Trabajo práctico final

Materia: "Captura y Almacenamiento de la Información"

Posgrado UNLP, Marzo 2020

Alumno: Germán Concilio.

Profesor: Bazzocco Javier

Introducción

En el proyecto "Sistema de apoyo a la detección y monitoreo inteligente de personas con COVID-19" [1] del Laboratorio de Sistemas de Información Avanzados (LSIA) de la Facultad de Ingeniería de la Universidad de Buenos Aires, se desea explotar la información almacenada.

El proyecto está en etapa de desarrollo de la aplicación móvil recolectora de información. El universo de datos recolectados contará con información de seguimiento de contactos entre personas generados por distintos mecanismos de detección de interacción social como: carga manual de DNI, proximidad por Bluetooth y Wi-fi. Estos contactos generados contarán serán enriquecidos con información temporal para poder determinar con más precisión el peligro de contagio.

La cantidad de información almacenada será directamente propocional a la cantidad de usuarios que adepten la aplicación. Estudios epidemológicos recientes [2][3] demuetran que la aplicación de este puede ser un mecanismo de batalla contra el virus sólo si es adoptada por más del 70% de la población. La institución espera que si nuestro proyecto es adoptado, la aplicación sea probada en un conjunto de usuarios de prueba de una localidad pequeña y luego se lanzaría para un rango más amplio como la Ciudad Autónoma de Buenos Aires.

Se necesita desarrollar un mecanismo de almacenamiento que permita explotar fácilmente los datos. El servicio debe permitir realizar de forma eficiente una serie de consultas relacionadas con la detección de contactos directos e indirectos de un paciente diagnosticado positivo de COVID-19.

Al día de hoy no existen bases de datos de "contact tracing" o seguimiento de contactos, ni contamos con datos propios para probar la arquitectura. Es por ello que se buscó una base de datos pública de contactos sociales que aproxima la tipología del grafo de contactos que se generará con la aplicación que vamos a desarrollar.

Set de datos elegido: Friendster

Friendster [4] era una red social que permitía a los usuarios conectarse con sus amigos. El elemento central del sitio era la 'lista de amigos', que mostraba los contactos del usuario. Este conjunto de datos contiene las conexiones entre todos los usuarios de Friendster. La red Friendster contiene usuarios conectados entre sí a través relaciones de amistad.

Cantidad de usuarios o nodos: ~100 millones Cantidad de contactos o conexiones: ~2.500 millones

El tamaño del conjunto de datos es de 22.8 GB descomprimido -comprimido 10GB-.

Descargar desde: https://archive.org/download/friendster-dataset-201107

Método de comparación entre tecnología de base de datos

1. Espacio de almacenamiento
2. Tiempo en devolver todos los contactos directos de un usuario
3. Tiempo en devolver todos los contactos indirectos de 1 salto de un usuario
4. Tiempo en devolver todos los contactos indirectos de hasta 2 saltos de un usuario
5. Tiempo en devolver todos los contactos indirectos de hasta 3 saltos de un usuario

Configuración del ambiente de pruebas

Hardware

Todas las pruebas fueron realizadas en una computadora personal con las siguientes caracterísitcas:

Sistema Operativo	CPU	Memoria	Espacio en Disco	Tipo de Volumen
Windows 10 x64	Core(TM) i7-5500U @2.40 GHz	8.00GB DDR3	1TB	SATA III

Software

1. MondoDB
2. Neo4j 4.0.3 Community Edition
3. TigerGraph 2.1.4 Developer Edition

Se evaluarán los resultados obtenidos en una base de datos NoSQL, MongoDB. Y luego se compararán los resultados con bases de datos orientadas a grafos. La

base de datos orientada a grafos más popular hoy en día es Neo4j, clasificada como #1 por DB-engine.

Evaluaremos también TigerGraph, que afirma ser la plataforma gráfica más rápida y escalable, después de lanzar su edición gratuita para desarrolladores de aplicaiones para combatir el COVID-19 [5], recientemente publicó sus resultados de referencia en Amazone Neptune. TigerGraph supera a estas bases de datos de grafos por un amplio margen en todas las pruebas de referencia. Además, TigerGraph demuestra un uso de almacenamiento más eficiente, reduciendo el tamaño de datos original en lugar de expandirlo, como es el caso de las otras bases de datos.

Descripción de pruebas y resultados

Carga de datos

MongoDB

Correr Jupyter carga_friendster_mongodb.ipynb

Neo4j

1 - Iniciar el servicio de neo4.

2 - Ingresar al puerto donde levanta la UI:

```
Started neo4j (pid 13595). It is available at http://localhost:5474/
```

4 - En la IU ir a "Database" (en el menu de la izq)

5 - Arriba aparece una consola para ejecutar comandos, correr:

```
create index on :User(id);
```

6 - Levantar el set de datos

Para eso debemos combinar los .txt porque originalmente viene partido. Para ello correr el jupyter notebook Combine files.ipynb

NOTA: Desde la UI tuve problema para poder levantar el archivo desde cualquier carpeta y tuve que copiarlo en

```
/var/lib/neo4j/import
```

Se puede configurar desde el archivo config (/etc/neo4j/neo4j.conf). Hay que comentar la linea 'dbms.directories.import=import' y sacarle el comentario a la linea 'dbms.security.allow_csv_import_from_file_urls=true' para que permita importar desde cualquier lugar.

Modificar las siguientes líneas

```
dbms.memory.heap.initial_size=1G
dbms.memory.heap.max_size=4G
dbms.memory.pagecache.size=1512m
```

7 - Para ejecutar esto sin error se debe agregar :auto USING PERIODIC COMMIT 100000... (Mint 19, Neo4j 4.0.3)

```
:auto USING PERIODIC COMMIT 100000
LOAD CSV FROM "file:///result.csv" as line FIELDTERMINATOR ":"
MERGE (u1:User {id:line[0]});

:auto USING PERIODIC COMMIT 100000
LOAD CSV FROM "file:///result.csv" as line FIELDTERMINATOR ":"
WITH line[1] as id2
UNWIND split(id2,",") as id
WITH distinct id
MERGE (:User {id:id});

:auto USING PERIODIC COMMIT 100000
LOAD CSV FROM "file:///result.csv" as line FIELDTERMINATOR ":"
WITH line[0] as id1, line[1] as id2
MATCH (u1:User {id:id1})
UNWIND split(id2,",") as id
MATCH (u2:User {id:id})
CREATE (u1)-[:FRIEND_OF]-(u2);
```

NOTA: En la computadora personal sólo se pudo cargar el archivo friends-000_____.txt Los experimentos se hicieron tomando estos ~3 millones de nodos porque la RAM no alcanzó para cargar todo.

TigerGraph

Antes de cargar los datos, se debe especificar un esquema gráfico, que para el caso del conjunto de datos Friendster solo requería definir un tipo de nodo con propiedad de id y un tipo de enlace. El trabajo de carga se escribe utilizando el lenguaje GSQL y no requirió ninguna preparación adicional del conjunto de datos original. Los datos se indexan durante la carga automáticamente.

Se parseó el archivo friends-000_____.txt. Se obtiene una relación por línea con el formato ID1, ID2. En el procesamiento se eliminaron los nodos vacíos, “notfound” y “private”.

```
Se cargaron 13070406 líneas en 1m 14s.
```

Configuración de las consultas realizadas

Todas las pruebas de rendimiento de consultas utilizan el mismo archivo con 10 nodos de inicio seleccionados al azar, y el tiempo promedio se mide en 10 ejecuciones de consultas para cada una de las pruebas. El tiempo de espera de la consulta para contactos directos se estableció en 180 segundos, y para las consultas de 2 saltos y 3 pasos se usó el tiempo de espera de 9000 segundos (2.5 horas), es decir, si después de la consulta de tiempo de espera no se completó, entonces se termina y el cálculo continúa a la siguiente consulta.

Se vio que los tiempos tuvieron bastante varianza dentro de cada salto debido a las diferencias entre la cantidad de relaciones de cada uno. Dicha variación se fue incrementando a medida que se aumentaba la cantidad de saltos.

Neo4j

```
 MATCH (u:User)-[:FRIEND_OF]->(u1:User)
WHERE u.id = "user_id"
RETURN u, u1

MATCH (u:User)-[:FRIEND_OF*2]->(u1:User)
WHERE u.id = "user_id"
RETURN u, u1

MATCH (u:User)-[:FRIEND_OF*3]->(u1:User)
WHERE u.id = "user_id"
RETURN u, u1
```

TigerGraph

Y el GSQL (TigerGraph) equivalente de la consulta:

```
CREATE QUERY kstep(VERTEX< User > start_node, INT k) for GRAPH friendster {
            int i = 0;
            Result = {start_node};
            Start = {start_node};

            WHILE (i < k) DO
                Start = SELECT v
                        FROM Start:u - (Friendship:e)->:v;
                Result = Result UNION Start;
                i = i + 1;
            END;

            PRINT Result.size();
}
```

Algunos ejemplos de resultados y tiempos de respuesta

MongoDB

User id: 745822	#Relaciones	Tiempo (ms)

User id: 745822	#Relaciones	Tiempo (ms)
contactos directos	144	642
contactos indirectos de 1 salto	8690	2060
contactos indirectos de 2 saltos	0	N/A (timeout)
User id: 101	**#Relaciones**	**Tiempo (ms)**
contactos directos	141	595
contactos indirectos de 1 salto	8771	1930
contactos indirectos de 2 saltos	0	N/A (timeout)
User id: 202	**#Relaciones**	**Tiempo (ms)**
contactos directos	15	610
contactos indirectos de 1 salto	264	1960
contactos indirectos de 2 saltos	0	N/A (timeout)

Neo4j

User id: 745822	#Relaciones	Tiempo (ms)
contactos directos	144	2
contactos indirectos de 1 salto	8690	44
contactos indirectos de 2 saltos	857949	14030
contactos indirectos de 3 saltos	0	N/A (timeout)
User id: 101	**#Relaciones**	**Tiempo (ms)**
contactos directos	141	4
contactos indirectos de 1 salto	8771	96
contactos indirectos de 2 saltos	527304	7561
contactos indirectos de 3 saltos	0	N/A (timeout)
User id: 202	**#Relaciones**	**Tiempo (ms)**
contactos directos	15	1
contactos indirectos de 1 salto	264	3
contactos indirectos de 2 saltos	8107	79
contactos indirectos de 3 saltos	0	N/A (timeout)

TigerGraph

User id: 745822	#Relaciones	Tiempo (ms)
contactos directos	144	810
contactos indirectos de 1 salto	8690	1500
contactos indirectos de 2 saltos	857949	1700
User id: 101	**#Relaciones**	**Tiempo (ms)**
contactos directos	141	300
contactos indirectos de 1 salto	8771	350
contactos indirectos de 2 saltos	527304	2350
User id: 202	**#Relaciones**	**Tiempo (ms)**
contactos directos	15	580
contactos indirectos de 1 salto	264	960
contactos indirectos de 2 saltos	8107	1550

Para algunos usuarios, es posible realizar más de tres saltos. La limitante es el tamaño del json resultante, el cual tiene que ser menor a 32MB. Si se aplican los filtros geoespaciales propios del dominio (cantidad mínima de tiempo en contacto y proximidad necesaria) el json no debería devolver tantos resultados con lo cual esto no representa un problema.

Resultados finales

Espacio de almacenamiento utilizado

Aquí se comparan los tamaños de almacenamiento de los datos cargados con el tamaño del conjunto de datos original. El tamaño de almacenamiento de Neo4j se midió después de que se creó el índice en los ID de nodo.

Dataset Original	MongoDB	Neo4j	TigerGraph
Friendster	104 MB	104 MB	4.51 GB | 200 MB

Rendimiento de las consultas

Avg ms(10 usuarios al azar)	MongoDB	Neo4j	TigerGraph
contactos directos	610 ms	2 ms	560ms
contactos indirectos de 1 salto	2 s	23 ms	1.2 s
contactos indirectos de 2 saltos	N/A (timeout)	4.6 s	2.2 s
contactos indirectos de 3 saltos	N/A (timeout)	N/A (9/10 timeout)	53.9 s

Conclusiones

Tiempo de carga

MongoDB es el más fácil de cargar por ser un NoSQL.

El tiempo de carga de TigerGraph no es sustancialmente mejor que el tiempo de Neo4j. Sin embargo, TigerGraph tiene una ventaja sobre Neo4j, es decir, no requiere

el procesamiento previo de los datos antes de la carga (por ejemplo, preparación de archivos de nodo). Además, dado que Neo4j no indexa automáticamente los datos durante la carga, esto implica que se requiere tiempo adicional para realizar la indexación antes de que los datos estén listos para usarse.

Espacio de almacenamiento

Se muestra que TigerGraph utiliza una compresión eficiente durante la ingestión de datos, lo que reduce el tamaño del gráfico cargado en la base de datos en comparación con su tamaño original.

Rendimiento de las consultas

MongoDB tiene una performance aceptable si sólo se necesitan los contactos directos o hasta 1 salto. Lo cual lo deja muy por debajo de las necesidades del trabajo aquí tratado. No sería adecuado adoptar este tipo de base de datos para un sistema de seguimiento epidemológico.

TigerGraph supera ampliamente a Neo4j en las consultas de contactos indirectos 3 o más saltos, terminando todas las consultas dentro del tiempo de espera establecido.

Neo4j también puede completar consultas de 1 y 2 saltos dentro del tiempo de espera establecido, de hecho aquí tiene un tiempo de respuesta menor al de TigerGraph. Para consultas de más de 2 saltos, las consultas de 9 de 10 caducaron, es decir, no se pudieron completar dentro del tiempo de espera de 9000 segundos. Dado que solo se completó una consulta, no es razonable proporcionar su tiempo de ejecución como promedio, ya que no refleja el valor promedio en absoluto.

Elección final

Se recomienta utilizar TigerGraph como tecnología de bases de datos del sistema de seguimiento epidemológico porque permite hacer consultas más complejas contempladas en el esquema de uso de la aplicación propuesta en un tiempo de respuesta adecuado, además de ser más efectivo en materia de almacenamiento de datos.

Referencias

- [1] "Sistema de apoyo a la detección y monitoreo inteligente de personas con COVID-19" - https://lsia.github.io/COVID-19/
- [2] Hellewell, J., Abbott, S., Gimma, A., Bosse, N.I., Jarvis, C.I., Russell, T.W., Munday, J.D., Kucharski, A.J., Edmunds, W.J., Sun, F., et al.: Feasibility of controlling covid-19 outbreaks by isolation of cases and contacts. The Lancet Global Health (2020)
- [3] Ferretti, L., Wymant, C., Kendall, M., Zhao, L., Nurtay, A., Bonsall, D.G., Fraser, C.: Quantifying dynamics of sars-cov-2 transmission suggests that epidemic control and avoidance is feasible through instantaneous digital contact tracing. medRxiv (2020)
- [4] J. Yang and J. Leskovec. Defining and Evaluating Network Communities based on Ground-truth. ICDM, 2012
- [5] https://www.tigergraph.com/stopcoronavirus/