

• Ex: Two documents - (d1, “the crew”) and (d2, “of the”)
  – Map(d1, “the crew”) => [(the, 1), (crew, 1)]
  – Map(d2, “of the”) => [(of, 1), (the, 1)]
    MapReduce runs its grouper module and calls reduce for every key
  – Reduce (the, [1,1]) => (the, 2)
  – Reduce (crew, [1]) => (crew, 1)
  – Reduce (of, [1]) => (of, 1)
Word Count in MapReduce

map(String key, String value) {
    // key: document name, line no
    // value: contents of line
    for each word w in value:
        emit(w, "1")
}

reduce(String key, Iterator values) {
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    emit(key, result)
}
• 2-slides.ppt
Example 1: Host size

• Suppose we have a large web corpus
• Let’s look at the metadata file
  – Lines of the form (URL, size, date, …)
• For each host, find the total number of bytes
  – i.e., the sum of the page sizes for all URLs from that host
• Solution:
  – Map(line_no, (URL, size, date, ...)) => [(URL_host, size)]
  – Reduce(URL_host, [size1, size2,...]) => (URL_host, sum([size1, size2,...]))
Example 2: Language model

• Statistical machine translation
  – Need to count number of times every 5-word sequence occurs in a large collection of documents

• Solution
  – Map(doc_id, document) => [(5-word seq, count),...]
  – Reduce(5-word seq, [count1, ...]) => (5-word seq, sum([count1, ...]))
Example 3: Distributed Grep

• Find all occurrences of the given pattern in a very large set of files

• Solution
  – Map(doc_id, document) => [(pattern, occ_line), ...]
  – Reduce(pattern, [occ_line1, ...]) => (pattern, [occ_line1, ...])
Example 4: Graph reversal

- Given a directed graph as an adjacency list:
  src1: dest11, dest12, ...
  src2: dest21, dest22, ...
- Construct the graph in which all the links are reversed
- Solution:
  - Map(src1, [dest11, dest12, ...]) => [(dest11, src1), (dest12, src1)]
  - Reduce(dest11, [src1, src4, ...]) =>(dest11, [src1, src4, ...])
Example 5: Reverse Web-Link Graph

• Determine in-coming links (Page rank)
• Solution:
  – Map(src_page_url, page_html) => [(link1, src_page_url), ...]
  – Reduce(link, [src_page_url1, ...]) => (link, [src_page_url1, ...])
Common mistakes to avoid

• **Mapper and reducer should be stateless**
  
  • Don't use static variables - after map + reduce return, they should remember nothing about the processed data!

  • Reason: No guarantees about which key-value pairs will be processed by which workers!

• **Don't try to do your own I/O!**

  • Don't try to read from, or write to, files in the file system

  • The MapReduce framework does all the I/O for you:
    
    - All the incoming data will be fed as arguments to map and reduce

    - Any data your functions produce should be output via emit
Design considerations

- **Key decision:** What should be done by *map*, and what by *reduce*?
  - *map* can do something to each individual key-value pair, but it can't look at other key-value pairs
    - Example: Filtering out key-value pairs we don't need
  - *map* can emit more than one intermediate key-value pair for each incoming key-value pair
    - Example: Incoming data is text, map produces (word,1) for each word
  - *reduce* can aggregate data; it can look at multiple values, as long as map has mapped them to the same (intermediate) key
    - Example: Count the number of words, add up the total cost, ...

- **Need to get the intermediate format right!**
  - If reduce needs to look at several values together, map must emit them using the same key!
• graph-algorithms.ppt