Arquitecturas de Persistencia Políglota
Caso de estudio:
MongoDB y Neo4j

Abelardo Moreno
Escuela de Computación
Licenciatura en Computación
Universidad Central de Venezuela
Caracas, Venezuela
Email: abelardo.moreno@gmail.com

David Fernandez
Escuela de Computación
Licenciatura en Computación
Universidad Central de Venezuela
Caracas, Venezuela
Email: davidchuki58@gmail.com

Alberto Suarez
Escuela de Computación
Licenciatura en Computación
Universidad Central de Venezuela
Caracas, Venezuela
Email: albertosuarez25@gmail.com

***Resumen*—En los últimos años se ha visto un auge en el uso de los sistemas de bases de datos NoSQL y junto a ello se ha popularizado la idea de aplicaciones de Persistencia Políglota. Esta consiste en que gracias a la gran variedad y cantidad de datos, y los diversos servicios que pueden dar las aplicaciones hoy en día, es probable que un único tipo de sistema de almacenamiento no sea capaz de cubrir de forma eficiente todas las necesidades de la aplicación. En este articulo se dará una idea general de las Aplicaciones de Persistencia Políglota dando información acerca de su funcionamiento, arquitectura y motivación; y ademas se hablara específicamente de como aplicar la Persistencia Políglota con MongoDB y Neo4j.**

Palabras Clave: NoSQL, Persistencia Políglota, MongoDB, Neo4j, Neo4j Doc Manager

1. ¿Que es Persistencia Políglota?

Para entender mejor a que se refiere con este término podríamos definir la palabra políglota, su definición es: alguien que habla o escribe varios idiomas; Entonces basado en esa definición pero llevandolo a base datos, podríamos definir persistencia políglota como un conjunto de aplicaciones que utilizan varias tecnologías básicas de base de datos que se usan para resolver problemas complejos de almacenamiento.
Diferentes bases de datos están diseñadas para resolver diferentes problemas. El uso de un único motor de base de datos para todos los requisitos suele conducir a soluciones no eficientes [1]; por lo tanto estos problemas complejos se fragmentan en partes más pequeñas y se aplican diferentes modelos de base de datos según las necesidades específicas de cada requerimiento del problema. Pero esta solucion tambien trae inconvenientes pues es necesario que la aplicación o un servicio maneje la comunicación con los diferentes tipos de almacenamientos y análisis de datos.

2. Origen

Debido a que persistencia políglota es una idea y forma de trabajo, no una aplicación ni una tecnología específica, esta no tiene un origen claro; no hay registros de cuales empresas o aplicaciones fueron las primeras en trabajar de este modo, pero si existe registros de cómo el término se originó.
En 2006, Neal Ford (Director, Software Architect, y Meme Wrangler en ThoughtWorks) [2] acuñó el término programación políglota, para expresar la idea de que las aplicaciones deberían ser escritas en una mezcla de idiomas para aprovechar el hecho de que diferentes lenguajes son adecuados para abordar diferentes problemas. Aplicaciones complejas combinan diferentes tipos de problemas, por lo que escoger el lenguaje adecuado para cada trabajo puede ser más productivo que tratar de encajar todos los aspectos en un solo idioma [3].
Con el paso del tiempo, se ha aplicado este método de trabajo al área de base de datos y este ha ido evolucionando en este campo, manteniendo la idea original la cual es trabajar con distintos tipos de tecnologías pero ahora en vez de lenguajes de programación se utilizan distintas base de datos dentro de una misma aplicación, buscando guardar los datos de la forma más optima, ya no se aplica el término de programación y se toma el de persistencia porque estamos trabajando con el objetivo de guardar una gran cantidad de datos a través del tiempo, al final utilizamos el término persistencia políglotas para definir este enfoque híbrido.

3. Arquitectura

Hay distintas enfoques para aplicar y estructurar persistencias políglota para resolver los problemas que tenga la organización con su aplicaciones y base de datos, los distintos tipos de enfoques son:

3.1. Enfoque sin Persistencia Políglota

La aplicación utiliza un único almacén de datos, en este enfoque se obliga al almacén a responder ante los distintos tipos de datos que son recibidos [fig1].

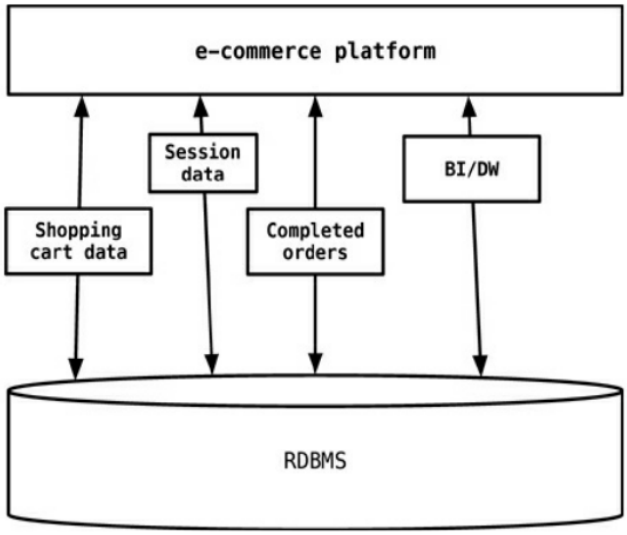

Figura 1. Ejemplo de aplicación ecommerce con una base de datos RBMS que se encarga de todos los requerimientos [1]

3.2. Enfoque con Persistencia Políglota

No es necesario que la aplicación utilice un único almacén de datos para todas sus necesidades. Se crean diferentes bases de datos para diferentes propósitos. La aplicación se encarga de mantener la consistencia de la bases de datos[fig2].

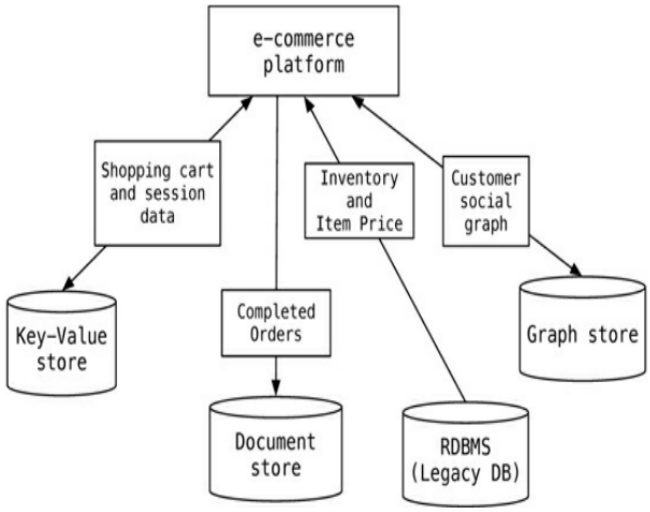

Figura 2. Ejemplo de aplicación ecommerce con varias bases de datos para cada requerimiento [1]

3.3. Enfoque Usando Servicios

En este enfoque se utilizan servicios para englobar una base de datos esto sustituyen la comunicación directa con la base de datos. Esto permite que la aplicación no se encargue de guardar los datos, todos los datos puedan ser guardadas en un lugar del servicio y consultadas por todas las aplicaciones . Las API proporcionadas por el servicio son más útiles que una única aplicación que habla con varias bases de datos[fig3].

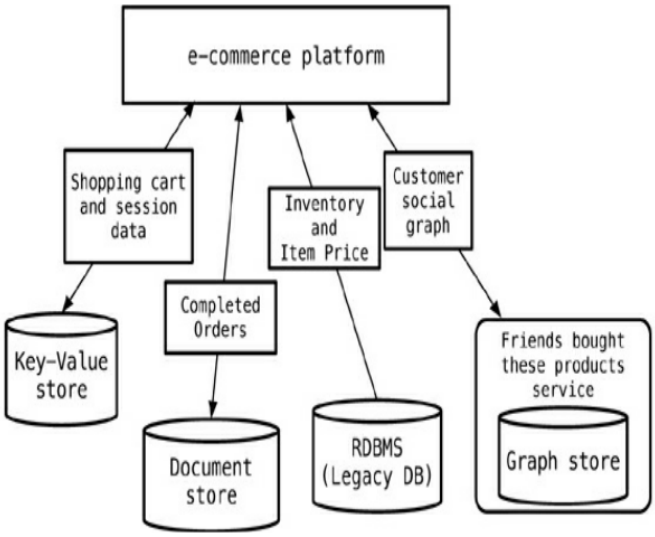

Figura 3. Ejemplo de aplicación ecommerce con un servicio que engloba la base de datos de grafos [1]

3.4. Enfoque Usando Servicios en todas las Base de Datos

Este enfoque es como el anterior pero se envuelve todas las bases de datos en servicios. Permite que las bases de datos dentro de los servicios evolucionen sin tener que cambiar las aplicaciones dependientes[fig4].

3.5. Enfoque Usando Dos capas de Servicios

En este enfoque se aplica las características del enfoque anterior pero se agregan capas adicionales de servicios. La nueva capa de servicio agregada se le trata como un maestro que administrara las capas de nivel inferior. Este servicio maestro permite que la aplicación se desvincule completamente de la tarea de encargarse de mantener la consistencia y el análisis de los datos de las distintas base de datos, ahora el servicio maestro es el que mantiene la consistencia y la entronización entre las distintas base de datos [fig5].

3.6. Enfoque Expandiendo Funcionalidades

Existen casos en los que no podemos cambiar el almacenamiento de datos. Debido a las aplicaciones legadas

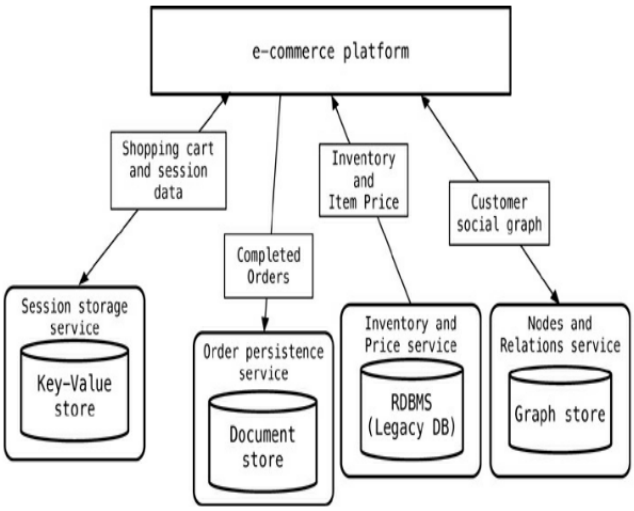

Figura 4. Ejemplo de aplicación ecommerce con servicios en cada base de datos [1]

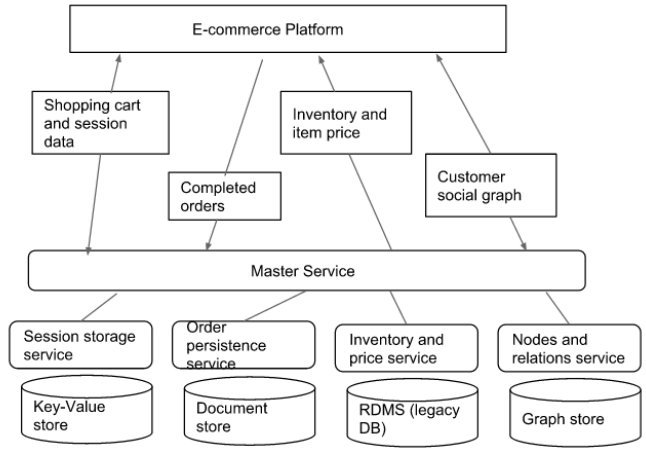

Figura 5. Ejemplo de aplicación ecommerce con servicios en cada base de datos y una capa adicional llamada servicio maestro que engloba los demás servicios

existentes y su dependencia del almacenamiento de datos existente. En estos casos podemos aplicar persistencia políglota de una forma diferente. en vez de una base de datos que maneje directamente requerimientos de la aplicación, podemos agregar una base de datos que trabaje en paralelo y ayude a una base de datos ya existentes, esto le agrega funcionalidades al sistema que ya tenemos para mejorar su desempeño[fig6].

4. Ventajas y Desventajas

La Persistencia Políglota presenta una serie de ventajas y desventajas que tienen que ser analizadas al momento de querer utilizar la Persistencia Políglota en algún proyecto de software.

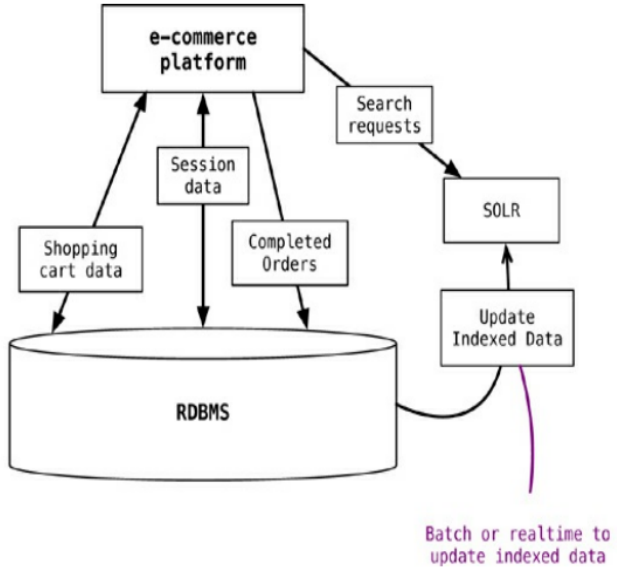

Figura 6. Ejemplo de aplicación ecommerce con una base de datos RBMS y una base de datos SOLR paralela indexar y optimizar la primera [1]

4.1. Ventajas

- Mejoras de rendimiento, ya que se utiliza una Base de Datos dependiendo de las necesidades funcionales que se tengan.
- Nos permite manejar mayor volúmen de datos, debido a que las Bases de Datos NoSQL están diseñadas para operar en cluster. Esto nos permite un escalamiento horizontal.
- Las consultas son mucho más flexibles porque se adaptan a la necesidad y están directamente enlazadas al formato de datos manejado por la base de datos.

4.2. Desventajas

- Dificultad para que la data almacenada entre las distintas bases de datos sea consistente.
- Ahora hay que decidir cuál Base de Datos se va a utilizar de entre todas las que hay disponibles, esto puede traer problemas si no se tiene mucha experiencia.
- La complejidad de la aplicación aumenta debido a que ahora hay que manejar mayor cantidad de Bases de Datos y ademas que funcionen en conjunto todas ellas.
- Se adquieren nuevas responsabilidades administrativas porque ahora hay que estar a cargo de muchas mas Bases de Datos y asegurar su funcionamiento.
- Si no se tiene ninguna experiencia con las Bases de Datos que se quieren utilizar es necesario entrenamiento, el cual lleva cierto tiempo y afecta en parte al desarrollo.

The real world of Database Administrator study by Dell (March 2015) [4]

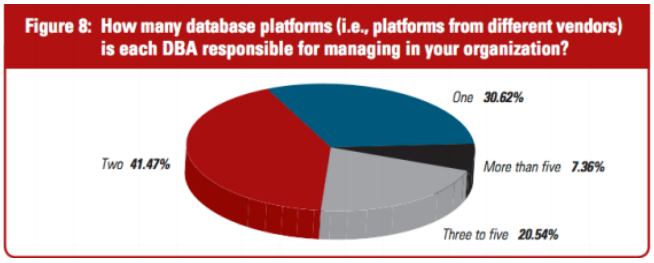

Figura 7. ¿De cuántas Bases de Datos son responsables los Administradores de Bases de Datos ? [4]

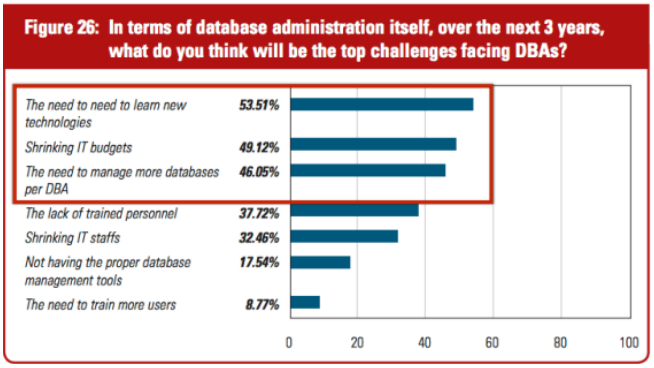

Figura 8. ¿En términos de Administración de Bases de Dato, en los próximos 3 años, cuáles serán los retos para los administradores de Bases de Datos ? [4]

5. Consideraciones

Existen varias consideraciones que se pueden tomar antes de querer utilizar Persistencia Políglota en alguno de nuestros proyectos. A continuación se van a explicar cada una de ellas.

5.1. Strategic Project

La mayoría de los proyectos de software de una empresa son denominados Utility Project, es decir; que son aquellos desarrollos que no le traen ninguna ventaja competitiva en el mercado a la empresa pero son importantes de todos modos. Para este caso de proyectos (Utility Project), no es tan beneficioso usar Persistencia Políglota debido a que la complejidad y el tiempo de desarrollo sería mayor y a su vez no valdría tanto la pena hacer dicho esfuerzo. En cambio para los proyectos denominados Strategic Project si se vería el beneficio, ya que en comparación al anterior estos si traen alguna ventaja competitiva a la empresa y por ende sería de gran utilidad usar las mejores herramientas para su desarrollo.

5.2. Rapid Time to Market

Si se necesita un Proyecto de Software que salga lo más rápido al mercado entonces necesitas que la productividad sea mucho mayor, por lo tanto si se cuenta con la suficiente experiencia el uso de Persistencia Políglota sería una buena opción, ya que para cada necesidad que se tenga se utilizaría la tecnología más adecuada y esto traería consigo un incremento en la productividad.

5.3. Data Intensive

Cuando hablamos de Data Intensive nos podemos referir a:

- Gran cantidad de data.
- Gran disponibilidad.
- Gran cantidad de tráfico: tanto lectura como escritura.
- Relaciones muy complejas.

Cualquiera de estos aspectos nos puede dar a entender que necesitamos algún tipo de almacenamiento No-Relacional, pero es en realidad la naturaleza de la data lo que nos permitirá escoger si usar Persistencia Políglota o no.

6. Casos de Uso

Un caso de uso muy común de Persistencia Políglota es el de una plataforma de comercio electrónico (e-commerce). Esta plataforma tratará muchos tipos de datos (por ejemplo, carrito de compras, inventario, pedidos completados, etc.). [5]

6.1. Modelo Tradicional

Un primer enfoque sería almacenar todos estos datos en una sola Base de Datos [fig9], lo que requeriría una gran cantidad de conversión de datos para que el formato de los datos sea igual.

Figura 9. Modelo Tradicional de Desarrollo en el que se usa una sola Base de Datos

6.2. Persistencia Políglota

Mediante Persistencia Políglota se pueden almacenar los datos en la base de datos más adecuada para cada tipo de datos [fig10].

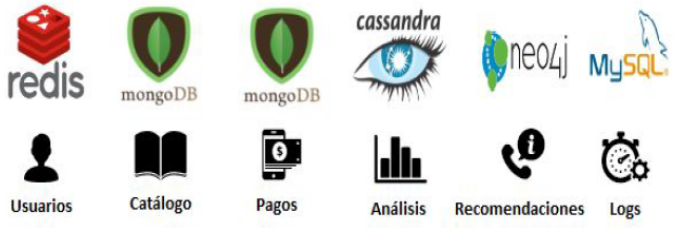

Figura 10. Mediante Persistencia Políglota se observan las distintas Bases de Datos utilizadas para cada funcionalidad

Por lo tanto, estamos utilizando una mezcla de soluciones RDBMS (por ejemplo, SQL Server) con soluciones NoSQL de las cuales hay cuatro tipos: Valor-clave, Documento, Gráfico, Columna (ver Tipos de bases de datos NoSQL). Una guía [fig11] sobre el tipo de base de datos a utilizar en función de la funcionalidad de los datos:

Funcionalidad	Consideraciones	Tipo de Base de Datos
Sesiones de Usuario	Acceso rápido para lectura y escritura. No necesita perdurar.	Clave-Valor (Redis)
Data Financiera	Se necesitan transacciones. Estructura fija.	Relacional (MySQL)
Carro de Compras	Alta disponibilidad.	Documental (MongoDB)
Recomendaciones	Buscar conexiones o enlaces entre amigos, productos comprados, etc.	Grafos (Neo4J)
Catálogo de Productos	Muchas lecturas y muy pocas escrituras.	Documental
Reportes	SQL se comporta bien con este tipo de funcionalidad	Relacional, Columnas
Analítica	Grandes análisis en clusters	Columnas

Figura 11. Cuadro con cada una de las funcionalidades de la aplicación e-commerce y porqué se usó cada una de las Base de Datos

7. Trabajos Relacionados

A continuación se presentan 3 ejemplos de desarrollos de software que han optado por la persistencia políglota y tienen gran éxito en la actualidad.

7.1. Wanderu

Wanderu [fig12] es una aplicación web que provee a los usuarios la búsqueda de tickets para bus y tren, con trayectos que pueden combinar múltiples compañías de transporte. La data de las rutas está almacenada en JSON, lo que permite que una Base de Datos como MongoDB sea una buena solución para almacenar dichas rutas.

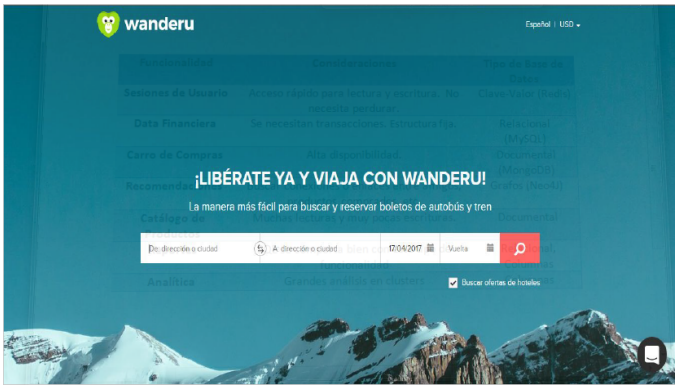

Figura 12. Interfaz Gráfica de Wanderu

Además, ellos también necesitan ser capaces de buscar el camino mas óptimo desde el origen hasta el destino. Esto es perfecto para una base de datos orientada a grafos como por ejemplo: Neo4j, ya que Neo4j puede manejar las relaciones entre cada una de las rutas o trayectos.
Wanderu no quería forzar MongoDB (un almacén de datos basado en documentos) para que manejara las relaciones de estilo gráfo porque la implementación habría sido costosa e ineficiente. En su lugar, utilizaron un enfoque de persistencia políglota para capitalizar las fortalezas de cada uno, decidiendo usar tanto MongoDB como Neo4j juntos. [6]

7.2. Twitter

Figura 13. Interfaz Gráfica de Twitter

Twitter [fig13] es un servicio de microblogging, con sede en San Francisco, California, con filiales en San Antonio (Texas) y Boston (Massachusetts) en Estados Unidos. Twitter, Inc. fue creado originalmente en California, pero está bajo la jurisdicción de Delaware desde 2007. Desde que Jack Dorsey lo creó en marzo de 2006, y lo lanzó en julio del mismo año, la red ha ganado popularidad mundial y se estima que tiene más de 500 millones de usuarios, generando 65 millones de tweets al día y maneja más de 800 000 peticiones de búsqueda diarias. Ha sido denominado como el ”SMS de Internet”. [7]

La red permite enviar mensajes de texto plano de corta longitud, con un máximo de 140 caracteres, llamados tweets, que se muestran en la página principal del usuario. Los usuarios pueden suscribirse a los tweets de otros usuarios – a esto se le llama 'seguir' y a los usuarios abonados se les llama 'seguidores', 'followers' y a veces tweeps' ('Twitter' + 'peeps', seguidores novatos que aún no han hecho muchos tweets). Por defecto, los mensajes son públicos, pudiendo difundirse privadamente mostrándolos únicamente a unos seguidores determinados. Los usuarios pueden twitear desde la web del servicio, con aplicaciones oficiales externas (como para teléfonos inteligentes), o mediante el Servicio de mensajes cortos (SMS) disponible en ciertos países. Si bien el servicio es gratis, acceder a él vía SMS comporta soportar tarifas fijadas por el proveedor de telefonía móvil. [7]

Entre las tecnologías de almacenamiento que utiliza actualmente Twitter tenemos [8]:

- MySQL para almacenar tanto tweets como usuarios.
- FlockDB (una Base de Datos orientada a grafos) en la cual almacenan las relaciones entre los usuarios (following, follower, etc).
- Redis para almacenar el timeline por su velocidad para lectura y escritura.
- Cassandra se dice que se ha empezado a utilizar para migrar lo que se tiene en MySQL y así proveer una mayor escalabilidad.

7.3. Linkedin

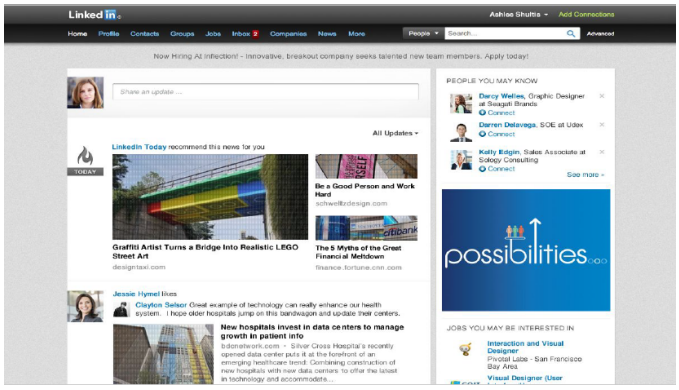

Figura 14. Interfaz Gráfica de Linkedin

LinkedIn [fig14] es una comunidad social orientada a las empresas, a los negocios y el empleo. Partiendo del perfil de cada usuario, que libremente revela su experiencia laboral y sus destrezas en un verdadero currículum laboral, la web pone en contacto a millones de empresas y empleados. Fundada en diciembre de 2002 por Reid Hoffman, Allen Blue, Konstantin Guericke, Eric Ly y Jean-Luc Vaillant, fue lanzada en mayo de 2003. [9]

Entre las tecnologías de almacenamiento que utiliza actualmente LinkedIn tenemos [10]:

- Oracle para almacenar la mayoría de sus datos.
- Voldemort (una Base de Datos clave-valor) en la cual almacenan los datos de los perfiles de los usuarios y así obtenerlos mucho más rápido.

8. Caso de Estudio: MongoDB y Neo4j

Hasta ahora hemos hablado de la persistencia políglota de forma general, en esta sección ahondaremos en un caso especifico, combinar las virtudes de MongoDB y Neo4j, el porque y como se combinarían estos sistemas manejadores de base de datos.

8.1. MongoDB

MongoDB[fig15] es una base de datos documental trabaja con archivos json, lo que permite que tenga un modelo de datos flexible, provee alta disponibilidad gracias al sharding ó escalabilidad horizontal y puede almacenar una gran cantidad de datos y accederlos de forma rápida. [11]
Tiene como desventaja el hecho de que realizar las consultas puede resultar ser un trabajo bastante complejo y tedioso.

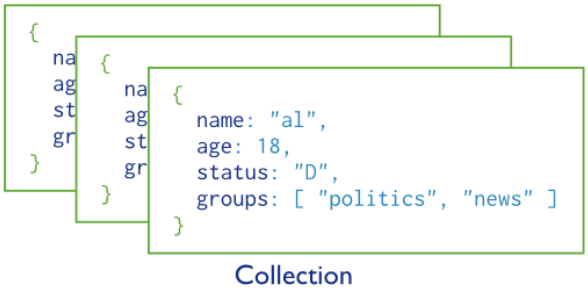

Figura 15. Colecciones en MongoDB [12]

8.2. Neo4j

Neo4j[fig16] es una base de datos de grafos, este implementa los grafos de forma nativa, a diferencia de otros sistemas manejadores de base de datos que no solamente están dedicados al manejo de grafos. [13]
Tambien provee el lenguaje de consultas Cypher, que hace que estás sean muy sencillas.
Una base de datos de grafos tiene como fortaleza la representación de relaciones y el recorrido de caminos.

8.3. MongoDB + Neo4j

Ya una vez vistas las ventajas que nos ofrece cada sistema manejador de base de datos, podemos darnos cuenta de que si combinamos ambos[fig17] podemos tener la capacidad de almacenar gran cantidad de datos de MongoDB junto con el poder de representar las relaciones y los caminos entre

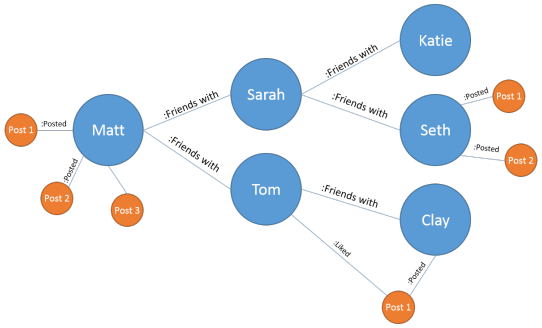

Figura 16. Grafo en Neo4j [14]

nodos fácilmente de Neo4j en una sola aplicación.
Esto nos es de utilidad en por ejemplo una aplicación de cursos de una universidad[fig18], donde los cursos pueden ser almacenados y manejados con MongoDB mientras que un sistema de recomendaciones de los cursos, basados en los que ya se han visto con anterioridad, puede ser manejado con Neo4j.

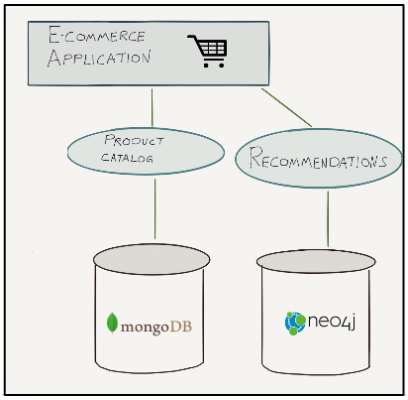

Figura 17. Ejemplo de una aplicacion utilizando MongoDB y Neo4j [15]

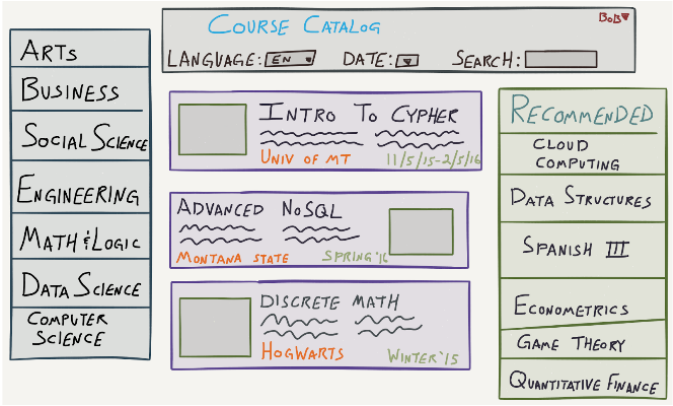

Figura 18. El catalogo de cursos es manejado con MongoDB y las recomendaciones con Neo4j [15]

8.4. Conectando MongoDB con Neo4j

Existen muchas formas y métodos de conectar estos dos sistemas manejadores de base de datos para que trabajen conjuntamente, aquí solo estudiaremos haremos referencia a las que hacen una conexión unidireccional de MongoDB a Neo4j, es decir, en donde solo los datos son transferidos de MongoDB a Neo4j.

8.4.1. ETL:. Un ETL o programa de extracción, transformación y carga, es la forma mas básica de transferir los datos de una base de datos a otra, pero es la que requiere mas trabajo, depende completamente del desarrollador de que forma sera implementado y las soluciones son a la medida, es decir, no siempre existe un único ETL que funcione para cada caso.

8.4.2. APOC:. Otra forma de conectar MongoDB con Neo4j es utilizando las APOC, acronimo de Awesome Procedures, estas son unas librerías que proveen herramientas para realizar la importación y exportación desde y hacia Neo4j de forma mas sencilla, es una alternativa mas general que el de un ETL, pero aun así requiere de un gran trabajo por parte del desarrollador. [16]

8.4.3. Neo4j Doc Manager:. Este método es el que hablaremos de forma mas detallada en este articulo, se trata de una herramienta proveída por los desarrolladores de Neo4j para automatizar la transferencia de los datos de MongoDB a Neo4j. [17]

8.5. Neo4j Doc Manager

Como se dijo anteriormente, esta es una herramienta proveída por los desarrolladores de Neo4j. Esta herramienta es una especie de extensión del Mongo Connector, que es la manera en la que MongoDB puede comunicarse con otros lenguajes de programación o sistemas manejadores de base de datos, pero no soporta de forma nativa la conexión con Neo4j, es por esto que se tuvo que crear el Neo4j Doc Manager.[fig19]
Usando el Neo4j Doc Manager, al momento de insertar un nuevo documento en MongoDB, este es transformado de forma inmediata a un grafo de Neo4j, permitiendo que los datos sean compartidos de forma fácil y rápida entre ambos manejadores de base de datos.

8.5.1. Requerimientos. Para utilizar el Neo4j Doc Manager es necesario tener los siguiente programas instalados:

- MongoDB con un set de replicas.
- Neo4j.
- Mongo Connector.
- Neo4j Doc Manager.

Ya con esto se puede empezar a trabajar con ambos manejadores de base de datos.

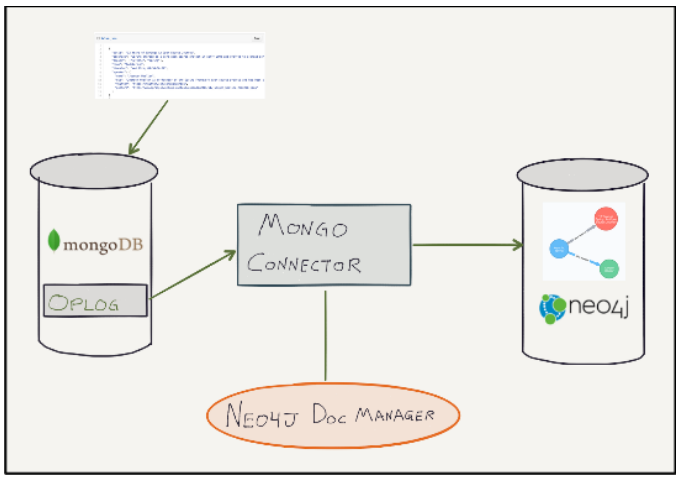

Figura 19. Conexion entre MongoDB y Neo4j utilizando el Neo4j Doc Manager [15]

8.6. Neo4j Doc Manager: Consideraciones

Aunque el Neo4j Doc Manager nos permite automatizar la conversión de los documentos de MongoDB a Neo4j, hay que tener en cuenta ciertas consideraciones para que esta conversión ocurra de manera exitosa.

8.6.1. Documento Simple. En este ejemplo podemos ver como se hace la conversión de un documento simple de MongoDB. [fig20]

```
db.personas.insert([
{
    nombre:"Abelardo",
    apellido:"Moreno"


},
{
    nombre:"David",
    apellido:"Fernandez"

},
{
    nombre:"Alberto",
    apellido:"Suarez"

}
])
```

Figura 20. Documento simple

Figura 21. Conversión en Neo4j

Vemos que cada documento se le asigna un nodo, y no existen relaciones entre ellos.[fig21]

8.6.2. Documento sin tomar en cuenta las consideraciones. En este ejemplo, usamos un documento de MongoDB que no ha sido modificado con las consideraciones para hacer una buena conversión utilizando el Neo4j Doc Manager. [fig22]

```
db.criaturas.insert([{
    "_id":"a02",
    "nombre":"ashwinder",
    "clasificacion":"beast",
    "tipo":"reptil",
    "region":["global"],
    "designacion":"XXX",
    "reproduccion":"oviparo",
    "tiempo_de_vida":0.001,
    "habitos":["colocan sus huevos en esquinas oscuras"],
    "caracteristicas":[{
            apariencia:"de serpiente delgada",
            adiestrable:"no",
            venenosa:"no",
            mordida:"ninguna",
            color:"gris"
            }],
    "otros":[{
            lengua:"parseltongue"
            }]

},
```

Figura 22. Documento sin seguir consideraciones

Se puede ver de que la conversión no nos genero un grafo que pueda ser utilizable, ya que hay nodos sueltos sin relaciones y no toda la información se encuentra desplegada como se quisiera.[fig23]

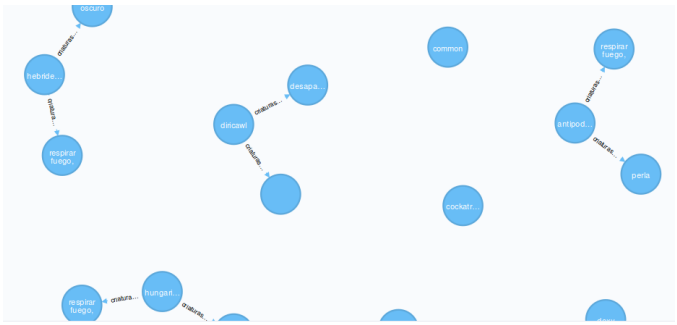

Figura 23. Grafo de documento sin seguir consideraciones

8.6.3. Consideraciones 1: Relaciones. Las relaciones son representadas a través de los subdocumentos dentro de un documento, es decir, por cada subdocumento se crearа un nuevo nodo y una relacion hacia el, en este ejemplo se puede observar un documento con múltiples subdocumentos.[fig24]

Una vez insertado, se observa como el grafo creado posee un nodo con tres relaciones hacia tres nodos diferentes, representando el documento de exposiciones con su información como quien la dicta, en donde y los temas.[fig25]

8.6.4. Consideraciones 2: Relaciones con arreglos. Otra manera de representar las relaciones es con los arreglos de

```
db.exposiciones.insert(
{
  "_id" : "a01",
  "contenido" : {
    "titulo" : "Persistencia Poliglota",
    "sitio" : {
      "lugar" : "CISI"
    }
  },
  "topics" : [
    "NoSQL",
    "mongo",
    "neo4j",
    "persistencia poliglota"
  ],
  "dictada" : {
    "nombre" : "Abelardo Moreno",
    "email" : "abelardo.moreno@gmail.com"
  }
}
)
```

Figura 24. Documento con múltiples subdocumentos

Figura 25. Grafo generado de documento con múltiples subdocumentos

```
db.exposiciones.insert(
{
  "_id" : "a01",
  "contenido" : {
    "titulo" : "Persistencia Poliglota",
    "sitio" : {
      "lugar" : "CISI"
    }
  },
  "topics" : [
    "NoSQL",
    "mongo",
    "neo4j"
  ],
  "bd":[{
    "nombre":"MongoDB",
    "herramienta":"mongo connector"
  },{
    "nombre":"Neo4j",
    "herramienta":"neo doc manager"
  }],
  "dictada": {
    "nombre" : "Abelardo Moreno",
    "email" : "abelardo.moreno@gmail.com"
  }
}
)
```

Figura 26. Documento con arreglo de subdocumentos

subdocumentos, en este caso por cada elemento del arreglo, se creara un nuevo nodo con una relacion hacia el.[fig26]

Vemos entonces que el grafo generado posee un los nodos que se corresponden a el arreglo de subdocumentos, cada uno identificado con la relación bd0 y bd1 que corresponde a su posición en el arreglo.[fig27]

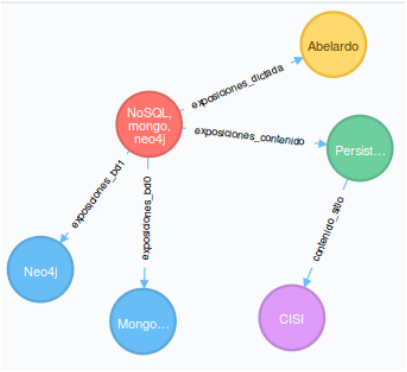

Figura 27. Grafo con relación a partir de arreglos de subdocumentos

8.6.5. Consideraciones 3: Relaciones entre diferentes colecciones. La ultima consideración a tener, es la capacidad de poder relacionar diferentes colecciones de MongoDB en un solo grafo, para esto se debe de crear una referencia en el documento hacia el documento de otra colección.
En la imagen se pueden ver como se referencia la colección lugar con la colección personas.[fig28]

```
db.lugar.insert({
        _id:"r01",
        "nombre":"UCV"
})

db.personas.insert(
{
    _id:"01",
    nombre:"Abelardo",
    lugar_id:"r01"
})

db.personas.insert(
{
    _id:"02",
    nombre:"David",
    lugar_id:"r01"
})

db.personas.insert(
{
    _id:"03",
    nombre:"Alberto",
    lugar_id:"r01"
})
```

Figura 28. Colecciones distintas referenciandose

Vemos entonces que en Neo4j se creo un grafo en el que los nodos personas están relacionados con el nodo lugar.[fig29]

8.6.6. Aplicaciones 1. Ya una vez vistas todas las consideraciones necesarias, podemos aplicarlas para crear un grafo

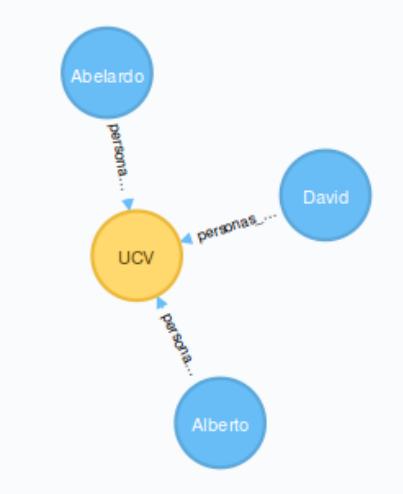

Figura 29. Grafo resultado de dos colecciones distintas referenciandose

que nos permita hacer consultas que nos puedan servir en nuestras aplicaciones, en este ejemplo tenemos un grafo con todos los estudiantes del curso y como estan relacionados entre ellos.[fig30]

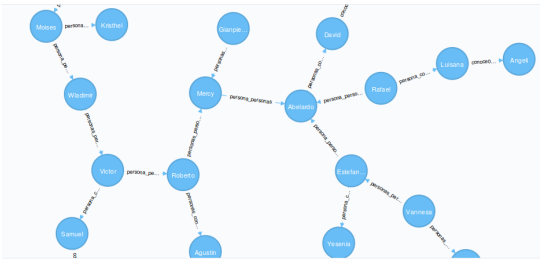

Figura 30. Grafo representando a los estudiantes del curso

A partir de este grafo podríamos realizar una consulta que nos diga, que en tal caso de que ocurra una epidemia y el paciente 0 sea un miembro del grafo, que otros miembros se infectarían.[fig31]

```
MATCH (miembro:persona {nombre:""})-[*1..2]-(salon)
RETURN DISTINCT miembro,salon
```

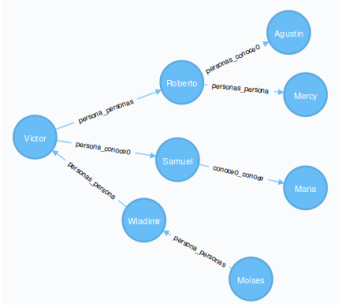

Figura 31. Consulta representando la propagación de una epidemia

8.6.7. Aplicaciones 2. En este ultimo ejemplo, tenemos un grafo que representa los productos de una tienda y quienes los han comprado.[fig32]

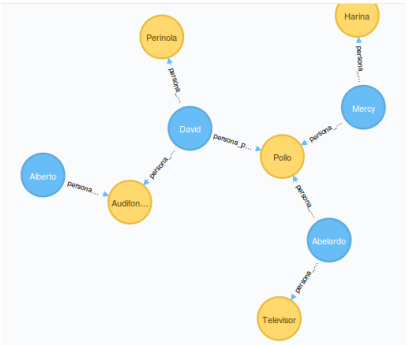

Figura 32. Grafo representando productos y clientes

A partir de este grafo podemos hacer consultas que nos puedan decir que productos recomendar a los clientes, a partir de los que ha comprado y los que otros clientes tambien han comprado.[fig33]

```
MATCH (person:persona)-->(producto),
      (producto)<--(similar:persona),
      (similar)-->(n)
WHERE person.nombre= 'Abelardo'
RETURN n
```

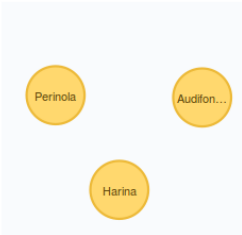

Figura 33. Consulta de recomendaciones a un usuario

8.7. Conclusión

Una vez vistos todos los beneficios que posee la persistencia políglota, no nos queda mas que llegar a la conclusión de que las tecnologías tienden hacia este paradigma y que en un futuro todas nuestras aplicaciones manejaran mas de un sistema manejador de base de datos, lo que implicara que nosotros como desarrolladores y administradores de base de datos tengamos que ser capaces de manejar todos estos diferentes manejadores y saber en que momento nos conviene combinar un manejador con otro.

Referencias

[1] M. F. Pramod J. Sadalage, "Nosql distilled a brief guide to the emerging world of polyglot persistence," 2013, [Online; accessed 20-may-2017].

[2] N. Ford, "about me," 2006, [Online; accessed 20-may-2017].

[3] ——, "Polyglot programming," 2006, [Online; accessed 20-may-2017].

[4] E. King, “The real world of database administrator,” 2015, [Online; accessed 17-may-2017].

[5] J. Serra, “What is polyglot persistence?” 2015, [Online; accessed 05-may-2017].

[6] R. Boyd, “Polyglot persistence case study: Wanderu + neo4j + mongodb,” 2015, [Online; accessed 05-may-2017].

[7] https://es.wikipedia.org/wiki/Twitter, “Twitter,” 2017, [Online; accessed 05-may-2017].

[8] Anónimo, “Which database system(s) does twitter use,” 2012, [Online; accessed 05-may-2017].

[9] https://es.wikipedia.org/wiki/LinkedIn, “Linkedin,” 2017, [Online; accessed 05-may-2017].

[10] Anónimo, “What is linkedin’s database architecture like?” 2010, [Online; accessed 05-may-2017].

[11] https://www.mongodb.com/what-is mongodb, “What is mongodb?” 2017, [Online; accessed 05-may-2017].

[12] https://docs.mongodb.com/v3.2/core/databases-and collections/, “Colecciones en mongodb,” 2017, [Online; accessed 05-may-2017].

[13] https://neo4j.com/developer/graph database/, “What is a graph database?” 2017, [Online; accessed 05-may-2017].

[14] http://ugamarkj.blogspot.com/2014/12/a-newbies-experience-with neo4j.html, “A newbie’s experience with neo4j,” 2017, [Online; accessed 05-may-2017].

[15] https://neo4j.com/blog/neo4j-doc-manager-polyglot-persistence mongodb/, “Neo4j doc manager: Polyglot persistence for mongodb and neo4j,” 2017, [Online; accessed 05-may-2017].

[16] https://github.com/neo4j-contrib/neo4j-apoc procedures, “Apoc, awesome procedures,” 2017, [Online; accessed 05-may-2017].

[17] https://github.com/neo4j contrib/neo4j_doc_manager, “Neoj4 doc manager,” 2017, [Online; accessed 05-may-2017].