Some Cloud Computing Topics

CSE 454

Aaron Kimball
Cloudera Inc.
3 Dec 2009
About me

- Aaron Kimball (M.S. 2008)
- Designed/taught CSE 490H
- I now work for Cloudera ("The Commercial Hadoop Company")
“I suspect that an overview on cloud computing that hits highlights on GFS, hadoop, bigtable, Ec2 would be great. (Or a subset or extended subset of those topics) would be appreciated by the students.”

– email from Dan 10/29/09
An outline?

- Big Data (Corporations are packrats)
- Big Computations (If you want it done right…)
- Big Computing Environments
Databases

- MySQL, Oracle, SQL Server…
- Store *structured data* along with large amount of *metadata*
  - A finite set of fields per record with well-defined types
  - Lots of bookkeeping information (table statistics, indices over one or more columns, constraints on data integrity…)
    - Really cool data structures! (e.g., B-Trees)

- **Pro**: REALLY FAST queries of certain types
  - Metadata can be tuned to make certain queries better
- **Con**: Metadata has time and space costs to create, maintain. Must also predict / control the *schema* of the information
- ... Take CSE 444 for more information
Example table

mysql> use corp;
Database changed

mysql> describe employees;

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>firstname</td>
<td>varchar(32)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>lastname</td>
<td>varchar(32)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>jobtitle</td>
<td>varchar(64)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>start_date</td>
<td>date</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>dept_Id</td>
<td>int(11)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>
Some things databases are good at

- Using an index to look up a particular row
- Grouping together rows with a shared key
  - And applying “aggregation” functions (SUM, AVG, STDDEV…)
- Enforcing data quality
  - e.g.: no duplicates; type-safety; other business-logic constraints
The problems...

- A hard drive writes at 50—60 MB/sec.
  - … but only if you’re writing in a straight line
  - Maintaining indexed data may drop this by 10x or more
  - Buffering / delayed writes can help recover this
- Performing database operations in parallel is complicated, and scalability is challenging to implement correctly
- Databases hold max 10 TB; queries can scan ~10%
Bigger data, better processing

- How do we store 1,000x as much data?
- How do we process data where we don’t know the schema in advance?
- How do we perform more complicated processing?
  - Natural language processing, machine learning, image processing, web mining…
- How do we do this at the rate of TB/hour?
What we need

- An **efficient** way to decompose problems into parallel parts
- A way to read and write data **in parallel**
- A way to **minimize** bandwidth usage
- A **reliable** way to get computation done
What does it mean to be reliable?

Ken Arnold, CORBA designer*: “Failure is the defining difference between distributed and local programming” *(Serious Über-hacker)*
Reliability Demands

- Support partial failure
  - Total system must support graceful decline in application performance rather than a full halt
Reliability Demands

- Data Recoverability
  - If components fail, their workload must be picked up by still-functioning units
Reliability Demands

- Individual Recoverability
  - Nodes that fail and restart must be able to rejoin the group activity without a full group restart
Reliability Demands

- Consistency
  - Concurrent operations or partial internal failures should not cause externally visible nondeterminism
Reliability Demands

- Scalability
  - Adding increased load to a system should not cause outright failure, but a graceful decline
  - Increasing resources should support a proportional increase in load capacity
A Radical Way Out…

- Nodes talk to each other as little as possible – maybe never
  - “Shared nothing” architecture
- Programmer should not explicitly be allowed to communicate between nodes
- Data is spread throughout machines in advance, computation happens where it’s stored.
Locality

- Master program divvies up tasks based on location of data: tries to have map tasks on same machine as physical file data, or at least same rack
- Map task inputs are divided into 64—128 MB blocks: same size as filesystem chunks
  - Process components of a single file in parallel
Fault Tolerance

- Tasks designed for independence
- Master detects worker failures
- Master re-executes tasks that fail while in progress
- Restarting one task does not require communication with other tasks
- Data is replicated to increase availability, durability
How MapReduce is Structured

- Functional programming meets distributed computing
- A batch data processing system
- Factors out many reliability concerns from application logic
MapReduce Provides:

- Automatic parallelization & distribution
- Fault-tolerance
- Status and monitoring tools
- A clean abstraction for programmers
Programming Model

- Borrows from functional programming
- Users implement interface of two functions:
  - map \((\text{in\_key}, \text{in\_value}) \rightarrow (\text{intermediate\_key}, \text{int\_value}) \text{ list}\)
  - reduce \((\text{intermediate\_key}, \text{int\_value \text{ list}}) \rightarrow (\text{out\_key}, \text{out\_value}) \text{ list}\)
map

map (in_key, in_value) ->
(intermediate_key, int_value) list
reduce

reduce (intermediate_key, int_value list) -> (out_key, out_value) list

initial

returned
Example: Filter Mapper

```fsharp
let map(k, v) =
    if (isPrime(v)) then emit(k, v)

(“foo”, 7) ➞ (“foo”, 7)
(“test”, 10) ➞ (nothing)
```
Example: Sum Reducer

```ml
let reduce(k, vals) =
  sum = 0
  foreach int v in vals:
    sum += v
  emit(k, sum)

(“A”, [42, 100, 312]) ➞ (“A”, 454)
(“B”, [12, 6, -2]) ➞ (“B”, 16)
```
Data store 1

Data store n

map
(key 1, values...)
(key 2, values...)
(key 3, values...)

map
(key 1, values...)
(key 2, values...)
(key 3, values...)

Input key*value pairs

(key 1, values...)
(key 2, values...)
(key 3, values...)

== Barrier == : Aggregates intermediate values by output key

key 1, intermediate values
reduce
final key 1 values

key 2, intermediate values
reduce
final key 2 values

key 3, intermediate values
reduce
final key 3 values
Example: Count word occurrences

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

reduce(String output_key, Iterator<int> intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += v;
    emit(output_key, result);
```
That’s how to process data in parallel

- … How to store all this data?
  - HDFS / GFS
Storage assumptions

▪ High component failure rates
  ▪ Inexpensive commodity components fail all the time
▪ “Modest” number of HUGE files
  ▪ Just a few million
  ▪ Each is 100MB or larger; multi-GB files typical
▪ Files are write-once (maybe appended-to)
▪ Large streaming reads
▪ High sustained throughput favored over low latency
GFS/HDFS Design Decisions

- Files stored as blocks
  - Much larger size than most filesystems (default is 64MB)
- Reliability through replication
  - Each block replicated across 3+ DataNodes
- Single master (NameNode) coordinates access, metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Familiar interface, but customize the API
  - Simplify the problem; focus on distributed apps
GFS Architecture

Figure from “The Google File System,”
Ghemawat et. al., SOSP 2003
The key insight...

- DataNodes (storing blocks of files) are the **same machines** as MapReduce worker nodes

- When scheduling tasks, MapReduce picks nodes based on **where data already is resident**
  - Data replication increases durability, also improves scheduling efficiency
Cloud computing: broader than any one app

Cloud computing is a method to address **scalability** and **availability** concerns for enterprise applications.
An evolving ecosystem

- Hardware Infrastructure
- Language-level Infrastructure
- Platform Infrastructure
- Application Infrastructure

- Breadth of applicability
- Rapid development in domain
Hardware as a service: virtualization

- Amazon EC2: Machines for rent by the hour, on demand.
  - But you don’t necessarily get a full machine (maybe just a slice)
- Google AppEngine: We give you “cycles,” who knows where
Virtualization in a nutshell

- Just a layer of software that responds to device drivers (hard drive, Ethernet, graphics) like the drivers/OS expect
- Software layer then does “something reasonable” with underlying actual resources
A fully-virtualized machine

- All applications run in a VM
- One or more VMs may share machine
Clouds: high-level

1) Requests for more service instances

2) Commands to allocate new virtual instances

3) Services provided to clients outside of cloud
Key promises of virtualization

- Users can get more machines in on-demand fashion
- Interface to new virtual nodes is through software API
  - So software on existing nodes can recognize over/underloading and make requests to provider to adjust on the fly.
- EC2 gives you more explicit control in terms of virtual nodes
- AppEngine takes this to the next level and does not expose any hardware to you whatsoever
  - Makes web application development simpler. Makes high-performance system design nearly impossible
- My next wish: explicit network topology control…
Some concluding summary…

- Processing lots of data requires lots of machines
- Using lots of machines in parallel requires
  - Some infrastructure to manage it for you (Hadoop)
  - The ability to decompose a problem into independent subtasks
- High performance requires data locality
  - (It’s not processing data that’s slow. It’s **moving** data that is.)
Want to play with Hadoop?

- We’ve got a virtual machine available online
  - Eclipse, and Hadoop, some exercises, and a tutorial all set up

- You’ll need VMWare Player/Fusion to run it

- It’s about a 1 GB download, 4 GB unpacked (so get this in 002 ;) )
Want a job this summer?

- No promises but last year we had a bunch of interns
  - We’ll probably need some more

- You get to play with Hadoop, other distributed systems, EC2…

- The catch: We’re pretty bad at making commitments this far out. Talk to us again around March/April.

- Want a full-time job? We’ve got a bunch of those :) 

- Send resumes/inquiries/etc to aaron@cloudera.com
Thanks for listening

Questions: aaron@cloudera.com