CS 5614: (Big) Data Management Systems

B. Aditya Prakash

Lecture #11: No-SQL, Map-Reduce and New Software Stack
(some slides from Xiao Yu)

NO SQL
Why No SQL?

**How to Write a CV**

- **Do you have any expertise in SQL?**
  - **No**

- Doesn't matter. Write: "Expert in No SQL"
RDBMS

- The predominant choice in storing data
  - Not so true for data miners since we put much in txt files.
- First formulated in 1969 by Codd
  - We are using RDBMS everywhere
Aside: RDBMS performance

Performance

Salary List

Majority of Webapps

Social network

Semantic Trading

custom

Data complexity

I DON'T ALWAYS USE RDBMS

BUT WHEN I DO, I DUMP EVERYTHING IN IT

Virginia Tech

Virginia Tech, “A NoSQL Overview and the Benefits of Graph Databases”

Prakash 2014

VT CS 5614
When RDBMS met Web 2.0

Big data

Connectivity

P2P Knowledge

Concurrency

Diversity

Cloud-Grid

Slide from Lorenzo Alberton, "NoSQL Databases: Why, what and when"
What to do if data is really large?

- Peta-bytes (exabytes, zettabytes ....)

- Google processed 24 PB of data per day (2009)

- FB adds 0.5 PB per day
**BIG data**

Data never sleeps. How much data is generated every minute?

- **YouTube users upload 48 hours of new video.**
- **Wordpress users publish 347 new blog posts.**
- **WordPress users publish 347 new blog posts.**
- **New websites are created.**
- **Flickr users add 2,083 new photos.**
- **Instagram users share 3,600 new photos.**
- **Twitter users send over 100,000 tweets.**
- **Facebook users share 684,478 pieces of content.**
- **Facebook users share 684,478 pieces of content.**
- **Foursquare users perform 2,083 check-ins.**
- **Foursquare users perform 2,083 check-ins.**
- **Apple receives about 47,000 app downloads.**
- **Apple receives about 47,000 app downloads.**
- **Brands & organizations on Facebook receive 34,722 “likes.”**
- **Brands & organizations on Facebook receive 34,722 “likes.”**

With no signs of slowing, the data keeps growing. These are just some of the more common ways that Internet users add to the big data pool. In truth, depending on the niche of businesses you’re in, there are virtually countless other sources of relevant data to pay attention to. Consider the following:

The global Internet population grew 9.5% percent from 2010 to 2011 and now represents 2.1 billion people.

These users are real, and they are out there leaving data trails everywhere they go. The team at Domo can help you make sense of this seemingly insurmountable heap of data, with solutions that help executives and managers bring all of their critical information together in one intuitive interface, and then use that insight to transform the way they run their business. To learn more, visit www.domo.com.
What’s Wrong with Relational DB?

- Nothing is wrong. You just need to use the right tool.
- Relational is hard to scale.
  - Easy to scale reads
  - Hard to scale writes
What’s NoSQL?

- The misleading term “NoSQL” is short for “Not Only SQL”.
- non-relational, schema-free, non-(quite)-ACID
  - More on ACID transactions later in class
- horizontally scalable, distributed, easy replication support
- simple API
Four (emerging) NoSQL Categories

- **Key-value (K-V) stores**
  - Based on Distributed Hash Tables/ Amazon’s Dynamo paper *
  - Data model: (global) collection of K-V pairs
  - Example: Voldemort

- **Column Families**
  - BigTable clones **
  - Data model: big table, column families
  - Example: HBase, Cassandra, Hypertable

*G DeCandia et al, Dynamo: Amazon's Highly Available Key-value Store, SOSP 07
** F Chang et al, Bigtable: A Distributed Storage System for Structured Data, OSDI 06
Four (emerging) NoSQL Categories

- **Document databases**
  - Inspired by Lotus Notes
  - Data model: collections of K-V Collections
  - Example: CouchDB, MongoDB

- **Graph databases**
  - Inspired by Euler & graph theory
  - Data model: nodes, relations, K-V on both
  - Example: AllegroGraph, VertexDB, Neo4j
Focus of Different Data Models

Slide from neo technology, “A NoSQL Overview and the Benefits of Graph Databases"
C-A-P “theorem”

Partition Tolerance

Consistency

Availability

RDBMS

NoSQL (most)
When to use NoSQL?

- Bigness
- Massive write performance
  - Twitter generates 7TB / per day (2010)
- Fast key-value access
- Flexible schema or data types
- Schema migration
- Write availability
  - Writes need to succeed no matter what (CAP, partitioning)
- Easier maintainability, administration and operations
- No single point of failure
- Generally available parallel computing
- Programmer ease of use
- Use the right data model for the right problem
- Avoid hitting the wall
- Distributed systems support
- Tunable CAP tradeoffs

from http://highscalability.com/

Prakash 2014
Key-Value Stores

<table>
<thead>
<tr>
<th>id</th>
<th>hair_color</th>
<th>age</th>
<th>height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1923</td>
<td>Red</td>
<td>18</td>
<td>6’0”</td>
</tr>
<tr>
<td>3371</td>
<td>Blue</td>
<td>34</td>
<td>NA</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table in relational db

Store/Domain in Key-Value db

Find users whose age is above 18?
Find all attributes of user 1923?
Find users whose hair color is Red and age is 19?
(Join operation) Calculate average age of all grad students?
Voldemort in LinkedIn

People You May Know

Viewers of this profile also viewed

Related Searches

Events you may be interested in

LinkedIn Skills

Jobs you may be interested in

Sid Anand, LinkedIn Data Infrastructure (QCon London 2012)
Voldemort vs MySQL

Sid Anand, LinkedIn Data Infrastructure (QCon London 2012)
Column Families – BigTable like

Sparse, distributed, persistent multi-dimensional sorted map indexed by \((row\_key, column\_key, timestamp)\)
BigTable Data Model

The row name is a reversed URL. The contents column family contains the page contents, and the anchor column family contains the text of any anchors that reference the page.
BigTable Performance

![Graph showing performance of BigTable with different operations and numbers of tablet servers.](image-url)
# Document Database - MongoDB

## Table in relational db

<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUMONT</td>
<td>Jean</td>
<td>43</td>
</tr>
<tr>
<td>PELLERIN</td>
<td>Franck</td>
<td>29</td>
</tr>
<tr>
<td>MATTHIEU</td>
<td>Nicolas</td>
<td>51</td>
</tr>
</tbody>
</table>

## Documents in a collection

```json
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc9"),
   "Last Name": "DUMONT",
   "First Name": "Jean",
   "Age": 43
},
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
   "Last Name": "PELLERIN",
   "First Name": "Franck",
   "Age": 29,
   "Address": "1 chemin des Loges",
   "City": "VERSAILLES"
}
```

## Open source, document db

Json-like document with dynamic schema

## Initial release 2009
mongoDB Product Deployment

And much more...
Graph Database

Data Model Abstraction:
• Nodes
• Relations
• Properties
Neo4j - Build a Graph

```java
NeoService neo = ... // Get factory

// Create Thomas 'Neo' Anderson
Node mrAnderson = neo.createNode();
mrAnderson.setProperty("name", "Thomas Anderson");
mrAnderson.setProperty("age", 29);

// Create Morpheus
Node morpheus = neo.createNode();
morpheus.setProperty("name", "Morpheus");
morpheus.setProperty("rank", "Captain");
morpheus.setProperty("occupation", "Total bad ass");

// Create a relationship representing that they know each other
mrAnderson.createRelationshipTo(morpheus, RelTypes.KNOWS);
// ...create Trinity, Cypher, Agent Smith, Architect similarly
```

Slide from neo technology, “A NoSQL Overview and the Benefits of Graph Databases"
A Debatable Performance Evaluation

Got neo4j to do a do a lookup in 2 seconds, that sql server did in 45 minutes. neo4j rocks!

6:28 AM Jun 30th from web

turboCodr
John Conwell
Conclusion

- Use the right data model for the right problem
MAP-REDUCE AND HADOOP
MapReduce

- Much of the course will be devoted to large scale computing for data mining

- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard

- **Map-reduce** addresses all of the above
  - Google’s computational/data manipulation model
  - Elegant way to work with big data
Single Node Architecture

Machine Learning, Statistics

“Classical” Data Mining
Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to do something useful with the data!
- Today, a standard architecture for such problems is emerging:
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them
Cluster Architecture

2-10 Gbps backbone between racks

1 Gbps between any pair of nodes in a rack

Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, [http://bit.ly/Shh0RO](http://bit.ly/Shh0RO)
Large-scale Computing

- Large-scale computing for data mining problems on commodity hardware

- Challenges:
  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - Machines fail:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to lose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!
Idea and Solution

- **Issue:** Copying data over a network takes time

- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability

- **Map-reduce addresses these problems**
  - Google’s computational/data manipulation model
  - Elegant way to work with big data
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS
  - **Programming model**
    - Map-Reduce
Hadoop VS NoSQL

- **Hadoop**: computing framework
  - Supports data-intensive applications
  - Includes MapReduce, HDFS etc.
    (we will study MR mainly next)

- **NoSQL**: Not only SQL databases
  - Can be built ON hadoop. E.g. HBase.
Problem:
- If nodes fail, how to store data persistently?

Answer:
- Distributed File System:
  - Provides global file namespace
  - Google GFS; Hadoop HDFS;

Typical usage pattern
- Huge files (100s of GB to TB)
- Data is rarely updated in place
- Reads and appends are common
Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks

- **Master node**
  - a.k.a. Name Node in Hadoop’s HDFS
  - Stores metadata about where files are stored
  - Might be replicated

- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data
Distributed File System

- Reliable distributed file system
- Data kept in “chunks” spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure

Bring computation directly to the data!

Chunk servers also serve as compute servers

Prakash 2014
Programming Model: MapReduce

**Warm-up task:**

- We have a huge text document
- Count the number of times each distinct word appears in the file

**Sample application:**

- Analyze web server logs to find popular URLs
Task: Word Count

Case 1:
– File too large for memory, but all <word, count> pairs fit in memory

Case 2:

□ Count occurrences of words:
– `words(doc.txt) | sort | uniq -c`
  • where `words` takes a file and outputs the words in it, one per a line

□ Case 2 captures the essence of **MapReduce**
– Great thing is that it is naturally parallelizable
MapReduce: Overview

- Sequentially read a lot of data
- **Map:**
  - Extract something you care about
- **Group by key:** Sort and Shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, Map and Reduce change to fit the problem
MapReduce: The **Map** Step

**Input** key-value pairs:

```
\( k \quad v \)
\( k \quad v \)
\( ... \)
\( k \quad v \)
```

**Intermediate** key-value pairs:

```
\( k \quad v \)
\( k \quad v \)
\( ... \)
```

MapReduce is a programming model for processing large datasets in a distributed environment. The **Map** step processes input data by applying a function to each key-value pair, generating intermediate key-value pairs. The **Reduce** step then aggregates these intermediate results to produce the final output.
MapReduce: The **Reduce** Step

**Intermediate key-value pairs**

- \( k \)
- \( v \)
- ...  
- \( k \)
- \( v \)
- \( k \)
- \( v \)

**Group by key**

**Key-value groups**

- \( k \)
- \( v \)
- \( v \)
- \( v \)
- ...  
- \( k \)
- \( v \)
- \( v \)

**Reduce**

**Output key-value pairs**

- \( k \)
- \( v \)
- \( k \)
- \( v \)
- ...  
- \( k \)
- \( v \)
More Specifically

- **Input:** a set of key-value pairs
- **Programmer specifies two methods:**
  - **Map**($k, v$) $\rightarrow <k', v'>*$
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every $(k,v)$ pair
  - **Reduce**($k'$, $<v'>*$) $\rightarrow <k', v''>$*
    - All values $v'$ with same key $k'$ are reduced together and processed in $v'$ order
    - There is one Reduce function call per unique key $k'$
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing -- is what we're going to need."
Word Count Using MapReduce

map(key, value):
// key: document name; value: text of the document
for each word w in value:
    emit(w, 1)

reduce(key, values):
// key: a word; value: an iterator over counts
result = 0
for each count v in values:
    result += v
emit(key, result)
Map-Reduce (MR) as SQL

- select `count(*)`  
  from DOCUMENT 
  group by word

Reducer

Mapper
Map-Reduce: Environment

Map-Reduce environment takes care of:

- Partitioning the input data
- Scheduling the program’s execution across a set of machines
- Performing the group by key step
- Handling machine failures
- Managing required inter-machine communication
**Map-Reduce: A diagram**

**MAP:**
Read input and produces a set of key-value pairs

**Intermediate**

**Group by key:**
Collect all pairs with same key (Hash merge, Shuffle, Sort, Partition)

**Reduce:**
Collect all values belonging to the key and output

**Input**
Big document

**Grouped**
k1:v, v, v, v, k2:v, k3:v, v, k4:v, v, v, k5:v

**Output**
Map-Reduce: In Parallel

All phases are distributed with many tasks doing the work
Map-Reduce

- **Programmer specifies:**
  - Map and Reduce and input files

- **Workflow:**
  - Read inputs as a set of key-value-pairs
  - Map transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k'v'-pairs with a given k’ are sent to the same reduce
  - Reduce processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files

- **All phases are distributed with many tasks doing the work**
Data Flow

- **Input and final output** are stored on a distributed file system (FS):
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data

- **Intermediate results** are stored on local FS of Map and Reduce workers

- **Output** is often input to another MapReduce task
Coordination: Master

- **Master node takes care of coordination:**
  - **Task status:** (idle, in-progress, completed)
  - **Idle tasks** get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its $R$ intermediate files, one for each reducer
  - Master pushes this info to reducers

- Master pings workers periodically to detect failures
Dealing with Failures

- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker

- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted

- **Master failure**
  - MapReduce task is aborted and client is notified
How many Map and Reduce jobs?

- $M$ map tasks, $R$ reduce tasks
- **Rule of a thumb:**
  - Make $M$ much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
- **Usually $R$ is smaller than $M$**
  - Because output is spread across $R$ files
Task Granularity & Pipelining

- **Fine granularity tasks**: map tasks >> machines
  - Minimizes time for fault recovery
  - Can do pipeline shuffling with map execution
  - Better dynamic load balancing

![Graphical representation of Task Granularity and Pipelining](image)
Refinements: Backup Tasks

- **Problem**
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things

- **Solution**
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first “wins”

- **Effect**
  - Dramatically shortens job completion time
Refinement: Combiners

- Often a Map task will produce many pairs of the form \((k, v_1), (k, v_2), \ldots\) for the same key \(k\)
  - E.g., popular words in the word count example

- Can save network time by pre-aggregating values in the mapper:
  - \(\text{combine}(k, \text{list}(v_1)) \rightarrow v_2\)
  - Combiner is usually same as the reduce function

- Works only if reduce function is commutative and associative
**Refinement: Combiners**

- **Back to our word counting example:**
  - Combiner combines the values of all keys of a single mapper (single machine):

  - Much less data needs to be copied and shuffled!
Refinement: Partition Function

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker

- **System uses a default partition function:**
  - \( \text{hash(key)} \mod R \)

- **Sometimes useful to override the hash function:**
  - E.g., \( \text{hash(hostname(URL))} \mod R \) ensures URLs from a host end up in the same output file
PROBLEMS SUITED FOR MAP-REDUCE
Example: Host size

- Suppose we have a large web corpus
- Look at the metadata file
  - Lines of the form: (URL, size, date, ...)
- For each host, find the total number of bytes
  - That is, the sum of the page sizes for all URLs from that particular host

- Other examples:
  - Link analysis and graph processing
  - Machine Learning algorithms
Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents

- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts
Example: Join By Map-Reduce

- Compute the natural join $R(A,B) \bowtie S(B,C)$
- $R$ and $S$ are each stored in files
- Tuples are pairs $(a,b)$ or $(b,c)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_3$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>$a_4$</td>
<td>$b_3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_2$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$b_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$b_3$</td>
<td>$c_3$</td>
</tr>
</tbody>
</table>

$\bowtie$

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_3$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_3$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$a_4$</td>
<td>$c_3$</td>
</tr>
</tbody>
</table>
Map-Reduce Join

- Use a hash function $h$ from B-values to $1...k$

- **A Map process turns:**
  - Each input tuple $R(a,b)$ into key-value pair $(b,(a,R))$
  - Each input tuple $S(b,c)$ into $(b,(c,S))$

- **Map processes** send each key-value pair with key $b$ to Reduce process $h(b)$
  - Hadoop does this automatically; just tell it what $k$ is.

- Each **Reduce process** matches all the pairs $(b,(a,R))$ with all $(b,(c,S))$ and outputs $(a,b,c)$. 
Cost Measures for Algorithms

In MapReduce we quantify the cost of an algorithm using

1. **Communication cost** = total I/O of all processes
2. **Elapsed communication cost** = max of I/O along any path
3. (**Elapsed**) **computation cost** analogous, but count only running time of processes

Note that here the big-O notation is not the most useful (adding more machines is always an option)
Example: Cost Measures

- For a map-reduce algorithm:
  - Communication cost = input file size + 2 × (sum of the sizes of all files passed from Map processes to Reduce processes) + the sum of the output sizes of the Reduce processes.
  - Elapsed communication cost is the sum of the largest input + output for any map process, plus the same for any reduce process.
Either the I/O (communication) or processing (computation) cost dominates
   – Ignore one or the other

Total cost tells what you pay in rent from your friendly neighborhood cloud (eg. AWS in HW3)

Elapsed cost is wall-clock time using parallelism
Cost of Map-Reduce Join

- **Total communication cost**
  \[= O(|R| + |S| + |R \bowtie S|)\]

- **Elapsed communication cost**
  \[= O(s)\]
  - We’re going to pick \(k\) and the number of Map processes so that the I/O limit \(s\) is respected
  - We put a limit \(s\) on the amount of input or output that any one process can have. \(s\) could be:
    - What fits in main memory
    - What fits on local disk

- With proper indexes, computation cost is linear in the input + output size
  - So computation cost is like comm. cost
Conclusions

- Hadoop is a distributed data-intensive computing framework
- MapReduce
  - Simple programming paradigm
  - Surprisingly powerful (may not be suitable for all tasks though)
- Hadoop has specialized FileSystem, Master-Slave Architecture to scale-up
NoSQL and Hadoop

- Hot area with several new problems
  - Good for academic research
  - Good for industry

= Fun AND Profit 😊
POINTERS AND FURTHER READING
Implementations

- **Google**
  - Not available outside Google

- **Hadoop**
  - An open-source implementation in Java
  - Uses HDFS for stable storage

- **Aster Data**
  - Cluster-optimized SQL Database that also implements MapReduce
Cloud Computing

- Ability to rent computing by the hour
  - Additional services e.g., persistent storage

- Amazon’s “Elastic Compute Cloud” (EC2)

- Aster Data and Hadoop can both be run on EC2
Reading

- Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
Resources

- **Hadoop Wiki**
  - Introduction
  - Getting Started
  - Map/Reduce Overview
    - [http://wiki.apache.org/lucene-hadoop/HadoopMapReduce](http://wiki.apache.org/lucene-hadoop/HadoopMapReduce)
    - [http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses](http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses)
  - Eclipse Environment

- **Javadoc**
  - [http://lucene.apache.org/hadoop/docs/api/](http://lucene.apache.org/hadoop/docs/api/)
Resources

- Releases from Apache download mirrors

- Nightly builds of source

- Source code from subversion
  - [http://lucene.apache.org/hadoop/version_control.html](http://lucene.apache.org/hadoop/version_control.html)
Further Reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]