CSE 414: Section 6 NoSQL, SQL++

November 1st, 2018

Query workload types

[“One Size Fits All”: An Idea Whose Time Has Come and Gone]

OL**T**P (Online **Transactional** Processing)

- Atomic operations (one or multi entities). E-commerce, webapps.
- A small number of records per query - “Latest state”

produce a lot of small datas; lots of writes to small number of rows.

OL**A**P (Online **Analytical** Processing)

- Analytics and data-warehousing. Reporting, decision support.
- Many records per query - “Aggregated stats” on “Bigger data”

Analyze large collections of data, alot more reads over a lot more rows

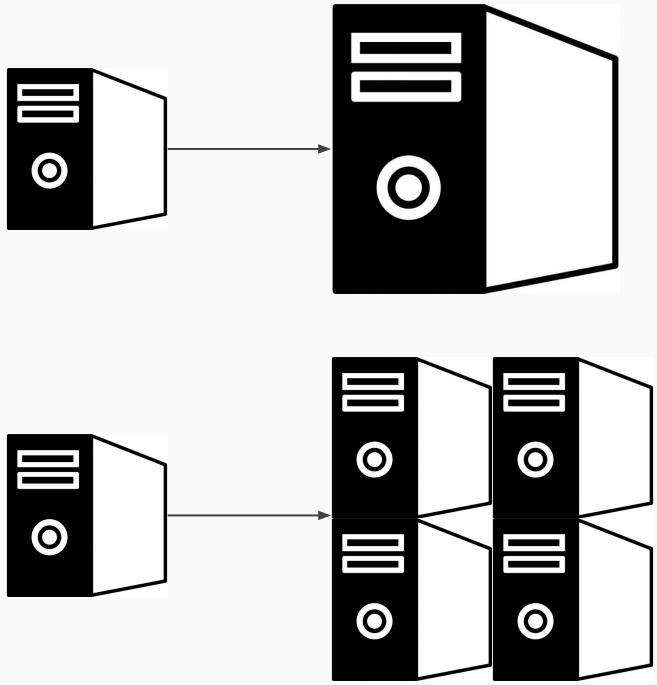
Scaling methods

Scale up (vertically) make your computer better

- Add more power to a single node
- diminishing returns physical limitations

Scale out (horizontally) add more computers

- Cheap commodity hardware
- Management / coordination complexity
  communication, parallelism, etc…



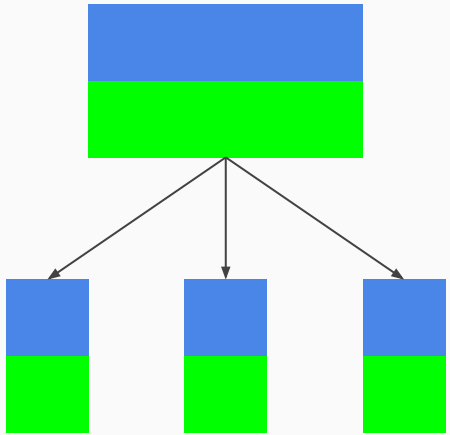
Partitioning & Replication

If you are adding more machines (scale out horizontally) how do you disperse the data?

Partitioning

easy to do writes, but reads are more difficult

Or “Sharding”, “Distribution, ”Fragmentation”



- Motivation:
  - BIG data - need to split up! (e.g. PB-level)
  - Availability: better write (and single-record read) throughput
- Challenge: fair share of requests
  - Choice of partitioning schemes
  - “Justin Bieber Effect” -> “hot spots”

one really popular piece of data on a specific machine; this machine gets accessed a lot and others barely used

Partitioning & Replication

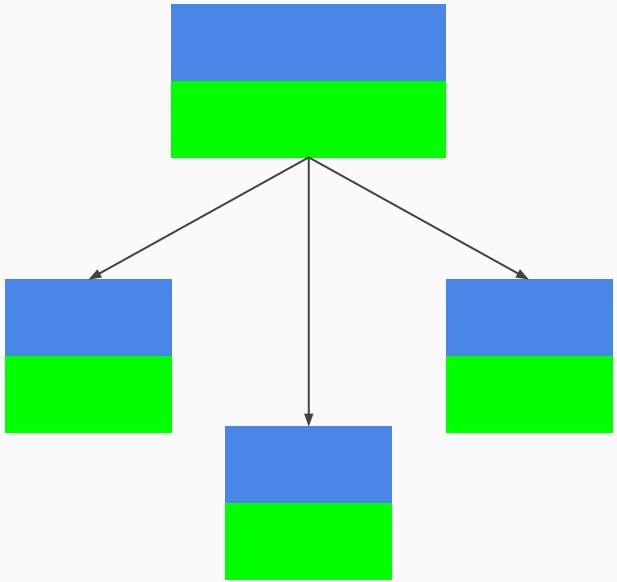
redundancy; same data replicated on multiple machines

Replication

- Motivation:
  - Fault-tolerance / durability: power / disk failures
  - Keep data close to the user (geographically)
  - Availability: better read (and potentially write) throughput
- Challenge: keeping data in sync
  - E.g. write to a leader and then propagate
  - Choice of consistency models

less risk of losing data

if you have multiple copies of data, how do you handle updates?
Make sure that ALL versions of that data are updated.



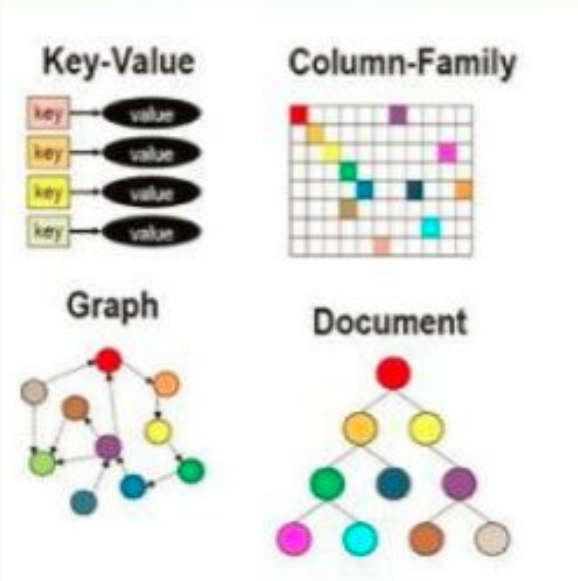
NoSQL

SQL vs. NoSQL Databases: What's the Difference?

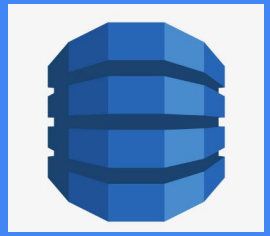
- No clear definition :\
  - Non-relational
  - + **scalability**, + availability, + flexibility
  - - consistency, - OLAP performance
  - Open source implementations
- Motivation
  - The need to scale
  - Lots of web apps mostly **OLTP** queries
    - Read/write intensive
    - but fewer joins & aggregates

Data Models

- Key-value stores
  - Opaque value
  - e.g., Project Voldemort, Memcached
- **Document stores** we use this one
  - “key-object”
  - e.g., SimpleDB, CouchDB, MongoDB, AsterixDB
- Extensible Record Stores
  - “column groups”
  - e.g., BigTable, HBase, Cassandra, PNUTS
- Graph
  - E.g. Neo4j



NoSQL: DynamoDB



Document and Key-Value Stores!?

Jeff Bezos wants his shopping carts full

Throughput motivated

NANI?

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed.

JSON and Semi-Structured Data

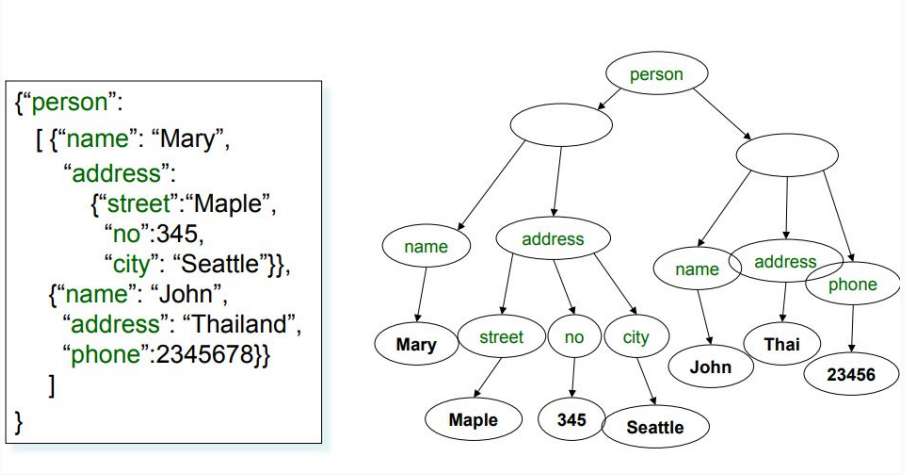
JSON, XML, Protobuf (also an IDL)

Familiar - as your HTTP request/response

- Good for data exchange
- Maps to OOP paradigm

Also - as a database file

- Flexible tree-structured model
- Query langs: XQuery, XPath, etc.



AsterixDB, SQL++

- A semistructured NoSQL style data model (ADM)
- Extends JSON with object database ideas

Know the following:

- DDL: type (open vs. closed), data types (e.g. multiset). Creating an index.
- DML: Heterogenous Collections, Nesting / Unnesting.
- (Asterix stores data as flattened tables behind the scenes)

What is SQL++?

Just like SQL but parsed for processing JSON data

SQL++ has keywords to handle collections of data (i.e. non-flat data)

Motivation for SQL++

Why SQL++? Why not some other query language?

- People are used to/like specifying data through SQL syntax
- SQL-like language enforces idea of physical data independence

Useful Keywords/Syntax for HW

`is_array( … )` -----> checks if value is an array

`split(s, d)` -----> splits string s on delimiter d

`[ … ]` -----> explicitly construct array

`(CASE WHEN … THEN … ELSE … END)` -----> combine with "`is_array(…)`"

no key nor a value — key without a value

`MISSING` -----> reserved keyword like "`NULL`"

`` ` … ` `` -----> backtick needed for accessing keys with names containing "-"