CS 102
Big Data
Fall 2017

Big Data Platforms & Services
Infrastructure
100 MW per data center

~ 1-10 power plants

~ 80,000 US households
1 Pbits/s

global bandwidth (2015)

~ 90 billion Shakespeares hourly
180 teraflops per TPU2 board

~ 15-30x faster than CPU/GPU
~ 130 Xbox

11.5 petaflops per pod

~ 10% of top supercomputer
Relational Databases

- Application
- Front end (SQL)
- Query processor
- Transaction processor
- File system

M

P ...

P

...
Relational Databases

Strengths
Declarative query language (SQL)
Query optimization
Transactions
  Atomicity, consistency, isolation, durability

Challenges at scale
Concurrency control
Reliability
Replication
BigTable

Basic idea: key-value store

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>34M</td>
</tr>
<tr>
<td>France</td>
<td>64M</td>
</tr>
<tr>
<td>Germany</td>
<td>82M</td>
</tr>
<tr>
<td>USA</td>
<td>307M</td>
</tr>
</tbody>
</table>

lookup(key) → value
scan(key range) → values
insert(key, value)
delete(key)
# BigTable

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
<th>Tablet server 1</th>
<th>Tablet server 2</th>
<th>Tablet server 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$v_1$</td>
<td>$k_1$</td>
<td>$v_1$</td>
<td>$k_5$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$v_2$</td>
<td>$k_2$</td>
<td>$v_2$</td>
<td>$k_6$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$v_3$</td>
<td>$k_3$</td>
<td>$v_3$</td>
<td>$k_7$</td>
</tr>
<tr>
<td>$k_4$</td>
<td>$v_4$</td>
<td>$k_4$</td>
<td>$v_4$</td>
<td>$k_8$</td>
</tr>
<tr>
<td>$k_5$</td>
<td>$v_5$</td>
<td>$k_5$</td>
<td>$v_5$</td>
<td>$k_9$</td>
</tr>
</tbody>
</table>
Spanner

= BigTable
  + transactions
  + global replication
Comparison

**SQL** Relational databases
Relations, queries, optimization, transactions
Costly to scale

**NoSQL** BigTable
Scale
No data structure, queries, optimization, (multi-row) transactions

**NewSQL** Spanner
Scale
Transactions
Global replication
Processing
MapReduce

Original application: building a text index

Loading

Tokenizing

Sorting

Flushing

Disk

Intermediate runs

Page stream

ant
dog
1

ant
dog
2

ant
dog
3

dog
cat

(ant, 1)
(dog, 1)

(ant, 3)
(dog, 3)

(dog, 2)

(cat, 2)

(dog, 2)

(dog, 3)

ant
dog

(ant, 1)
(dog, 3)

(dog, 1)

(dog, 2)

(dog, 3)

1

2

3
MapReduce

Intermediate runs

( ant, 1 )
( ant, 3 )
( cat, 2 )
( dog, 1 )
( dog, 2 )
( dog, 3 )
( ant, 5 )
( cat, 4 )
( dog, 4 )
( dog, 5 )
( eel, 6 )

Merge

( ant, 1 )
( ant, 3 )
( ant, 5 )
( cat, 2 )
( cat, 4 )
( dog, 1 )
( dog, 2 )
( dog, 3 )
( dog, 4 )
( dog, 5 )
( eel, 6 )

Final index

( ant: 1, 3, 5 )
( cat: 2, 4 )
( dog: 1, 2, 3, 4, 5 )
( eel: 6 )
MapReduce

- **Page stream**
- **Loading**
  - Sample data:
    - ant
    - dog
    - cat
- **Tokenizing**
  - Results:
    - (ant, 1)
    - (dog, 1)
    - (cat, 2)
    - (ant, 3)
    - (dog, 2)
    - (dog, 3)
- **Sorting**
  - Results:
    - (ant, 1)
    - (ant, 3)
    - (cat, 2)
    - (dog, 1)
    - (dog, 2)
    - (dog, 3)
- **Intermediate runs**
- **Disk**
- **Flushing**
MapReduce

Intermediate runs

( ant, 1 )
( ant, 3 )
( cat, 2 )
( dog, 1 )
( dog, 2 )
( dog, 3 )
( ant, 5 )
( cat, 4 )
( dog, 4 )
( dog, 5 )
( eel, 6 )

Merge

( ant, 1 )
( ant, 3 )
( ant, 5 )
( cat, 2 )
( cat, 4 )
( dog, 1 )
( dog, 2 )
( dog, 3 )
( dog, 4 )
( dog, 5 )
( eel, 6 )

Reduce

( ant: 1, 3, 5 )
( cat: 2, 4 )
( dog: 1, 2, 3, 4, 5 )
( eel: 6 )

Final index
MapReduce

Framework

Map → Reduce

Map → Reduce

Map → Reduce
MapReduce

Example: counting word occurrences

map(String docId, String docBody):
    for each w /* word */ in docBody
        emitIntermediate(w, 1)

E.g., map(42, “cat dog cat bat dog”) emits
[“cat”, 1], [“dog”, 1], [“cat”, 1],
[“bat”, 1], [“dog”, 1]
MapReduce

Example: counting word occurrences

reduce(String key, Iterator values):
  int result = 0
  for each v in values result += v
  emit(result)

E.g., reduce("dog", { 1, 1, 1, 1 }) emits 4
Spark

MapReduce limitations

Complex programming
  Real applications require multiple (many!) MR steps
  Lots of boilerplate code

Rigid / inefficient model
  Only map, shuffle, reduce data transformation primitives
  Fixed execution sequence
  Heavy use of disks
Spark

Improvements

Efficiency → 2-100x faster
  General computation graph
  Rich set of data transformation primitives
  In-memory processing

Usability → 5x less code
  Rich core APIs
  Interactive shell

<table>
<thead>
<tr>
<th>SQL</th>
<th>Streaming</th>
<th>ML</th>
<th>GraphX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark Core API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Python</td>
<td>Scala</td>
<td>Java</td>
</tr>
</tbody>
</table>
Spark

Generalized distributed data flow

Resilient distributed data sets (RDDs)
  Fault-tolerant collection of elements for parallel processing
Transformations
  Lazy execution (not computed immediately)
  E.g., map, reduce, filter, sample, distinct, join, union
Actions
  Trigger computations
  E.g., collect, count, save as text file
Spark

Generalized distributed data flow
Example: counting word occurrences

```python
spark = SparkSession\ 
    .builder\ 
    .appName("PythonWordCount")\ 
    .getOrCreate()

lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
counts = lines.flatMap(lambda x: x.split(' ')) \ 
    .map(lambda x: (x, 1)) \ 
    .reduceByKey(add)
output = counts.collect()
for (word, count) in output:
    print("%s: %i" % (word, count))

spark.stop()
```
BigQuery

Data warehouse for analytics
SQL-like queries on read-only data

Idea #1
Column store

Vertical partitioning

Horizontal partitioning
Idea #2
Hierarchical execution

Client

Root server

Intermediate servers

Leaf servers

Local storage
BigQuery

```
select sum(A) from T
```

```
select sum(A) from T_A
```

```
select sum(A) from T_{A1}
```

```
select sum(A) from T_{A2}
```

```
select sum(A) from T_{B1}
```

```
select sum(A) from T_{B2}
```

```
```
```
```
```
```
TensorFlow

Software library for numerical computation | machine learning

Low-level core API
Complete programming control (similar to Spark)

High-level APIs
E.g., tf.contrib.learn for ML training and evaluation
Platforms
Cloud Computing

“The fight to dominate cloud computing will increase competition and innovation”
(Battle of the clouds, October 2009)

“Tech giants are waging a price war to win other firms’ computing business”
(Silver lining, August 2014)

“As cloud-computing prices keep falling, the whole IT business will change”
(The cheap, convenient cloud, April 2015)
Cloud Computing

What is it?
On-demand access to shared computing resources
Infrastructure/platform/software as a service
Hardware virtualization
Business model (focus on business, not infra, differentiators)
Cloud Platforms

Amazon Web Services
Google Cloud
Microsoft Azure

Application  Big data
Compute
Virtual machine
Network
File system
Storage
Compute and Application Engines

Virtual machines
1-32 CPUs, 1-200 GB RAM
0-3 TB local HDD or SSD
0-64 TB network storage
Linux or Windows

Networking
Virtual networks
Load balancing
Domain name system
Storage

Basic storage
Distributed files

Relational database
E.g., MySQL instances (16 cores, 100 GB RAM, read replicas)

Distributed data store
E.g., BigTable (hundreds of PBs, millions of ops/second)
Big Data

Procedural and semi-declarative processing
MapReduce, Spark

Declarative queries
SQL-like query language, e.g., BigQuery
Integrations with, e.g., Tableau visualization

Interactive processing
E.g., on top of Jupyter (Python, SQL, JavaScript)
Other Services

Machine learning
Libraries, e.g., TensorFlow
Vision, speech recognition, translation APIs

Tools
Management
Development
Identity and security
Summary

Infrastructure
Storage
Processing
Cloud platforms

Questions?