Bases de datos NoSQL
Clase 2

Arroyo Joaquin
Belmonte Marina

Universidad Nacional de Rosario
Licenciatura en Ciencias de la Computación
Bases de Datos Avanzadas

15 de mayo de 2024

Resúmen

- Repaso de Modelos NoSQL
- Implementaciones de Modelos NoSQL
  - Redis
  - Apache HBase
  - MongoDB
  - Neo4j
- Dos artículos que estudian el rendimiento de bases de datos SQL versus NoSQL en aplicaciones reales

Repaso

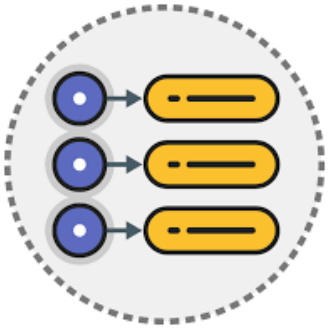
Como vimos en la clase 1, existe una amplia variedad de modelos de datos *NoSQL*. Entre los más conocidos encontramos:

1. Clave-Valor
2. Columnar
3. Documental
4. Grafo

Índice - Modelos NoSQL

Clave-Valor - Repaso

Este modelo almacena pares del tipo (*Key*, *Value*).



- **Key**: Un identificador único para acceder al valor asociado.
- **Value**: Puede ser cualquier tipo de dato, desde texto, números y documentos, hasta listas o incluso otros pares clave-valor.

Clave-Valor - Ejemplos

Algunas de las implementaciones más conocidas son:

- **Amazon DynamoDB**
- **Voldemort**
- **Riak**
- **Redis**

Nos vamos a centrar en esta última.

Clave-Valor - Redis

Redis significa "**RE**mote **DI**ctionary **S**erver"

- Base de datos de código abierto.
- Rápida y versátil.
- Diseñada como un almacén de estructuras de datos en memoria.

Características de Redis

- Estructuras de datos: strings, hashes, listas, conjuntos, etc.
- Operaciones atómicas: Agregar, incrementar, intersección, unión, etc.
- Replicación de datos incorporada.
- Trabaja con un conjunto de datos en memoria.

Operaciones de Redis

Redis ofrece más de **400** operaciones e implementa la interfaz teórica para cada uno de los tipos de datos mencionados.

Algunos ejemplos son:

- *Strings: SET*, *GET* y *DEL*
- *Hashes: HSET*, *HGET* y *HDEL*
- *Listas: LPUSH*, *LPOP*, *RPUSH*, *RPOP*, *LRANGE*
- etc.

Ejemplo de Uso

SET: Agrega un par (*key*, *value*).

- Complejidad temporal: $\mathcal{O}(1)$.

```
redis> SET subject1 "Base de Datos Avanzadas"
OK
```

Ejemplo de Uso

GET: Obtiene el valor de una *key*.

- Complejidad temporal: $\mathcal{O}(1)$.

```
redis> GET subject1
"Base de Datos Avanzadas"
```

Ejemplo de Uso

LPUSH: Inserta valores en la cabeza de una lista.

- Complejidad temporal: $\mathcal{O}(N)$.

```
redis> LPUSH mylist "NoSQL"
redis> LPUSH mylist "Datos" "De" "Bases"
```

Ejemplo de Uso

LRANGE: Retorna los elementos en un rango especificado de una lista.

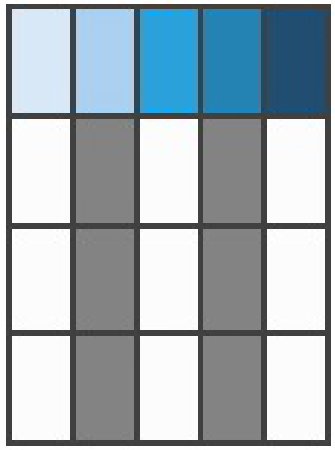
- Complejidad temporal: $\mathcal{O}(S + N)$.

```
redis> LRANGE mylist 0 2
1) "Bases"
2) "De"
3) "Datos"
```

Índice - Modelos NoSQL

Columnar - Repaso

Las bases de datos orientadas a columnas almacenan datos verticalmente por columnas en lugar de horizontalmente por filas, permitiendo un acceso más eficiente a datos específicos.



Algunas de sus ventajas:

- Mayor eficiencia en consultas sobre columnas específicas.
- Convenientes para análisis de grandes volúmenes de datos.
- Adecuadas para aplicaciones de business intelligence y análisis predictivo.

Columnar - Ejemplos

Algunos ejemplos de bases de datos columnares son:

- **Apache Cassandra**
- **Apache HBase**
- **ClickHouse**

Nos vamos a centrar en Apache HBase.

Columnar - Apache HBase

- Base de datos distribuida y escalable.
- Desarrollada por la Apache Software Foundation.
- Modelo de datos basado en tablas y columnas flexibles.
- Almacenamiento de datos en archivos HFile en Hadoop HDFS.
- Utiliza partición distribuida y replicación para alta disponibilidad.
- Ofrece diferentes niveles de consistencia.

Operaciones Básicas en Hbase

HBase proporciona un conjunto de comandos específicos que se utilizan para realizar diversas operaciones administrativas y de consulta.

Algunos ejemplos son:

1. **create**: Crear una nueva tabla.
2. **put**: Insertar un valor en una celda de datos.
3. **get**: Recuperar datos de una fila.
4. **delete**: Eliminar una celda de datos.
5. **drop**: Eliminar una tabla.

Ejemplo de Uso

```
hbase(main):001:0> create 'usuarios',
                   'datos_personales', 'datos_contacto'
0 fila(s) en 1.2340 segundos
```

Ejemplo de Uso

```
hbase(main):002:0> put 'usuarios', '1001',
'datos_personales:nombre', 'Juan'
0 fila(s) en 0.0510 segundos

hbase(main):003:0> put 'usuarios', '1001',
'datos_personales:apellido', 'Pérez'
0 fila(s) en 0.0110 segundos

hbase(main):004:0> put 'usuarios', '1001',
'datos_contacto:email', 'juan@example.com'
0 fila(s) en 0.0090 segundos
```

Ejemplo de Uso

```
hbase(main):005:0> get 'usuarios', '1001',
{COLUMN => 'datos_personales'}
COLUMN                                  CELL
 datos_personales:nombre                valor=Juan
 datos_personales:apellido              valor=Pérez
1 fila(s) en 0.0190 segundos
```

Ejemplo de Uso

```
hbase(main):006:0> delete 'usuarios', '1001',
'datos_contacto:email'
0 fila(s) en 0.0230 segundos

hbase(main):007:0> get 'usuarios', '1001'
COLUMN                              CELL
 datos_personales:nombre            valor=Juan
 datos_personales:apellido          valor=Pérez
1 fila(s) en 0.0190 segundos
```

Operaciones Avanzadas en HBase

1. Escaneo de Rango
2. Filtros de Columnas y Filas
3. Transacciones y Consistencia
4. Replicación de Datos
5. Optimización de Rendimiento
6. Seguridad

Índice - Modelos NoSQL

Documental

Las bases de datos documentales almacenan datos en documentos individuales, que pueden ser estructurados o semi-estructurados como archivos JSON o XML.

Son especialmente adecuadas para aplicaciones donde los datos tienen una estructura flexible y variable como por ejemplo:

- **Contenido web**
- **Análisis de registros**
- **Gestión de datos de productos**
- etc.

Documental - Ejemplos

Algunos ejemplos de bases de datos documentales son:

- **MongoDB**
- **CouchDB**

Nos vamos a centrar en MongoDB.

Documental - MongoDB

- Base de datos de código abierto, orientada a documentos y altamente escalable.
- Desarrollada por MongoDB Inc.
- Modelo de datos flexible basado en documentos BSON.
- Utiliza almacenamiento basado en archivos de mapeo directo o motor WiredTiger.
- Maneja replicación a través de conjuntos de réplicas para alta disponibilidad.
- Distribuye datos en clústeres de servidores llamados fragmentos.
- Lenguaje de consulta poderoso y flexible.

Operaciones comunes en MongoDB

- **Insertar un documento**

```
db.users.insertOne({
    name: "John Doe",
    age: 30,
    email: "john@example.com"
})

db.users.insertMany([
  { name: "Alice", age: 25 },
  { name: "Bob", age: 30 },
  { name: "Charlie", age: 50 }
])
```

Operaciones comunes en MongoDB

- **Consultar documentos**

```
db.users.find({ age: { $gt: 25 } })
```

- **Actualizar documentos**

```
db.users.updateOne(
    { name: "John Doe" },
    { $set: { age: 35 } }
)

db.users.updateMany(
    { status: "active" },
    { $set: { status: "inactive" } }
)
```

Operaciones comunes en MongoDB

- **Eliminar documentos**

```
db.users.deleteOne({ name: "John Doe" })
db.users.deleteMany({ age: { $gt: 40 } })
```

- **Contar documentos**

```
db.users.countDocuments({ age: { $lt: 30 } })
```

- **Valores Distintos**

```
db.users.distinct("city", { country: "USA" })
```

- **Agregar datos**

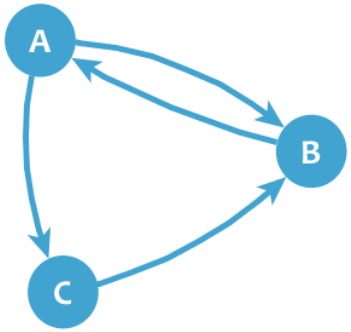
```
db.sales.aggregate([
  { $match: { status: "completed" } },
  { $group: { _id: "$product",
              totalAmount: { $sum: "$amount" } } }
])
```

Índice - Modelos NoSQL

Grafo - Repaso

Las bases de datos en grafo utilizan estructuras de grafo para almacenar, consultar y relacionar datos.

Los datos se representan mediante nodos (entidades) y arcos (relaciones entre entidades).



Pueden asignarse propiedades a nodos y arcos para capturar más detalles.

Son ideales para almacenar datos interconectados.

Aplicaciones comunes incluyen: **Redes sociales**, **Sistemas de recomendación**, **Redes de trasportes**, etc.

Grafo - Ejemplos

Algunos ejemplos de bases de datos en grafos son:

- **Neo4j**
- **GraphBase**
- **Infinite Graph**
- **FlockDB**

Nos vamos a centrar en Neo4j.

Grafo - Neo4j

- Plataforma líder en bases de datos de grafos.
- Diseño nativo de grafo para un rendimiento óptimo y escalabilidad excepcional.
- Utiliza Cypher como su lenguaje de consulta principal.
- Ofrece una interfaz integrada de visualización de grafos.
- Altamente escalable y diseñado para manejar grandes volúmenes de datos.
- Garantiza propiedades ACID.
- Edición empresarial.

Cypher: Lenguaje de Consulta de Neo4j

- Sintaxis intuitiva y expresiva.
- Orientado a patrones.
- Declarativo.
- Soporte completo para operaciones CRUD.
- Amplio soporte para funciones y operadores para operaciones avanzadas.

Operaciones Básicas de Neo4j

Estas son algunas de las cláusulas básicas más comunes en Cypher y su sintaxis asociada:

- **Crear nodos y relaciones**

```
CREATE (a:Person {name: 'John', age: 30})

CREATE (b:Person:Employee {name: 'Alice', role:
'Manager'})

CREATE (a)-[:FRIENDS_WITH]->(b)
```

- **Especificar patrones**

```
MATCH (p:Person {name: 'John'}) RETURN p

MATCH (p:Person {name: 'John'})-[r]->() RETURN p, r
```

- **Filtrar resultados**

```
WHERE n.name = 'John' AND friend.age > 25
```

Operaciones Básicas de Neo4j

- **Devolver datos**
  `RETURN n, friend`
- **Actualizar propiedades**
  `SET n.age = 30`
- **Eliminar nodos y relaciones.**
  `DELETE n, friend`
- **Ordenar resultados**
  `ORDER BY n.name DESC`
- **Limitar número de resultados**
  `LIMIT 10`
- **Crear índices**
  `CREATE INDEX ON :Person(name)`

Ejemplo de Consulta en Neo4j

Para una demostración de como estas operaciones podrían ser utilizadas, supongamos que queremos encontrar los amigos en común entre dos usuarios en una red social y alguna información adicional sobre esos amigos en común:

```
MATCH (userA:User {username: 'UsuarioA'})
    -[:FRIENDS_WITH]-(commonFriend)-[:FRIENDS_WITH]-
    (userB:User {username: 'UsuarioB'})
RETURN commonFriend.username AS commonFriendUsername,
       commonFriend.age AS commonFriendAge
```

Empresas que Utilizan Neo4j

- Walmart
- Volvo
- eBay
- Cisco

Índice - Articulos

1. Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data.
2. Performance investigation of selected SQL and NoSQL databases.

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

Este estudio fue llevado a cabo en el año 2022 por tres investigadores pertenecientes a los siguientes institutos

Instituto de Ingeniería de Coimbra[1], *Centro de Informática y Sistemas de la Universidad de Coimbra*[2] e *Instituto Tecnológico de São Paulo*[3]

João Antas[1], Rodrigo Rocha Silva[2,3] y Jorge Bernardino[1,2]

y tuvo como objetivo:

- Estudiar diferentes sistemas de bases de datos, con el fin de ayudar a seleccionar el más adecuado para almacenar, gestionar y minar datos relacionados con el COVID-19.
- Llevar a cabo un proceso de Minería de Datos, empleando pruebas de clasificación de datos del software Orange Data Mining.

Nos vamos a centrar en mostrar como se logró el primer objetivo.

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

Para ambos experimentos, utilizaron una computadora con las siguiente características:

- Windows 10
- Procesador Intel Core i7-8750H 2.20GHz
- 16 GB de RAM
- 256 GB SSD de Almacenamiento

Además las versiones de las tecnologías utilizadas fueron las siguientes:

- SQL Server versión 2017
- MongoDB versión 4.4
- Cassandra versión 3.11.10

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

Los datos fueron obtenidos de distintas fuentes, como hospitales, datos públicos, etc., y fueron almacenados en formatos CSV o XML antes de ser incorporados a las bases de datos. Y para evaluar la escalabilidad de las bases de datos, crearon dos datasets de distintos tamaños.

Para el primer experimento, utilizaron seis queries diferentes, con el objetivo de evaluar el tiempo de ejecución, la RAM utilizada y el porcentaje de CPU de cada consulta.

Vamos a mostrar los resultados obtenidos sobre las consultas 1, 2 y 3, que se presentan a continuación.

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

```
select Id_Regiao, AnoNascimento, count (Id_ExameCovid) as TotalExames,
sum (case when Resultado = 'DETECTADO' or Resultado = 'REAGENTE' then 1 else 0 end) as PosReag,
sum (case when Resultado = 'NAO DETECTADO' or Resultado = 'NAO REAGENTE' then 1 else 0 end) as
NegNReag,
sum (case when Resultado = 'DETECTADO' or Resultado = 'NAO DETECTADO' then 1 else 0 end) as
ExamesPCR,
sum (case when Resultado = 'DETECTADO' then 1 else 0 end) as Positivos,
sum (case when Resultado = 'NAO DETECTADO' then 1 else 0 end) as Negativos,
sum (case when Resultado = 'REAGENTE' or Resultado = 'NAO REAGENTE' then 1 else 0 end) as ExamesSoro,
sum (case when Resultado = 'REAGENTE' then 1 else 0 end) as Reagentes,
sum (case when Resultado = 'NAO REAGENTE' then 1 else 0 end) as NaoReagentes,
sum (case when Sexo = 'M' then 1 else 0 end) as ExamesM,
sum (case when Sexo = 'M' and (Resultado = 'DETECTADO' or Resultado = 'REAGENTE') then 1 else 0 end) as
PosReagM,
sum (case when Sexo = 'M' and (Resultado = 'NAO DETECTADO' or Resultado = 'NAO REAGENTE') then 1 else
0 end) as NegReagM,
sum (case when Sexo = 'F' then 1 else 0 end) as ExamesF,
sum (case when Sexo = 'F' and (Resultado = 'DETECTADO' or Resultado = 'REAGENTE') then 1 else 0 end) as
PosReagF,
sum (case when Sexo = 'F' and (Resultado = 'NAO DETECTADO' or Resultado = 'NAO REAGENTE') then 1 else
0 end) as NegReagF
from ExameCovid, Paciente
where Paciente.Id_Paciente = ExameCovid.Id_Paciente
group by Id_Regiao, AnoNascimento
order by TotalExames DESC;
```

Figura: Query Region (Query 1 - sin subrayado) y Query RegionYear (Query 2)

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

Esta query fue seleccionada utilizando un registro de auditoría que controlaba todas las consultas realizadas en la base de datos de SQL Server.

```
select count(*) from SymptomsCovid;
```

Figura: Query de Orange Data Mining (Query 3)

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

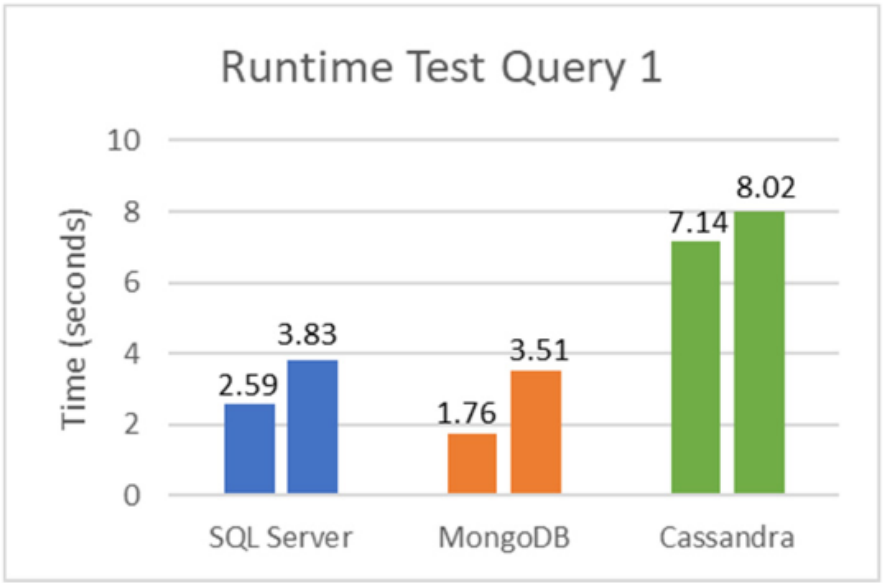


Figura: Prueba de tiempo de ejecución para la Query 1.

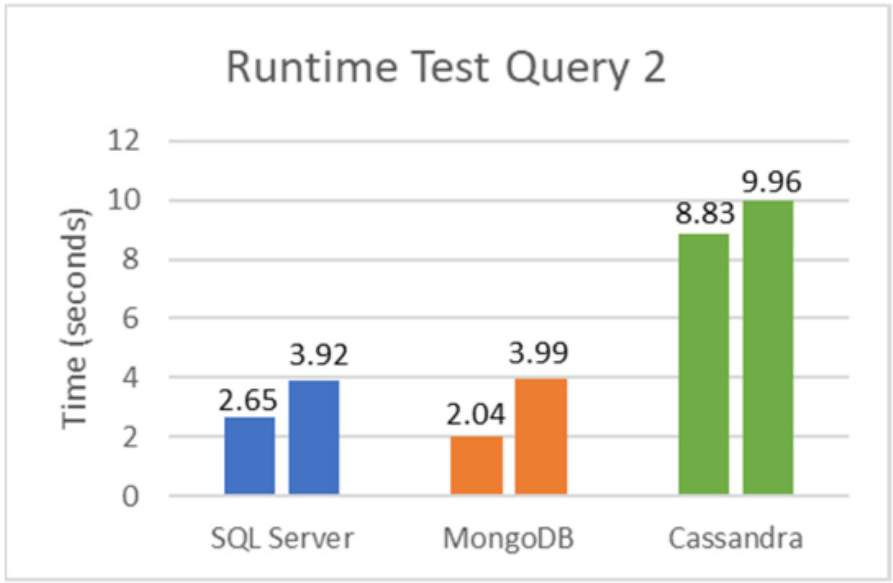


Figura: Prueba de tiempo de ejecución para la Query 2.

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

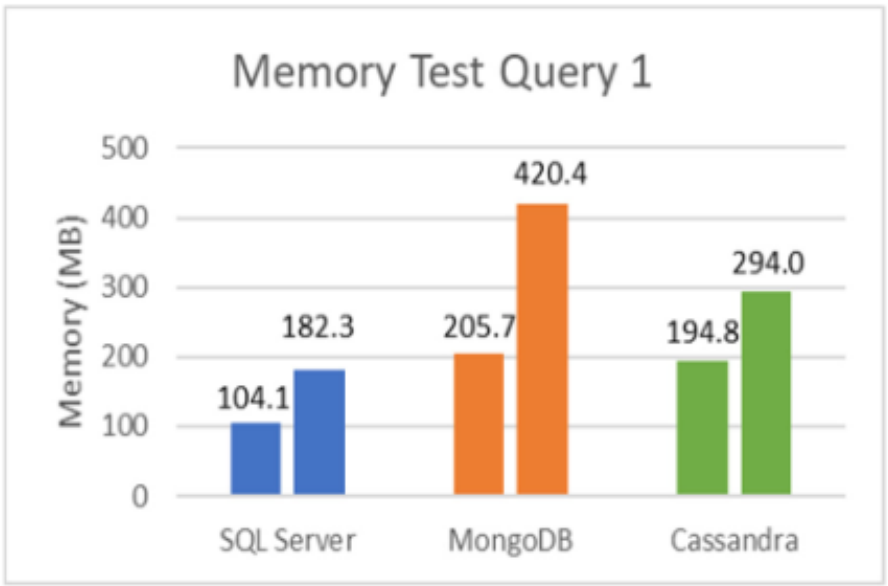


Figura: Memoria RAM utilizada para la Query 1.

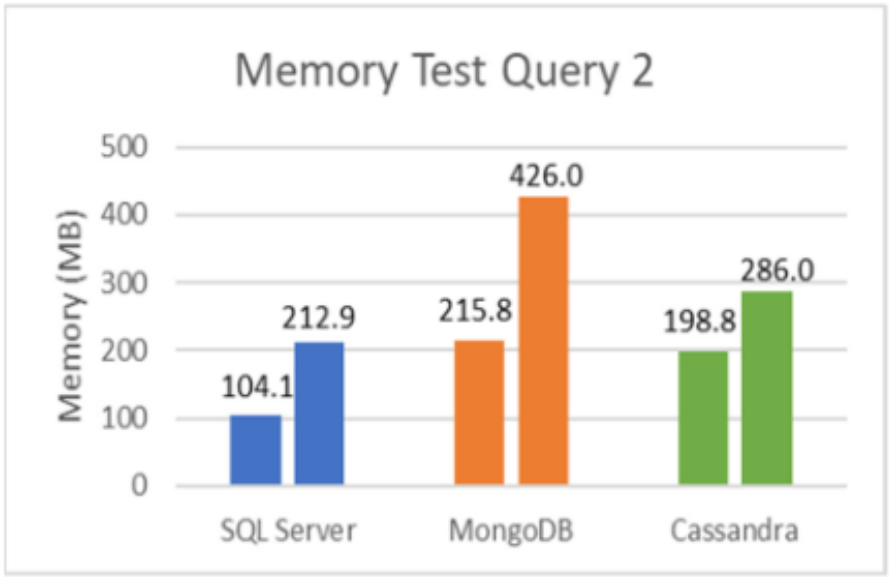


Figura: Memoria RAM utilizada para la Query 2.

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

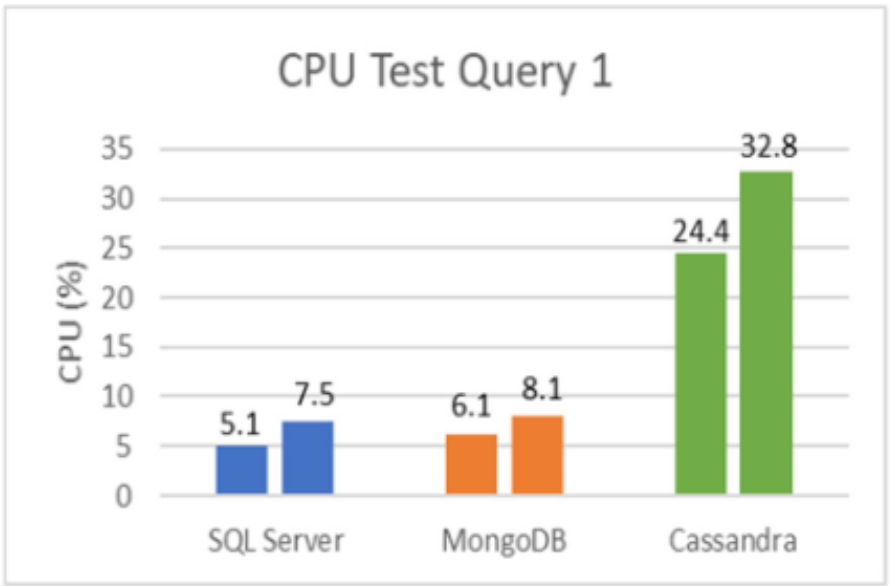


Figura: Porcentaje de CPU utilizado para la Query 1.

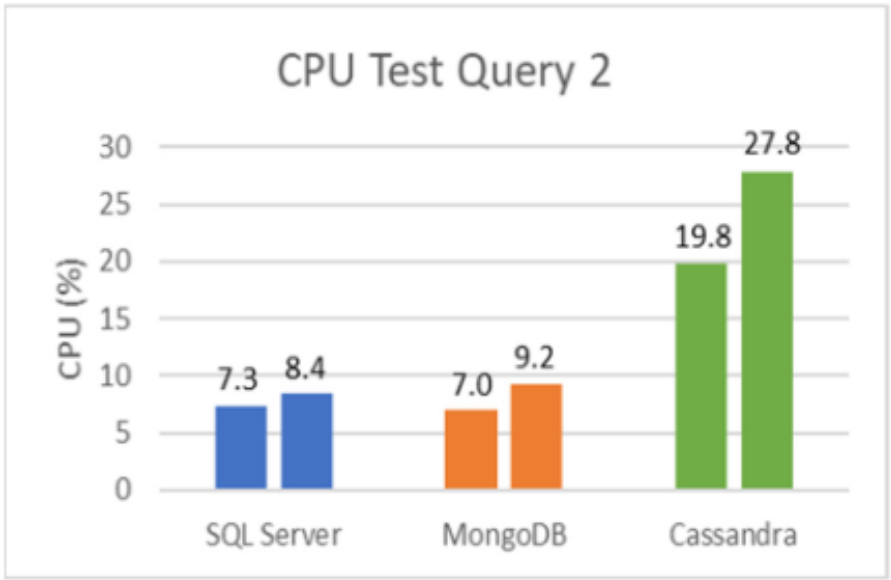


Figura: Porcentaje de CPU utilizado para la Query 2.

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

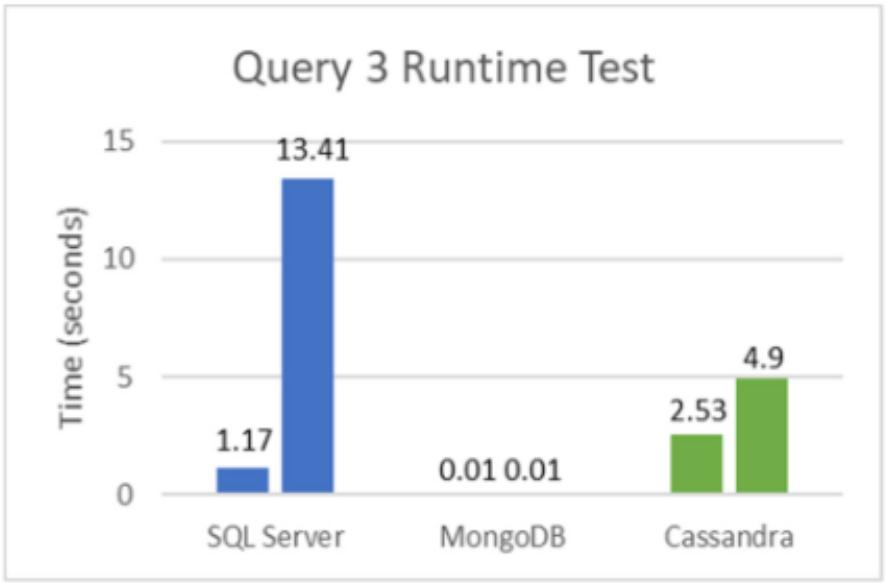


Figura: Prueba de tiempo de ejecución para la Query 3.

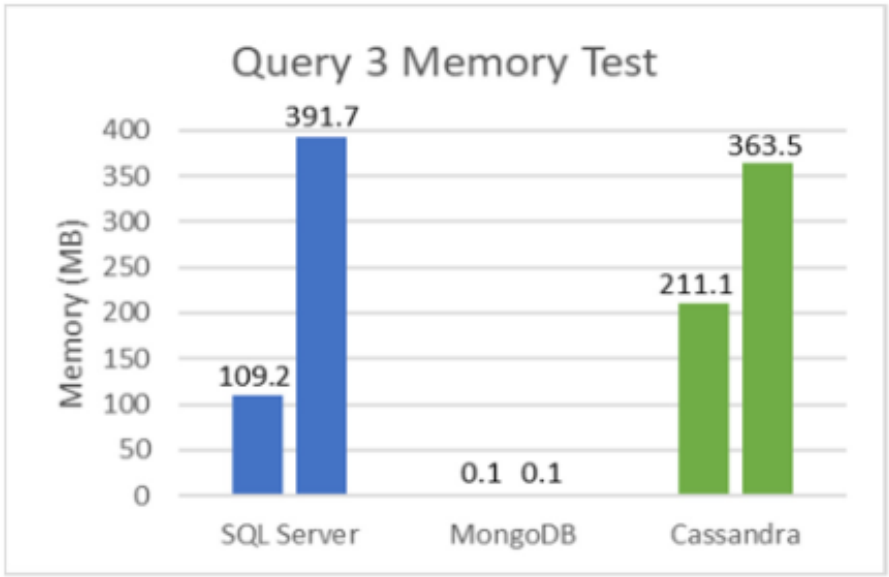


Figura: Memoria RAM utilizada para la Query 3.

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data

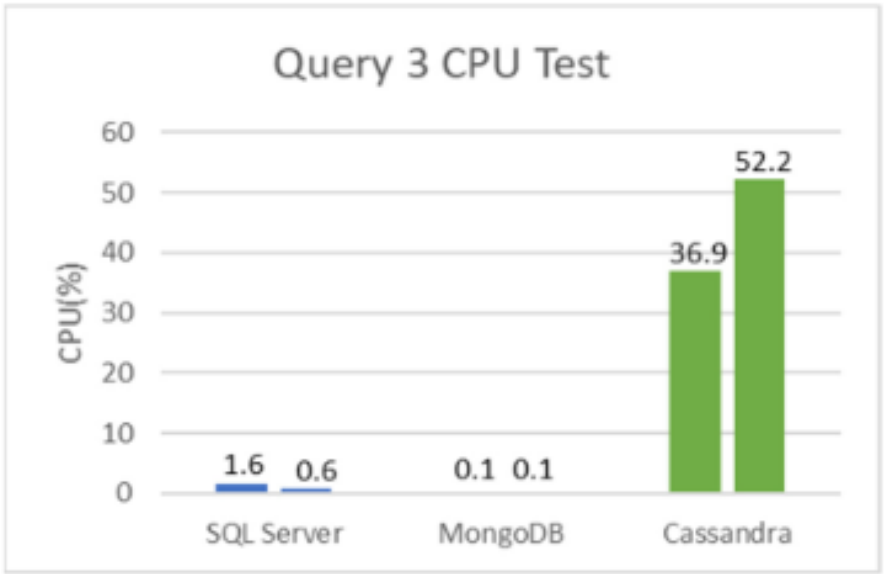


Figura: Porcentaje de CPU utilizado para la Query 3.

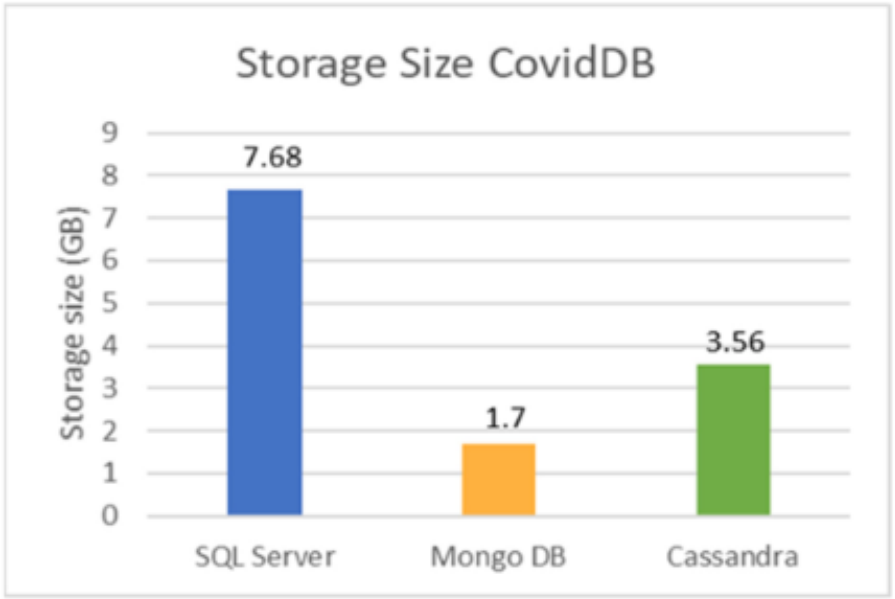


Figura: Tamaño de la base de datos de COVID-19.

Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data - Conclusiones

Cabe destacar que los experimentos que no se mostraron, utilizaron consultas con *joins*, y SQL Server tuvo un mejor rendimiento.

- SQL Server debería ser la elección si los datos son muy estructurados y necesitan consultas con *joins*.
- Si se utiliza un gran volumen de datos no estructurados y no es necesario realizar demasiadas consultas con *joins*, MongoDB o Cassandra se consideran las soluciones más adecuadas.

Índice - Articulos

1. Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data.
2. Performance investigation of selected SQL and NoSQL databases.

Performance investigation of selected SQL and NoSQL databases

Este artículo fue presentado en el año 2015 en la Conferencia Internacional sobre Ciencia de la Información Geográfica (AGILE) por tres investigadores de la *Universidad de Bundeswehr*:

Stephan Schmid, Eszter Galicz y Wolfgang Reinhardt.

- Trata sobre la creciente importancia de los datos espaciales, en el mundo actual.
- Explora las bases de datos NoSQL como una posible alternativa ante el dominio de las bases de datos relacionales para almacenar y manipular este tipo de datos.

Para los experimentos utilizaron tres bases de datos:

PostgreSQL, **MongoDB** y **CouchBase**.

Performance investigation of selected SQL and NoSQL databases

Para la representación de datos espaciales en PostgreSQL utilizaron **PostGis**, y en las dos siguientes utilizaron el formato **GeoJSON**, el cuál permite representar **Geometrías** (Puntos, Polígonos, Colección de Geometrías, etc.), **Características** y **Coleciones de Características**.

Mencionan que al utilizar las estructuras de datos GeoJSON, el enfoque sin esquemas tiene algunas restricciones.

Sin embargo, la representación geográfica debe seguir dicha estructura para poder establecer un índice geoespacial.

Performance investigation of selected SQL and NoSQL databases

Para los experimentos, los autores utilizaron una computadora con las siguientes características:

- Microsoft Windows Server 2008 R2
- 8 core CPU 2,5 GHz
- 10GB RAM

Utilizaron tres datasets para los experimentos, los cuales fueron obtenidos OpenStreetMap.

Level	Region	Size
Subregion	Niederbayern	38.9 MB
State	Bayern	501 MB
Country	Germany	2.1 GB

Cuadro: Datos de prueba utilizados de OpenStreetMap.

Performance investigation of selected SQL and NoSQL databases

Y eligieron dos tipos de consultas para el análisis:

1. Consultas sobre información de atributos.
2. Consultas que utilizan la geo-función *within*.

```
Select * from points WHERE osm_id = '1082817686'
```

Figura: Query 1

```
Select * from points WHERE
(ST_Within (wkb_geometry, ST_GeomFromGeoJSON('
    {
    "type": "Polygon",
    "coordinates": [
        [[12.782592773437498,
        48.38817819201506 ],
        [12.782592773437498,
        48.54843286654265,
        [13.1231689453125,
        48.54843286654265],
        [13.1231689453125,
        48.38817819201506],
        [12.782592773437498,
        48.38817819201506]]],
    "crs": {
        "type": "name",
        "properties": {
        "name": "EPSG:4326" }
        }
    ]
')) is true)
```

Figura: Query 2

Performance investigation of selected SQL and NoSQL databases

Para realizar la simulación en condiciones realistas, las consultas se realizaron con una determinada cantidad de usuarios, la cuál va en aumento:

100, 250 y 500 usuarios.

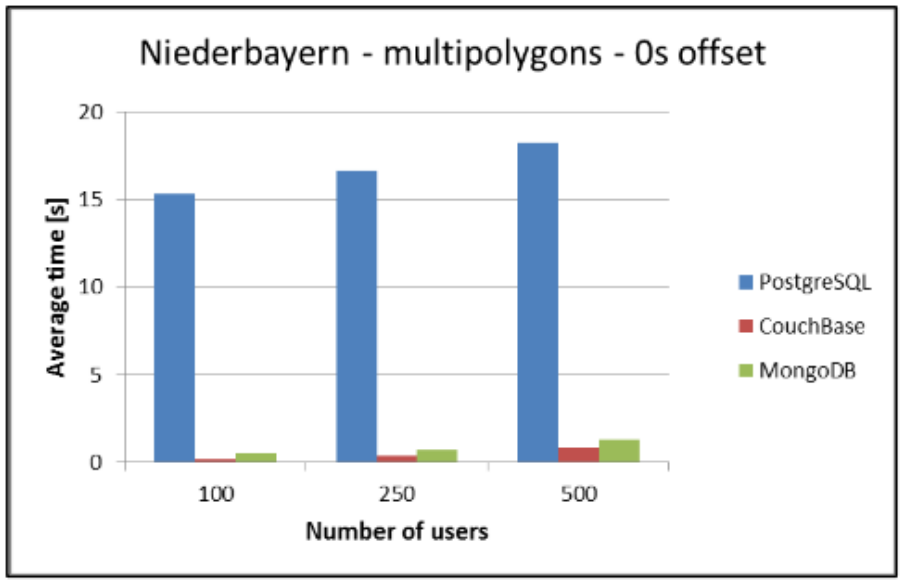


Figura: Resultados Query 1

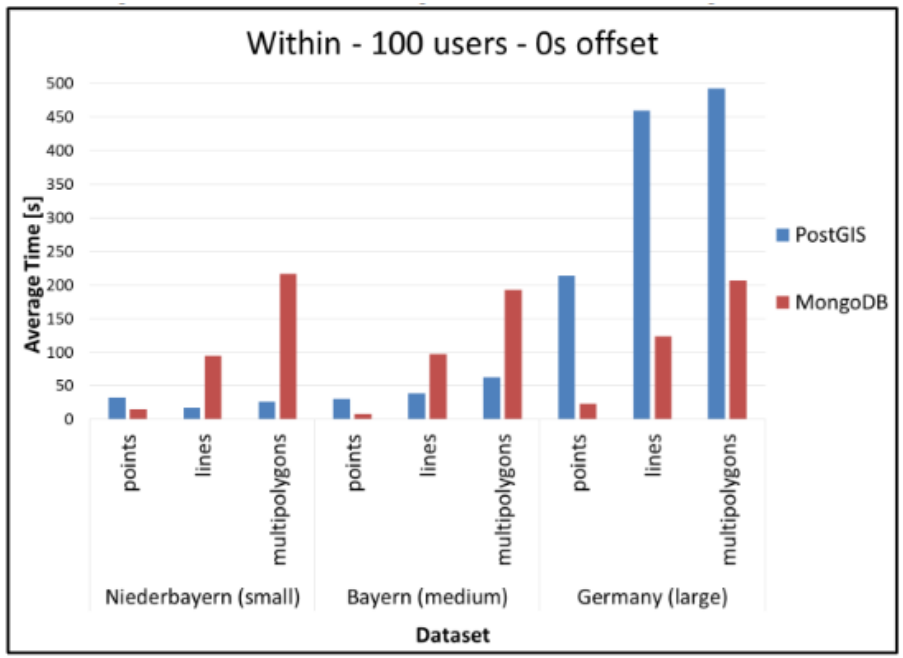


Figura: Resultados Query 2

Performance investigation of selected SQL and NoSQL databases - Conclusión

- Las consultas con el uso de geo-funciones llevan más tiempo que las consultas sobre información de atributos.
- Para solicitudes puramente sobre información de atributos, las bases de datos NoSQL son superiores en comparación con las bases de datos SQL.
- Los resultados muestran claramente que las bases de datos NoSQL son una alternativa posible, al menos para consultar información de atributos.

Dudas?

Referencias

1 redis.io Accedido el 15.04.2024

2 hbase.apache.org Accedido el 13.05.2024

3 mongodb.com Accedido el 13.05.2024

4 neo4j.com Accedido el 06.05.2024

5 Antas, J., Silva, R.R., Bernardino, J.: Assessment of sql and nosql systems to store and mine covid-19 data. MDP Comput. Suv. (2022)

6 Schmid, S., Galicz, E., Reinhardt, W.: Performance investigation of selected sql and nosql databases (2015)

7 datatracker.ietf.org/doc Accedido el 10.05.2024