# NoSQL

Michal Barla

# Literature

Pramod J. Sadalage and Martin Fowler: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, 2012 by Addison-Wesley Professional.

# Use case

- Simple Web Analytics
    - Id, user_id, url, pageviews_count
- It starts to be successful...
- ...and you start to see timeouts during inserts into database

Solution: batch processing. You add a queue & worker, which inserts a batch of 1000 records at once

# Use case

- Simple Web Analytics
  - Id, user_id, url, pageviews_count
- It starts to be successful...
- ...and even more successful
  - Worker cannot keep with the pace, the queue is growing
  - You add more workers, database is clearly a bottleneck
  - Solution? Sharding (horizontal partitioning)
  - You need to rewrite the whole backend of your application

# Use case

- Simple Web Analytics
  - Id, user_id, url, pageviews_count
- It starts to be successful...
- ...and even more successful
- ...and more & more successful
  - You find out that you have chosen too few shards
  - Welcome to Resharding Hell

# Use case

- Simple Web Analytics
    - Id, user_id, url, pageviews_count
- It starts to be successful...
- ...and even more successful
- ...and more & more successful
- You have so many hardware that you are starting to see disk failures quite often
    - You need replication

# NoSQL

- We encounter problems related to management and analysis of data, where relational-database based approaches are not the most convenient ones

- NoSQL means not RDBMS

# What is in “offer” or RDBMS

- Efficient
- Reliable
- Convenient
- Secure
- Multi-user
- Storage and access to vast amount of persisted data

# What is in “offer” or RDBMS

- Efficient
- Reliable
- Convenient
- Secure
- Multi-user
- Storage and access to vast amount of persisted data

Sometimes this package is more than we actually need

# What is in “offer” or RDBMS

- Efficient
- Reliable
- **Convenient**
- Secure
- Multi-user
- Storage and access to vast amount of persisted data

# RDBMS are convenient

- Simple data model
- Declarative query language
- Transactions
- Data constraints

- Everything plays well together, is more or less standardized
- Everybody is familiar with it
- Relational Database as an integration point between applications

# Problems

- Impedance mismatch
  - In memory data structures vs relational model
  - tuple /row vs nested records, lists, hashes
- ORM is not a solution
  - If you forget about your database, it will surely remind you that it is there
- Distributed world
  - Rise of the Web & logging of everything we do on it

# When it is not that convenient

- We have non-relational data
  - Complicated preprocessing to get data into tables
- We do not need complicated queries
  - Maybe a simple key-value fetch would be enough
- We do not need total data safety & consistency
  - But rather want it to be a bit faster

# End of database as an integration point

- HTTP protocol & text-based API (XML, JSON)
  - These are rich data structures!
- No need to expose my data structure
  - No need to keep it standardized

# New types of applications - a tradeoff

- Simple data model
- Reliability is not that critical - we can redo
- Persistence - we are ok with simple, huge text files

- HUGE amounts of data ⇒ requirement for a distributed setup
- Speed of processing is an important factor

# Brewer’s CAP Theorem

- Consistency
- Availability
- Partition Tolerance

# Brewer’s CAP Theorem

- Consistency
  - Two customers will not buy the last airplane ticket
  - Relational databases can handle thanks to their ACID properties (Transactions)
- Availability
- Partition Tolerance

# Brewer’s CAP Theorem

- Consistency
- Availability
  - The service must be available
  - Amazon: +0.1s in response time means 1% drop in sales
  - Google: +0.5s of latency lowers the traffic by 1/5
- Partition Tolerance

# Brewer’s CAP Theorem

- Consistency
- Availability
- Partition Tolerance
  - If we have a distributed system, partitions **will** exist if there is a connectivity problem between servers in zone A and servers in zone B
  - If there is a server, which can handle the request, than the system should continue to operate and work correctly

# Brewer’s CAP Theorem

- You can have only two of those
    - Consistency
    - Availability
    - Partition Tolerance

# Brewer’s CAP Theorem

- You can have only two of those
    - Consistency
    - Availability
    - **Fixed: Partition Tolerance**

# Brewer’s CAP Theorem

- You can have only two of those
  - Consistency
  - Availability
  - **Fixed: Partition Tolerance**
- What would you pick?
  - Consistency
  - Availability

# BTW: Why do we need a distributed system?

- Data do not fit into a single machine
- Geographical proximity
  - Orders from Europe in DC in Ireland
  - Orders from New Zealand in DC in Australia
- Sharding
- Replication
  - Master-slave
  - peer-to-peer

# Brewer’s CAP Theorem

- If you need to scale then scale horizontally
  - Scale out instead of scale up
- You should sacrifice consistency
  - And learn to live with *eventual consistency*

# NoSQL

- Naturally distributed
    - Sharding & replication built-in
- Lower consistency, Higher availability
    - Eventual consistency
- Flexible schema
    - Schemaless...really?
- Often without nice declarative queries

# Weblog example

- CSV: UserID, URL, timestamp, additional-info
- Query: Find all records
  - For a given UserID
  - For a given URL
  - Between two timestamps
  - Having something special in additional-info
- None requires SQL features
- All can be executed in parallel

# Weblog example

- CSV: UserID, URL, timestamp, additional-info
- Query: Find all pairs of UserIDs, which access the same URL
- OK, we would benefit from a JOIN here
- ...but honestly, it is a weird query :)

# Weblog example

- CSV: UserID, URL, timestamp, additional-info
- CSV: UserID, name, age, gender
- Query: Find an average age of users accessing this URL
- SQL is nice for such use cases
- ...consistency is not that important

# Wikipedia

- A huge set of structured and unstructured data
- Leading paragraph of all pages about US Presidents prior to 1900
- What would be the schema?
- Consistency is not important at all

# Polyglot persistence

Individual parts of an application have different requirements for storage

Financial data - safety, ACID, relational databases

E-shop cart - something fast, we can survive glitches from time to time

# What NoSQL databases work with?

- Relational DB - rows & tables
- NoSQL DB - aggregates
  - Complex data structure, with arrays, nesting
- A unit of work for most use cases
  - consistency , atomicity on a level of an aggregate
- Perfect for horizontal scaling
  - Data of one aggregate always on a single node

# Example: users

```
{
“id” : 1,
name”: “Martin”,
“billingAddress”: [{city: Chicago}]
}
```

# Orders

```
{
“id” : 99,
“customer_id” : 1,
“orderItems”: [{“productId”: 27, “price”: 35.47,
“productName”: “NoSQL Distilled” },{...}],
“shippingAddress” : {“city”: {…}},
“paymentInfo”: {…}
}
```

# Or  maybe customers with orders?

```
{
“id” : 1,
name”: “Martin”,
“billingAddress”: [{city: Chicago}],
orders” : [{
        “id” : 99,
        “customer_id” : 1,
        “orderItems”: [{“productId”: 27, “price”: 35.47, “productName”: “NoSQL Distilled” },{...}],
        “shippingAddress” : {“city”: {…}},
        “paymentInfo”: {…}
}]
}
```

# How to define aggregates?

- I need to think about my uses cases when I design aggregates
    - Compared to SQL approach
- Customer view
- Manager View
    - GROUP BY accross aggregates

# Aggregates and ACID

- Atomicity usually on an aggregate level
- Everything else must be handled on an application side

# Types of NoSQL systems

- MapReduce family - “computing layer”
- Key-value storage
- Column-oriented storage
- Document storage
- Graph databases