# NoSQL Databases and Real Time Analytics

Rebecca Simmonds

# Introduction

- First year PhD student at Newcastle University
- Started identifying noSQL architectures
- Compiled information about them
- Moved onto real time analytics
- Now investigating the combination of them both

# Overview

- Background knowledge
- Relational databases
- NoSQL databases
- Comparison
- Real world applications
- Real time analytics

# Brewer’s Theorem

- Consistency: all client see the most recent data simultaneously.
- Availability: all requests receive a response, whether they are successful or not.
- Partition Tolerance: the system continues to operate even though there are message losses or part of the system fails.

# Relational Databases

- SQL querying language for analyzing the data
- Stores data in tables, rows and columns
- Displays the relations of the data
- Problems: horizontal scalability, partitioning, unstructured data and availability

# VoltDB

- Has the ability to scale to 120 partitions on 39 servers with 1.6 million complex transactions per second
- NewSQL
- ACID transactions
- No joins has to be done through stored procedures or at the application level
- Rigid schema

# Relational Cloud

- MIT based project
- CryptDB
- Scalable
- Transactional
- Workload aware partitioning

# NoSQL

- New architecture for data storage
- Doesn’t use SQL
- “Not only SQL”
- A definition is not agreed upon but usually they consist of a combination of the following functions. . .

# NoSQL

- The ability to horizontally scale
- The ability to distribute (partition) data over many servers
- A simple call level interface
- A weaker concurrency model than the ACID transactions of most relational (SQL) database systems

# NoSQL Database Categories

- Key-value store
- Document store
- Column/Extensible store
- Graph store

# Key Values Stores

- Stores the data value by key
- Simple
- Get, put and remove
- Schemaless
- Allows different data types for storage providing flexibility

# DynamoDB

- Amazon’s original noSQL implementation
- AP system
- Allows for tunable consistency
- Effective for multiple scales, even 100’s of servers
- Really elastic and is used by Amazon’s EC2

# Riak

- Key value store
- Elastic
- Horizontally scalable
- AP system with tunable consistency
- Good performance, with pluggable storage
- REST interface

# Document Stores

- Stores documents
- A document holds a collection of fields
- Each document can hold a different amount of attributes
- Retrieved by using the key that each document is given
- Allows for more complex queries than key value stores

# MongoDB

- Document store
- AP system
- Eventually consistent
- Automatic sharding
- Scalable
- Schemaless

# CouchDB

- Document store
- Quick read and writes
- Eventually consistent
- But acid transactions are available at document level
- Provides tools to improve its use

# Column or Extensible Stores

- Based on Google's BigTable
- Column-oriented data storage instead of rows
- Key based searching
- Horizontally Scalable
- More flexible and dynamic than document stores

# BigTable

- The first column store
- One big table made up of columns
- Partitioned into tablets
- Flexible schema
- Scales to 100s of servers
- Atomic operations at row level

# Cassandra

- Column store
- Structure is a keyspace which holds column families
- Tunable consistency
- Automatically brings nodes into the cluster, so very elastic
- Scalable across 100s of nodes

# Graph Stores

- Data is stored as a graph representation
- Uses graph nodes and relations
- Good for more complex relational information, e.g. friend of a friend
- Best for analytical work loads that need traversal

# Neo4j

- Schemaless so easily evolved
- ACID transactions
- Highly available and fault tolerant
- Scalable to billions of graph nodes
- Partitioning is manual and can be hard

| Database | Consistency | Availablility | Partition Tolerance | Schema | Scalability |
|---|---|---|---|---|---|
| **VoltDB** | yes | no | yes | yes | Near linear scalability |
| **Relational Cloud** | yes | yes | no | no | Scalable |
| **DynamoDB** | no | yes | yes | no | Scalable |
| **Riak** | no | yes | yes | no | Scalable |
| **MongoDB** | no | yes | yes | no | Scalable |
| **Neo4j** | yes | yes | no | no | Scalable |
| **BigTable** | no | yes | yes | no | Scalable to 100s of servers |
| **Cassandra** | no | yes | yes | no | Scalable |

# Other categories

- **Joins:** whether it allows data to be joined from separate partitions
- **Open source:** whether or not the source code can be obtained freely.
- **Partitioning type:** provides how and if the database is partitioned (sharded).
- **Logging:** includes whether the database includes logging of the operations it completes.

# Other Categories

- **Locking:** describes whether the database provides locks.
- **Storage:** describes what storage it uses.
- **Performance:** How quickly the system performs reads and writes
- **Application and work loads**: these are the workloads and applications the database would be best suited to

# Real World Application

- VoltDb is used by companies such a Booyah games, Eonblast and GetCo
- VoltDB was used because of its linear scaling, consistency and performance
- Riak provides a new storage architecture for Mozilla labs
- Riak is used because of its security, availability and fault tolerance

# Real World Application

- MongoDB is used by a company called Pixable
- MongoDb is used because it is simple, has flexible sharding and its replication options
- Cassandra has been implemented by Netflix and OOyala
- Cassandra is used because of its asynchronous replication, no down time when the schema is changed, fast, scalable, more analytics, cheap and can scale without decreasing the performance.

# Real Time Analytics

# The Next Step Data Collection. . .

- Using the  architectures
- Traditional methods of data analysis
- Using streams to collect the data
- Data analysis in real time or across the database

# Traditional Data Collection

- Data warehouses
- A database used to analyse the data
- Stores the data so there is an archive
- Data is then processed and can be reprocessed
- Data mining and analytical online processing

# Stream Data Collection

- Continuous flow of data that is being produced in near real time
- As the data arrives and a complex event processing system can be used to process it
- Provides real time information
- Social networks, financial markets, health markets

# Current Work with the Stream

- Introducing a real time aspect to the storage
- Decided to use Twitter
- Using a small percentage of Twitter's firehose to collect data
- Different tools for analysis
- Currently using twitter4j
- Moving onto using complex event processing systems e.g. ESPER

# Part One revisited

- Now researching a combination of traditional and streaming data collection
- Using noSQL because of its scalability
- Using Cassandra for the implementation
- Scalable, provide more complex querying techniques, elastic and dyanmic structure

# Queries

- Decided to devise some general queries to use across the architecture
- There are three categories of query:
- Historic
- Real time
- Real time + historic

# Historic Query Example

- Query to identify a trend within a certain time bound

Select count(tweet(hash)) as TweetCount From H[Range τ − now]

Having (TweetCount > x and τ <= t) OR (TweetCount < x and τ <= t’)

# Real Time

- A query to retrieve all tweets with a specified hash tag, from a specified location from the stream

Select tweet as t
From S[now]
Where t.loc = loc and t.hash = h

# Real Time + Historic

- A query to notify the user when a tweet has been rewteeted x number of times, on the stream and within the historic store

Select NotifyUser() From S[Now], H Where retweet > x

# Part One + Part two

- Using streaming to collect the data
- Then analyse it in real time
- Modeling a Cassandra architecture that will allow the execution of all three categories of query
- Combining real time and historic queries is the interesting part

# Future work

- Implement a set of general queries
- Conduct experiments which will execute the queries across a database
- Investigate the performance and ease of these executions
- Scale out

# Any Questions