NoSQL

Source:
Pramod J. Sadalage and Martin Fowler
NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Pearson Education, 2013

Software Engineering
Rochester Institute of Technology

Objectives

- Introduce some key concepts behind the NoSQL family of databases
  - Why NoSQL?
  - Why relational databases?
  - Impedance mismatch – relational models are limiting

- Application and integration databases
- Distributed database clusters
- Introduce aggregate data models
  - Key-Value data models
  - Document data models
  - Column-family store data models

Why NoSQL?

Why Relational Databases?

- For decades, relational databases have been the default choice for serious data storage
  - **Persistence**: provide a “backing” store for volatile memory
    - More flexible than a file system for storing large amounts of memory and accessing it in small bits
  - **Concurrency**: multiple applications accessing shared data
    - Transactions
  - **Integration**: multiple applications store their data in a single database
  - **Standard**: developers and database professionals can easily move between projects and technologies

Impedance Mismatch

- Impedance Mismatch: the difference between the relational model and the in-memory data structures
  - Relational model: everything is a relation – tables of rows
  - Structure and relationships have to be mapped
    - Rich, in-memory structures have to be **translated** to relational representation to be stored on disk
    - Translation: impedance mismatch

Impedance Mismatch

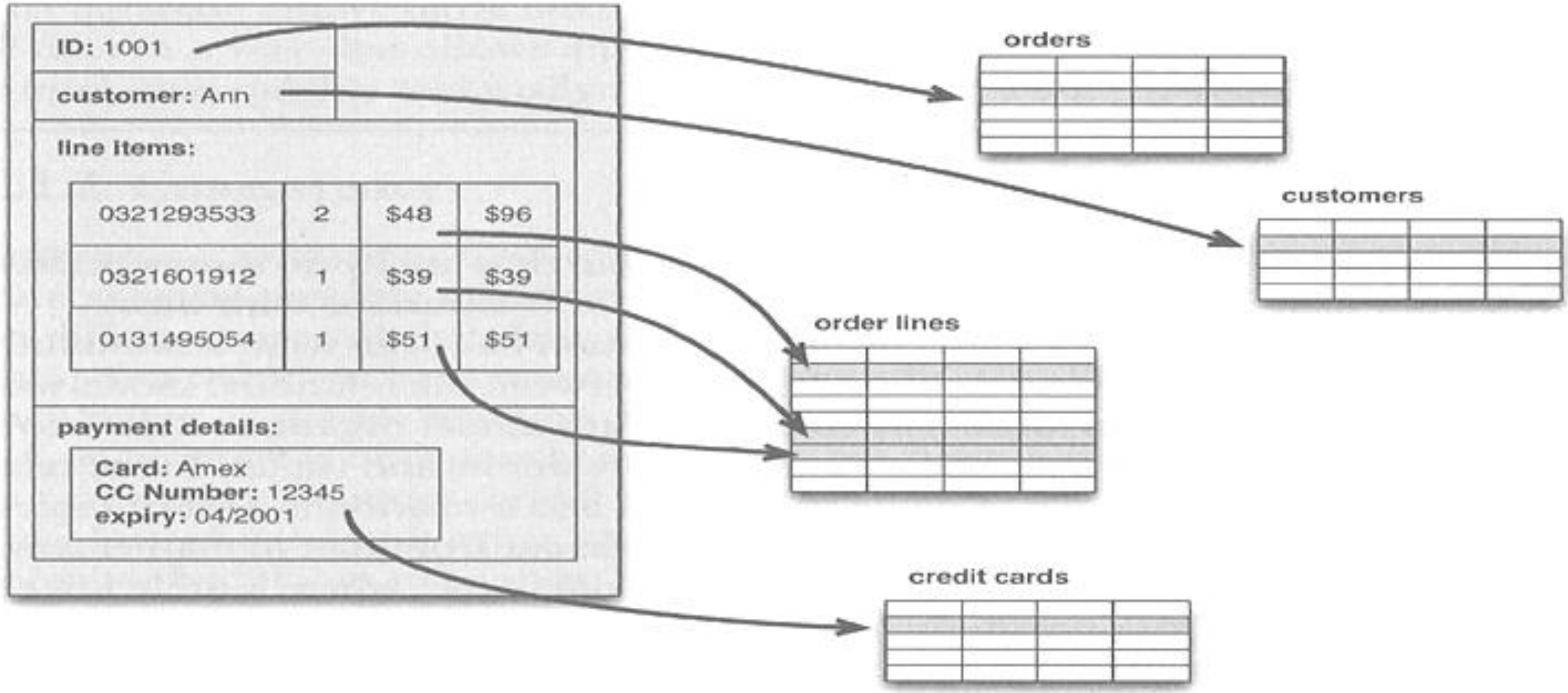


Figure 1.1 *An order, which looks like a single aggregate structure in the UI, is split into many rows from many tables in a relational database*

Impedance Mismatch (cont.)

- Object-oriented programming languages became popular in 1990s, but object-oriented databases did not
- Entrenched relational database vendors
  - Database as an **integration mechanism**
  - **Standard** data manipulation language (SQL)
  - Professional divide between application developers and database administrators
  - Object-relational mapping frameworks ease the grunt work
    - But often result in serious **performance issues**

Application and Integration Databases

- Integration databases are a convenient and powerful method for **integrating multiple applications** developed by different teams
  - Common data model, common data store
- Integrate many applications becomes (dramatically) more **complex** than any single application needs
  - Changes to the data model must be **coordinated**
  - **Different** structural and performance **needs** for different applications
  - Database **integrity** becomes an issue
- Instead, treat the database as an <u>application</u> database
  - Single application, single development team
  - Provide alternate integration mechanisms

Alternate Integration Mechanism: Services

- More recent push to use Web Services where applications integrate over HTTP communications
  - XML-RPC, SOAP, REST
- Results in **more flexibility** for exchange data structure
  - XML, JSON, etc.
  - Text-based protocols
- Results in letting application developers **choose database**
  - Application databases
  - Relational databases are often still an appropriate choice

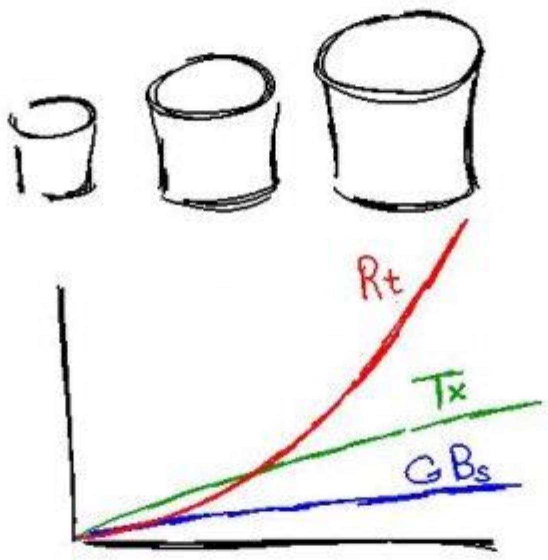
The Attack of the Clusters

- The 2000s saw the web grow enormously
  - Web use tracking data, social networks, activity logs, mapping data, etc.
  - Huge websites serving huge numbers of visitors
- To handle the increase in data and traffic required more computing resources
- Instead of building bigger machines with more processors, storage, and memory, use **clusters** of small, commodity machines
  - Cheaper, more resilient
- But relational databases are not designed to be run on clusters

Clustered Relational Databases

- Clustered relational databases such as Oracle Real Application Clusters (RAC) and Microsoft SQL Server still work against a **single database** disk subsystem
  - A cluster-aware file system and a highly-available disk subsystem
  - Disk subsystem is a **single point of failure**
- Can also partition the database into functionally distinct subsets ("**sharding**")
  - Each application has to keep track of which database server to talk to for each bit of data
  - <u>Lose cross-shard querying, referential integrity, transactions, or consistency control</u>
- Commercial relational database licenses are typically per node, raising overall cost for clusters

“sharding”



Size and Transaction Volume (linear growth):

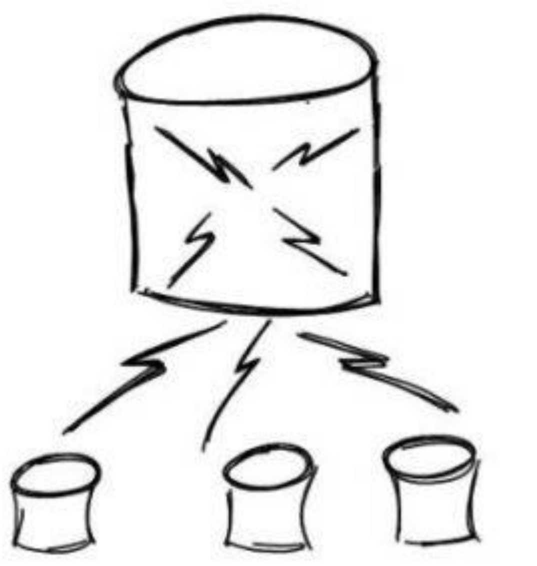
- CPU
- Memory
- Disk

Response time (exponential growth)

You cannot add an unlimited number of CPUs (or processing cores) and see a commensurate increase in performance without also improving the memory capacity and performance of the disk drive subsystem

http://www.agildata.com/database-sharding/

“sharding”



Advantages (smaller databases):

- Easier to manage
- Faster
- Reduce costs

Disadvantages (challenges):

- Reliability (backups, redundancy, failover, disaster recovery)
- Distributed queries (cross-shard joins)
- Auto-increment key management
- Multiple shard schemes (session/transaction or statement based sharding)

http://www.agildata.com/database-sharding/

New Opportunities

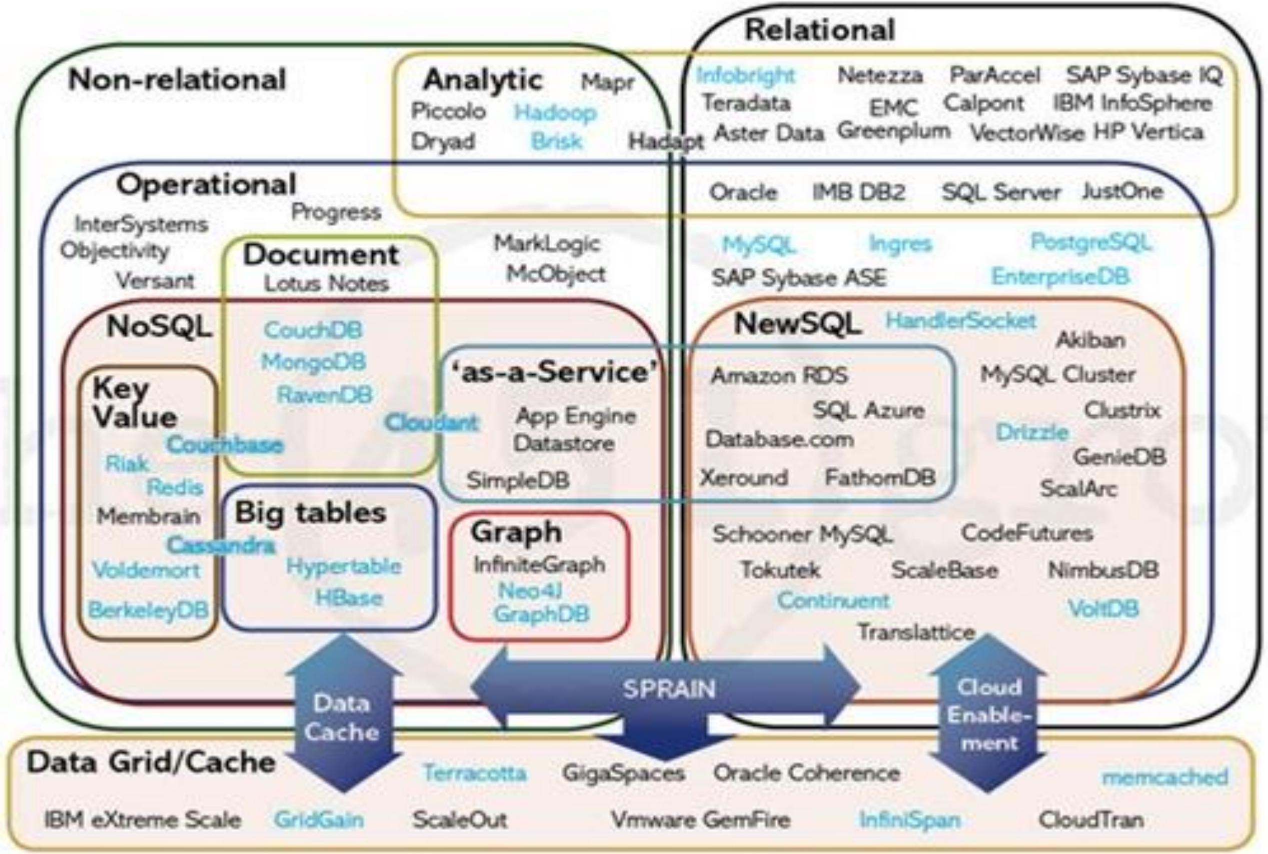
- The mismatch between relational databases and clusters led some organizations to consider **alternative approaches** to data storage
  - Google and Amazon have been very influential
    - Large clusters and huge amounts of data
    - Google: BigTable paper; Amazon: Dynamo paper
- Few organizations need the scale of Google or Amazon, but many organizations are seeing an **exponential increase** in data storage and use
- New styles of databases explicitly designed to run on clusters

The Emergence of NoSQL

- Historical note: ‘NoSQL’ was first used to name an open-source relational database development led by Carlo Strozzi
  - Based on ASCII storage of tables manipulated through Unix shell scripts instead of SQL
  - No influence on databases under the current use of the term ‘NoSQL’
- Current use of the phrase came from a conference meetup discussing “open-source, distributed, nonrelational databases”
  - Talks from Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB, MongoDB
- No generally accepted definition of ‘NoSQL’
  - Often “Not only SQL”

Characteristics of NoSQL Databases

- They do not use SQL and the relational model
  - Some do have **query languages** which are similar to SQL to be easy to learn and use
- Mostly open-source projects
- **Designed to be distributed – clustered**
  - No expectation of ACID properties
  - Range of options for consistency and distribution
- **Schema free**
  - Freely add fields to records without having to define any changes in structure first
  - Non-uniform data and custom fields
- A noDefinition of NoSQL: An ill-defined set of mostly open-source databases, mostly developed in the early 21st century, and mostly not using SQL

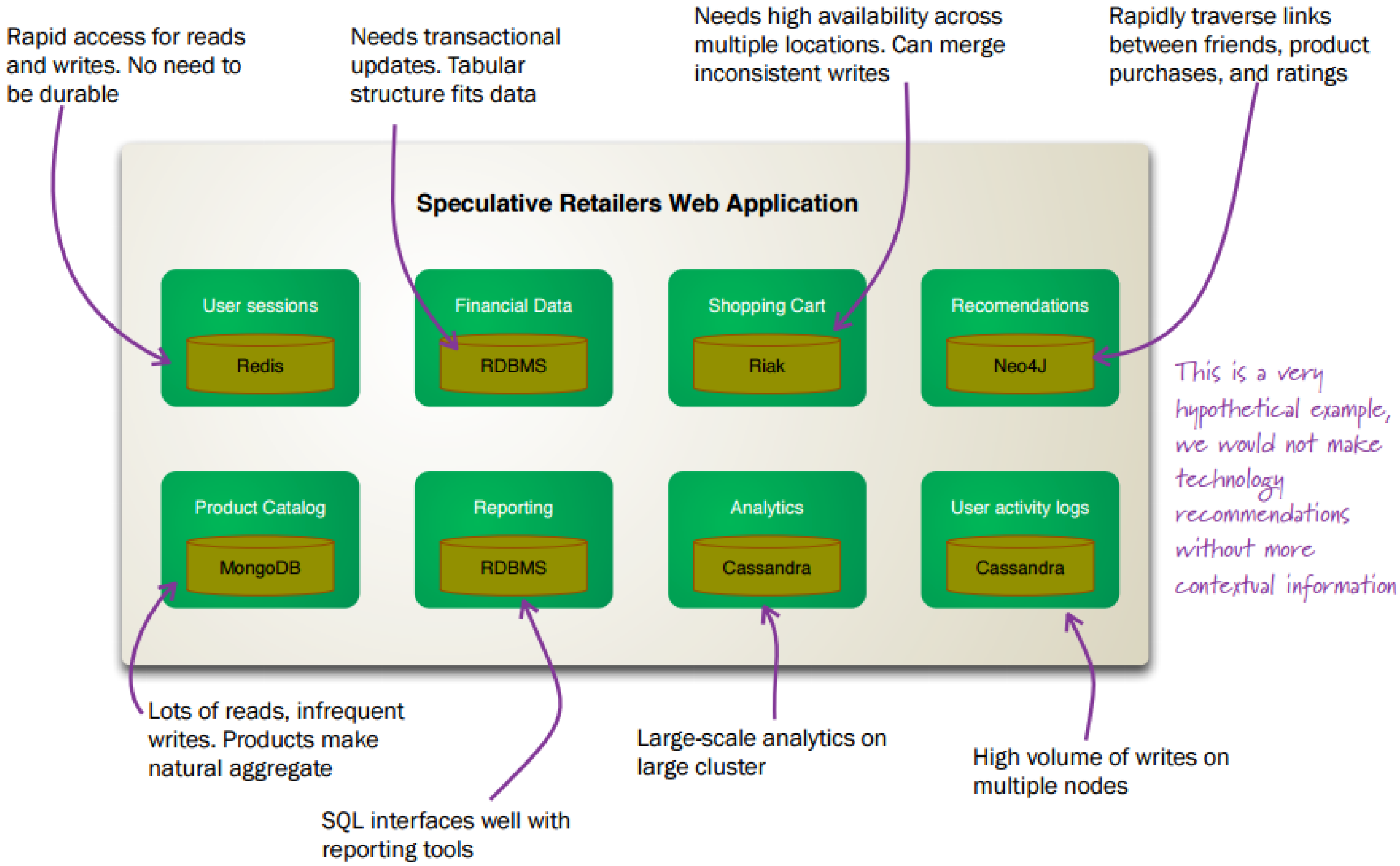


https://blogs.the451group.com/information_management/2011/04/15/nosql-newsql-and-beyond/

Software Engineering
Rochester Institute of Technology

Polyglot Persistence (Fowler)

what might Polyglot Persistence look like?



This is a very hypothetical example, we would not make technology recommendations without more contextual information