

Databases 101

Rayid Ghani



Slides liberally borrowed and customized from lots of excellent online sources

What we'll cover

- Intro to Databases
- Types of Databases
 - Features
 - Pros and Cons
- Working with databases
 - Getting data in (Joe already covered)
 - Analysis/Querying – SQL
 - Getting data out

Why do we need Databases?

- Store data
- Organize data
- Use data efficiently

History of Databases

- Punch Cards
- Flat Files
- Relational
- NoSQL

Relational Databases

- Represent relations
- Traditionally motivated by the need for transaction processing and analysis
- Use SQL for querying
- Typically normalized but more and more denormalized for analytical reasons

Data models

- A **data model** is a collection of concepts for describing data
 - The relational model of data is the most widely used model today
- A **schema** is a description of a particular collection of data, **using the given data model**
 - E.g. every *relation* in a relational data model has a *schema* describing types, etc.

SQL Database Examples

- Commercial
 - Microsoft SQL Server
 - Oracle
 - IBM DB2
 - Teradata
 - Sybase SQL Anywhere
 - ...
- Open Source (with commercial options)
 - Sqlite
 - MySQL
 - Postgres
 - Ingres

Transactions – ACID Properties

- **Atomic** – All of the work in a transaction completes (commit) or none of it completes
- **Consistent** – A transaction transforms the database from one consistent state to another consistent state.
Consistency is defined in terms of constraints.
- **Isolated** – The results of any changes made during a transaction are not visible until the transaction has committed.
- **Durable** – The results of a committed transaction survive failures

Designing a Relational Database

- Entities (keys) and attributes
- Relationships (between entities) and attributes
- Constraints

Why NoSQL?

- Initially motivated by web applications
- Hack was to front a relational DB with a cache for reading and writing
- Scaling Issues
 - Master slave is slow and expensive
 - Sharding is not ubiquitous and joins don't work

What is NoSQL?

- Stands for **Not Only SQL** or **Not SQL** (people argue about this all the time)
- Class of non-relational data storage systems
- “Usually” do not require a fixed table schema nor do they use the concept of joins
- **All NoSQL dbs relax one or more of the ACID properties (CAP theorem)**

CAP Theorem

- Three properties of a system: consistency, availability and partitions
- You can have at most two of these three properties for any shared-data system
- To scale out, you have to partition. That leaves either consistency or availability to choose from
 - In almost all cases, you would choose availability over consistency

Availability

- Traditionally, thought of as the server/process available five 9's (99.999 %).
- However, for large node system, at almost any point in time there's a good chance that a node is either down or there is a network disruption among the nodes.
 - Want a system that is resilient in the face of network disruption

Consistency Model

- A consistency model determines rules for visibility and apparent order of updates.
- For example:
 - Row X is replicated on nodes M and N
 - Client A writes row X to node N
 - Some period of time t elapses.
 - Client B reads row X from node M
 - Does client B see the write from client A?
 - Consistency is a continuum with tradeoffs
 - For NoSQL, the answer would be: maybe
 - CAP Theorem states: Strict Consistency can't be achieved at the same time as availability and partition-tolerance.

Eventual Consistency

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- Known as **BASE** (**B**asically **A**vailable, **S**oft state, **E**ventual consistency), as opposed to ACID

When to NoSQL

- Unknown/flexible schema
- Bursty usage (easy scaling)
- Simple-ish queries
- Fast read access

NoSQL Database Types

- Key-Value Pair
- Document Database
- Column Database
- Graph Database

What kinds of NoSQL

- Key/Value Stores or ‘the big hash table’ .
 - Riak
 - Redis
 - Amazon S3 (Dynamo)
 - Voldemort
 - Scalaris
- Schema-less which comes in multiple flavors, column-based, document-based or graph-based.
 - Cassandra (column-based)
 - HBase (column-based)
 - CouchDB (document-based)
 - Neo4J (graph-based)

Key/Value

Pros:

- very fast
- very scalable
- simple model
- able to distribute horizontally

Cons:

- many data structures (objects) can't be easily modeled as key value pairs

Key-Value Stores

- Memcached – Key value stores.
- Membase – Memcached with persistence and improved consistent hashing.
- AppFabric Cache – Multi region Cache.
- Redis – Data structure server.
- Riak – Based on Amazon's Dynamo.
- Project Voldemort – eventual consistent key value stores, auto scaling.

Beyond Key-Values but Schema-Less

Pros:

- Schema-less data model is richer than key/value pairs
- eventual consistency
- many are distributed
- still provide excellent performance and scalability

Cons:

- typically no ACID transactions or joins

Document Stores

- Schema Free.
- Usually JSON like interchange model.
- Query Model: JavaScript or custom.
- Aggregations: Map/Reduce.
- Indexes are done via B-Trees.

Document Store Examples

- Example: CouchDB
 - <http://couchdb.apache.org/>
 - BBC
- Example: MongoDB
 - <http://www.mongodb.org/>
 - Foursquare, Shutterfly
- Store as JSON (JavaScript Object Notation)

CouchDB JSON Example

```
{  
  "_id": "guid goes here",  
  "_rev": "314159",  
  
  "type": "abstract",  
  
  "author": "Keith W. Hare"  
  
  "title": "SQL Standard and NoSQL Databases",  
  
  "body": "NoSQL databases (either no-SQL or Not Only SQL)  
          are currently a hot topic in some parts of  
          computing.",  
  "creation_timestamp": "2011/05/10 13:30:00 +0004"  
}
```

Column Store

- Each storage block contains data from only one column
- Example: Hadoop/Hbase
 - <http://hadoop.apache.org/>
 - Yahoo, Facebook
- Examples: Vertica, Ingres VectorWise
 - Column Store integrated with an SQL database

Column Stores

- More efficient than row (or document) store if:
 - Multiple row/record/documents are inserted at the same time so updates of column blocks can be aggregated
 - Retrievals access only some of the columns in a row/record/document

Graph Stores

- Useful for storing triples (nodes and edges)
- Scale vertically, no clustering.
- You can use graph algorithms easily.
- Neo4J, OrientDB, FlockDB



NoSQL - Common Advantages

- Cheap, easy to implement (open source)
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
 - Down nodes easily replaced
 - No single point of failure
- Easy to distribute
- Don't require a schema
- Can scale up and down
- Relax the data consistency requirement (CAP)

What am I giving up?

- Joins (mostly)
- group by
- order by
- ACID transactions
- SQL
- Easy integration with other applications that support SQL

Typical NoSQL API

- Basic API access:
 - `get(key)` -- Extract the value given a key
 - `put(key, value)` -- Create or update the value given its key
 - `delete(key)` -- Remove the key and its associated value
 - `execute(key, operation, parameters)` -- Invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map etc).

Searching

- Relational
 - `SELECT `column` FROM `database`, `table` WHERE `id` = key;`
 - `SELECT product_name FROM rockets WHERE id = 123;`
- Cassandra (standard)
 - `keyspace.getSlice(key, "column_family", "column")`
 - `keyspace.getSlice(123, new ColumnParent("rockets"),
getSlicePredicate());`

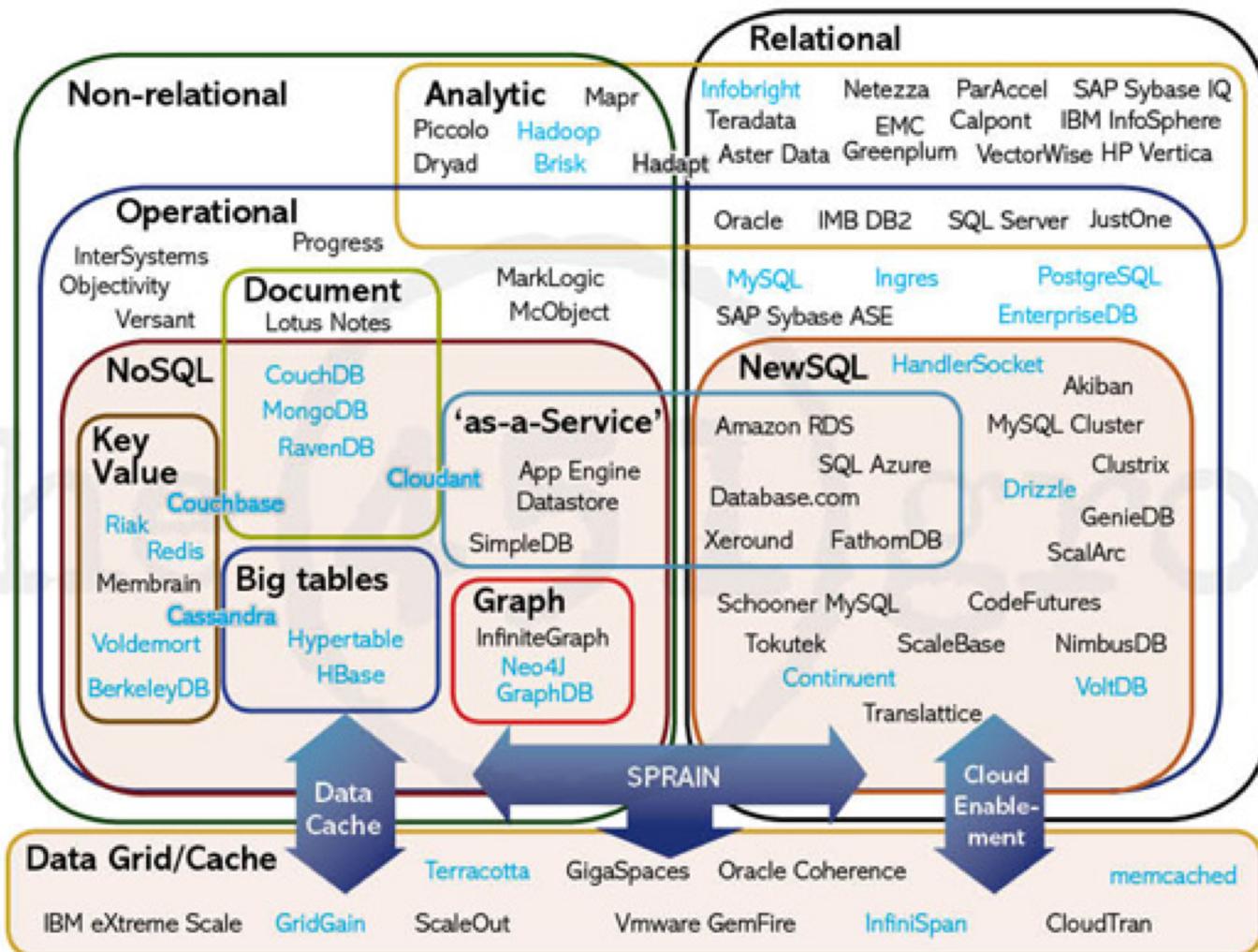
Where would I use them?

- Log Analysis
- Social Networking Feeds (many firms hooked in through Facebook or Twitter)
- Data that is not easily analyzed in a RDBMS such as time-based data
- Large data feeds that need to be massaged before entry into an RDBMS

NoSQL Distinguishing Characteristics

- Large data volumes
- Scalable replication and distribution
 - Potentially thousands of machines
 - Potentially distributed around the world
- Queries need to return answers quickly
- Mostly query, few updates
- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE

Database Map



Visual Guide to NoSQL Systems

Availability:
Each client can
always read
and write.

Data Models

Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented

A

CA

RDBMSs
(MySQL,
Postgres,
etc)
Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak

Pick Two

C

Consistency:
All clients always
have the same view
of the data.

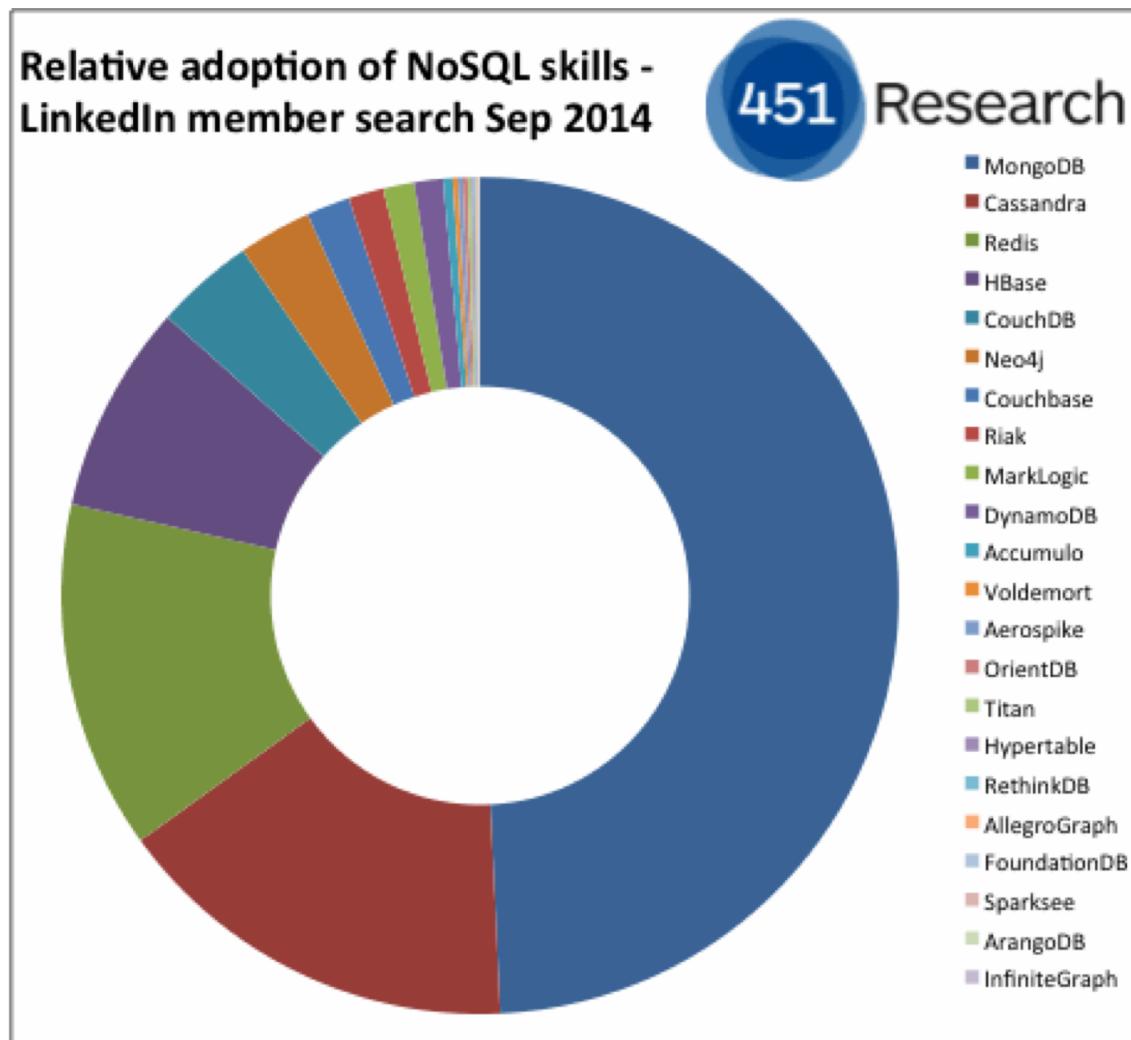
CP

BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalarmis
Berkeley DB
MemcacheDB
Redis

P

Partition Tolerance:
The system works
well despite physical
network partitions.

Popularity



Intro to SQL

Data Import and Export for MySQL

- Import
 - LOAD DATA INFILE ‘filename’ INTO TABLE tablename;
 - Mysqldump
- Export
 - SELECT * FROM tablename INTO OUTFILE ‘filename’;

Data Import and Export for Postgres

- Import
 - `\copy tablename FROM filename.csv' WITH
DELIMITER ',' CSV HEADER;`
- Export
 - `\copy (SELECT * FROM tablename WHERE) TO
'filename.csv' WITH DELIMITER ',' CSV HEADER;';`

SQL

- Data Definition Language (DDL)
 - Create/alter/delete tables and their attributes
 - Following lectures...
- Data Manipulation Language (DML)
 - Query one or more tables – discussed next !
 - Insert/delete/modify tuples in tables

Data Types in SQL

- Atomic types:
 - Characters: CHAR(20), VARCHAR(50)
 - Numbers: INT, BIGINT, SMALLINT, FLOAT
 - Others: MONEY, DATETIME, ...
- Every attribute must have an atomic type

SQL Query – Basic form

```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

Operators

- Like
- Distinct
- Order by
- Group by
- Aggregation
- Joins

Aggregation

```
SELECT  S  
FROM    R1,...,Rn  
WHERE   C1  
GROUP BY a1,...,ak  
HAVING  C2
```

Evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes a_1, \dots, a_k
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Modifying the Database

- Insertions
 - `INSERT INTO R(A1,..., An) VALUES (v1,..., vn)`
- Deletions
 - `DELETE FROM PURCHASE WHERE seller = 'Joe'`
`AND product = 'Brooklyn Bridge'`
- Updates
 - `UPDATE PRODUCT SET price = price/2 WHERE`

References

- “NoSQL -- Your Ultimate Guide to the Non - Relational Universe!”
<http://nosql-database.org/links.html>
- “NoSQL (RDBMS)”
<http://en.wikipedia.org/wiki/NoSQL>
- PODC Keynote, July 19, 2000. *Towards Robust. Distributed Systems*. Dr. Eric A. Brewer. Professor, UC Berkeley. Co-Founder & Chief Scientist, Inktomi .
www.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
- “Brewer's CAP Theorem” posted by Julian Browne, January 11, 2009. <http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>

Web References

- “Exploring CouchDB: A document-oriented database for Web applications”, Joe Lennon, Software developer, Core International. <http://www.ibm.com/developerworksopensource/library/os-couchdb/index.html>
- “Graph Databases, NOSQL and Neo4j” Posted by Peter Neubauer on May 12, 2010 at: <http://www.infoq.com/articles/graph-nosql-neo4j>
- “Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase comparison”, Kristóf Kovács. <http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis>
- “Distinguishing Two Major Types of Column-Stores” Posted by Daniel Abadi on March 29, 2010 http://dbmsmusings.blogspot.com/2010/03/distinguishing-two-major-types-of_29.html

Web References

- “MapReduce: Simplified Data Processing on Large Clusters”, Jeffrey Dean and Sanjay Ghemawat, December 2004.
<http://labs.google.com/papers/mapreduce.html>
- “Scalable SQL”, ACM Queue, Michael Rys, April 19, 2011
<http://queue.acm.org/detail.cfm?id=1971597>
- “a practical guide to noSQL”, Posted by Denise Miura on March 17, 2011 at <http://blogs.marklogic.com/2011/03/17/a-practical-guide-to-nosql/>
- Mongodb book <http://openmymind.net/mongodb.pdf>

Contact Information

Rayid Ghani

`rayid@uchicago.edu`