RECAP: NOSQL

NoSQL vs. Relational Databases

What are the big differences between relational databases and NoSQL systems?
What are the trade-offs?

RECAP: KEY–VALUE
Key–Value = a Distributed Map

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>country:Afghanistan</td>
<td>capital:city:Kabul,continent:Asia,pop:31108077#2011</td>
</tr>
<tr>
<td>country:Albania</td>
<td>capital:city:Tirana,continent:Europe,pop:30114054#2013</td>
</tr>
<tr>
<td>city:Kabul</td>
<td>country:Afghanistan,pop:3476000#2013</td>
</tr>
<tr>
<td>city:Tirana</td>
<td>country:Albania,pop:30114054#2013</td>
</tr>
<tr>
<td>user:10239</td>
<td>basedIn:city:Tirana,post:{103,10430,201}</td>
</tr>
</tbody>
</table>

Amazon Dynamo(DB): Model

- Named table with primary key and a value

<table>
<thead>
<tr>
<th>Countries</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>capital:Kabul,continent:Asia,pop:31108077#2011</td>
</tr>
<tr>
<td>Albania</td>
<td>capital:Tirana,continent:Europe,pop:30114054#2013</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cities</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kabul</td>
<td>country:Afghanistan,pop:3476000#2013</td>
</tr>
<tr>
<td>Tirana</td>
<td>country:Albania,pop:30114054#2013</td>
</tr>
</tbody>
</table>

Amazon Dynamo(DB): Object Versioning

- Object Versioning (per bucket)
  - PUT doesn’t overwrite: pushes version
  - GET returns most recent version

Other Key–Value Stores

RECAP: DOCUMENT STORES

- Document-type depends on store
  - XML, JSON, Blobs, Natural language
- Operators for documents
  - e.g., filtering, inv. indexing, XML/JSON querying, etc.
**MongoDB: JSON Based**

- Can invoke Javascript over the JSON objects
- Document fields can be indexed

```javascript
db.inventory.find({ continent: { $in: [ 'Asia', 'Europe' ] }})
```

**Document Stores**

**Key–Value = a Distributed Map**

<table>
<thead>
<tr>
<th>Countries</th>
<th>Primary Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>capital:Kabul,continent:Asia,pop:31108072011</td>
<td></td>
</tr>
<tr>
<td>Albania</td>
<td>capital:Tirana,continent:Europe,pop:30114052013</td>
<td></td>
</tr>
</tbody>
</table>

**Tabular = Multi-dimensional Maps**

<table>
<thead>
<tr>
<th>Countries</th>
<th>Primary Key</th>
<th>capital</th>
<th>continent</th>
<th>pop-value</th>
<th>pop-year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>Kabul</td>
<td>Asia</td>
<td>3110807</td>
<td>2011</td>
<td></td>
</tr>
<tr>
<td>Albania</td>
<td>Tirana</td>
<td>Europe</td>
<td>3011405</td>
<td>2013</td>
<td></td>
</tr>
</tbody>
</table>

**Bigtable: The Original Whitepaper**

Why did they write another paper? MapReduce solves everything, right?

Bigtable: A Distributed Storage System for Structured Data

**Bigtable used for ...**
Bigtable: Data Model

“**a sparse, distributed, persistent, multi-dimensional, sorted map.**”

- **sparse**: not all values form a dense square
- **distributed**: lots of machines
- **persistent**: disk storage (GFS)
- **multi-dimensional**: values with columns
- **sorted**: sorting lexicographically by row key
- **map**: look up a key, get a value

Bigtable: in a nutshell

**(row, column, time)** → value

- **row**: a row id string
  - e.g., “Afganistan”
- **column**: a column name string
  - e.g., “pop-value”
- **time**: an integer (64-bit) version time-stamp
  - e.g., 18545664
- **value**: the element of the cell
  - e.g., “31120978”

Bigtable: in a nutshell

**(row, column, time)** → value

**(Afganistan, pop-value, t4)** → 31108077

Bigtable: Sorted Keys

<table>
<thead>
<tr>
<th>Primary Key</th>
<th>capital</th>
<th>pop-value</th>
<th>pop-year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia:Afghanistan</td>
<td>Kabul</td>
<td>31143292</td>
<td>t1 2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31120978</td>
<td>t2 2011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31108077</td>
<td>t4 2011</td>
</tr>
<tr>
<td>Asia:Azerbaijan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Europe:Albania</td>
<td>Tirana</td>
<td>2912380</td>
<td>t1 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3011405</td>
<td>t3 2013</td>
</tr>
<tr>
<td>Europe:Andorra</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Benefits of sorted keys vs. hashed keys?

Bigtable: Tablets

<table>
<thead>
<tr>
<th>Primary Key</th>
<th>capital</th>
<th>pop-value</th>
<th>pop-year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia:Afghanistan</td>
<td>Kabul</td>
<td>31143292</td>
<td>t1 2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31120978</td>
<td>t2 2011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31108077</td>
<td>t4 2011</td>
</tr>
<tr>
<td>Asia:Azerbaijan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Europe:Albania</td>
<td>Tirana</td>
<td>2912380</td>
<td>t1 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3011405</td>
<td>t3 2013</td>
</tr>
<tr>
<td>Europe:Andorra</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Take advantage of locality of processing!

Bigtable: Distribution

Pros and cons versus hash partitioning?

Horizontal range partitioning
Bigtable: Column Families

- Group logically similar columns together
  - Accessed efficiently together
  - Access-control and storage: column family level
  - If of same type, can be compressed

<table>
<thead>
<tr>
<th>Primary Key</th>
<th>Capital</th>
<th>pop-value</th>
<th>pop-year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia:Afghanistan</td>
<td>Kabul</td>
<td>3114292</td>
<td>t1 2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3122078</td>
<td>t2 2011</td>
</tr>
<tr>
<td>Asia:Azerbaijan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Europe:Albania</td>
<td>Tirana</td>
<td>2912388</td>
<td>t1 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3014005</td>
<td>t2 2013</td>
</tr>
<tr>
<td>Europe:Andorra</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bigtable: Versioning

- Similar to Apache Dynamo (so no “fancy” slide)
  - Cell-level
  - 64-bit integer time stamps
  - Inserts push down current version
  - Lazy deletions / periodic garbage collection
  - Two options:
    - keep last \( n \) versions
    - keep versions newer than \( t \) time

Bigtable: SSTable Map Implementation

- 64k blocks (default) with index in footer (GFS)
- Index loaded into memory, allows for seeks
- Can be split or merged, as needed

<table>
<thead>
<tr>
<th>Primary Key</th>
<th>Capital</th>
<th>pop-value</th>
<th>pop-year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia:Afghanistan</td>
<td>Kabul</td>
<td>3114292</td>
<td>t1 2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3122078</td>
<td>t2 2011</td>
</tr>
<tr>
<td>Asia:Azerbaijan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Europe:Albania</td>
<td>Tirana</td>
<td>2912388</td>
<td>t1 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3014005</td>
<td>t2 2013</td>
</tr>
<tr>
<td>Europe:Andorra</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bigtable: Buffered/Batched Writes

- How to handle writes?

Bigtable: Redo Log

- If machine fails, Memtable redone from log

Bigtable: Minor Compaction

- When full, write Memtable as SSTable
**Bigtable: Merge Compaction**

- Merge *some of the SSTables* (and the Memtable)

**Bigtable: Major Compaction**

- Merge *all SSTables* (and the Memtable)
- Makes reads more efficient!

**Bigtable: Hierarchical Structure**

- **CHUBBY**: Distributed consensus tool based on PAXOS
  - Maintains consistent replicas
  - Five replicas: one master and four slaves
  - Co-ordinates distributed locks
  - Stores location of main “root tablet”

**Bigtable: Consistency**

- **CHUBBY**: Distributed consensus tool based on PAXOS
  - Maintains consistent replicas
  - Five replicas: one master and four slaves
  - Co-ordinates distributed locks
  - Stores location of main “root tablet”

**Bigtable: A Bunch of Other Things**

- **Locality groups**: Group multiple column families together; assigned a separate SSTable
- **Select storage**: SSTables can be persistent or in-memory
- **Compression**: Applied on SSTable blocks; custom compression can be chosen
- **Caches**: SSTable-level and block-level
- **Bloom filters**: Find negatives cheaply ...

**Aside: Bloom Filter**

- Create a bit array of length $m$ (init to 0’s)
- Create $k$ hash functions that map an object to an index of $m$ (even distribution)
- **Index $o$**: set $m[\text{hash}_i(o)]$ to 1
- **Query $o$**:
  - any $m[\text{hash}_i(o)] = 0$ = not indexed
  - all $m[\text{hash}_i(o)] = 1$ = might be indexed
Bigtable: an idea of performance

• Values are 1 kilobyte in size
• Results from 2006 paper

Why are random (disk) reads so slow?

The read sizes are 1 kb, but a different 64 kb block must be sent over the network (almost) every time.

Bigtable: examples in Google (2006)

<table>
<thead>
<tr>
<th>Project name</th>
<th>Table size (TB)</th>
<th>Compatibility (rows/columns)</th>
<th># of columns in family</th>
<th>Poolable</th>
<th># of index families</th>
<th># of users</th>
<th>In memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud</td>
<td>90</td>
<td>70% 80%</td>
<td>200</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>No</td>
</tr>
<tr>
<td>Google Search</td>
<td>90</td>
<td>70% 80%</td>
<td>200</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>Yes</td>
</tr>
<tr>
<td>Google Talk</td>
<td>90</td>
<td>70% 80%</td>
<td>200</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>No</td>
</tr>
<tr>
<td>Globel</td>
<td>90</td>
<td>70% 80%</td>
<td>200</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>No</td>
</tr>
<tr>
<td>Bigtable</td>
<td>90</td>
<td>70% 80%</td>
<td>200</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>No</td>
</tr>
</tbody>
</table>

Bigtable: Apache HBase

Open-source implementation of Bigtable ideas

The Database Landscape

GRAPH DATABASES
Any data can be represented as a directed labelled graph (not always neatly).

When is it a good idea to consider data as a graph?
- When you want to answer questions like:
  - How many social hops is this user away?
  - What is my Erdős number?
  - What connections are needed to fly to Perth?
  - How are Einstein and Godel related?
Graph Databases: Index Nodes

- Fred -> (Fred, IS_FRIEND_OF, Jim)
  (Jim, LIKES, iSushi)

Graph Databases: Index Relations

- LIKES -> (Ted, LIKES, Zushi_Zam)
  (Jim, LIKES, iSushi)

Graph Databases: Graph Queries

- (Fred, IS_FRIEND, ?friend)
- (?friend, LIKES, ?place)
- (?place, SERVES, ?sushi)
- (?place, LOCATED_IN, New_York)

Graph Databases: Path Queries

- (Fred, IS_FRIEND*, ?friend_of_friend)
  (?friend_of_friend, LIKES, Zushi_Zam)

What about scalability?
Graph Database: Index-free Adjacency

Fred IS_FRIEND_OF Ted
Fred IS_FRIEND_OF Jim
Jim LIKES iSushi
iSushi SERVES Sushi
iSushi LOCATED_IN New York
Ted LIKES Zushi Zam

Fred IS_FRIEND_OF Ted
Fred IS_FRIEND_OF Jim
Jim LIKES iSushi
iSushi SERVES Sushi
iSushi LOCATED_IN New York
Ted LIKES Zushi Zam

Leading Graph Database

Leading Graph Database

http://db-engines.com/en/ranking

SPARQL

http://db-engines.com/en/ranking

RECAP
Recap
- Relational Databases don’t solve everything
  - SQL and ACID add overhead
  - Distribution not so easy
- NoSQL: what if you don’t need SQL or ACID?
  - Something simpler
  - Something more scalable
  - Trade efficiency against guarantees

NoSQL: Trade-offs
- Simplified transactions (no ACID)
- Simplified (or no) query language
  - Procedural or a subset of SQL
- Simplified query algebra
  - Often no joins
- Simplified data model
  - Often map-based
- Simplified replication
  - Consistency vs. Availability

Simplifications enable scale to thousands of machines. But a lot of relational database features are lost!

NoSQL Overview Map

Types of NoSQL Store
- Key-Value Stores (e.g., Dynamo):
  - Distributed unsorted maps
  - Some have secondary indexes
- Document Stores (e.g., MongoDB):
  - Map values are documents (e.g., JSON, XML)
  - Built-in document functions/indexable fields
- Table/Column-Based Stores (e.g., Bigtable):
  - Distributed multi-dimensional sorted maps
  - Distribution by Tablets/Column-families
- Graph Stores (e.g., Neo4J)
  - Stores vertices and relations: Index-free adjacency
  - Query languages for paths, reachability, etc.
- Hybrid/mix/other (e.g., Cassandra)

Categories are far from clean: aside from graph stores, most NoSQL stores are just fancy (sometimes sorted) maps basically.

Bigtable
- Column family store: (row, column, time) → value
- Sorted map, range partitioned
- PAXOS for locks, root table
- Tablets: horizontal table splits
- Column family: logical grouping of columns stored close together
- Locality groups: grouping of column families
- SSTable: sequence of 64k blocks
- Batch writes
- Compactions: merge SSTables

Questions
Schedule

• No evaluated activities allowed this week
  – No task deadline either
• Current week 11? Semester continues until week 15?
• Rough plan for rest of course:
  – Week 11 Wednesday: "open lab"
  – Week 12 Monday: Projects in Lab
    • Week 12 Tuesday: Lab 8 & 9 due
  – Week 12 Wednesday: Projects in Lab
  – Week 13 Monday: Project Reports Due, Presentations Given
  – Week 13 Wednesday: HBase lab
  – Week 14 Monday: Graph data lecture
  – Week 14 Wednesday: Unmarked lab
  – Week 15 Monday: Wrap-up, exam preparation
  – Week 15 Wednesday: Not sure really 😊