CIS 560 – Database System Concepts

Lecture 29

NoSQL

November 13, 2013

Credits for slides: Wisdom, Alberton, Pokorný, Hoekstra.

Reminders

- Assignment 9 (query optimization) due 11/15
- Exam 2 (assignments 6-9) – 11/20
- Project - DB implementation and queries due 11/22

Outline

So far:

- Relational DBMS, SQL

Next:

- NoSQL

NoSQL: The Name

- "SQL" = Traditional relational DBMS
- Recognition over past decade or so:
  Not every data management/analysis problem is best solved (*exclusively*) using a traditional relational DBMS
- "NoSQL" = "No SQL" =
  Not using traditional relational DBMS
- "No SQL" ≠ Don't use SQL language

(Not only SQL)

- NoSQL "Definition" from http://nosql-databases.org/

***NoSQL DEFINITION**: Next Generation Databases mostly addressing some of the points: being **non-relational**, **distributed**, **open-source** and **horizontally scalable**. The original intention has been **modern web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free, easy replication support, simple API, eventually consistent / BASE** (not ACID), a **huge data amount**, and more. So the misleading term "nosql" (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.*

NoSQL: The Name

- "SQL" = Traditional relational DBMS
- Recognition over past decade or so:
  Not every data management/analysis problem is best solved using a traditional relational DBMS
- "NoSQL" = "No SQL" =
  Not using traditional relational DBMS
- "No SQL" ≠ Don't use SQL language

✳ "NoSQL" = "Not Only SQL"

Not every data management/analysis problem is best solved using a traditional DBMS

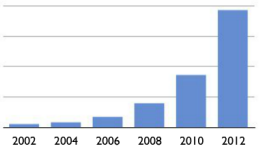
Database Management System (DBMS) provides….

… efficient, reliable, convenient, and safe multi-user storage of and access to massive amounts of persistent data.

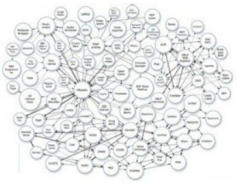
Not every data management/analysis problem is best solved using a traditional DBMS

- Convenient
  - Relational DBMS (simple data model, declarative query language, transaction guarantees)
- Multi-user
- Safe
- Persistent
- Reliable
- Massive
- Efficient

When RDBMS met Web 2.0



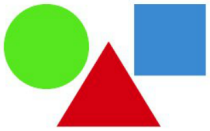
Big data



Connectivity

P2P Knowledge

Concurrency



Diversity

Cloud-Grid

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

Scaling Up

- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
- Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
  - Master-slave
  - Sharding

Scaling RDBMS – Master/Slave

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to slaves

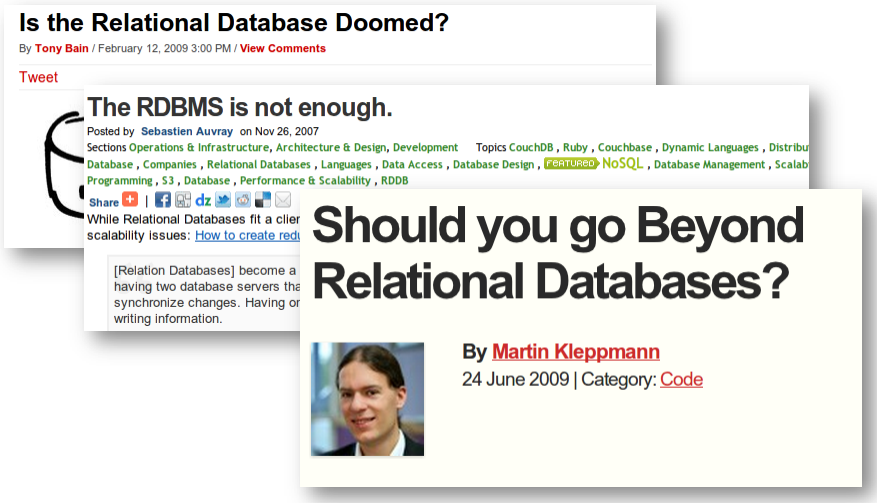
Scaling RDBMS - Sharding

- Partition or sharding
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Scaling RDBMS – Multi-Master Replication

- The multi-master replication system is responsible for propagating data modifications made by each member to the rest of the group, and resolving any conflicts that might arise between concurrent changes made by different members.
- INSERT only, not UPDATES/DELETES
  - Data is versioned upon update
  - Data is never DELETED, only inactivated
- No JOINs, thereby reducing query time
  - JOINs are expensive with large volumes and don't work across partitions.
  - This involves de-normalizing data
  - Denormalization leads to even larger databases, reduces query time.

The Death of RDBMS?



Why NoSQL?

- For data storage, an RDBMS cannot be the be-all/ end-all
- Just as there are different programming languages, need to have other data storage tools in the toolbox
- A NoSQL solution is more acceptable to a client now than even a year ago

How did we get here?

- Explosion of social media sites (Facebook, Twitter) with large data needs
- Rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Just as moving to dynamically-typed languages (Ruby/Groovy), a shift to dynamically-typed data with frequent schema changes
- Open-source community

Dynamo and BigTable

- Three major papers were the seeds of the NoSQL movement
  - BigTable (Google)
    - Column families
  - Dynamo (Amazon)
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data store
    - Eventual consistency
  - CAP Theorem (discuss next...)

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm
- Not a backlash/rebellion against RDBMS
- SQL is a rich query language that cannot be rivaled by the current list of NoSQL offerings

CAP Properties

- Three properties of a system
  - Consistency (all copies have same value)
  - Availability (system can run even if parts have failed)
  - Partitions (network can break into two or more parts, each with active systems that can not influence other parts)

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

Partition tolerance

- Ensures that write and read operations are redirected to available replicas when segments of the network become disconnected

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.

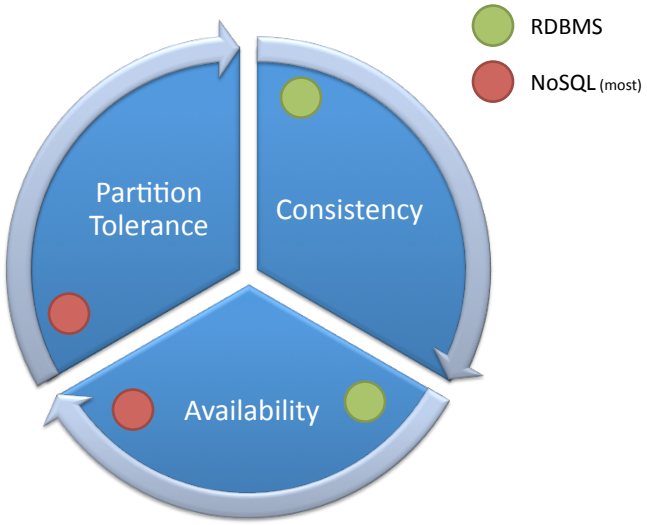
Eventual Consistency

- 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) properties, as opposed to ACID
  - Soft state: copies of a data item may be inconsistent
  - Eventually Consistent – copies becomes consistent at some later time if there are no more updates to that data item
  - Basically Available – possibilities of faults but not a fault of the whole system

CAP Properties

- Three properties of a system
  - Consistency (all copies have same value)
  - Availability (system can run even if parts have failed)
  - Partitions (network can break into two or more parts, each with active systems that can not influence other parts)
- Brewer's CAP "Theorem": for any system sharing data it is impossible to guarantee simultaneously all of these three properties
- Very large systems will partition at some point
  - it is necessary to decide between C and A
  - traditional DBMS prefer C over A and P
  - most Web applications choose A (except in specific applications such as order processing)

CAP theorem



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/

When to use NoSQL?

- Not everybody should jump on the NoSQL wagon... It's all about *your* requirements:
  - Do you have Big Data? A few hundred gigabytes isn't considered that much these days and it's easily manageable using well known techniques. Unless you are planning to scale to 10x, 100x?
  - Do you need all the relational features of RDBMS? Would simple lookups be sufficient?
  - Do you have millions of dollars to spend on licenses? If yes, then you probably already have geo-distributed Oracle clusters and you should stop worrying. If not, it's worth researching these solutions.