NoSQL Databases

NoSQL

- Introduction
- Aggregate data model
- Distribution Models
- Consistency
- Map-Reduce
- Types of NoSQL Databases
  - Key-Value
  - Document
  - Column Family
  - Graph

Introduction

Relational Databases

- Relational databases are the prevalent solution for storing enterprise data
- Some of the main benefits of RDBMS are
  - Access to persistent data
  - ACID Properties
  - Integration
    - Where multiple applications share data
  - Standardized model

Impedance Mismatch

- A major disadvantage with a RDBMS is the *impedance mismatch* with the object model
- This is partly mitigated by the availability of object-relational mapping frameworks
- The typical DB model uses a RDBMS as an integration database
  - Providing a data source for multiple applications
- An alternative is to create separate application databases
  - And use web services to integrate the application databases

Impedance mismatch – a term borrowed from electrical engineering where (very broadly) the output does not match the input

Data, Data Everywhere

- The volume of data has grown dramatically in recent years
  - Caused by the onset of the web as a vehicle for trade, information and social networking
  - With the growth in data came a dramatic growth in users
- Managing the increase in data requires more computing resources
  - Scale up or out

Clusters, and More Clusters

- There are two basic methods for scaling computing resources
  - Vertical scaling ("up")
    - Buy bigger, more powerful machines
  - Horizontal scaling ("out")
    - Buy more, cheaper, machines
- Many small machines in a cluster ends up being cheaper than scaling up
  - And provides more resilience

RDBMS and Clusters

- RDBMS are not designed to run on clusters
  - Though many DBMS products support distributed databases
    - Primarily through writing to a highly available disk subsystem
    - Though this subsystem may still be a single point of failure
- Organizations sought alternatives to RDBMS
  - Google Dynamo
  - Amazon BigTable
  - Neither of these products use SQL
    - They are NoSQL databases

NoSQL

- There is no one definition of NoSQL
  - And NoSQL databases differ significantly
  - There is even disagreement over whether it stands for No SQL or Not only SQL
- Common NoSQL database characteristics
  - Does not use relational model
  - Runs well on clusters
  - Open source (not always)
  - Built for 21st century web applications
  - Schema-less

Aggregates

Types of NoSQL Database

- NoSQL databases move away from the relational data model
- There are four main types of NoSQL database
  - Key-value
  - Document
  - Column store
  - Graph
- The data models used by the first three have some similarities

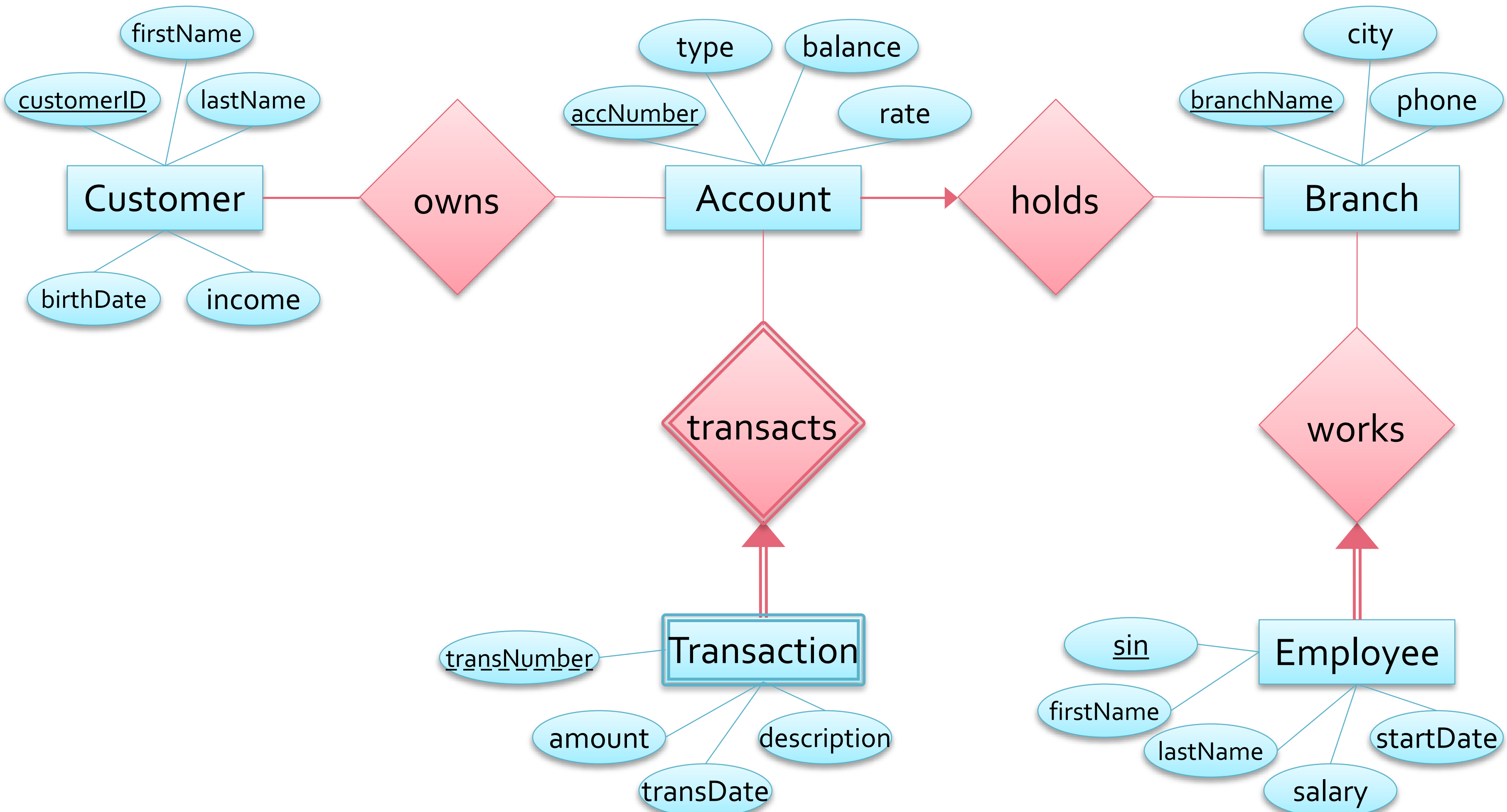
Aggregates

- Modern programming language data structures have more complex structure than relations
  - That allow lists and other structures to be nested
  - We will refer to such structures as *aggregates*
    - A collection of related objects to be treated as a unit
- Using aggregates makes it easier for databases to operate in clusters
  - The aggregate can be used as a unit for replication and sharding

Relations versus Aggregates

- A relational database captures relationships using foreign keys in tables
  - Combining tables entails joins
    - Which may be expensive
  - It does not capture the structure of an aggregate
- The same relational schema could have many different corresponding aggregates
  - Let's look at our bank example

Bank ERD



firstName
customerID
lastName
Customer
birthDate
income
owns
type
balance
accNumber
rate
Account
holds
city
branchName
phone
Branch
transacts
works
transNumber
Transaction
amount
description
transDate
sin
Employee
firstName
lastName
startDate
salary

Bank Aggregates

- There are many different possible aggregates
  - For example a Branch object could include a container holding its associated accounts
  - Or Account objects could be independent of the Branch and contain a reference to the branch
- There are similar variations between the relationships between other entities
- The aggregates that are chosen should reflect the way that data is manipulated
  - Data related to a single aggregate should be maintained at the same cluster

Schema-Less Databases

- NoSQL databases are schema-less
- Before storing data in a relational database the schema has to be defined
  - Tables, columns and their domains are defined
- NoSQL databases are much more casual
  - Key-value store allows any data to be associated with the key
  - Document databases do not make restrictions on what is contained in a document
  - Column family databases allow any data to be stored in any column

Advantages and Disadvantages

- Advantages
  - Freedom and flexibility
    - New data can be added as required
    - Old data can be retained since un-needed columns do not have to be deleted
  - Easier to deal with non-uniform data
- Disadvantages
  - In practice most programs rely on schemata
    - If the schema is only implicit it must be deduced
    - The schema is moved from the data store to the application

Distribution Models

Introduction

- NoSQL databases handle large amounts of data by scaling out
  - Running on clusters of machines
- The data to be stored needs to be distributed across the cluster
  - Sharding
  - Replication

Sharding

- Sharding distributes data between nodes
  - The goal is for users to get all, or most of, their data from one server
- Sharding methods
  - By physical location
    - Locate the Vancouver accounts in Vancouver servers
  - Locate aggregates that are likely to be accessed together or in sequence in the same location
  - Many NoSQL databases perform automatic sharding

Sharding and Performance

- Sharding can improve both read and write performance
  - Sharding allows horizontal scaling for both reads and writes
- However sharding does not improve resilience
  - Since sharding distributes data across many machines there is a larger chance of failure
    - Particularly compared to a single machine that is highly maintained

Replication

- Replication is the process of maintaining multiple copies of data
  - To improve read performance
  - And improve availability and resilience
- Replication works better for read-intensive databases
  - Since all copies of the data have to be updated when processing writes
  - There are two replication schemes that handle writes in different ways
- Replication may lead to inconsistency

Master – Slave Replication

- In master-slave replication one copy is maintained as the definitive data source
  - All updates are performed on this *master* copy and then propagated to the slaves
  - Read requests are handled by the slaves
- Since the master handles all updates it is not good for write-intensive systems
- If the master fails one of the slaves is appointed as the new master
  - Either manually or automatically

Peer to Peer Replication

- Master-slave replication does not improve write scalability
  - Resilience is improved for slaves but not the master
  - Master is a bottleneck and a single point of failure
- In peer to peer replication all replicas accept writes and have equal weight
  - There is a trade-off between availability and inconsistency
    - Read inconsistency can occur when changes have not been propagated to all replicas
    - Write inconsistency can occur when two updates are performed at the same time

Sharding and Replication

- Sharding and replication can be combined
  - Each shard is replicated
- In master – slave replication there is one master for each shard
- Peer to peer replication of shards is commonly used for column family databases

Consistency

ACID

- Relational databases guarantee consistency
  - Through ensuring that transactions are processed atomically as if they occurred in isolation
  - Databases interleave actions of transactions to improve throughput
  - While identifying and preventing conflicts that could leave the database in an inconsistent state
    - Often through locking
- When actions of two transactions conflict the database prevents one from starting before the other has finished
  - Guaranteeing consistency becomes more difficult when the data resides on multiple servers

Lost Update Example

Consider two transactions that affect a single bank account which initially holds $1,000. In one transaction Bob (T1) is going to deposit $500 and in the second interest of 10% is going to be added to the account.

T1	T2	A
Read(A)		1,000
	Read(A)	1,000
Write(A)		1,500
	Write(A)	1,100

T1	T2	A
Read(A)		1,000
Write(A)		1,500
	Read(A)	1,500
	Write(A)	1,650

This interleaving results in a *lost update* caused by an *unrepeatable read* which leaves the database in an inconsistent state

In this version the database locks A which prevents T2 from acting on A (including reading it) until T1 has completed

No Free Lunch

- It's important to understand that locking mechanisms come with a cost
  - A reduction in throughput
    - i.e. less transactions are processed in a given time
- Controlling concurrency through locking in a distributed system is time consuming
  - The process is more complex
  - And it takes time to communicate across the network
- NoSQL databases often relax their requirements for consistency

Distributed Concurrency

- Controlling concurrency on a distributed system becomes more complex
  - Particularly if the system is using peer to peer replication
    - Where any node can process updates
- Conflicts must be detected across nodes
  - And updates must be processed in the same order
    - Note that in the example the account's balance would be different if T2 occurred before T1

Atomic Transactions

- A common claim is that NoSQL databases to not guarantee ACID transactions
  - That is, they *drop acid*
    - Specifically they do not support atomic transactions
- Aggregate oriented databases *do* support atomic transactions
  - But only within aggregates
  - And not necessarily across aggregates
- Updates that affect multiple aggregates may result in inconsistent reads for some time
  - Known as the *inconsistency window*

> The typical inconsistency window for Amazon's SimpleDB is claimed to be less than one second

Distributed Read Consistency

- An additional issue for distributed databases is replication consistency
  - Where updates reach different replicas at different times
  - This may result in two users reading different values at the same time
- Over time replicas will have the same values
  - That is they will be *eventually consistent*
  - Replication consistency issues may result in an increase in the size of the inconsistency window

CAP Theorem

- The CAP theorem states that it is only possible to maintain two out of three properties
  - Consistency
  - Availability
    - If a node is available it can read and write data, or
    - Every request received by a non-failing node in the system must result in a response
  - Partition tolerance
    - The cluster can survive communication breakdowns that separate it into multiple partitions

Consistency Tradeoff

- A distributed system must have partition tolerance
  - Unlike a single server system which can therefore provide consistency and availability
  - Without partition tolerance the implication is that if a partition occurs all the nodes in the cluster go down
- As partition tolerance is required, distributed systems must choose between availability and consistency
  - This is not an either / or choice
    - Most often it is necessary to reduce consistency to increase availability

CAP Example – Hotel Booking 1

- Consider Bob and Kate trying to book the last hotel room in the Grand Hotel in Vancouver
  - On a peer to peer system with two nodes
    - Bob is in Vancouver and Kate is in London
  - If consistency is to be ensured then London must confirm Kate's booking with Vancouver
- If the network link fails then neither node can book rooms
  - Which sacrifices availability

CAP Example – Hotel Booking 2

- An alternative is to use master-slave replication
  - All bookings for Vancouver hotels will be processed by the Vancouver node
    - Vancouver is the master
- What happens if the network connection fails?
  - Bob can still book the last room but Kate cannot
  - Kate can see that a room is available but cannot book it
  - There is an availability failure in London
    - As Kate can talk to the London node but the node is unable to update data

CAP Example – Hotel Booking 3

- A third alternative is to allow both nodes to accept reservations when the connection fails
  - This increases availability
  - But may result in both Bob and Kate booking the last room
    - A consistency failure
- This reduction in consistency may be acceptable
  - The hotel does not *lose* bookings
  - And may keep a few spare rooms even when fully booked in case a room has to be vacated

Dealing with Inconsistency

- There may be situations where some inconsistency is permissible
  - These situations are domain dependent and would have to be identified
    - By talking to the client
- Deciding to deal with some inconsistent updates (or reads) can be very useful
  - The tradeoff may be for more availability or performance

ACID and BASE

- NoSQL databases are said to follow the BASE properties rather than the ACID properties
  - Basically Available
  - Soft state
  - Eventually consistent
- It is debatable how useful this is, or how well defined the properties are
  - But its cute …

Types of Eventual Consistency

- Read-Your-Writes consistency
  - Once a process has updated a record it will always read the updated value
- Session consistency
  - Read-Your-Writes consistency over a session
- Monotonic Read consistency
  - Once a process sees a version of a value it will never see an earlier version of that value
- Monotonic Write consistency
  - Updates are executed in the order in which they are performed

Key-Value Stores

Key-Value Databases

- A key-value store has two columns
  - The key – the primary key for the store
  - The value – which can be anything
- The value in a key-value store is not understood by the store
  - It is the responsibility of the application that is accessing the value
- The structurally simplest NoSQL database

Key-Value Store Features

- Consistency
  - Applicable only for single key operations
  - Eventual consistency a popular model
- Transactions
  - Varies considerably between products
- Query features
  - Key-values stores support querying by the key
  - Querying by attributes of the value column is not supported
- Scaling – by sharding
  - The value of the key determines the node on which the key is stored

Key Design

- Keys should be well designed
  - Use a naming convention
  - Use meaningful and unambiguous names
  - Use consistently named values for ranges
    - e.g. dates
  - Use a common delimiter
  - Keep keys short while complying with the above
  - Take implementation limitations into account

Value Design

- Generally the value will be driven by the application but some design issues remain
  - What aggregates are to be used?
    - One key – many values (e.g. customer123)
    - Many keys – many values (e.g. customer123 address, customer 123name etc.)
  - Large or small values?
    - Large values reduce the number of reads
    - But the time to read and write values increases

Key-Value Store Uses

- Uses for which key-value stores are suitable include
  - Session information
  - Storing configuration and user data information
  - Shopping carts
- Examples of key-value stores
  - [Riak]
  - Redis
  - Amazon WebServices Simple Db (and DynamoDB)
  - Project Voldemort

Document Databases

Document Databases

- Document databases store … documents …
  - Often XML, JSON etc.
- Documents are self-describing hierarchical tree structures
  - Documents are similar but do not have to be identical
    - And can have different schema
- Document databases are similar to key-value stores
  - Except that the value is a document
  - And the document can be examined, rather than just being obtained

Document Database Features

- Consistency
  - Using MongoDB as an example, the database uses master-slave replication
  - The level of consistency can be specified
    - That is, the number of nodes to which an update has to be propagated before it is deemed successful
    - Making write consistency stronger reduce availability
- Availability
  - Availability is improved through replication
    - Data is duplicated across nodes
    - Allowing data to be read even when the master node is unavailable

Document Database Features 2

- Queries
  - Document databases allow documents to be queried without first retrieving the entire document
    - Different document databases provide different query features
- Scaling
  - Scaling for reads is supported by adding more slaves
  - Scaling for writes is supported by sharding
    - A shard key is selected that determines how documents are broken into shards

Documents and Collections

- Documents can be grouped into collections
  - Grouping similar documents together
    - Documents in a collection do not have to have identical structure
    - But should contain documents of the general type
  - Documents in a single collection will typically be processed by the same application code
    - If not, consider if the collection should be split
- Documents in a collection can be operated on as a group

Document Database Uses

- Uses for which document databases are suitable include
  - Event logging
  - Content management systems
  - Web analytics
  - E-commerce
- Examples of document databases
  - MongoDB
  - CouchDB

Column-Family Stores

Column-Family Stores

- Column-family stores group column families and are a refinement of columnar databases
  - A columnar database stores each column separately
    - Which greatly increases the speed of aggregate operations on column data
    - But makes accessing an entire row inefficient
- Column-families contain multiple related columns
  - But may still break down what would be a single table in a relational database into multiple tables
  - Rows in a column-family do not have to have the same columns as other rows

Column-Family and RDBMS

- Column family DBs *appear* similar to relational databases
  - They have rows and columns
  - Rows are identified by unique identifiers
- There are important differences
  - Column family DBs do not support multi-row transactions
  - In some column family DBs column values are not typed
    - The data is interpreted at the application level
  - Column family DBs are typically de-normalized
    - The same key may identify different column families
    - Column values may include lists and other structures

DeNormalization Example

Row key

Customer ID

Customer column family

lname	fname	address	...
Bobson	Bob	123 Bobville Rd	

Product column family

16735	56122	98735
Thermal Detonator	Pulse Rifle	BFG

- Different customers rows can have different numbers of products
- The product ID is used as a column name
  - Data is stored in column name order
  - Column values, the product names are repeated for each customer
  - But joins are not required to return customer information

Column-Family Store Features

- Consistency
  - Column-family stores use peer to peer replication
  - The level of consistency can be specified
    - By selecting the number of nodes that have to respond to a read or write before it is committed
- Availability
  - Availability is improved through replication
    - Using peer to peer replication improves availability
    - Which can be further improved by reducing consistency

Column-Family Store Features 2

- Queries
  - Column-families can be queried
    - Although query languages are not as rich as SQL
    - Do not, for example, allow joins or subqueries
  - Columns can be indexed to improve efficiency
    - Data in rows are sorted by column names
- Scaling
  - Scaling for reads or writes is achieved by adding additional nodes

Column-Family Store Uses

- Uses for which column-family stores are suitable include
  - Event logging
  - Content management systems, blogging sites
  - Counters
- Examples of column-family stores
  - Cassandra
  - HBase
  - Google BigTable
    - Designing BigTable schema

Graph Databases

Graph Databases

- Graph databases are designed to efficiently store relationships
  - Nodes map to entities and edges to relationships
- Nodes have properties such as *name*
- Edges have types such as *likes*
  - Edges can be set to bidirectional
  - Different edges in the same graph can have different types
    - For example *likes* and *employee*
- Traversing relationships in a graph database is fast
  - The relationships are stored persistently

Graph Operations

- Graph databases support a set of specialized operations
  - Union
    - Combines two graphs by taking the union
  - Intersection
    - Combines two graphs by taking the union
  - Traversal
    - Traverses the graph from a given node, visiting all connected nodes

Graph Properties

- Isomorphism
  - Two graphs are isomorphic if they have corresponding vertices and edges between vertices
- Order – number of vertices
- Size – number of edges
- Degree – the number of edges of a vertex
- Closeness
  - A measure of distance between vertices
- Betweenness
  - A measure of how connected vertices are
  - Can be used to identify vulnerable paths

Graph Database Uses

- Uses for which graph databases are suitable include
  - Storing connected data such as social networks
  - Location based services
  - Recommendation engines
  - Tracking infections
- Examples of graph databases
  - Neo4j
  - Titan
  - OrientDB