Facultad de Ingeniería

Bases de Datos No Relacionales

Obligatorio 2

Keshet Hertz - 209074

Evgeny Sidagis - 222144

Mathias Lantean - 180094

Índice

Parte 1: Suscripciones, Feed, Analytics interno, Mapa de calor

1. **Recomendar una o la cantidad de bases de datos que considere necesarias para cada uno de los subsistemas mencionados.** Se deberá indicar que motor se usará, los motivos para haberlo seleccionado y que ventajas y desventajas presenta su uso.

Suscripciones

Para este primer subsistema, conviene utilizar una base de datos relacional, dado que tiene que si o si ser transaccional y mantener la consistencia de los datos. Además, se asume que la suscripción como tal no va a variar a lo largo del tiempo, manteniendo el mismo modelo de datos.

Esto abre la posibilidad a trabajar sobre un cierto número de bases de datos relacionales SQL, entre ellas MySql, PostreSql o MemSql.

De estos, se elige PostgreSql.

Por qué PostgreSql?

Elegimos este proveedor debido a que, si bien no es tan popular como sus competidores, provee un sistema mucho más íntegro y consistente que estos. PostgreSql es completamente ACID-compliant, proveyendo control multi-version para mantener la consistencia de los datos en todo momento.

Esta base de datos permite además realizar ciertas operaciones complejas, y almacenar tipos de datos que otros sistemas no ofrecen.

Cabe destacar que el sistema es Open-Source, y contiene una extensa documentación para realizar sus operaciones.

Debido a que no se hace mención de la velocidad de las transacciones en ningún momento, se evita hacer mención de una de las principales desventajas de PostgreSql, que es ser relativamente más lenta que, por ejemplo, mySQL.

Ventajas

- **Acid-Compliant.** Seguir completamente los estándares de ACID aseguran la consistencia e integridad de los datos en todo momento.
- **Open-Source:** Ser de código abierto le provee una ventaja competitiva sobre bases de datos con un sistema de licencias, además esto permite una mayor y vasta cantidad de documentación disponible para el manejo de PostgreSql.
- **Extensible:** En el caso de tener que agregar un nuevo tipo de dato a las subscripciones, este sistema lo permite sin mayor dificultad.

Desventajas

- **Popularidad:** Esta base de datos no es muy popular comparada a su competencia, como serlo mySql, lo cual limita la cantidad de desarrolladores que saben manejar la base correctamente
- **Velocidad:** Si bien la brecha de velocidad y performance entre mySql y PostgreSql se ha ido achicando con el paso de los tiempos, mySql sigue siendo más performante para el caso de transacciones con datos simples. En cambio, PostgreSql es más adecuada a proyectos grandes, con sistemas más complejos y con un volumen de datos mayor.

Feed

Para el subsistema de Feed, se decidió usar una base de datos no relacional ya que las consultas tienen que ser rápidas y la pérdida de datos es aceptable. Entre las base de datos no relacionales se consideraron Redis y MongoDB. Se eligió usar Redis.

Por qué Redis?

Elegimos Redis ya que las escrituras y consultas tienen que ser rápidas y justamente Redis es conocido por su velocidad. Además, este tiene un alto nivel de disponibilidad, escalabilidad y es de fácil configuración.

Redis también ofrece diversas estructuras de datos, entre ellos la lista, que en este caso nos sirve para el feed. Esta estructura de datos nos permite agregar nuevos datos de manera ordenada en la lista, lo que hace muy performante la lectura ya que no se requiere ordenar cada vez.

Además, Redis también ofrece el patrón de Publicador/Suscriptor, que en este caso puede ser muy útil para suscribirse a eventos que agregan elementos al Feed, haciéndolo más escalable y fácil.

Ventajas

- **Velocidad:** Gracias a su almacenamiento en memoria, el rendimiento es muy rápido.
- **Fácil configuración:** Redis es simple y fácil de usar.
- **Alto nivel de disponibilidad y escalabilidad**
- **Diversas estructuras de datos:** Aparte de Strings, también es compatible con diferentes estructuras de datos como Hash, List, Set y otras.
- **Soporte patrón Publicador/Suscriptor:** Se puede desarrollar una aplicación de alto rendimiento utilizando el mecanismo Publisher-subscriber de Redis.

Desventajas

- **Alta capacidad de memoria:** Requiere de gran espacio en memoria RAM.
- **No recomendado para consultas complejas:** Redis no tiene la capacidad de realizar consultas complejas como las bases de datos relacional.

Analytics interno

Para el subsistema de Analytics interno, decidimos usar una base de datos orientada a columnas. Éstas son ideales para ambientes con foco en la lectura.

Entre las base de datos orientadas a columnas consideradas se encuentra Vertica y MonetDB. Finalmente nos decidimos por Vertica dado que no encontramos diferencias sustanciales entre una y otra (más que el mapeo de la estructura de datos en memoria a disco).

Por qué Vertica?

Ya que no es necesario que la información esté actualizada en tiempo real y se pueden dar cada 24hs, es decir que se actualizarán una vez al día, lo que necesita este subsistema es prestarle especial atención a la lectura de los datos.

Por otro lado, podemos asumir que este sistema gestionará una gran cantidad de datos y si además analizamos el tipo de consulta que los analistas de negocio realizarán notaremos que son consultas sobre todo el conjunto de datos y no en registros particulares.

Vertica justamente se ajustaría muy bien en este escenario dado que permite acelerar el tiempo de lectura al analizar grandes cantidades de datos.

A su vez Vertica permitiría optimizar el uso de los servidores ya que nos permitiría la compresión de los datos y permite consultas sobre los mismos (sin necesidad de descompresión), hace un uso eficiente de la memoria RAM y dada la forma en que los datos son almacenados en disco se reducen los tiempos de accesos al mismo.

Ventajas

- **Velocidad:** Rápido en grandes cantidades de datos.
- **Escalabilidad:** Cuenta con una arquitectura escalable de forma horizontal.
- **Eficiencia:** Cada SELECT solamente lee lo que necesita.
- **Compresión:** Se puede operar sobre datos comprimidos.
- **Alta disponibilidad**
- **Consultas distribuidas**: cualquier nodo puede iniciar consultas y utilizar el resto de los nodos para su ejecución.
- **Mejor uso de los recursos de hardware**

Desventajas

- Los INSERT y UPDATE son complejos.
- No sirve para implementar sistemas OLTP.

Mapa de calor

En lo que refiere al subsistema de mapas de calor sin duda creemos que es conveniente utilizar Elasticsearch con Kibana.

Por qué Elasticsearch con Kibana?

Elasticsearch es una base de datos diseñada para la búsqueda distribuidas y análisis de datos mientras que Kibana nos ofrece una consola que nos permite explorar, visualizar y compartir información sobre los datos de forma interactiva y administrar y monitorear el stack. Es de destacar que todas las consultas se realizan utilizando la sintaxis JSON.

Elasticsearch proporciona búsquedas y análisis casi en tiempo real para todo tipo de datos. Ya sea que tenga texto estructurado o no estructurado, datos numéricos o datos geoespaciales, Elasticsearch puede almacenarlos e indexarlos de manera eficiente de una manera que se admitan búsquedas rápidas.

Para la funcionalidad de mapa de calor Elasticsearch nos otorgaría una gran facilidad de inserción de datos de tipo geo_point y geo_shape.

Los campos de tipo geo_shape facilitan la indexación y la búsqueda con formas geográficas arbitrarias, como rectángulos y polígonos mientras que los campos de tipo geo_point aceptan pares de latitud-longitud, que se pueden utilizar:

- Para encontrar puntos geográficos dentro de un cuadro delimitador, dentro de una cierta distancia de un punto central, o dentro de un polígono o dentro de una consulta geo_shape.
- Para agregar documentos geográficamente o por distancia desde un punto central.
- Para integrar la distancia en la puntuación de relevancia de un documento.
- para clasificar documentos por distancia.

En particular Elasticsearch nos permite generar de forma directa mapas de calor, simplemente es necesario agregar una layer de mapa de calor y contar con índice que debe contener al menos un campo mapeado como geo_point o geo_shape. Las capas de mapas de calor agrupan datos de puntos para mostrar ubicaciones con densidades más altas.

Ventajas

- Elasticsearch es compatible para ejecutarse en todas las plataformas porque está desarrollado en Java.
- Velocidad de búsqueda. La Performance en lectura que permite una búsqueda casi en tiempo real.
- Orientado a documentos distribuidos, también permite la replicación de datos y clustering aumentando de esta forma la disponibilidad
- Código abierto, no necesita licencia para utilizarlo
- Se integra con gran variedad de tecnologías (hay librerías para una gran variedad de lenguajes distintos)

Desventajas

- ElasticSearch tiene como principal desventaja que se puede presentar inconsistencias entre nodos (split brain).
- No tiene soporte para varios lenguajes a la hora de manejar datos de solicitudes y respuestas. Solo permite JSON.

Parte 2: TileDB

2. **¿Qué tipo de base de datos es TileDB? ¿Podría aplicarse en algunos de los escenarios planteados?** Deberá describir el modelo de datos que TileDB soporta, ejemplos concretos de ese modelo y analizar si es aplicable o no para los escenarios que se plantean en este obligatorio.

TileDB es un motor de base de datos capaz de almacenar y acceder a densos y/o escasos arrays multidimensionales, que puede ayudar a modelar cualquier tipo de datos de forma eficiente. Funciona como una librería C++ embedida en Windows, MacOS y Linux.

El sistema funciona en base a dos tipos de Arrays Multidimensionales, los Densos y los Escasos. Los Arrays Densos no permiten valores nulos, mientras que los Escasos permiten valores nulos.

En lo que respecta a ejemplos concretos del modelo, existen varios ejemplos del funcionamiento de TileDB junto a otra base de datos, como MariaDB o PrestoDB. A modo general de la estructura del modelo de TileDB, se puede denotar en el modelo descrito a continuación.

Como se mencionó anteriormente, TileDB utiliza arrays multidimensionales. Cada array está conformado por dimensiones y atributos. Cada dimensión tiene un dominio, y son estos dominios de dimensión los cuales orientan el espacio lógico del array. Cada coordenada (combinación de valores) dentro de un dominio de dimensión, la cual identifica de forma única a un elemento del array, se denomina “celda”. Estas celdas pueden ser nulas o contener una tupla de atributos de valor. Cada atributo es un tipo primitivo, o un vector de valor primitivo de tamaño fijo o variable.

Por otra parte, como se hacía referencia previamente, TileDB tiene arrays densos y escasos. Para el caso de los densos, ninguna tupla puede ser nula, es decir, todas tienen un atributo de valor. En el caso de los escasos, en cambio, las celdas pueden ser nulas. Todas las operaciones de TileDB pueden ser realizadas sobre cualquiera de los dos tipos de arrays, pero esta elección entre escaso y denso puede afectar severamente la performance de la aplicación. Típicamente, los arrays son almacenados en formato escaso cuando se supera una fracción dada de celdas nulas, lo cual depende mucho de la aplicación.

TileDB presenta ordenamiento global de celdas, dado que si bien los arrays son multidimensionales, los storages, sea disco o memoria, son unidimensionales. La forma de ordenarlo tiene un gran impacto sobre la performance, dado que afecta qué celdas se encuentran cerca otras. Una feature deseable es que celdas similares se encuentren juntas, para minimizar la búsqueda, lecturas y lecturas de caché.

Esto implica que la mejor forma de ordenar esta dictada por la forma que tiene la aplicación, si una aplicación lee fila a fila, conviene ordenar fila a fila.

Este sistema provee múltiples formas de hacer el Global Cell Order, lo cual le da un nivel de control altísimo al usuario para especificarlo de la forma más performante posible.

Todos los dominios de dimensión tienen un tipo fijo. En arrays densos solo se soportan *ints*, mientras que en escasos también acepta *floats.*

Como un ejemplo concreto de TileDB:

Un sistema de series de análisis del tiempo climático

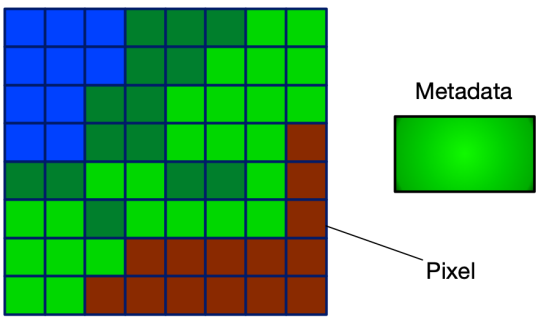
- Como esquema se usa:
    - Array Denso, sin duplicados
    - Dimensiones
        - Bandas
        - X
        - Y
        - Cada una de estas dimensiones tiene su propio dominio. Además, son tipadas como UINT64
    - Atributos
        - TOR values, Float32

Como esquema llevado al obligatorio, para específicamente el mapa de calor: Esto sería similar al tiempo climático, dado que también lleva un sistema de coordenadas, y se podría aplicar bajo Raster, que son imágenes en 2d que pueden mantener cualquier número de valores.

LiDAR

(from NYU Spatial Data Repository)

Raster



Como esquema se usa:

- Array Denso, sin duplicados.
- Dimensiones
  - X, UINT64
  - Y UINT64
- Atributos
  - Calor, UINT64
  - Actividad, float32

La idea principal del funcionamiento es que cada “celda” o coordenada, por ejemplo celda(0,1) este “pintada” con un número que indique la intensidad o su calor, y que además está asociado a esto un número que denomine la actividad dada en dicha celda.

¿Se podría utilizar en algunas partes de este obligatorio?

Ser usable en cierta situación no significa que sea la opción más eficaz para abordar el problema, y lamentablemente para TileDB, en la scope de este proyecto, es un poco overkill comparado a las alternativas ya planteadas en la parte anterior. En el caso de suscripciones, es preferible el uso de una base de datos relacional, dado que no es necesario tener tal poder de almacenamiento, y TileDB nos ofrece "eventual consistency vía fragments", no sigue ACID, lo cual es integro para esa sección.

Siguiendo, para el Feed se eligió utilizar Redis, debido a su velocidad, escalabilidad y facilidad de uso. No consideramos eficiente tener que aprender un nuevo sistema para tener un mayor poder de almacenamiento, pero sacrificando propiedades de Redis, como las ya mencionadas en la sección de Feed.

Para el caso de Analytics, creemos que las opciones utilizadas de bases orientadas a columna son superiores, nuevamente debido a que la escala de TileDB es demasiada.

Por último, para el mapa de calor TileDB podría ser la opción superior a ElasticSearch, más que nada dado que TileDB fue diseñada originalmente para procesar información geoespacial, cosa que el mapa de calor requiere.

En comparación con el resto de subsistemas no cubiertos por la Parte 1 para esta entrega, creemos que TileDB nuevamente es usable, pero no factible, dado que hay alternativas con un alcance más pequeño y más simple de utilizar.

Componentes no vistos en la Parte 1:

Registro de Perfil de Usuarios

Para esta sección se eligió utilizar MongoDB, debido particularmente a la facilidad que este brinda para agregar nuevos atributos.

Por qué MongoDB?

Elegimos MongoDB porque podemos editar el esquema sin necesidad de realizar migraciones de la DB por lo que no habrá tiempos muertos. MongoDB provee una base de datos flexible, dándonos libertad de modificar y almacenar datos de distintos tipos. Cabe destacar que MongoDB también es una base de datos con una disponibilidad extremadamente alta, y es de alta velocidad. MongoDB está orientada a procesar y almacenar altos volúmenes de datos, lo cual es de particular importancia para bases de datos.

Ventajas

- Alta disponibilidad.
- Escalabilidad: al ser una base de datos distribuida puede escalar de forma horizontal y vertical.
- Aporte gran velocidad
- Soporta los principales lenguajes de programación.

Desventajas

- No es adecuada para aplicaciones con transacciones complejas.
- No tiene JOINs.

Comunidades

Para el subsistema de comunidades decidimos usa una base de datos orientada a grafos Neo4j

Por qué Neo4j?

Este servicio se encuentra enfocado principalmente a las relaciones entre entidades. Para este punto se busca modelar con detalle cómo interactúan las comunidades, por lo que parecía bastante lógico utilizar una base de datos de gráficos para almacenar los datos de este servicio ya que nuestros datos representan, en gran medida, un grafo. Es ágil. Queríamos un sistema de rendimiento realmente rápido.

Ventajas

- Es ágil, tiene un alto rendimiento
- Gran capacidad de almacenamiento de nodos y relaciones (la compresión de punteros dinámicos permitió a Neo4j expandir el

espacio de direcciones disponibles, pudiendo representar grafos de cualquier tamaño, ya no existe el límite de 34 mil millones de nodos)
- Neo4j tiene su propio lenguaje de consulta CYPHER que es muy intuitivo y fácil de usar.
- Neo4j admite API en casi todos los lenguajes como Java, Python, PHP, NodeJS, etc.
- Garantiza ACID

Desventajas

- Dificultades para escalar horizontalmente
- No soporta nativamente propiedades complejas para nodos y relaciones

Actividades

Para el subsistema de actividades, elegimos usar una base de datos NoSQL, tomando en cuenta que es más flexible a la hora de recoger datos. Entre las bases NoSQL consideramos Cassandra y MongoDB. Decidimos utilizar Cassandra.

Por qué Cassandra?

Elegimos Cassandra ya que se busca que el subsistema tenga un alto nivel de tolerancia a fallas y eligiendo disponibilidad sobre consistencia. Cassandra cumple ambos requisitos. Esta tecnología facilita el uso de redundancia, podemos tener múltiples instancias maestras de la base de datos, haciéndolo más tolerante a fallos; si una instancia falla se siguen utilizando las otras instancias. La alta disponibilidad también es una característica principal de Cassandra.

Ventajas

- Alta disponibilidad.
- Cantidad de recursos disponibles.
- Tolerante a fallos.

Desventajas

- No existe el concepto de conexiones JOIN en Cassandra.
- No permite ordenar resultados en tiempo de consulta.
- No garantiza ACID.

Explorador

Para la exploración decidimos utilizar dos bases distintas, por un lado Elasticsearch y por el otro una base orientada a grafos Neo4j.

El uso de Elasticsearch es porque nos va a permitir buscar rápidamente los

distintos retos así como también realizar consultas sobre la tabla de ranking. También nos permite realizar búsquedas geográficas para identificar los segmentos en los que los usuarios utilizaron rutas similares para realizar actividades parecidas.

Por otro lado, para modelar los clubes y las relaciones que se describen perfectamente se podría implementar perfectamente con estructuras de grafos, por lo que Neo4j.

Las ventajas y desventajas de ambas bases de datos fueron descritas en puntos anteriores.

Estadísticas de Actividades

Para Estadísticas de Actividades se decidió usar una base de datos orientada a columnas. Las bases de datos basadas en columnas son creadas para la velocidad, leyendo de inmediato los datos que se busca ya que se omiten datos no necesarios.

Por qué Vertica?

Elegimos Vertica ya que se pide que la información debe ser accedida en el menor tiempo posible. Vertica es conocida por ser rápida en grandes cantidades de datos gracias a su arquitectura, realizando consultas de manera distribuida y en paralelo. Además es eficiente y con alta disponibilidad.

Ventajas

- **Velocidad:** Rápido en grandes cantidades de datos.
- **Escalabilidad:** Cuenta con una arquitectura escalable de forma horizontal.
- **Eficiencia:** Cada SELECT solamente lee lo que necesita.
- **Compresión:** Se puede operar sobre datos comprimidos.
- **Alta disponibilidad**
- **Consultas distribuidas**: cualquier nodo puede iniciar consultas y utilizar el resto de los nodos para su ejecución.
- **Mejor uso de los recursos de hardware**

Desventajas

- Los INSERT y UPDATE son complejos.
- No sirve para implementar sistemas OLTP.

Parte 3: Entidades usadas para el registro de actividades

activity		
activityid	< text >	**K C↑**
date	< date >	**K C↓**
title	< text >	
userid	< text >	**K**
url	< text >	
comment	< text >	
type	< text >	
duration	< double >	
distance	< double >	
text	< text >	
photo	< text >	
description	< text >	
difficulty	< text >	
location	< varchar >	
averageSpeed	< double >	
cadence	< double >	
calories	< double >	

```
CREATE TABLE IF NOT EXISTS activity (
        activityid TEXT,
        date DATE,
        title TEXT,
        userid TEXT,
        url TEXT,
        comment Text,
        type Text,
        duration double,
        distance double,
        text text,
        photo text,
        description text,
        difficulty text,
        location varchar,
        averageSpeed double,
        cadence double,
        calories double,
        Primary Key(userid,date, activityid)
)WITH CLUSTERING ORDER BY (date DESC, activityId ASC);
```

Parte 4 y 5

Se crearon dos microservicios básicos los cuales implementan el sistema de Actividades y por otra parte el sistema de Registro y perfil de usuarios. Ambos microservicios fueron desarrollados en node.js, en particular se utilizó express para el sistema de Registro y perfil de usuario mientras que koa fue utilizado para desarrollar el sistema de Actividades.

El código se encuentra disponible en github, junto con el manual de instalación correspondiente https://github.com/MathiasLantean/BDNR_209074_222144_180094

Cabe destacar que la implementación de los subsistemas se enfocaron especialmente en investigar y comprender cómo funcionan las bases de datos no tradicionales, por lo que existen muchos aspectos de seguridad y funcionalidades de las APIs REST que podrían ser mejorados.

Referencias

Suscripciones

https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

https://blog.mdcloud.es/postgresql-vs-mysql-cual-usar-para-mi-proyecto/

https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres

https://www.tecnologias-informacion.com/transaccionales.html

Feed

https://aws.amazon.com/es/redis/

Analytics interno

PPT de aulas: BBDD orientadas a columnas

Mapa de Calor

https://www.elastic.co/guide/en/kibana/current/heatmap-layer.html

https://medium.com/engineering-tyroo/using-elasticsearch-for-reporting-analytics-3bb1d7c84c19

https://www.javatpoint.com/advantages-and-disadvantages-of-elasticsearch

TileDB

https://thenewstack.io/tiledb-managing-big-data-storage-in-multiple-dimensions/

https://people.csail.mit.edu/stavrosp/papers/vldb2017/VLDB17_TileDB.pdf

https://www.youtube.com/watch?v=PSopznR3DbY&ab_channel=RadiantEarthFoundation

https://docs.tiledb.com/geospatial/

https://docs.tiledb.com/main/solutions/tiledb-embedded/internal-mechanics/consistency

https://docs.tiledb.com/main/

Registro y perfil de usuarios:

https://severalnines.com/database-blog/battle-nosql-databases-comparing-mongodb-cassandra

https://data-flair.training/blogs/advantages-of-mongodb/

Actividades

https://cassandra.apache.org/

Comunidades

https://neo4j.com/blog/neo4j-3-0-massive-scale-developer-productivity/
http://alronz.github.io/Factors-Influencing-NoSQL-Adoption/site/Neo4j/Results/Strengths%20and%20Weaknesses/
https://www.trustradius.com/products/neo4j/reviews?qs=pros-and-cons

Estadísticas de Actividades

https://gravitar.biz/bi/base-datos-columnar/

https://www.vertica.com/