

Cloud Computing: Servicios y Aplicaciones



UNIVERSIDAD
DE GRANADA



T5. Data as a Service

SGBD NoSQL

- NoSQL (not only SQL) es una categoría general de sistemas de gestión de bases de datos que difiere de RDBMS: no hay esquemas, no permiten JOINS, no intentan garantizar ACID y escalan horizontalmente
- Son almacenamiento estructurado

Objetivos de BD NoSQL

- Distribución
- Escalabilidad: horizontal y prácticamente ilimitada
- Disponibilidad: mediante distribución y replicación
- Flexibilidad: incorporación de nuevos tipos de datos posteriormente a la creación de la BD

Almacenamiento

- A diferencia de RDBMS no se define por adelantado el esquema para incorporar información real
- En BD distribuidas:
 - Almacenes basados en filas y columnas
 - Consistencia eventual
 - *Sharding*
 - Replicación maestro-maestro
 - Particionado

ACID vs. BASE

- En los RDBMS las transacciones **ACID** (Atomicidad, Consistencia, Aislamiento y Durabilidad) garantizan la consistencia y estabilidad de las operaciones → Gestión de bloqueo
 - **Atomicidad**: garantía de ejecución o no de la operación
 - **Consistencia**: integridad
 - **Aislamiento**: ejecución concurrente coherente con ejecución secuencial
 - **Durabilidad**: persistencia

ACID vs. BASE

- NoSQL, almacenamiento según modelo BASE:
 - Basic availability
 - Soft-state
 - Eventual consistency
- BASE: alternativa flexible a ACID para cuando no se requiere un modelo estricta al modelo relacional

Características para NoSQL

- Garantías de consistencia débiles
- Arquitectura distribuida, elementos redundantes
- Estructuras de datos sencillas: arrays asociativos o almacenes para pares clave-valor

Características para NoSQL

- Fáciles de usar en clusters
- Guardan datos persistentes
- No hay esquemas fijos, sino dinámicos
- Sistemas de consultas propios
- Propiedades ACID en un nodo del cluster

¿Por qué son necesarias?

- Desafíos presentados por las aplicaciones web modernas, computación ubicua y Big Data
 - Datos a escala web
 - Procesamiento masivo de datos
 - *Alta frecuencia* de lecturas y escrituras
 - Las aplicaciones sociales (no bancarias) no necesitan el mismo nivel de ACID

¿Por qué son necesarias?

- Desafíos:
 - La BDR no escala con el tráfico a un coste aceptable
 - El tamaño del esquema crece desproporcionadamente.
 - Muchos datos temporales
 - La BD se desnormaliza por rendimiento o conveniencia
 - Consultas sobre relaciones jerárquicas complejas; recomendaciones o inteligencia de negocio
 - Transacciones locales no muy durables

Tipos de BDDD

- Orientadas a columnas
- Orientadas a documentos
- De clave-valor
- Basadas en grafos

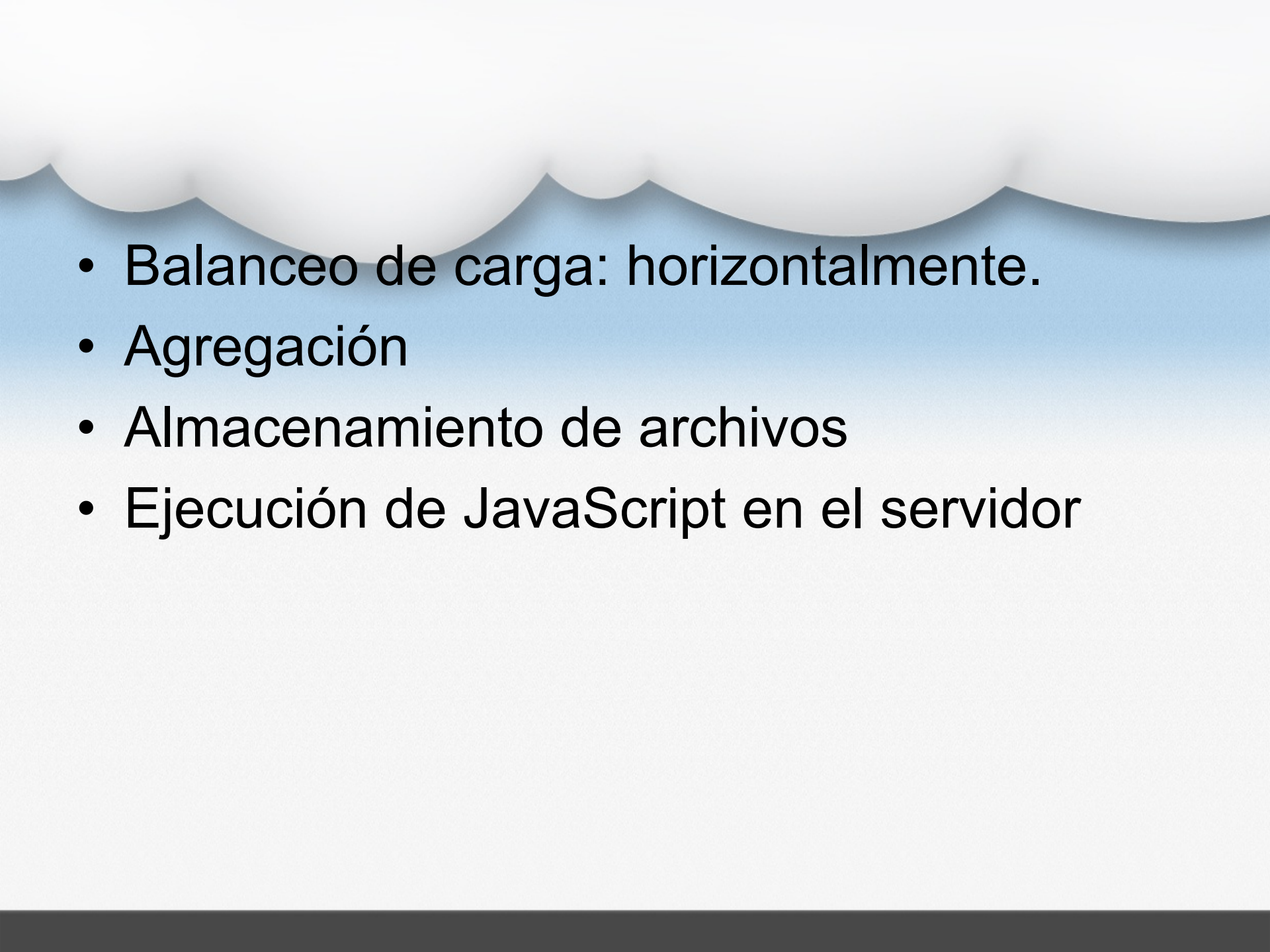


mongoDB

- SGBD potente, flexible y escalable y de propósito general
- Orientada a documentos, no relacional
 - Facilidad de uso
 - Escalable
 - Funcionalidad amplia
 - Optimizando la eficiencia (en tiempo)
- <http://mogodb.org>

- Existen versiones para múltiples plataformas: Linux, Windows, OS X
- Disponible como servicio:
<https://mms.mongodb.com>
- Desarrollada en C++

- BD basada en documentos:
 - Los documentos (objetos) se acoplan perfectamente en los tipos de datos de lenguajes de programación
 - Los documentos incrustados y las colecciones reducen la necesidad de *joins*
 - Dispone de un esquema dinámico que facilita el polimorfismo
- Alto rendimiento:
 - Lecturas y escrituras rápidas
 - Índices potentes: sobre colecciones, subcolecciones y documentos incrustados
- Alta disponibilidad: replicación y restablecimiento automático del maestro
- Fácil escalabilidad:
 - *Sharding* automático
 - Lecturas eventualmente-consistentes

- 
- Balanceo de carga: horizontalmente.
 - Agregación
 - Almacenamiento de archivos
 - Ejecución de JavaScript en el servidor

Conceptos

- Documento: unidad básica de datos («fila»)
- Colección: grupo de documentos («tabla»)
- Una base de datos alberga múltiples colecciones
- Una instancia de MongoDB puede alojar múltiples bases de datos

Documento

Conjunto ordenado de pares <clave, valor>

```
{  
  "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),  
  "Last Name": "PELLERIN",  
  "First Name": "Franck",  
  "Age": 29,  
  "Address": {  
    "Street": "1 chemin des Loges",  
    "City": "VERSAILLES"  
  }  
}
```

Utilidades

- mongo: shell interactivo
- mongostat: Estadísticas de una instancia
- mongotop: tiempo de ejecución de operaciones
- mongoimport/mongoexport
- mongodump/mongorestore

mongooshell

- Se comporta como shell de Unix
- Soporta expresiones en JavaScript
- Mandatos útiles:
 - help
 - db.help()
 - show dbs
 - use <database>
 - show collections
 - show users

MYSQL EXECUTABLE	ORACLE EXECUTABLE	MONGODB EXECUTABLE
mysqld	oracle	mongod
mysql	sqlplus	mongo

SQL TERM	MONGODB TERM
database	database
table	collection
index	index
row	document
column	field
joining	embedding & linking

SQL

```
CREATE TABLE users (name VARCHAR(128), age  
NUMBER)
```

```
INSERT INTO users VALUES ('Bob', 32)
```

```
SELECT * FROM users
```

```
SELECT name, age FROM users
```

```
SELECT name, age FROM users WHERE age  
= 33
```

```
SELECT * FROM users WHERE age > 33
```

```
SELECT * FROM users WHERE age <= 33
```

MONGODB

```
db.createCollection("users")
```

```
db.users.insert({name: "Bob", age: 32})
```

```
db.users.find()
```

```
db.users.find({}, {name: 1, age: 1, _id:0})
```

```
db.users.find({age: 33}, {name: 1, age:  
1, _id:0})
```

```
db.users.find({age: {$gt: 33}})
```

```
db.users.find({age: {$lte: 33}})
```

SQL	MONGODB
<code>SELECT * FROM users WHERE age > 33 AND age < 40</code>	<code>db.users.find({age: {\$gt: 33, \$lt: 40}})</code>
<code>SELECT * FROM users WHERE age = 32 AND name = 'Bob'</code>	<code>db.users.find({age: 32, name: "Bob"})</code>
<code>SELECT * FROM users WHERE age = 33 OR name = 'Bob'</code>	<code>db.users.find({\$or:[{age:33}, {name: "Bob"}]})</code>
<code>SELECT * FROM users WHERE age = 33 ORDER BY name ASC</code>	<code>db.users.find({age: 33}).sort({name: 1})</code>
<code>SELECT * FROM users ORDER BY name DESC</code>	<code>db.users.find().sort({name: -1})</code>
<code>SELECT * FROM users WHERE name LIKE '%Joe%'</code>	<code>db.users.find({name: /Joe/})</code>
<code>SELECT * FROM users WHERE name LIKE 'Joe%'</code>	<code>db.users.find({name: /^Joe/})</code>
<code>SELECT * FROM users LIMIT 10 SKIP 20</code>	<code>db.users.find().skip(20).limit(10)</code>
<code>SELECT * FROM users LIMIT 1</code>	<code>db.users.findOne()</code>

SQL	MONGODB
<code>SELECT DISTINCT name FROM users</code>	<code>db.users.distinct("name")</code>
<code>SELECT COUNT(*) FROM users</code>	<code>db.users.count()</code>
<code>SELECT COUNT(*) FROM users WHERE AGE > 30</code>	<code>db.users.find({age: {\$gt: 30}}).count()</code>
<code>SELECT COUNT(AGE) FROM users</code>	<code>db.users.find({age: {\$exists: true}}).count()</code>
<code>UPDATE users SET age = 33 WHERE name = 'Bob'</code>	<code>db.users.update({name: "Bob"}, {\$set: {age: 33}}, {multi: true})</code>
<code>UPDATE users SET age = age + 2 WHERE name = 'Bob'</code>	<code>db.users.update({name: "Bob"}, {\$inc: {age: 2}}, {multi: true})</code>
<code>DELETE FROM users WHERE name = 'Bob'</code>	<code>db.users.remove({name: "Bob"})</code>
<code>CREATE INDEX ON users (name ASC)</code>	<code>db.users.ensureIndex({name: 1})</code>
<code>CREATE INDEX ON users (name ASC, age DESC)</code>	<code>db.users.ensureIndex({name: 1, age: -1})</code>
<code>EXPLAIN SELECT * FROM users WHERE age = 32</code>	<code>db.users.find({age: 32}).explain()</code>

Ejemplos

```
x = 200
```

```
Math.sin(Math.PI / 2);
```

```
function factorial (n) {  
  if (n <= 1) return 1;  
  return n * factorial(n -1);  
}
```

```
factorial(5)
```

```
db.runCommand( { create : "jmbs",  
autoIndexId : true } )
```

Create

```
db.post = { "title" : "My Blog Post",  
            "content" : "Here's my blog post.",  
            "date" : new Date() }
```

```
db.blog.insert(post)
```

```
db.blog.find()
```

Cassandra



- Cassandra provides a scalable, high-availability datastore with no single point of failure.
- Originally developed by A. and P. Malik to solve their Inbox-search problem.
- Arquitectura basada en peer-to-peer frente a maestro-esclavo.
- “Tunable consistency”

Titan: Distributed Graph DB



- Titan is a scalable graph database optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster. Titan is a transactional database that can support thousands of concurrent users executing complex graph traversals in real time.

Titan (2)

- Elastic and linear scalability for a growing data and user base.
- Data distribution and replication for performance and fault tolerance.
- Multi-datacenter high availability and hot backups.
- Support for ACID and eventual consistency.

Titan (3)

- Support for various storage backends:
 - Apache Cassandra
 - Apache HBase
 - Oracle BerkeleyDB
- Support for global graph data analytics, reporting, and ETL through integration with big data platforms: