Big Data and NoSQL

Very short history of DBMSs

- The seventies:
  - IMS – end of the sixties, built for the Apollo program (today: Version 15) and IDS (then IDMS), hierarchical and network DBMSs, navigational
- The eighties – for twenty years:
  - Relational DBMSs
  - The nineties: client/server computing, three tiers, thin clients

Object Oriented Databased

- In the nineties, Object Oriented databases were proposed to overcome the impedence mismatch
- They influenced Relational Databases, and disappeared

Big Data

- Mid 2000s, Big Data:
  - Volume:
    - DBMSs do not scale enough for some applications
  - Velocity:
    - Computational speed
    - Development velocity:
      - DBMS require upfront schema design and data cleaning
  - Variety:
    - Schemas conflict with variety

BigData platforms

- The google stack:
  - Hardware: each Google Modular Data Center houses 1.000 Linux servers with AC and disks
  - GFS: distributed and redundant FS
  - MapReduce
  - BigTable, on top of GFS
- Hadoop – open source
  - HDFS, Hadoop MapReduce
  - HBase
  - SQL on Hadoop: Apache Hive, IBM Jaql, Apache Pig, Cloudera Impala

NoSQL

- Giving up something to get something more
- Giving up:
  - ACID transactions, to gain distribution
  - Upfront schema, to gain
    - Velocity
    - Variety
  - First normal form, to reduce the need for joins
- Different from NewSQL

Types of NoSQL systems

- Key-value stores (Amazon Dynamo, Riak, Voldemort…)
- Document databases:
  - XML databases: MarkLogic, eXist
  - JSON databases:
    - CouchDB, Membase, Couchbase
    - MongoDB
- Sparse table databases:
  - HBase
- Graph databases:
  - Neo4j

NewSQL

- Column databases
- In memory databases

NoSQL

Why NoSQL

- Impedance mismatch
- Restrictive schema
- Integration databases -> application databases
- Cluster architecture
  - Google BigTable
  - Amazon Dynamo

NoSQL

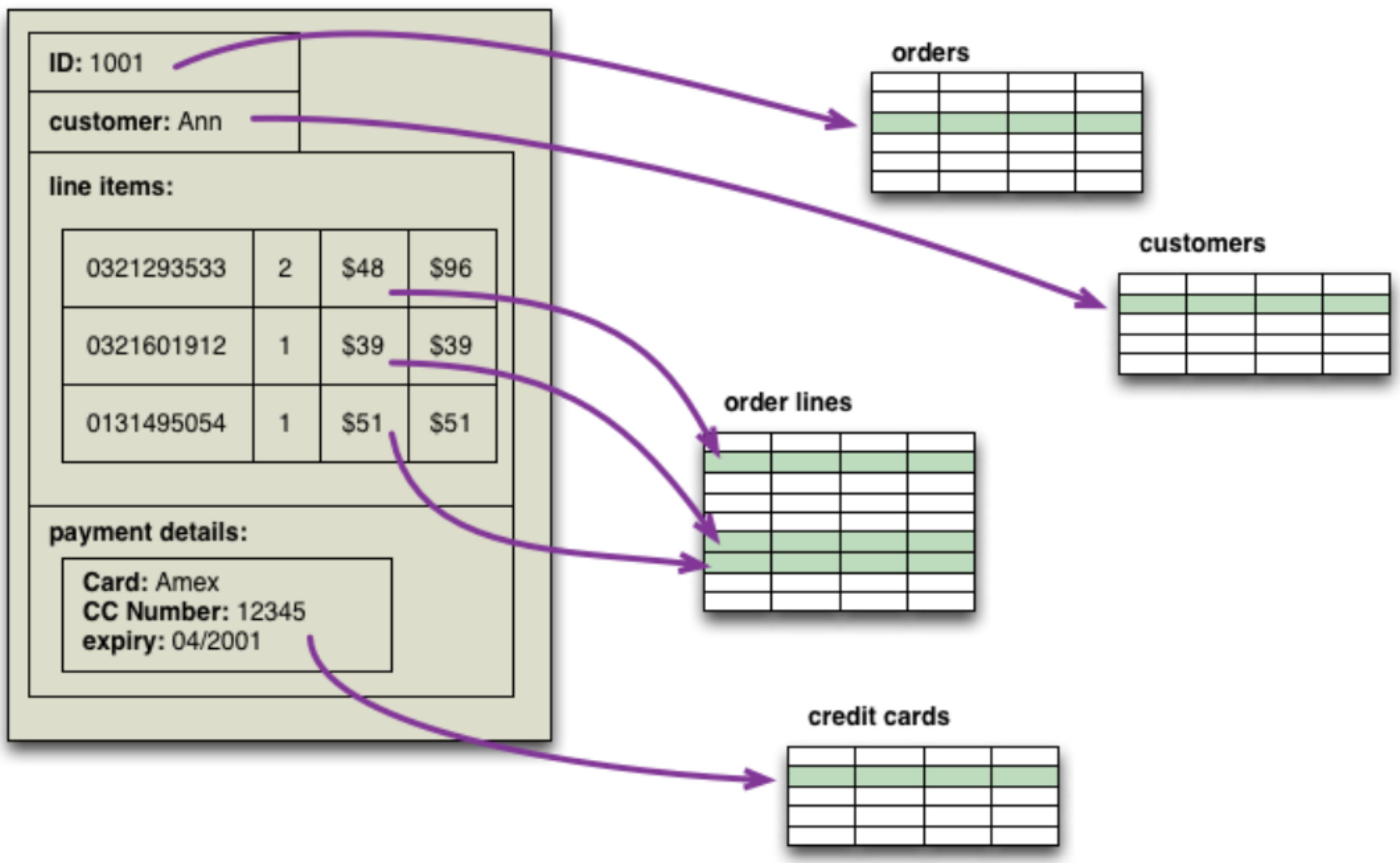
- A set of ill-defined systems that are not RDMBS
- Usually do not support SQL
- Are usually Open Source (not always)
- Often cluster-oriented (not always), hence no ACID
- Recent (after 2000)
- Schema free
- Oriented toward a single application
- It is more a ‘movement’ than a technology

Aggregate data model

- NoSQL data models:
  - Aggregate data models:
    - Key-value
    - Document
    - Column family
  - Graph model

Aggregate orientation

http://martinfowler.com/bliki/AggregateOrientedDatabase.html



Aggregate data models

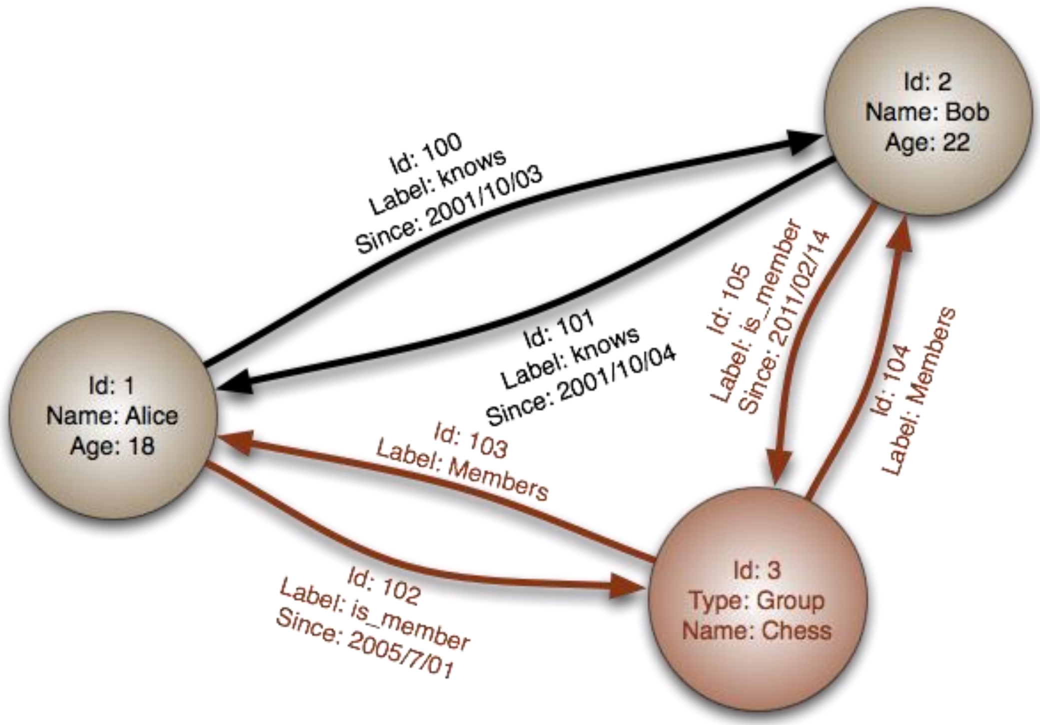
- Key value stores: the database is a collection of <key,value> pairs, where the value is opaque (Dynamo, Riak, Voldemort)
- Document database: a collection of documents (XML or JSON) that can be searched by content (MarkLogic, MongoDB)
- Column-family stores: a set of <key, record> pair (BigTable, HBase, Cassandra)
  - Columns are grouped in ‘column families’

Key-value stores implementation

- Implementation model:
  - Key-based distribution of the pairs on a huge farm of inexpensive machines
  - Constant time access
  - Constant time parallel execution on all the pairs
  - Flexible fault-tolerance
  - MapReduce execution model
  - Amazon Dynamo, Riak, Voldemort

Graph databases

- Set of triples <nodeid, property, nodeid> (FlockDB, Neo4J)



Schemaless databases

- Schema first vs. schema later
- Homogeneous vs. non homogeneous

Materialized views

- OLAP applications greatly benefit from materialized views
- Materialized views can be used to regain the flexibility of the relational model

Distribution Models

- Sharding: splitting data among nodes according to a key
- Master-slave replication
  - No update conflict
  - Read resilience
  - Master election
- P2P replication
  - No single point of failure
- Sharding + replication

Consistency

- Write-write conflicts: avoiding to lose an update
- Read consistency:
  - Fresh data
  - No intermediate data
  - Session consistency
- Transactional consistency
  - Writing values that are based on data that is not valid any more

The CAP Theorem

- You cannot have all of:
  - Consistency
  - Availability
  - Partition tolerance
- A trade-off between consistency and latency
- Relaxing consistency
  - Two writes in the same cart
- Relaxing durability

Consistency

- Quorums: in a P2P system, an operation is successful if it gets a quorum of confirmations
  - The write quorum:
    - W > N/2
  - The read quorum:
    - R+W > N
- Version stamps:
  - Counter, GUID, content hash, time-stamp
  - Consistent write after read

Map-Reduce

- Map: maps each object to a set of <key, value> pairs
- Shuffle: collect all pairs with the same <key> to the same node
- Reduce: for each set {<k,v1>,…,<k,vn>} produce a result
- Combine-Reduce:
  - If reduce is associative, all same-key pairs can be combined locally before shuffling

Map-Reduce

- Map: <key1, value1> -> set(<key2,value2>)
- Combine: <key2,set(value2)> -> <key2,value2>
- Reduce: <key2,set(value2)> -> <key2,value2>
- Input of Map and output of Reduce must be put somewhere
  - HDFS
  - Main memory (Spark)
- Examples
  - OrderLine(Product, Amount, Date): group by product

Key-Value Databases

- Basically, a persistent hash table
- Sharding + replication
- Consistency
  - Single object
  - Riak: for each bucket (data space):
    - Newest write wins / create siblings
    - Setting read / write quorum
- Query
  - By key
  - Full store scan (not always provided)
- Uses: session information, user profiles, shopping cart data by userid…

Document Databases: MongoDB

- One instance, many databases, many collections
- JSON documents with _id field
- Sharding + replication

Consistency

- Master/slave replication
  - Automated failover, server maintenance, disaster recovery, read scaling
- Master is dynamically re-elected over fail
- One can specify a write quorum
- One can specify whether reads can be directed to slaves

Querying

- CouchDB: query via views (virtual or materialized)
- MongoDB:
  - Selection, projection, aggregation

Column-family Stores

- A ‘column-family’ (similar to a ‘table’ in relational databases) is a set of <key,record> pairs
- Records are not necessarily homogeneous
- Confusing terminology
  - Column: a field such as «age:=35»
  - Supercolumn: «address:={city:=‘Pisa,…}»
  - Row: a pair key-record (record: set of columns):
    - <johnsmith_001657, {name:=‘John’, age:=35}>
  - Column family: set of related rows
  - Keyspace: set of column families

Consistency

- In Cassandra:
  - The DBA fixes the number of replicas for each keyspace
  - the programmer decides the quorum for read and write operations (1, majority, all…)
  - Transactions:
    - Atomicity at the row level
    - Possibility to use external transactional libraries

Queries (Cassandra)

- Row retrieval:
  - **GET** *Customer*[‘*johnsmith00012*’]
- Field (column) retrieval:
  - **GET** *Customer*[‘*johnsmith00012*’][‘*age*’]
- After you create an index on age:
  - **GET** *Customer* **WHERE** *age = 35*
- Cassandra supports CQL:
  - Select-project (no join) SQL

Graph Databases

- A graph database stores a graph
- We will talk later about a specific graph model: RDF
- Example: Neo4J

Consistency

- Graph databases are usually not sharded and transactional
- Neo4J supports master-slave replication
- Data can be sharded at the application level with no database support, which is quite hard

Querying: Cypher

```
MATCH (me {name:"Giorgio"})
RETURN me
```

Querying: Cypher

```
MATCH (expert)
    -[:WORKED_WITH]->
    (neodb:Database
        {name:"Neo4j"})
RETURN neodb, expert
```

Querying: Cypher

```
MATCH (me {name:"Giorgio"})
MATCH (expert)
      -[:WORKED_WITH]->
      (neodb:Database {name:"Neo4j"})
MATCH path = shortestPath( (me)-[:FRIEND*..5]-(expert) )
RETURN neodb, expert, path
```

Querying: Cypher

MATCH pattern matches

WHERE filtering conditions

RETURN what to return

ORDER BY properties to order by

SKIP nodes to skip from the top

LIMIT limit results

Polyglot Persistence

- Transactional RDBMSs, DSSs and NoSQL systems have different strength and it is natural to combine all of them
- However, such a heterogeneous environment can create huge problems of maintenance and security

Sources

- P. J. Sadalage, M Fowler, NoSQL Distilled, Addison Wesley