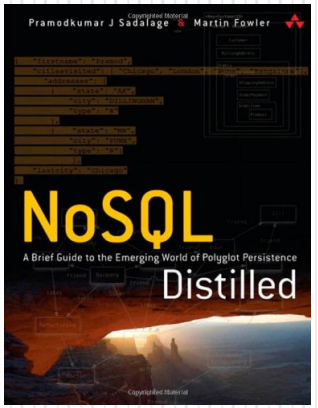
NoSQL systems: Implementation

Riccardo Torlone

Università Roma Tre



Key-Value Database

- A simple hash table accessible only through its primary key
- Basically a table with two columns: ID and VALUE
- The value is a blob that the system just stores: it can be text, JSON, XML, and anything else.
- Operations:
  - get the value for the key
  - put a value for a key (if the key already exists the corresponding value is overwritten)
  - delete a key from the data store.

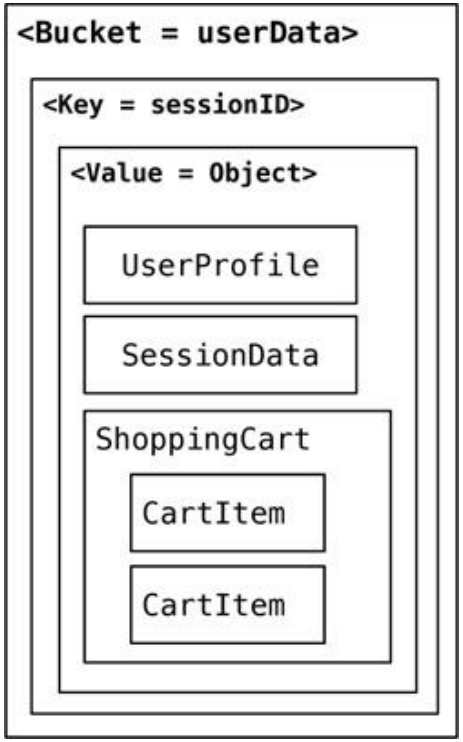
Oracle	Riak
database instance	Riak cluster
table	bucket
row	key-value
rowid	key

Popular key-value databases

- Redis (often referred to as Data Structure server) [http://redis.io/]: it is more general since it supports storing lists, sets, hashes and can do range, diff, union, and intersection operations,
- Riak [http://basho.com/riak/],
- Amazon DynamoDB (not open-source) [https://aws.amazon.com/dynamodb/],
- Microsoft Azure Cosmos DB [https://azure.microsoft.com/services/cosmos-db]
- Memcached DB and its flavors [http://memcached.org/],
- Hazelcast (in-memory database) [https://hazelcast.com/],
- Oracle NoSQL (not open-source) [http://www.oracle.com/technetwork/database/database-technologies/nosqldb/]

Riak

- Store keys into buckets, which are just a way to group the keys



- Usually, one bucket store aggregates of the same type (domain buckets)
- Different types of aggregates can be stored in the same bucket

Consistency in Key-Value Databases

- Peer-to-peer architecture.
- Consistency applicable only for operations on a single key (get, put, delete).
- Optimistic writes can be performed
- Eventually consistent model in distributed implementations
- Riak has two ways of resolving update conflicts
  - the newest write wins and older writes loose
  - both values are returned allowing the client to resolve the conflict.

Consistency set up in Riak

- During the bucket creation

```
Bucket bucket = connection
    .createBucket(bucketName)
    .withRetrier(attempts(3))
    .allowSiblings(siblingsAllowed)
    .nVal(numberOfReplicasOfTheData)
    .w(numberOfNodesToRespondToWrite)
    .r(numberOfNodesToRespondToRead)
    .execute();
```

- w=nVal: data in every node must be consistent
  - But it decreases the write performance of the cluster
- allowSiblings(false): last write wins.

Transactions

- Different products have different specifications of transactions
- Generally speaking, there are no guarantees on the writes
- Riak uses the concept of quorum
  - set of the w and r values during the write API call
  - n = 5 (replication factor), w =3, r =2 : the write is successful only when it is written on at least three nodes, the read is successful only when it is read on at least two nodes
  - the cluster can tolerate n - w = 2 nodes being down for writes
  - r+w>n guarantees a correct read

Query Features

- Only query by the key!
- It is not possible to use some attribute of the value column
- What if we don't know the key?
  - Some system allow the search inside the value (e.g., Riak Search)
  - The key needs to be suitably chosen (e.g., session ID for storing session data)

CRUD operations in Riak

```
Bucket bucket = getBucket(bucketName);
IRiakObject riakObject = bucket.store(key, value).execute();
```

```
Bucket bucket = getBucket(bucketName);
IRiakObject riakObject = bucket.fetch(key).execute();
byte[] result = riakObject.getValue();
String value = new String(result);
```

Riak provides an HTTP-based interface: all operations can also be performed from the Web browser or on the command line using cURL (a command line tool for doing all sorts of URL manipulations and transfers)

Sharding

- The value of the key determines on which node the key is stored
- Settings in Riak:
  - `n` (number of nodes to store the key-value replicas),
  - `r` (number of nodes to read before a read is considered successful),
  - `w` (number of nodes to write before a write is considered successful).
  - These settings allow us to fine-tune node failures for reads or writes.
  - They can be set as defaults during bucket creation.
  - Rule of quorums: `r+w>n`

Suitable Use Cases

- Storing Session Information
  - Using the `sessionId` as key.
- User Profiles, Preferences
  - Using the `userId` as key.
- Shopping Cart Data
  - Using the `sessionId` or the `userId` as key.

When Not to Use

- Relationships among Data
- Multioperation Transactions
- Query by Data
- Operations by Sets

Document Databases

- The database stores and retrieves documents
- A key-value store where the value is an examinable document
- Documents:
  - can be XML, JSON, BSON, and so on
  - are self-describing, hierarchical tree data structures
  - can consist of scalar values, collections, and maps
  - are structurally similar but not identical to each other

Oracle	MongoDB
database instance	MongoDB instance
schema	database
table	collection
row	document
rowid	_id
join	DBRef

Documents in a document database

```
{ "firstname": "Martin",
  "likes": [ "Biking", "Photography" ],
  "lastVisited":"Boston",
}

{ "firstname": "Pramod",
   "citiesvisited": [ "Chicago", "London", "Pune", "Bangalore" ],
   "addresses": [
     { "state": "AK", "city": "DILLINGHAM", "type": "R" },
     { "state": "MH", "city": "PUNE", "type": "R" }
    ],
  "lastcity": "Chicago"
}
```

Document structure

- The schema of the data can differ across documents,
- Collection is a set of "similar" documents
- In documents, there are no empty attributes - if a given attribute is not found, it was not set or not relevant to the document
- New attributes can be created without the need to define them or to change the existing documents

Popular document databases

- MongoDB [http://www.mongodb.org/],
- CouchDB [http://couchdb.apache.org/],
- Couchbase (https://www.couchbase.com/)
- RethinkDB [https://www.rethinkdb.com/],
- RavenDB [http://ravendb.net/],
- Terrastore [https://code.google.com/p/terrastore/],
- OrientDB [http://www.orientechnologies.com/],
- Each product has some features that may not be found in others.

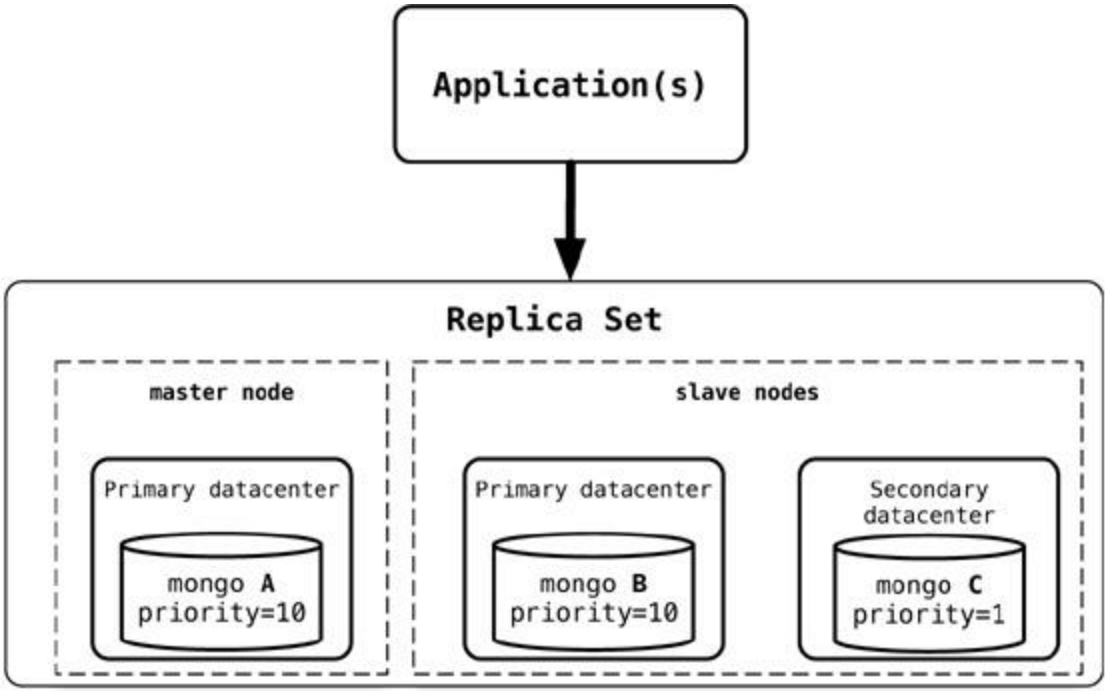
MongoDB

- Each MongoDB instance has multiple databases
  - Similar to a database schema in a RDBMS
- Each database can have multiple collections
  - Similar to a table in a RDBMS
- When we store a document, we have to choose which database and collection this document belongs

  db.collection.insert(document)

CAP in MongoDB

- Master-slave architecture
- A MongoDB database makes use of **replica sets** for consistency and availability



Replica sets

- The replica-set nodes elect the master, or primary, among themselves nodes
  - That closer to the other servers or having more RAM
  - Users can affect this by assigning a priority to a node
- All requests go to the master node
- Data is replicated to the slave nodes and the clients can get to the data even when the primary node is down.
- If the master node goes down, the remaining nodes in the replica set vote among themselves to elect a new master.
- Using replica sets gives you the ability to have a highly available document data store.

Consistency

- You can choose to replicate the writes to all the slaves or a given number of slaves before it returns as successful

```
shopping.insert( { item: "envelopes", qty : 100, type: "Clasp" },
    { writeConcern: { w: "majority", wtimeout: 5000 } } )
```

- You can make sure that writes are written to the master and some slaves by setting WriteConcern to REPLICAS_SAFE (which means at least two):
  - For all writes:

```
DBCollection shopping = database.getCollection("shopping");
shopping.setWriteConcern(REPLICAS_SAFE);
```

  - Individually for each specific write:

```
WriteResult result = shopping.insert(order, REPLICAS_SAFE);
```

- You can increase the read performance by allowing reading from slaves by setting slaveOk
  - For all read:

```
Mongo mongo = new Mongo("localhost:27017");
mongo.slaveOk();
```

  - Individually for each specific read:

```
DBCollection collection = getOrderCollection();
BasicDBObject query = new BasicDBObject();
query.put("name", "Martin");
DBCursor cursor = collection.find(query).slaveOk();
```

Query Features

- Document databases provide different query features.
- CouchDB allows you:
  - to specify complex queries (called views) on documents which can be either materialized or dynamic
  - to implement a view via map-reduce
- In general:
  - You can query the data inside the document without having to retrieve the whole document by its key and then introspect the document.

MongoDB query language

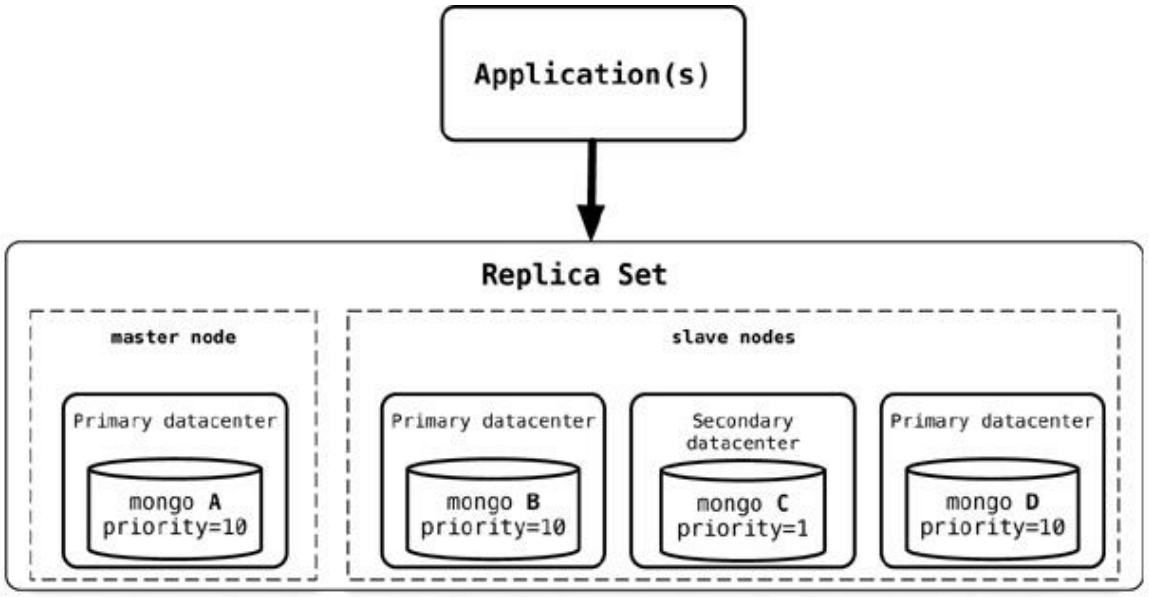
- Expressed via JSON with simple constructs

```
// in orders
{
"orderId":99,
"customerId":"883c2c5b4e5b",
"orderDate":”2014-04-23”,
"orderItems":[
  { "product": {"id":27, "name":"NoSQL Distilled"}, "price": 32.45 }
  { "product": {"id":55, "name":"Java 4 all"}, "price": 41.33 }
 ],
}

db.orders.find()
db.orders.find({"customerId":"883c2c5b4e5b"})
db.order.find({customerId:"883c2c5b4e5b"}, {orderId:1,orderDate:1})
db.orders.find({"orderItems.product.name":/NoSQL/})
```

Scaling for reads in MongoDB

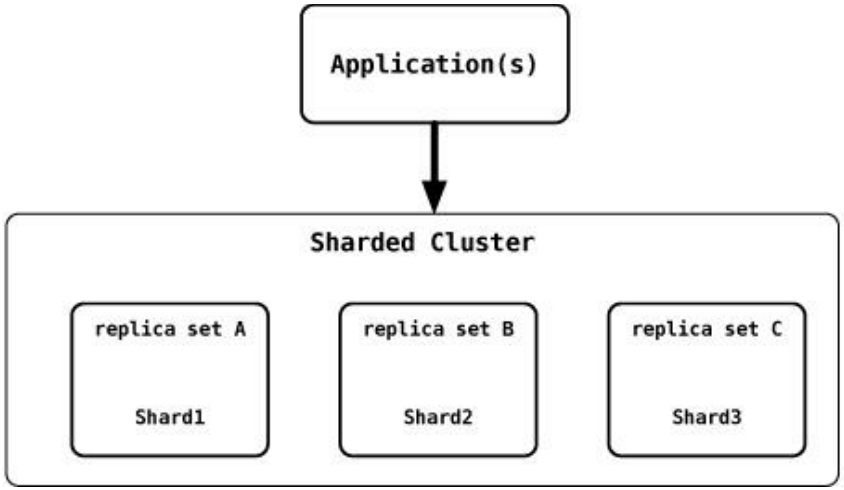
- Heavy-read loads can be supported by adding more read slaves



- Once the new node is started, it needs to be added to the replica set with: `rs.add("mongod:27017");`

Scaling for writes in MongoDB

- Achieved by sharding the data
- The data is then split by certain field and moved to different nodes



- Sharding on the first name of the customer:
  db.runCommand({shardcollection:"ecommerce.customer",key:{firstname:1}})

Suitable Use Cases

- Event Logging
- Content Management Systems, Blogging Platforms
- Web Analytics or Real-Time Analytics
- E-Commerce Applications

When Not to Use

- Complex Transactions Spanning Different Operations
- Queries against Varying Aggregate Structure

Column(-Family) Stores

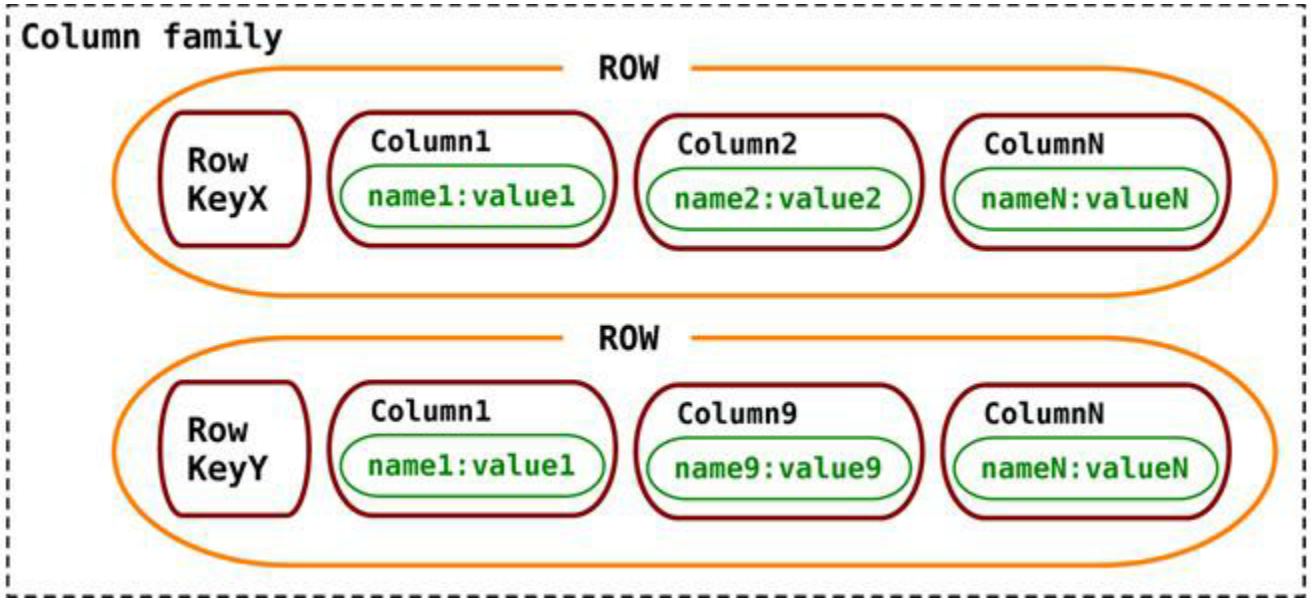
- Data with keys mapped to values
- Values grouped into multiple Column families
- Each column family is a collection of name-value pairs
- Column families group related data that is often accessed together

RDBMS	Cassandra
database instance	cluster
database	keyspace
table	column family
row	row
column (same for all rows)	column (can be different per row)

Popular column-family stores

- Cassandra [http://cassandra.apache.org/],
- HBase [https://hbase.apache.org/],
- Hypertable [http://hypertable.org/],
- BigTable [https://cloud.google.com/bigtable/],
- SimpleDB [https://aws.amazon.com/simpledb/]

Cassandra data model



- In Cassandra the cluster does not have a master node
- Any read and write can be handled by any node in the cluster

Cassandra columns

- The basic unit of storage in Cassandra is a column.
- A column is a name-value pair
  - The name behaves as the key
  - It is always stored with a timestamp
- The timestamp is used to expire data, resolve write conflicts, deal with stale data, and do other things.

```
{
  name: "fullName",
  value: "Martin Fowler",
  timestamp: 12345667890
}
```

Cassandra rows

- A row is a collection of columns attached or linked to a key
- a collection of similar rows makes a column family
- standard column family: the columns are simple

```
//column family
{
//row
  "pramod-sadalage" : {
     firstName: "Pramod",
     lastName: "Sadalage",
     lastVisit: "2019/12/12"
  }
//row
  "martin-fowler" : {
     firstName: "Martin",
     lastName: "Fowler",
     location: "Boston"
  }
}
```

Super columns

- A super column consists of a name and a value which is a map of columns

```
{
  name: "book:978-0767905923",
  value: {
    author: "Mitch Albon",
    title: "Tuesdays with Morrie",
    isbn: "978-0767905923"
  }
}
```

- Cassandra puts the standard and super column families into keyspaces (databases in RDBMS).
- Keyspaces have to be created so that column families can be assigned to them:

```
create keyspace ecommerce
```

Super column family

```
{//row
name: "billing:martin-fowler",
value: {
  address: {
    name: "address:default", value: {fullName: "Martin Fowler", street:"100 N. Main Street", zip: "20145"}
  },
  billing: {
     name: "billing:default", value: {creditcard: "8888-8888-8888-8888", expDate: "12/2016" }
   }
 }
{//row
name: "billing:pramod-sadalage",
value: {
  address: {
    name: "address:default", value: {fullName: "Pramod Sadalage", street:"100 E. State Parkway", zip: "54130"}
  },
  billing: {
      name: "billing:default", value: {creditcard: "9999-8888-7777-4444", expDate: "01/2016« }
      }
  }
}
```

Consistency in Cassandra

- During keyspace (database) creation, the replication factor can be set
- Writes:
  - data is recorded in a commit log and written to an in-memory structure called memtable
  - by default, this is sufficient to consider the write successful
  - consistency setting = QUORUM: the write must propagate to the majority of the nodes before it is considered successful
  - LWW (last write wins) for resolving write-write conflicts
- Reads:
  - consistency setting = ONE (the default): the data from the first replica is returned even if it is stale.
  - consistency setting = QUORUM: majority of the nodes are accessed and the column with the newest timestamp is returned
  - consistency setting = ALL: all nodes will have to respond to reads or writes

Transactions and Availability

- Cassandra does not have transactions in the traditional sense
  - Atomicity at the row level
- There is no master and every node is a peer
- Availability governed by the $(R + W) > N$ formula
- You can tune the availability by changing R and W for a fixed N
- You can add nodes to the cluster to improve the capacity of the cluster

Query Features in Cassandra

```
use ecommerce;
CREATE COLUMN FAMILY Customer
WITH comparator = UTF8Type
AND key_validation_class=UTF8Type
AND column_metadata = [
        {column_name: city, validation_class: UTF8Type}
        {column_name: name, validation_class: UTF8Type}
        {column_name: web, validation_class: UTF8Type}
];
SET Customer['mfowler']['city']='Boston';
SET Customer['mfowler']['name']='Martin Fowler';
SET Customer['mfowler']['web']='www.martinfowler.com';

GET Customer['mfowler'];
GET Customer['mfowler']['web'];


DEL Customer['mfowler']['city'];
DEL Customer['mfowler'];
```

Indexing

- You can index columns other than the keys for the column family

```
UPDATE COLUMN FAMILY Customer
WITH comparator = UTF8Type
AND column_metadata = [{column_name: city,
                        validation_class: UTF8Type,
                        index_type: KEYS}];

GET Customer WHERE city = 'Boston';
```

Cassandra Query Language (CQL)

- An SQL-like language

```
CREATE COLUMNFAMILY Customer (
        KEY varchar PRIMARY KEY,
        name varchar,
        city varchar,
        web  varchar);

INSERT INTO Customer (KEY,name,city,web)
VALUES ('mfowler',
          'Martin Fowler',
          'Boston',
          'www.martinfowler.com');

SELECT * FROM Customer
SELECT name,web FROM Customer
SELECT name,web FROM Customer WHERE city='Boston'
```

Suitable Use Cases

- Event Logging
- Content Management Systems, Blogging Platforms
- Analytics

When Not to Use

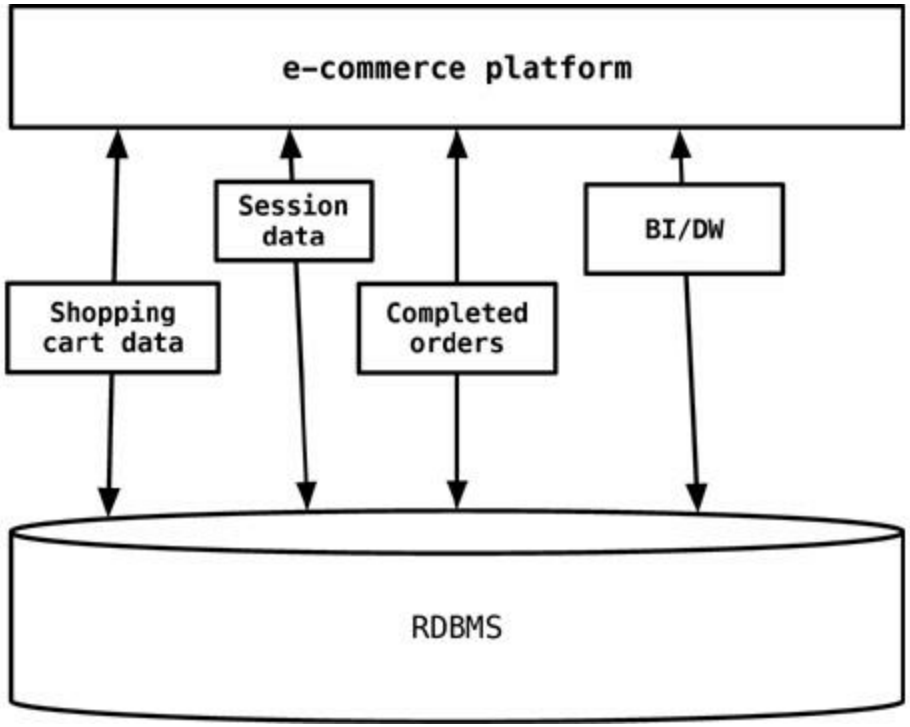
- Systems that require ACID transactions for writes and reads.
- You do not want deal with write-write conflicts

Polyglot Persistence

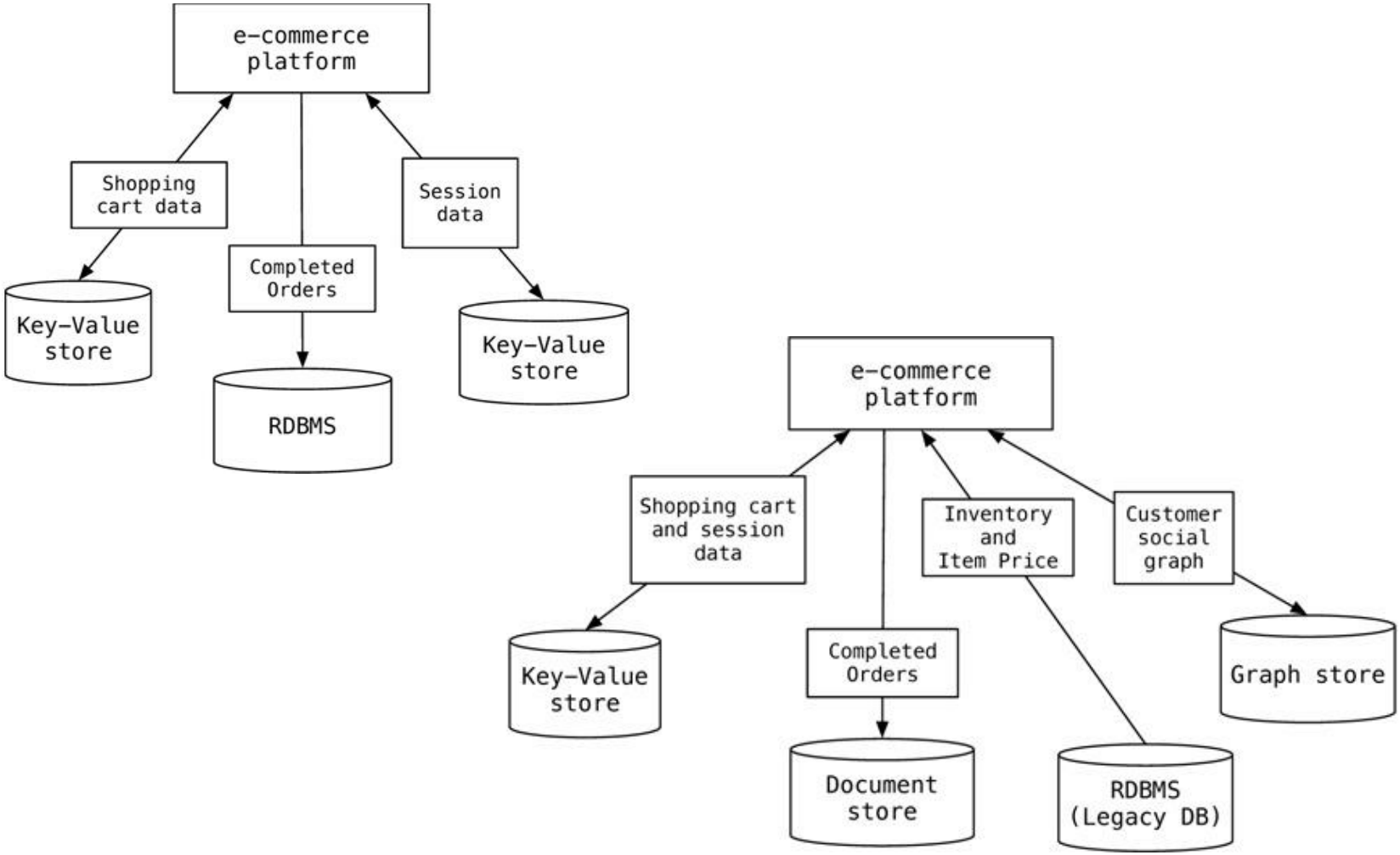
- Different databases are designed to solve different problems.
- Using a single database engine for all requirements
  - storing transactional data,
  - caching session information,
  - traversing graph of customers,
  - performing OLAP operations,
  - …

- …usually leads to non-performant solutions.
- Different needs for availability, consistency, or backup requirements.

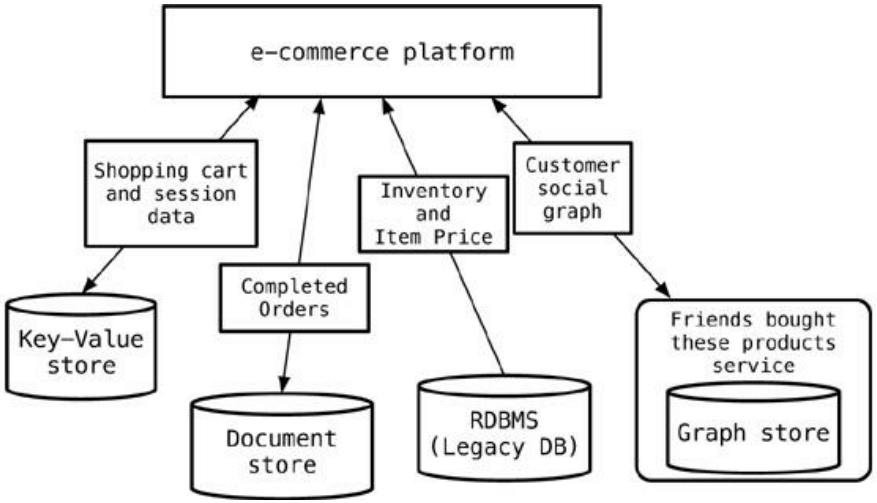
Traditional approach

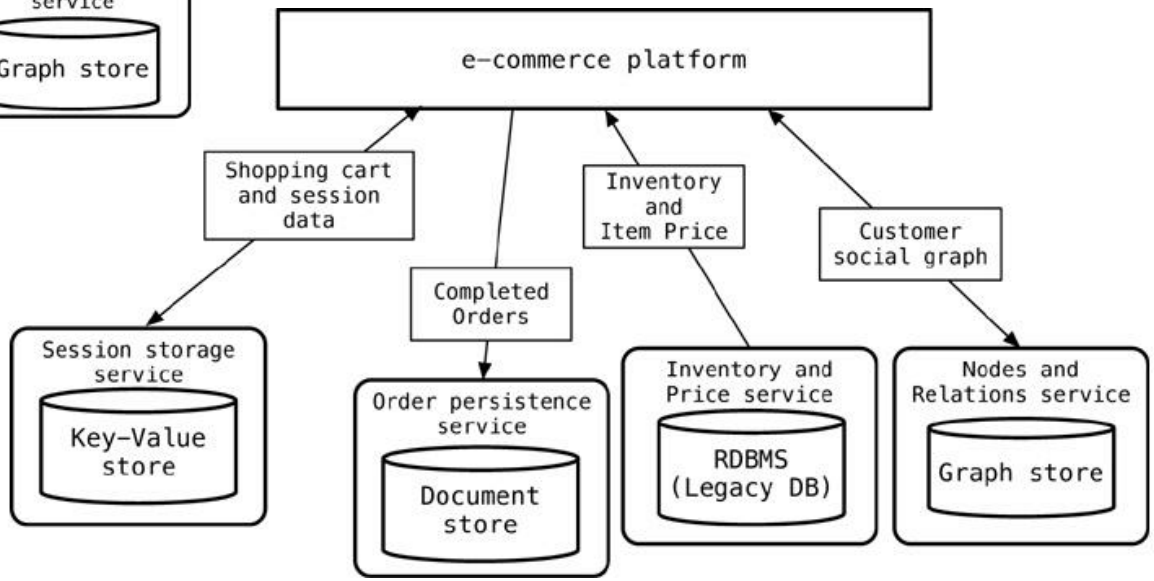


Polyglot Data Store Usages

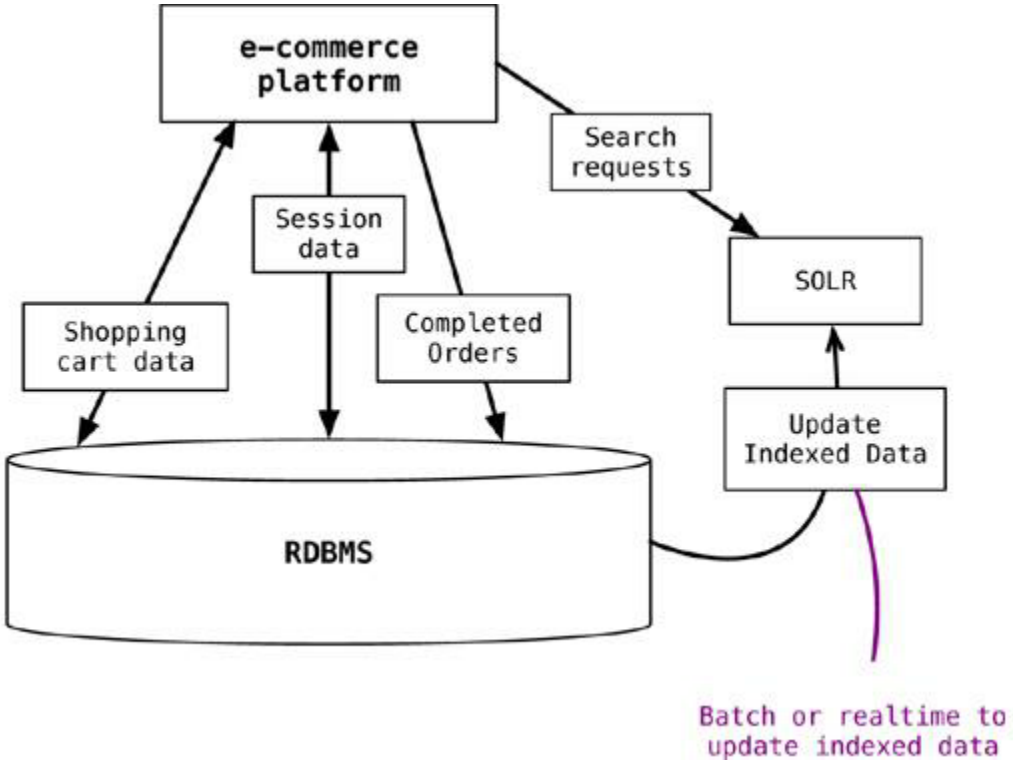


Service-oriented Usage





Expanding for Better Functionality



Polyglot Persistence in Enterprises

- In the world of polyglot persistence, the DBAs will have to become more poly-skilled
  - learn how some of these NoSQL technologies work,
  - how to monitor these systems,
  - back them up and take data out of and put into these systems.
- Security: the ability to create users and assign privileges to see or not see data at the database level. Most of the NoSQL databases do not have very robust security features. The responsibility for the security lies with the application.
- Issues such as licensing, support, tools, upgrades, drivers, auditing, and security come up.
- Many NoSQL technologies are open-source and have an active community of supporters; also, there are companies that provide commercial support.
- Tool vendors and the open-source community are releasing tools such as MongoDB Monitoring Service, Datastax Ops Center, or Rekon browser for Riak.
- ETL tools need to access NoSQL data stores. The ETL tool vendors are including NoSQL databases: Pentaho can talk to MongoDB and Cassandra.
- Every enterprise runs analytics of some sort. The need to scale for writes are a great use case for NoSQL databases.

Choosing Your Database

- The two main reasons to use NoSQL technology are:
  - To improve programmer productivity by using a database that better matches an application's needs.
  - To improve data access performance via some combination of handling larger data volumes, reducing latency, and improving throughput.
- It's essential to test expectations about programmer productivity and/or performance before committing to using a NoSQL technology.
- Service encapsulation supports changing data storage technologies as needs and technology evolve. Separating parts of applications into services also allows you to introduce NoSQL into an existing application.
- Most applications, particularly nonstrategic ones, should stick with relational technology—at least until the NoSQL ecosystem becomes more mature.

Key points

- Polyglot persistence is about using different data storage technologies to handle varying data storage needs.
- Polyglot persistence can apply across an enterprise or within a single application.
- Encapsulating data access into services reduces the impact of data storage choices on other parts of a system.
- Adding more data storage technologies increases complexity in programming and operations, so the advantages of a good data storage fit need to be weighed against this complexity.
- NoSQL is just one set of data storage technologies. As they increase comfort with polyglot persistence, we should consider other data storage technologies whether or not they bear the NoSQL label.