Comparación de Performance de distintos modelos de Bases de Datos en el contexto de una REST API

Jaime Mauricio Calcagno
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
jaime.calcagno@fing.edu.uy

***Resumen*—En el contexto actual, con las cada vez mas pujantes Arquitecturas orientadas a microservicios y/o *headless*, donde las API van ganando cada vez mayor importancia, es relevante poder comparar el comportamiento de distintas bases de datos. Para esto se presenta un caso de estudio clásico de modelado, se resuelve el mismo utilizando un modelo relacional (implementado sobre MySQL) y dos modelos no relacionales, uno documental (MongoDB) y otro de grafos (Neo4J). Las tres soluciones de bases de datos son embebidas en una REST API construida sobre Spring Boot en Java, sin ningún tipo de ORM en el medio, y sobre esta API se realizan pruebas de *performance* sobre escritura y lectura. Es así que se implementa IMDBdnr, una REST API para manejar información de películas, series y episodios, con su relación a actores y directores. Los datos utilizados son los que IMDb [1] expone de manera gratuita.**

I. Introducción

En trabajos previos, como el de Fraczek, Konrad y Malgorzata Plechawska-wojcik [Fraczek and B(2017)] se compara el rendimiento de los distintos tipos de bases de datos en una aplicación web. Con esto como puntapié inicial, se quiere comparar el rendimiento de distintas bases de datos en el contexto de una REST API. Para esto en se comienza presentando un caso de estudio en la la Sección II, con una realidad a modelar y con una descripción del comportamiento esperado, para poder en base a esto luego tomar decisiones de diseño. Luego en la Sección III se presentan tres modelados posibles al caso de estudio, un modelo relacional III-A, un modelo documental III-B y finalmente con un modelo de grafos III-C. A continuación en la Sección IV se define la interfaz de la API, la cual realiza siguiendo los lineamientos REST [Fielding(2000)] [Fielding(2008)] [res(2016)]. En la Sección V se presenta la implementación y la *suite* de pruebas en la Sección VI. Finalmente se presentan los resultados obtenidos en la Sección VIII y las conclusiones IX sobre los mismos

II. Caso de Estudio

El problema presentado fue elegido de manera tal que en lo previo, ninguno de los tres modelos de bases de datos elegidos tuviera una clara ventaja sobre los demás. También era relevante que su modelado fuera de relativa simplicidad y que dentro de lo posible existiera algún estudio previo sobre el mismo, intentando de esta manera minimizar el impacto de un mal modelado por sobre el del rendimiento de la base de datos en los resultados obtenidos. De esta manera se decide tomar como ejemplo IMDb [2] y construir una API llamada **IMDbdnr** la cual será una simplificación de de IMDb dado que ademas de ser un un caso de estudio sencillo se cuenta con un numeroso conjunto de datos provistos gratuitamente por IMDb[3].

Entonces en IMDbdnr se tienen distintos **Títulos**, de los cuales se sabe su *id*, *nombre* y *duración en minutos*. Estos títulos pueden ser **Películas**, de las que también se sabe que *año* fueron emitidas, **Series**, de las que se conoce su **año de comienzo** y si existe, su *año de fin*. Una Serie puede tener **Episodios**, los cuales son un **Título** en sí mismos y se conoce su *año* de emisión. Cada título puede también tener una lista de **Géneros** asociados. Por otro lado se tienen **Personas** de las cuales se conoce su *identificador*, *nombre*,*año de nacimiento* y potencialmente su *año de muerte*, estas personas pueden ser *directores* de un título o bien ser *actores o actrices*, en cuyo caso sabemos que *personajes* interpretaron.

De los actores o actrices, nos interesa conocer si tienen **co-actores** en común.

Esta aplicación se debe considerar en un contexto de un sitio similar a IMDb, donde los tiempos de lectura deben prevalecer por sobre los tiempos de escritura dado que son datos relativamente estáticos. En este sitio, se quisiera poder buscar los títulos por su título, nombre de personajes, nombre de actores, nombre de directores, géneros o tipo de titulo. En el caso de las películas podríamos querer buscarlas por año y en el caso de un episodio por el identificador de la serie a la que pertenece.

III. Modelado

Para el modelado de la realidad presentada en la Sección II se comienza realizando el Modelo Entidad - Relación (1) de la problemática y luego el Diagrama de Entidad - Relación (2).

En base a esta primer aproximación a la realidad, se presenta un modelo relacional de la misma con la estructura de tablas presentada en la Figura 3.

[1]IMDb Datasets https://www.imdb.com/interfaces/

[2]IMDb https://www.imdb.com/

[3]IMDb Datasets https://www.imdb.com/interfaces/

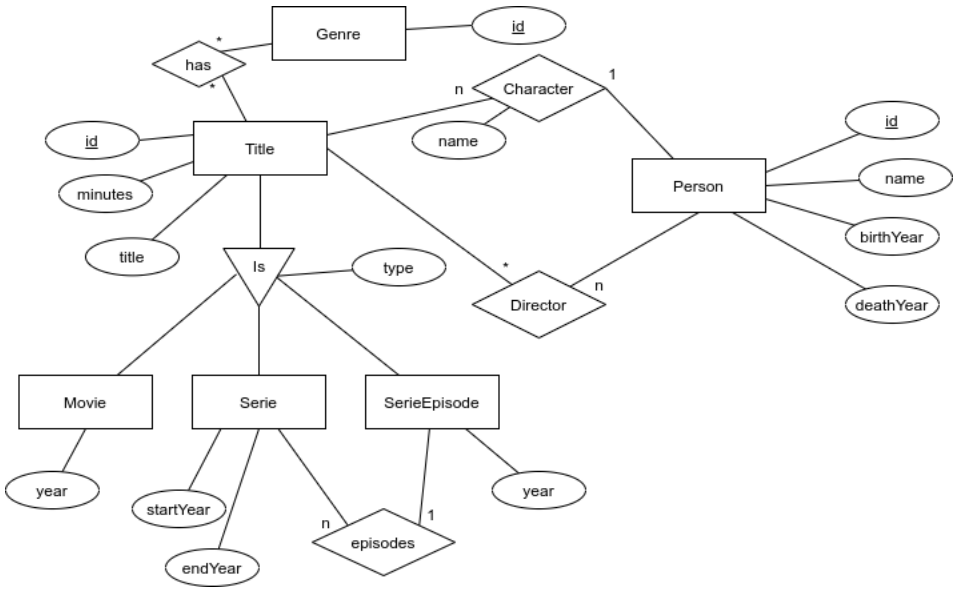

Figura 1: Modelo Entidad Relación

III-A. Modelo Relacional

En el modelado relacional presentado, se intenta obtener una modelado en cuarta forma normal, evitando repeticiones y valores vacíos. Se presenta una tabla **titles** con la información básica de los títulos, luego las tablas **movies, series** y **serie_episodes** contienen la información específica para películas, series y episodios correspondientemente. En el caso de los episodios se tiene una referencia a la serie mediante una ***foreign key***. La información de personas se guarda en la tabla **people** y las relaciones de directores y actores mediante **title_directors** y **title_characters**, en este último caso se tiene también el nombre del personaje interpretado que puede ser más de uno. Finalmente se registran los distintos géneros de cada título en la tabla **title_genres**.

III-B. Modelo Documental

Para el modelado documental, se presentan dos colecciones de documentos, una colección de títulos polimórfica **titles**, donde se almacenan tanto películas, como series o episodios. Debido a que las relaciones entre títulos y actores, directores o géneros son las llamadas uno a poco se siguen las recomendaciones de modelado propuestas por MongoDb[4] [5] (base de datos documental elegida) y se las modela a todas como un arreglo de documentos embebidos. Para las personas relacionadas (ya sean directores o actores) se guarda una proyección básica de los mismo (id y nombre) que es la información necesaria.

La colección de personas **people** tiene la información básica de las personas y dos arreglos de documentos embebidos: uno con proyecciones de los títulos en los que actuó la persona **actedIn**, donde además de nombre e identificador del título se indica el nombre de los personajes en el mismo, y otro arreglo con una proyección básica de los títulos que dirigió **directed** con nombre e identificador.

Ejemplo de un elemento de la colección **titles**

```
{
  id: 'tt0000879',
  title: 'Goodfellas',
  minutes: 146,
  year: 1909,
  characters: [
    {
      actor: {
        id: 'nm1747784',
        name: 'Robert De Niro'
      },
      characters: ['Jimmy Conway']
    }
  ],
  directors: [
    {
      id: 'nm0023107',
      name: 'Martin Scorsese
    }
  ],
  genres: [Mafia'],
  type: 'movie'
}
```

Ejemplo de un elemento de la colección **people**

```
{
  id: 'nm1747784',
  name": 'Robert De Niro',
  birthYear": 1943,
  actedIn: [
    {
      id: 'tt0000879',
      title: 'Goodfellas',
      characters: ['Jimmy Conway']
    }
  ]
}
```

La consistencia de las relaciones se mantiene en momento de actualización / creación para obtener siempre las lecturas

[4]Model One-to-Many Relationships with Document References https://docs.mongodb.com/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/

[5]William Zola, Rules of Thumb for MongoDB Schema Design https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1

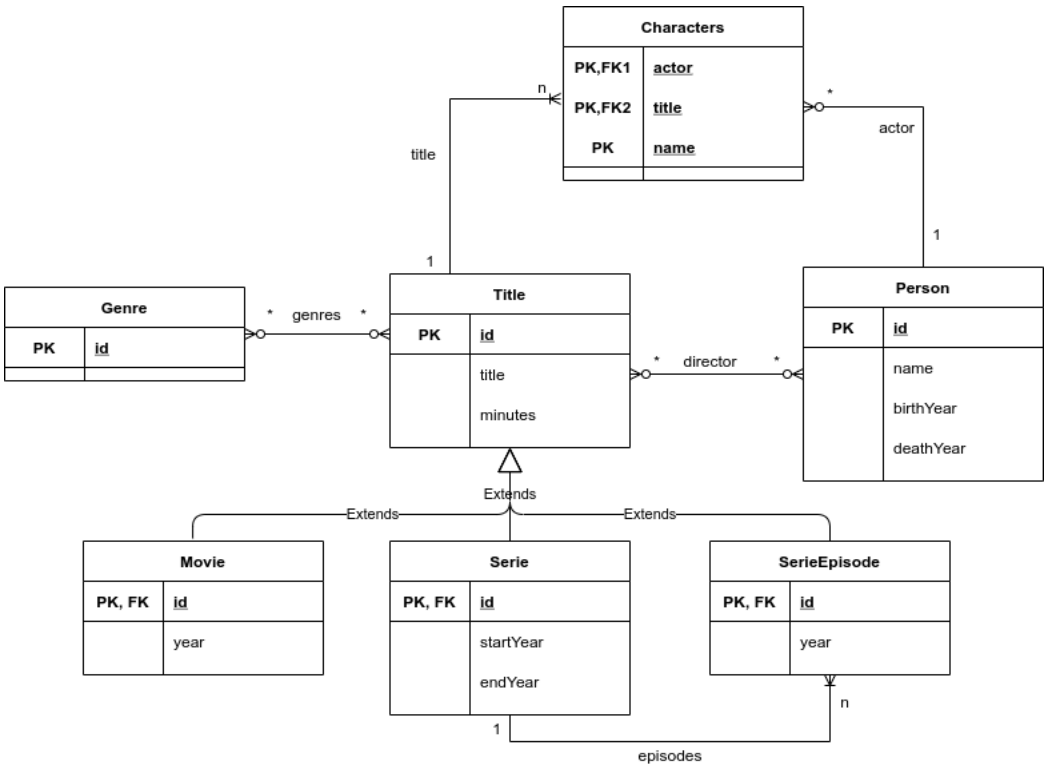

Figura 2: Diagrama Entidad Relación

en una sola lectura, a costo de tener escrituras más lentas, de acuerdo a lo especificado en la descripción de la realidad II.

Las colecciones cuentan ambas con restricciones de unicidad sobre el atributo **id**.

III-C. Modelo de Grafo

Para el modelado de grafos, se modelan los títulos, personas y géneros como nodos indicados con los ***labels*** **Title, Person** y **Genre**. A su vez cada tipo de titulo también se modela como un ***label*** distinto teniendo los ***labels*** **Movie, Serie** y **SerieEpisode**. Las relaciones entre un título y su director se modelan con una arista marcada con el *label* **DIRECTED**, entre un título y sus actores mediante otra arista con el label *label* **ACTED_IN** y a su vez, en cada caso de marca el nodo Persona con los *labels* **Actor** o **Director** según corresponda. En el caso de las aristas **ACTED_IN** tienen como atributo los *roles* o personajes que el actor tuvo en el título. Esto se puede ver en el Diagrama se Base de Datos de Grafos o GDBS, como en la Figura 4.

En este caso también es necesario agregar una restricción de unicidad por el atributo **id** sobre las personas y títulos.

IV. Presentación REST API

La interfaz se plantea siguiendo los lineamientos del protocolo REST [Fielding(2000)] [Fielding(2008)] [res(2016)], se exponen los recursos **Titles, Movies, Series, SerieEpisodes y People** como se ve en el Cuadro I.

La documentación de la API se implementó siguiendo la especificación OpenAPI 2.0 [ope()].

Para cada recurso se expone la creación mediante POST, obtener una instancia por id mediante GET en la URI */<endpoint>/id* del elemento. En la URI de cada recurso, se puede borrar el mismo mediante un DELETE o actualizarlo mediante un PUT.

Los actores y personajes o directores, solamente son configurables a nivel del titulo mediante los recursos */<endpoint>/id/characters* y */<endpoint>/id/directors*. Sobre cada una de estas URI se puede obtener datos mediante GET, eliminar datos mediante DELETE o agregar personajes o directores de un título mediante PUT.

Finalmente, se dispone de una búsqueda genérica para personas, títulos, películas o el recurso correspondiente según lo expresado en la Sección II. Las búsquedas no se implementan con paginado por simplicidad, pero sin dudas es un punto pendiente.

Cuadro I: URI Recursos

Recurso	**URI**	**Descripción**
Titles	*/titles*	todos los títulos
Movies	*/movies*	películas
Series	*/series*	series
SerieEpisodes	*/serie-episodes*	episodios
People	*/people*	personas

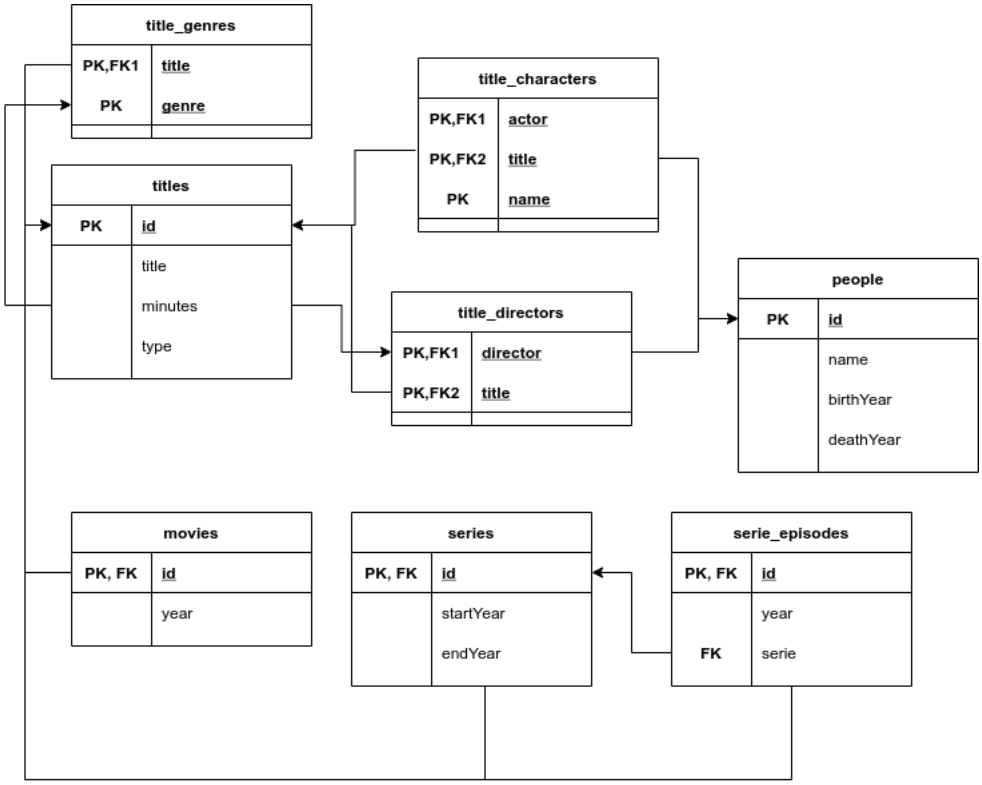

Figura 3: Estructura de Tablas, Modelo Relacional

V. Implementación de la Aplicación

Por cuestiones de tiempo y alcance de este trabajo, no se implementa todo el alcance de la aplicación sino que se limita a Películas, Personas y sus relaciones.

La aplicación **IMDbdnr** es construida sobre el marco de Spring [spr(e)] [6] y sus conectores de Spring Data [spr(a)] para conectarse a las distintas bases de datos. En todas las implementaciones se utilizó una conexión directa a base de datos, sin ningún tipo de ORM [Martin Lorenz and Rudolph()] en el medio para aislar el ruido que pudiera ingresar un ORM en el análisis de los resultados.

Para el alcance de este experimento se decidió utilizar MySQL[7] para el modelo relacional, MongoDb[8] como base de datos documental y Neo4J[9] como base de datos de grafos. A continuación se enumeran las versiones de los manejadores de bases de datos y conectores a las mismas [spr(b)] [spr(c)] [spr(d)]

- Para el modelo relacional se utilizó MySQL en su versión 8.0.25 y las librerías de *spring-jdbc* versión 5.3.8.
- Para el modelo documental se utilizó MongoDb en su versión 5.0.0 y las librerías de *spring-data-mongodb* versión 3.2.1
- Para el modelo de grafos se utilizó Neo4J en su versión 4.1.10 y las librerías de *spring-data-neo4j* versión 6.1.1

La versión de *spring-web* utilizada fue la 5.3.8 junto a *springfox-swagger2* versión 2.7.0 para la documentación de la API.

En los casos donde la consistencia de datos debió ser manejada a nivel de aplicación, como por ejemplo en las proyecciones **actedIn y directed** del modelo documental, o actualizaciones en cascada de algún nombre, esto se realizó siempre a nivel de aplicación y en tiempo de ejecución.

En otros casos fue necesario unir información de distintas fuentes para armar estas proyecciones al retornar datos de consultas, como en el modelo relacional.

Si bien estas manipulaciones podrían afectar el rendimiento de la aplicación e introducir un factor exógeno a la base de datos en sí misma, su consideración es pertinente dado que en una situación real son factores a tomar en cuenta a la hora de evaluar que base de datos utilizar.

En cuanto las búsquedas se resolvieron de la siguiente manera:

VI. Suite de Pruebas

La suite de pruebas fue generada mediante JMeter[10] (versión 5.4.1) y se utilizaron los datos proporcionados por IMDb en sus datos públicos. Si bien en los modelados de grafos y documentales no es necesaria la existencia de las personas

[6]Building a RESTful Web Service https://spring.io/guides/gs/rest-service/
[7]MySQL https://www.mysql.com/
[8]MongoDB https://www.mongodb.com/es
[9]Neo4J https://neo4j.com/
[10]Apache JMeter https://jmeter.apache.org/

para crear una película con sus relaciones, como si lo es en el modelado relacional debido a las claves foráneas, por consistencia se crean siempre primero las personas para que existan previamente.

JMeter permite la configuración de grupos de hilos, es decir que ese conjunto de operaciones serán ejecutadas en bloque y luego se pasa al siguiente grupo (configurado así). Esto se utiliza para poder organizar las invocaciones, creando primero las entidades y luego realizando consultas sobre las mismas. Se realizaron pruebas de validación de los datos obtenidos para tener controlar algún tipo de consistencia en la aplicación, para evitar tener resultados de buen rendimiento pero semánticamente incorrectos.

Las pruebas se dividen en dos partes, una donde se hace foco en las operaciones de alta, baja y modificación con consultas simples para validar estas. Luego se ejecuta un módulo de *browsing* o navegación, donde se intenta simular el comportamiento de un usuario, consultando películas y navegando dentro de de sus actores y directores.

1. Grupo de carga de Personas
   a) Se crean personas con información básica (id, nombre, año de nacimiento y muerte).
   b) Se obtienen por id las distintas personas previamente creadas y se valida la información obtenida.
2. Grupo de carga básica de Películas
   a) Se crean películas con información básica (id, título, año y minutos).
   b) Se obtienen por id las distintas películas previamente creadas y se valida la información obtenida.
3. Personajes de Películas
   a) Se agregan personajes a las películas. Se agregan de aun actor por simplicidad de construcción de la prueba.
   b) Se obtienen los personajes de la película por id y se valida que existan personajes creados para el actor indicado.
   c) Se obtiene la persona / actor y se valida que la proyección actedIn esté actualizada con la película correspondiente.
4. Directores de Películas
   a) Se agregan directores a las películas. Se agregan de a uno por simplicidad de construcción de la prueba.
   b) Se obtienen los directores de la película por id y se valida que exista el director agregado.
   c) Se obtiene la persona / director y se valida que la proyección directed esté actualizada con la película correspondiente.
5. Navegación por Películas
   a) Desde una lista de películas, se consulta las mismas por nombre o id, se toma alguno de sus actores o directores y se consulta por el mismo.
   b) Se realiza una búsqueda de películas por actor, director o título y luego se consulta por cada una de ellas

A continuación se detallan los casos ejecutados, se generar dos conjuntos de datos, uno *full* y otro *basic* para distintas pruebas.

Carga de Personas	118948	446
Carga de Películas	100000	250
Actores y Personajes	250000	400
Directores	100000	234
Browsing [11]	10000	0
Búsqueda y Navegación[12]	15000	0

Cuadro II: Casos de prueba

VII. Infraestructura de despliegue

La aplicación fue desplegada en una instancia de ec2 AWS EC2[13] de tipo ***m5n.xlarge***[14] mientras que JMeter fue ejecutado en una máquina local.

Detalle de ec2:

- Sistema Operativo: Linux
- Procesador: 3.1GHz x 4
- Memoria RAM: 16GiB
- Disco Duro: SSD

Detalle de máquina local:

- Sistema Operativo: Ubuntu 10.04LTS / 64 bits
- Procesador: Intel Core I5-6300U 2.4GHzx4
- Memoria RAM: 32 b
- Disco Duro: SSD 480Gb

VIII. Ejecución de pruebas y resultados obtenidos

Las pruebas se ejecutaron en dos partes, primero las pruebas de cargas con 200 hilos concurrentes con un *ramp up* de 200 segundos, es decir, cada segundo se inician un nuevo hilo. Los resultados se pueden ver en V, III y IV[15].

Por otro lado las pruebas de navegación y búsqueda se realizaron con la misma configuración con la única diferencia que ambos grupos de hilos se ejecutaban en paralelo, por lo que habían 400 hilos concurrentes. Los resultados se pueden observar en VI, VII y IX. En este caso las pruebas fueron limitadas a 10 minutos.

Finalmente, en las pruebas de co-actores se obtuvieron los valores [16]

[13]AWS EC2 https://aws.amazon.com/es/ec2

[14]https://aws.amazon.com/es/ec2/instance-types/m5/

[15]Debido a la demora, en el caso de Neo4J no se ejecutan los sets de pruebas completos

[16]No se implementó en Mongo

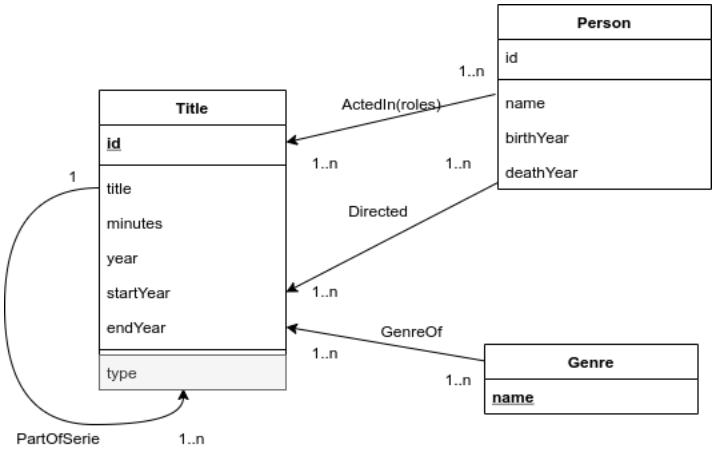

Figura 4: Diagrama de Base de Datos de Grafos

label	avg ms	min ms	max ms	tp/sec
Create People	157	135	1176	417.92
Check People	151	134	600	419.42
Create Movie	184	135	1747	361.58
Check Movie	168	135	1634	361.85
Add Actors	159	135	1080	359.98
Check Actors	155	134	891	360.10
Check actedIn	156	134	896	360.13
Add Directors	162	136	1366	292.72
Check Directors	158	134	1253	292.88
Check directed	159	135	902	292.95

Cuadro III: Resultados Carga MySQL

label	avg ms	min ms	max ms	tp/sec
Create People	478	140	2003	183.96
Check People	441	140	1765	183.77
Create Movie	3353	52	102845	40.16
Check Movie	1150	1	2568	40.04
Add Actors	1835	19	3582	42.16
Check Actors	1257	52	3129	42.04
Check actedIn	763	19	2183	41.94
Add Directors	1289	23	2869	46.97
Check Directors	697	33	2076	46.64
Check directed	381	40	2214	46.42

Cuadro IV: Resultados Carga Neo4J

IX. Conclusiones y trabajo futuro

IX-A. Conclusiones

En las pruebas de carga, el comportamiento se ajusta bastante al esperado. Mongo y MySQL se comportan de manera bastante similar, un poco mejor Mongo. En las operaciones de lectura, donde Mongo tiene la información ya lista las diferencias son más sensibles. Al crear personas o películas tenemos en promedio 157 y 184 contra 141 y 143 pero al consultarlas tenemos 151 y 168 contra 139 y 140. Por otro lado, Neo4J presenta números sensiblemente superiores, 478 y 3353 para las creaciones respectivamente y 441 / 1150 para las consultas.

En los escenarios de búsqueda y navegación los resultados son abrumadoramente favorables a Mongo, donde en el mismo lapso de tiempo se pudieron realizar más consultas, teniendo lógicamente mejores promedio. Pero sin dudas lo que más salta a la vista es que en MySQL se llegan a producir errores lo cual impacta directamente en el rendimiento.

La tasa de errores en el ambiente de Neo4J en los escenarios de navegación fueron demasiados altos, el análisis de los errores muestra que fue por conexiones de mas de 6000ms a la base de datos, es probable que esto lleve a errores de implementación y/o definición de la API.

Los malos resultados de Neo4J y sus alta taza de error llevan a pensar que hay errores de implementación, instalación y configuración que generan los datos obtenidos, por lo cual este trabajo en este sentido simplemente sería un punto de partida

IX-B. Trabajo Futuro

Si bien con la elección de un caso de estudio sencillo se intentó mitigar el efecto de un mal diseño, siempre es un factor a tomar en cuenta. Por lo tanto un trabajo a futuro

label	avg ms	min ms	max ms	tp/sec
Create People	141	134	700	444.96
Check People	139	134	590	445.23
Create Movie	143	134	711	413.28
Check Movie	140	134	483	413.65
Add Actors	147	134	874	381.05
Check Actors	147	134	1421	381.16
Check actedIn	146	134	1420	381.16
Add Directors	161	134	792	292.40
Check Directors	159	134	926	292.57
Check directed	159	134	1043	292.62

Cuadro V: Resultados Carga Mongo

label	samples	avg ms	min ms	max ms	%Err
Check Movie Data	1969	7899	136	54490	3.2
Find Movies By Title	800	13125	201	171427	1.3
Check Director	1890	8519	135	67380	3.14
Check Movie Data Title	1376	10602	136	53753	2.29
Check Actor	7136	9609	135	62199	11.98
Find Movies By Actor	745	20653	182	147037	1.2
Check Movie Data Actor	4595	8732	136	61862	7.6
Find Movies By Director	635	15993	306	114311	1.06
Check Movie Data Director	888	10654	136	62502	1.50

Cuadro VI: Resultados Búsqueda y Navegación MySQL

label	samples	avg ms	min ms	max ms	%Err
Check Movie Data	580	95267	178	438039	27.759
Check Director	393	21891	189	374584	7.888
Check Actor	715	36139	189	436749	12.168
Find Movies By Title	200	405719	10022	538667	40.000
Check Movie Data Title	98	43265	712	423783	16.327
Find Movies By Actor	239	127798	261	255824	50.47
Check Movie Data Actor	307	12433	642	164262	5.86
Find Movies By Director	200	197447	97969	296955	50
Check Movie Data Director	300	12543	567	164372	6.51

Cuadro VII: Resultados Búsqueda y Navegación Neo4J

a realizar es con el mismo caso de estudio, misma API e implementación, realizar las pruebas con otro diseño del modelado. El trabajo de índices puede ser otro foco donde experimentar más, probando distintas estrategias además de las implementadas en este experimento.

Agregar otro modelo de bases de datos no relacionales, como en el experimento de Fraczek, Konrad y Malgorzata [Fraczek and B(2017)] se utiliza Cassandra [17].

Las instalaciones fueron *on premise* mediante imágenes de *docker*[18] en la misma máquina virtual. Sería interesante probar mediante instalaciones en la nube de Mongo Neo4J y alguna soluciones relacional.

Las búsquedas no fueron resueltas mediante *free text search*, sería interesante agregar este aspecto para tomar conclusiones. En cuanto a las búsquedas, sería interesante realizar algún trabajo comparativo de paginado y como el mismo puedo o no afectar el rendimiento.

Finalmente, podría ser interesante, buscar un modelo de persistencia políglota modelando cada sub-problema con un tipo de base de datos distinta.

En cuanto al rendimiento de la base de datos relacional, sería altamente interesante realizar pruebas tomando un

[17]Apache Cassandra https://cassandra.apache.org/

[18]Docker https://www.docker.com/

label	samples	avg ms	min ms	max ms	%Err
Check Movie Data	10000	318	135	9268	0.00
Find Movies By Title	5210	1159	135	10129	0.00
Check Director	10267	313	134	5431	0.00
Check Movie Data Title	6530	1112	134	5439	0.00
Check Actor	46717	306	134	5446	0.00
Find Movies By Actor	5181	7185	352	11893	0.00
Check Movie Data Actor	25801	722	135	5827	0.00
Find Movies By Director	5071	5337	44	10797	0.00
Check Movie Data Director	5490	764	135	3955	0.00

Cuadro VIII: Resultados Búsqueda y Navegación Mongo

modelo	avg ms	min ms	max ms
Neo4J	432	158	1357
MySQL	139	134	277

Cuadro IX: Resultados Búsqueda y Navegación Mongo

modelo relacional desnormalizado y ver se pueden puedo mejorar los rendimientos obtenidos, en espacial en cuanto a las búsquedas.

REFERENCIAS

[ope()] OpenAPI Specification. URL https://swagger.io/specification/v2/.

[spr(a)] Sprint Data, a. URL https://spring.io/projects/spring-data.

[spr(b)] Sprint Data JDBC, b. URL https://spring.io/projects/spring-data-jdbc.

[spr(c)] Sprint Data Mongo, c. URL https://spring.io/projects/spring-data-mongodb.

[spr(d)] Sprint Data Neo4J, d. URL https://spring.io/projects/spring-data-neo4j.

[spr(e)] Spring Framework, e. URL https://spring.io/projects/spring-framework.

[res(2016)] What is REST API?, 2016. URL https://restfulapi.net/.

[Fielding(2000)] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures, 2000. URL https://www.ics.uci.edu/~fielding/pubs/dissertation/top.html.

[Fielding(2008)] Roy Thomas Fielding. REST APIs must be hypertext-driven, 2008. URL https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

[Fraczek and B(2017)] Konrad Fraczek and Malgorzata Plechawska-wojcik B. Comparative analysis of relational and non-relational databases in the context of performance in web applications. volume 716, pages 153–164, 2017. ISBN 978-3-319-58273-3. doi: 10.1007/978-3-319-58274-0. URL http://link.springer.com/10.1007/978-3-319-58274-0.

[Martin Lorenz and Rudolph()] Gunther Hesse Martin Lorenz and Jan-Peer Rudolph. Object-relational Mapping Revised - A Guideline Review and Consolidation. URL https://pdfs.semanticscholar.org/94ee/b9cb199220361c9ad257d505634f37ccb8f8.pdf.