Bases de Datos en Tiempo Real: Estado del Arte

Agustín Gurvich

FCEIA, UNR

31/05/2022

Resumen

En las clases anteriores vimos los conceptos teóricos necesarios para el desarrollo del tema:

- Necesidades prácticas
- Definiciones de consistencia
- Deadlines
- Concurrencia
- Scheduling
- Seguridad

¡Pero en ningún momento hablamos de aplicaciones prácticas!

Resumen

La realidad es que estos sistemas recién están comenzando.

La mayoría de los sistemas profesionales son desarrollos propios, pues cada proyecto tiene requerimientos muy diversos (recordemos lo visto en la clase 1).

Lo que sí podemos encontrar, es software comercial que simula el tiempo real con estrategias más generales.

Software comercial

Casi todo el software comercial provee bases de datos de estilo NoSQL. Con Santi vimos algunas como MongoDB y Cassandra.

Vamos a ver otros tres ejemplos: Firebase, RethinkDB y OrientDB

Google Firebase

Google Firebase es una plataforma de Google hecha para el desarrollo de aplicaciones móviles.

Provee servicios integrados como Analytics, Auth o Cloud Services, pero vamos a concentrarnos en el que nos interesa: Realtime Database

Google Firebase

Firebase provee una base de datos en tiempo real estructurada como un árbol JSON, y tiene una API para realizar comunicaciones remotas. Se integra con varias tecnologías comercialmente disponibles.
Una característica interesante es que la API utiliza el sistema de Server Sent Events, pudiendo así disminuir el costo de la aplicación local.

Google Firebase

Para realizar actualizaciones en tiempo real, la base de datos local se convierte en una instancia de la base de datos general.

Cuando se realiza una modifiación en la BD local, la API se encarga de almacenar el cambio en la nube y notificar al resto de las instancias del cambio para que puedan actualizarse.

También provee la oportunidad de realizar tareas offline, almacenar la información de manera local y al recuperar la conectividad notificar al servidor principal.

RethinkDB

RethinkDB es una base de datos en tiempo real que utiliza el formato JSON, desarrollada teniendo en mente las aplicaciones web.

Al igual que Firebase, su arquitectura está diseñada par evitar que las aplicaciones realicen polling, lo que ayuda en su rendimiento. En su lugar, se le puede indicar a RethinkDB cómo enviar los cambios en la BD a los usuarios para que no tengan que preocuparse por tener los datos actualizados.

Los desarrolladores diseñaron RethinkDB desde cero utilizando C++, y es un software de código abierto.

RethinkDB

Algo interesante es el lenguaje de querys diseñado para RethinkDB.

No utiliza SQL, sino su propio lenguaje, ReQL. Provee operaciones como joins y funciones de agregado. Una característica destacable es la posibilidad de embeber sentencias javascript y utilizar el esquema map-reduce.

Si bien nos aseguran que la actualización de documentos individuales es atómica, no pueden decir lo mismo de múltiples documentos o de querys no deterministas.

OrientDB

OrientDB es una base de datos de código abierto que provee almacenamientos en forma de grafos (la más usada), documentos, clave-valor y objetos.

Si bien tiene soporte SQL, al almacenar las tablas como grafos también provee la oportunidad de recorrer las a través de las aristas en lugar de usar joins. Para esto, OrientDB guarda las referencias a las otras tablas como punteros, mejorando la velocidad de la recuperación de datos.

A Study on Real-Time Database Technology and Its Applications

En el año 2020 se publicó una tesis de la Eastern Illinois University llamado "A Study on Real-Time Database Technology and Its Applications".

El estudio se basó en decidir qué base de datos en tiempo real disponible comercialmente es la mejor opción a la hora de desarrollar una aplicación de viajes compartidos (como carpoolear).
La aplicación en cuestión fue desarrollada dentro de la universidad, con el objetivo de facilitar viajes entre los estudiantes.

A Study on Real-Time Database Technology and Its Applications

El estudio comparó 4 bases de datos comercialemnte disponibles:

- Google Firebase
- MongoDB
- Cassandra
- OrientDB

Las conclusiones fueron bastante esperadas.

A Study on Real-Time Database Technology and Its Applications

La conclusión fue que MongoDb y Firebase, aunque sean de pago, otorgan demasiadas ventajas como para ser ignoradas por su precio.

La compatibilidad con Google Maps, la sencillez de la tecnología y la facilidad de realizar querys son algunas de las cosas más importantes. También se destaca el desempeño de las aplicaciones cuando entran grandes volúmenes de datos.

A Study on Real-Time Database Technology and Its Applications

Las conclusiones destacadas fueron las siguientes:

- Firebase está mantenida por Google, así que su almacenamiento en la nube es extremadamente confiable
- MongoDB es excelente si se decide guardar la información de manera local
- Los 4 productos ofrecen la escalabilidad suficiente
- Si se quiere masificar la aplicación, el almacenamiento en la nube es necesario

Como mejora del estudio plantean que, si se agregara un mecanismo para verificar la velocidad de las querys al sistema, las conclusiones del estudio podrían haber sido mucho maś concretas.

Refining Timing Constraints Of Applications In DeeDS

Jonas Mellin, Jorgen Hansson y Sten F. Andler, de la universidad de Skövde en Suecia desarrollaron en 1996 el Distributed Active Real-Time Database System, también llamado DeeDs.

En 1997 publicaron "Refining Timing Constraints Of Applications In DeeDS", donde revisitaron el sistema DeeDS para adecuarlo y refinarlo luego de haber obtenido la experiencia de prototiparlo.

En el sistema DeeDS, las transacciones tienen un scheduling dinámico y pueden tener deadlines duras, firmes o suaves.

Refining Timing Constraints Of Applications In DeeDS

Luego de hacer un recorrido por la arquitectura y el diseño de DeeDS, el paper continúa explicando las experiencias de la implementación del sistema. Los puntos destacados son los siguientes:

- Sincronización con los reinicios y abortos de transacciones: debieron evitar el polling y proponen utilizar un mecanismo de manejo de excepciones
- Transacciones con largo tiempo de ejecución: proponen basar estas transacciones en snapshots de la base de datos, para evitar acaparar recursos (cuando sea posible)
- Identificadores de objetos: como DeeDS es una abse de datos orientada a objetos, los lenguajes de programación asociados a ella no pueden mantener la integridad referencial. Proponen mejorar la forma de guardar y manejar las referencias.
- Espacio de direcciones: fue complicado manejar las ubicaciones de las variables por la cantidad de procesos corriendo en el sistema

NoSQL real-time database performance comparison

Este paper, publicado en 2017 por Diogo Augusto Pereira, Wagner Ourique de Morais y Edison Pignaton de Freitas en el international Journal of Parallel, Emergent and Distributed Systems presenta una comparación de performance entre 3 grandes bases de datos NoSQL para evaluar su viabilidad como base de datos en tiempo real: Couchbase, MongoDB y RethinkDB.

NoSQL real-time database performance comparison

La prueba consistió en una serie de operaciones sobre diferentes versiones de estos softwares: Couchbase Server 4.6.0, MongoDB Enterprise 3.4 y RethinkDB 2.3.5, por ser las versiones representativas del momento.

NoSQL real-time database performance comparison

La prueba consistió en una serie de operaciones sobre diferentes versiones de estos softwares: Couchbase Server 4.6.0, MongoDB Enterprise 3.4 y RethinkDB 2.3.5, por ser las versiones representativas del momento.

Las especifiaciones técnicas de la máquina utilizada son:

- CPU: Intel® Xeon® CPU 2.20 GHz 64bits - 8 núcleos
- RAM: 30 GB
- HD: 10 GB SSD
- SO: Ubuntu 16.04.1 LTS

NoSQL real-time database performance comparison

Para equiparar el campo de juego, se desarrolló una API REST programada usando el framework Express y corriendo sobre Node.js que permitía realizar las siguientes operaciones:

- POST
- PATCH
- GET by Id
- GET (hasta 100 documentos)
- DELETE

NoSQL real-time database performance comparison

Los datos utilizados, en formato JSON, tenían la siguiente forma:

```
‘id’: ‘999999’,
‘number’: ‘9999999’,
‘date’: ‘10/10/2016 10:10:10’,
‘customer’: ‘ABCDE ABCDE ABCDE ABCDE’,
‘amount’: ‘999999.99’
```

NoSQL real-time database performance comparison

Se calculo el promedio de tiempo de respuesta, la mediana, el 90 percentil (la situacion en el 90% de los casos) y la cantidad de requests hechas al servidor.

Se realizaron 2 iteraciones del proceso: una con un solo hilo, y otra con múltiples hilos para representar situaciones de alta demanda.

NoSQL real-time database performance comparison

Los resultados hallados fueron los siguientes (un solo hilo):

- En las operaciones POST y PATCH, MongoDB y Couchbase tuvieron los menores tiempos de respuesta (4ms y 2ms)
- En las operaciones GET, Couchbase tuvo el menor tiempo de GET by Id (1 ms) pero MongoDB el menor GET (3ms contra 22ms de Couchbase)
- En la operación DELETE, MongoDB y Couchbase tuvieron los menores tiempos de respuesta (2ms)

NoSQL real-time database performance comparison

Los resultados hallados fueron los siguientes (múltiples hilos):

- En las operación POST MongoDB tuvo menores tiempos, pero en PATCH lo superó Couchbase
- En las operaciones GET, Couchbase y RethinkDB tuvieron los menores tiempos de GET by Id, pero MongoDB el menor GET
- En la operación DELETE, Couchbase tuvo el menor tiempo de respuesta, pero Rethink tuvo algunos casos excepcionalmente veloces

NoSQL real-time database performance comparison

Los resultados hallados fueron los siguientes (múltiples hilos):

- En las operación POST MongoDB tuvo menores tiempos, pero en PATCH lo superó Couchbase
- En las operaciones GET, Couchbase y RethinkDB tuvieron los menores tiempos de GET by Id, pero MongoDB el menor GET
- En la operación DELETE, Couchbase tuvo el menor tiempo de respuesta, pero Rethink tuvo algunos casos excepcionalmente veloces

Algo que puede explicar los malos resultados de Couchbase a la hora de hacer un GET es el lenguaje específico que utiliza, N1QL. Ayuda al usuario que sea más cercano a SQL, pero la performance depende de cómo se escribe la query y quizás podría mejorarse con técinas de conversión de querys.

Improving the quality of service of real-timedatabase systems through a semantics-based scheduling strategy

En un desarrollo planteado por Fehima Achour, Emna Bouaziz y Wassim Jaziri en 2021, estos investigadores decidieron trabajar sobre mejoras en el quality of service (QoS) para gestores de bases de datos en tiempo real.

En este paper propusieron el protocolo AEDF-TAL-DS, que se nutre de conceptos tales como feedback de usuario, ontologías y relaciones semánticas entre los elementos de las querys.

Improving the quality of service of real-timedatabase systems through a semantics-based scheduling strategy

Por ejemplo, podemos tener estas dos querys en simultáneo:

1. Lista de usuarios mayores a 20 años
2. Lista de personas mayores a 20 años

Bajo el contexto correcto, ¡son la misma consulta!

Improving the quality of service of real-timedatabase systems through a semantics-based scheduling strategy

Por ejemplo, podemos tener estas dos querys en simultáneo:

1. Lista de usuarios mayores a 20 años
2. Lista de personas mayores a 20 años

Bajo el contexto correcto, ¡son la misma consulta!

El protocolo es llamado "advanced earliest deadline first based on transactions aggregation links and data semantic link" porque trata de juntar las querys que son semánticamente similares entre si y darles mayor prioridad.

Improving the quality of service of real-timedatabase systems through a semantics-based scheduling strategy

Al comparar este enfoque con EDF, se pudo ver un 15% de mejora en casos de alta demanda.

Improving the quality of service of real-timedatabase systems through a semantics-based scheduling strategy

Al comparar este enfoque con EDF, se pudo ver un 15% de mejora en casos de alta demanda.

Algo interesante a destacar es que, al agregar datos, este sistema no es tan afectado por la sobrecarga. En efecto, agregar información solamente crea nuevos enlaces entre las consultas, por lo que en el peor de los casos (querys sin relación) se degradaría a EDF.

Improving the quality of service of real-timedatabase systems through a semantics-based scheduling strategy

Al comparar este enfoque con EDF, se pudo ver un 15% de mejora en casos de alta demanda.

Algo interesante a destacar es que, al agregar datos, este sistema no es tan afectado por la sobrecarga. En efecto, agregar información solamente crea nuevos enlaces entre las consultas, por lo que en el peor de los casos (querys sin relación) se degradaría a EDF.

Un trabajo a futuro planteado por esta investigación es aplicar este protocolo para bases de datos espacio-temporales.

A Priority Heuristic Policy in Mobile Distributed Real-Time Database System

En 2018 Prakash Kumar Singh y Udai Shanker publicaron un estudio donde propusieron realizar estudios más exaustivos sobre las bases de datos en tiempo real para móbiles y distribuidas (MDRTDBS).

Ellos plantearon que la mayoría delos estudios se realizan haciendo uso de la política EDF, pero que en los medios móviles no suele ser la mejor opción. Para solucionar este problema, plantearon una nueva política heurística que asigna a una query su prioridad en base a cuántos locks de escritura necesita y según el tamaño de los datos que utilizará:

$$Init\_P = 1/Nw + 1/S \quad where\ Nw \geq 1$$

A Priority Heuristic Policy in Mobile Distributed Real-Time Database System

La comparación se realizó contra EDF y otra política (HP_NL) propuesta por Shanker en 2005.

Los resultados mostraron que un aumento en el tiempo de preprocesamiento (*think time*) implica una menor falla en la deadline. Si bien las mejoras son muy ajustadas, son más que bienvenidas en un ambiente donde se está constantemente corriendo a contrarreloj.

Una mejora muy interesante que plantean es tener en cuenta los problemas que generan los medios de comunicación no guiados a la hora de asignar las prioridades.

RT-MongoDB: A NoSQL Database with Differentiated Performance

En 2021 Remo Andreoli, Tommaso Cucinotta y Dino Pedreschi publicaron en el 11th International Conference on Cloud Computing and Services Science un proyecto que habían desarrollado sobre MongoDB: RT-MongoDB.

Su idea fue tomar el servicio estándar de MongoDB y expandirlo a un esquema de tiempo real aprovechándose del parámetro *nice* del sistema Unix, y el usuario tiene control total sobre este valor.
Es posible modificar manualmente el parámetro para darle a las querys importantes mayor prioridad.

RT-MongoDB: A NoSQL Database with Differentiated Performance

Como resultado, lograron una buena primer aproximación para adaptar MongoDB a un sistema de tiempo real para aplicaciones en línea (streaming de audio, video, videojuegos).

Para continuar la investigación, propusieron tres líneas a futuro:

- Trabajar con scheduling a nivel del disco además de CPU (para tener un control más fino sobre el sistema)
- Apoyarse en otras investigaciones sobre MongoDB para obtener mejores resultados
- Realizar un nuevo estudio para agregar un sistema de prioridades (el proyecto actual no contempla casos de usuarios codiciosos)

Implementación en Java de un Modelo de Consistencia Temporal para Datos de Tiempo Real

En el CACIC 2011 se presentó el trabajo "Tipo de Dato Abstracto para Sistemas de Bases de Datos de Tiempo Real", donde se desarrolló un TAD específico para representar la consistencia temporal en una base de datos en tiempo real junto con la noción de datos derivados.

En 2012, los autores (Carlos E. Buckle, Damián P. Barry, José M. Urriza) decidieron implementar dicho modelo utilizando el lenguaje JAVA.

Implementación en Java de un Modelo de Consistencia Temporal para Datos de Tiempo Real

Explican cómo representaron en JAVA los datos disponibles en una base dedatos de este estilo:

- RTDBase: Valores que se actualizan en base a sensores externos. Pueden considerarse como continuos o discretos según la periodicidad de su actualización
- RTDDerived: Los datos tales que es posible calcularlos uitlizando la información (consistente) disponible en la BD

Implementación en Java de un Modelo de Consistencia Temporal para Datos de Tiempo Real

Explican cómo representaron en JAVA los datos disponibles en una base dedatos de este estilo:

- RTDBase: Valores que se actualizan en base a sensores externos. Pueden considerarse como continuos o discretos según la periodicidad de su actualización
- RTDDerived: Los datos tales que es posible calcularlos uitlizando la información (consistente) disponible en la BD

El testeo se desarrolló como un sistema para realizar consultas y chequear la validez de los objetos, resultando estas pruebas exitosas.