OLTP con MongoDB

Marcelo Sureda
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
marcelo.sureda@fing.edu.uy
Ernesto Gerez
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
egerez@fing.edu.uy

Resumen

Las bases de datos (BD) NoSQL fueron creadas como respuesta a limitaciones que se percibían en el mundo de las BD relacionales. En los últimos años las BD no relacionales han ido ganando terreno y adeptos, incluso llegando a ofrecer funcionalidades que se consideraban anteriormente características y fortalezas exclusivas de las BD relacionales. Tal es el caso de la BD documental MongoDB que liberó hace unos pocos años la funcionalidad de integridad transaccional para transacciones multidocumento. Esto motiva el presente trabajo, cuyo objetivo es evaluar la capacidad de MongoDB de cumplir con los requerimientos esperados por la industria con respecto a las soluciones de sistemas transaccionales, principalmente: rendimiento e integridad. A partir de un conjunto de datos obtenido de un sitio público y con un desarrollo *ad-hoc* de un motor transaccional se evalúan algunas características de MongoDB. Se despliega esta solución en plataformas disponibles de forma gratuita en la nube y tras las pruebas realizadas, se comprueba un nivel de cumplimiento razonable con las expectativas iniciales.

I. Introducción

El modelo relacional permite manejar datos de forma estructurada y con baja redundancia. Esto permite consultar, manipular y almacenar los datos de forma eficiente. Las bases de datos relacionales se convirtieron en el estándar *de facto* para el manejo de datos en sistemas informáticos. Sin embargo, estas estructuras son aptas para manejar cierto volumen de datos que ciertas aplicaciones superan ampliamente hoy en día. Además, la heterogeneidad de los datos que se manejan en aplicaciones modernas hace que no sea posible definir a priori estructuras de almacenamiento apropiadas en el modelo relacional. Esto fuerza a las organizaciones a adoptar tecnologías que sacrifican estructura y eficiencia en el almacenamiento en pos de acceso rápido, escalabilidad y versatilidad[1].

Los sistemas OLTP (*Online Transactional Processing*) se enfocan en el procesamiento de transacciones. Típicamente consisten en realizar muchas operaciones en las que se agrega y/o actualiza una pequeña cantidad de datos cada vez. Esto es en oposición a los sistemas OLAP (*Online Analytical Processing*) en los que habitualmente se realizan pocas consultas que implican acceso a una gran cantidad de datos por operación[2]. Para mantener la integridad de las transacciones los sistemas OLTP idealmente deben cumplir las características del modelo ACID (atómicas, consistentes, independientes, durables)[3]. Dicho modelo surgió en forma concurrente con las bases de datos relacionales y ambos, junto con el lenguaje SQL, se convirtieron en los pilares de la arquitectura soportada por los principales sistemas gestores de bases de datos en su momento[4]. Por este motivo la inmensa mayoría de los sistemas OLTP están implementados con bases de datos relacionales. Sin embargo, en los últimos años esta tendencia se está revirtiendo. MongoDB[5] tiene soporte de integridad transaccional para transacciones multidocumento a partir de la versión 4.0[6].

El presente trabajo tiene como objetivo explorar la capacidad de MongoDB para ser utilizado como motor de operaciones transaccionales. Concretamente se pretende estudiar su desempeño ante la ejecución de varias transacciones concurrentes que involucren uno o más documentos. También se desea evaluar hasta qué punto se comportan las transacciones en MongoDB conforme a las cuatro propiedades ACID.

II. Antecedentes

Se toma como referencia un artículo[7] que presenta un estudio de rendimiento de MongoDB en transacciones multidocumento.

Durante el proyecto se intentaron seguir, dentro de lo posible, las decisiones presentadas en el artículo referentes a tecnologías y lineamientos generales sobre la forma de desplegar la solución para los experimentos.

En dicho trabajo se utiliza el modelo de datos TPC-C tomado de *Transaction Processing Performance Council*[8] que emula un sistema comercial con diferentes tipos de transacciones. Implementar el modelo TPC-C excede el alcance de este proyecto, por lo que se elige un modelo de datos más simple que permita medir el desempeño de MongoDB.

III. Modelo de datos

Los experimentos se realizan emulando un sistema de transacciones bancarias simplificado, en el que una operación puede ser un depósito a una cuenta, un retiro desde una cuenta, o una transferencia de dinero entre dos cuentas. Las primeras dos operaciones involucran un único documento y las transferencias dos documentos correspondientes a las dos cuentas que participan: cuenta origen y cuenta destino.

Las operaciones se definen en base a un juego de datos[9] obtenido de la plataforma Kaggle. Hay más de 116000 operaciones bancarias presentes contra un conjunto de diez cuentas, que consisten en retiros y depósitos solamente, y contiene campos que no se adecuan a los objetivos del trabajo. Los campos son:

- Número de cuenta
- Fecha de la operación y fecha valor
- Detalle o descripción de la transacción
- Monto de retiro
- Monto de depósito
- Saldo de la cuenta

El juego de datos original incluye un campo de saldo que refleja el monto de dinero en la cuenta luego de la operación. Con base en el modelo, se crean nuevas operaciones de tipo transferencia entre cuentas, para tener transacciones multidocumento contra MongoDB, ya que los retiros o depósitos afectan un solo documento. Para las transferencias se agregan campos con un segundo número de cuenta y el saldo de dicha cuenta luego de la transferencia. Además, en la planilla original el valor del saldo no refleja fielmente lo que corresponde a la cuenta ya que no se distinguen saldos entre distintas cuentas. En nuestro ajuste de datos, se recalculan los saldos para tener el valor correcto tras cada operación para cada una de las cuentas. Se agrega un número de secuencia como identificador global, siendo utilizado para distinguir las operaciones que lograron completarse a un momento dado. Luego de los ajustes nuestro modelo presenta los siguientes campos:

- Número de cuenta primaria u origen
- Número de cuenta secundaria o destino para las transferencias
- Número de secuencia
- Monto de la operación según corresponde:
  - Monto de retiro
  - Monto de depósito
  - Monto de transferencia
- Saldo de la cuenta origen
- Saldo de la cuenta destino

Se convierte el conjunto de datos disponible desde el formato original de planilla Excel a un archivo de texto plano con formato *JSON*. De esta forma se tienen los datos disponibles en una formato más adecuado al entorno de trabajo. A continuación se presentan ejemplos de operaciones de documento único y multidocumento.

Listado 1 Ejemplo de transacción de retiro (documento único)

```
{
  "ACCOUNT NO": "409000405747",
  "DESTINATION ACCOUNT": null,
  "SEQUENCE": 2941,
  "WITHDRAWAL AMT": 30000000,
  "DEPOSIT AMT": 0,
  "BALANCE AMT": 200000000
}
```

Listado 2 Ejemplo de transacción de transferencia (multidocumento)

```
{
  "ORIGIN ACCOUNT": "409000405747",
  "DESTINATION ACCOUNT": "1196428",
  "SEQUENCE": 202940,
  "TRANSFER AMT": 170000,
  "BALANCE Origin Account": -189567348,
  "BALANCE Destination Account": 237011721.61
}
```

Cada uno de estos objetos *JSON* representa una transacción que se impactará contra la base de datos. Se agrupan en diferentes archivos de entrada que serán leídos por los motores de transacción con el objetivo de paralelizar las ejecución durante el experimento con varios procesos, asegurando la concurrencia contra la base de datos. En la base de datos hay una colección de cuentas predefinida con un estado inicial. La colección contiene las diez cuentas identificadas en el juego de datos obtenidos de la plataforma Kaggle. En cada ejecución se reinicia la colección al estado inicial para validar fácilmente al final del experimento la consistencia del procesamiento y la integridad de las transacciones ejecutadas. El estado inicial de la colección de cuentas se presenta a continuación.

Listado 3 Estado inicial de la colección de cuentas

```
[
  {"Account": "409000611074", "Balance": 0},
  {"Account": "409000493201", "Balance": 0},
  {"Account": "409000425051", "Balance": 0},
  {"Account": "409000405747", "Balance": 0},
  {"Account": "409000438611", "Balance": 0},
  {"Account": "409000493210", "Balance": 0},
  {"Account": "409000438620", "Balance": 0},
  {"Account": "1196711", "Balance": 0},
  {"Account": "1196428", "Balance": 0},
  {"Account": "409000362497", "Balance": 0}
]
```

IV. Solución

En esta sección se dan detalles sobre el diseño, implementación y despliegue de la solución utilizada para ejecutar el experimento propuesto.

IV-A. Implementación

Se desarrolla un conjunto de herramientas y componentes en lenguaje *Python*. Incluye un simulador de transacciones encargado de leer los archivos de entrada con las transacciones en formato *JSON* y actualizar la cuenta o cuentas en la colección de la base de datos. Las herramientas desarrolladas proveen las siguientes funcionalidades:

- Conectar a la base de datos MongoDB configurada
- Inicializar la colección de cuentas
- Leer el conjunto de archivos *JSON* de entrada
- Interpretar el tipo transacción a procesar
- Administrar un *pool* de procesos disponibles para lanzar transacciones en paralelo y de forma concurrente para estresar la base de datos. Se utliza un *pool* de procesos para sortear la limitante de *Python* a la hora de usar *multithreading*, ya que con múltiples hilos, *Python* no permite ejecución realmente concurrente[10].
- Actualizar la base de datos procesando transacciones de documento único
- Actualizar la base de datos procesando transacciones multidocumento

La actualización de la cuenta al procesar una transacción implica lo siguiente:

- Actualizar el campo *Balance* sumando o restando el monto según corresponda a la transacción.
  - Retiros: Decrementa el campo *Balance* de la cuenta
  - Depósitos: Incrementa el campo *Balance* de la cuenta
  - Transferencias: Decrementa el campo *Balance* de la cuenta origen e incrementa el campo *Balance* de la cuenta destino. Este tipo de transacciones son multidocumento por afectar dos documentos diferentes en la colección de cuentas.
- Carga el campo *LastTxnSequence* con el número de secuencia de la transacción siendo procesada
- Carga el campo *LastUpdateTS* con la hora actual del sistema

Esas actualizaciones permiten validar la última transacción procesada, y validar que el saldo corresponde a lo esperado. A continuación, un ejemplo de una cuenta en la colección luego de haber impactado una transacción:

Listado 4 Cuenta actualizada luego de procesar transacción

```
{
  "_id": {"$oid": "62c232038926b557a08d49ee"},
  "Account": "409000493201",
  "Balance": 281383.32,
  "LastTxnSequence": 2137,
  "LastUpdateTS": { "$timestamp": {"t": 1656893958, "i": 1912} }
}
```

El código de las herramientas, incluyendo el motor que simula las transacciones, está disponible en el repositorio de *Gitlab* en el enlace https://gitlab.fing.edu.uy/marcelo.sureda/bdnrproject

IV-B. Infraestructura y despliegue

Para un ejercicio orientado a obtener métricas de rendimiento el equipamiento con que se cuenta es de fundamental importancia. El servidor de base de datos debe contar con ciertas características que permitan satisfacer los requerimientos mínimos del experimento. Un punto primordial a considerar es que para soportar transacciones multidocumento el servidor debe estar desplegado en un *cluster* con *replica sets*[11].

La opción lógica, siguiendo los lineamientos del artículo tomado como referencia[7], es utilizar un *cluster* con *replica sets* en MongoDB Atlas[12]. Acá las restricciones están impuestas por la posibilidad de acceder a un *cluster* gratuito desplegado

en la nube con el modelo de *Platform as a Service* (*PaaS*). Atlas provee a través de tres diferentes proveedores de servicios de computación en la nube (AWS, Google Cloud Platform y Azure) algunas opciones gratuitas.

Para la ejecución del cliente donde corre el motor simulando transacciones, hay dos opciones disponibles: ejecución en una máquina local, o desplegar una máquina virtual en la nube. La opción local genera el problema de la latencia de red para obtener las métricas de desempeño del sistema. La única opción disponible para el equipo con la cuenta de estudiante es el servicio de Azure.

Para evitar problemas con la latencia de red, o al menos minimizarlos, se debe desplegar la máquina virtual cliente en la misma región que el servidor. En Azure hay restricciones respecto a las regiones en las que se puede desplegar un servidor gratuito, y también el tipo de servidor que se permite aprovisionar. Considerando estas limitaciones, sólo se consigue desplegar servidor y cliente en la región *East Asia* de Azure localizada en Hong Kong. Las características de cada uno se describen a continuación:

- Servidor MongoDB Atlas
  - Sandbox M0 (Free Cluster)
  - Version 5.0.9
  - Replica Set con tres nodos: uno primario, dos secundarios
- Máquina virtual Cliente
  - Tamaño: Standard B1s
  - CPU: 1 CPU Virtual Intel(R) Xeon(R) Platinum 8171M CPU @ 2.60GHz
  - Memoria: 1 GiB
  - SO: Linux Ubuntu 20.04

El *cluster* gratuito M0, además del hecho de no poder ser alterado en la configuración provista por Atlas en cuanto a memoria o capacidad computacional, tiene las siguientes limitantes[13]:

- No permite pruebas de fallas como conmutación de servidor primario al secundario.
- Las únicas métricas disponibles son las siguientes:
  - Cantidad de conexiones
  - Tamaño lógico de la base
  - Volumen de tráfico de red
  - Cantidad de operaciones: inserciones, consultas, actualizaciones, eliminaciones, y comandos
- No provee acceso al panel de rendimiento en tiempo real.
- Permite sólo hasta 100 operaciones por segundo.
- Atlas limita el tráfico de red a los servidores que excedan el límite impuesto de operaciones por segundo.

Hay otras limitantes, como que el escalado automático del almacenamiento está deshabilitado, pero no afectan el ejercicio. La limitante que impacta las pruebas directamente es la cota de 100 operaciones por segundo. Se puede considerar como restrictiva para el experimento, pero se estima que es un número razonable como cota superior para un volumen transaccional en el mercado uruguayo. Sosteniendo en el sistema una carga de 100 transacciones por segundo durante diez horas (cosa altamente improbable) se le permitiría a toda la población del Uruguay realizar al menos una operación bancaria durante ese corto período.

V. Verificación y pruebas

Los datos para este experimento, cuyo modelo fue descrito en la sección III, están disponibles en el repositorio de *Gitlab*: https://gitlab.fing.edu.uy/marcelo.sureda/bdnrproject

Para capturar las métricas del servidor MongoDB se utiliza la herramienta *mongostat*[14]. Permite obtener las estadísticas internas de MongoDB con mayor precisión y libera de responsabilidad al motor de transacciones de realizar los cálculos correspondientes. La herramienta *mongostat* provee información de la cantidad de operaciones realizadas, más específicamente, de los actualizaciones (columna *update* en la información provista por *mongostat*) que corresponden a las transacciones por segundo realizadas desde el motor de transacciones. En esa métrica están basadas algunas conclusiones del proyecto, ya que se toma como referencia del rendimiento de la base de datos.

Como se detalló en la sección IV, el motor de transacciones lanza en paralelo y de forma concurrente las operaciones con la intención de estresar la base de datos y observar la consistencia del procesamiento en las multidocumento. Se realizan diferentes ejecuciones enviando a MongoDB transacciones de documento único, transacciones multidocumento, y combinando ambos tipos.

También se varía el parámetro *Write Concern*[15] que determina cómo se realizan las escrituras en los *replica sets*. Se hacen pruebas con el valor de *Write Concern = 'majority'* que obliga a que la escritura se propague a la mayoría de los nodos en el *cluster* antes de devolver el mensaje con la confirma31.1ción al cliente. En nuestro caso, eso corresponde al servidor primario y alguno de los secundarios. También se prueba con valores de *Write Concern* = 1 (sólo actualización en el nodo primario) y *Write Concern* = 3 (propagación del cambio a todos los nodos).

Los experimentos con transacciones multidocumentos se realizaron tres veces y los resultados se promediaron. Las medidas de tráfico se miden en bytes/s. A continuación se presentan los resultados.

	Updates	Tráfico de entrada	Tráfico de salida	Conexiones
Documento único (sin sesión)	48.7	212.2	171.3	31.0
Documento único (wc=1)	48.7	20.7	15.6	31.1
Documento único (wc=2)	48.9	21.2	16.9	31.2
Documento único (wc=3)	48.6	21.2	17.1	31.1
Multidocumento (wc=1)	38.9	24.2	19.0	14.3
Multidocumento (wc=2)	38.8	24.2	18.8	14.9
Multidocumento (wc=3)	38.9	24.2	18.7	14.9

Ejecutando el comando *top* en la máquina virtual de Azure se verifica que no hay un cuello de botella en el cliente. Con todos los procesos del motor de transacciones corriendo en paralelo la CPU consume menos del 20 % de su capacidad, y el uso de memoria física ronda el 50 %. Tampoco se detectan problemas con la red. La latencia promedio contra el servidor se encuentra en 2 ms (0,002 segundos), y se mantiene constante durante el experimento.

VI. Conclusiones y trabajo futuro

Se comprueba que MongoDB permite realizar el tipo de operaciones planteadas al principio del proyecto soportando una carga transaccional que se considera aceptable.

Como era de esperar, el rendimiento de la base de datos fue menor al tener que procesar transacciones multidocumento, pero no fue un descenso considerable: alrededor del 20 % respecto a las transacciones de documento único.

Las transacciones multidocumento, desde el punto de vista de su integridad, no son afectadas por otras transacciones concurrentes. Se valida esto con el campo de saldo de la cuenta, al confirmar que se corresponda con las transacciones ejecutadas. Se comprueba de forma exitosa, en todos los casos, verificando ejecuciones parciales y del conjunto completo de datos.

Al contrario de los que se suponía, la variación del parámetro *Write Concern* no afecta en nuestras pruebas el rendimiento de MongoDB respecto a las transacciones multidocumento. En teoría, a un mayor valor de *Write Concern* corresponde un mayor tiempo de demora en el *cluster* propagando la actualización, y por consiguiente, una mayor espera por parte del cliente del mensaje indicando que la escritura fue exitosa. Por esta causa las transacciones deberían tener un mayor tiempo de ejecución y, por lo tanto, se observaría un menor rendimiento general del sistema. Sin embargo, no se detectan grandes cambios en las métricas del sistema entre las diferentes ejecuciones variando el parámetro *Write Concern*. Se especula que eso se debe a haber ejecutado las pruebas en un ambiente restringido con tres nodos en el *cluster* gratuito de Atlas. En este ambiente reducido la variación puede ser imperceptible. También se supone que podría tenerse un comportamiento distinto poniendo el parámetro `'j'` de *Write Concern* con valor `TRUE`. Esta configuración obliga a MongoDB guardar el cambio al *Journal* de la base de datos en cada nodo, y no solamente a impactar el cambio en memoria antes de enviar el acuse de recibo.

Dentro del trabajo a futuro se pueden mencionar los siguientes puntos:

- Propiedades ACID.
  Hay un conjunto de pruebas diseñadas para validar el cumplimiento de las propiedades ACID para las bases de datos[8], que no llegó a ser implementado.
- Implementar lecturas.
  De la mano del punto anterior se podrían implementar lecturas en forma concurrente con las actualizaciones, a forma de consultas de saldo. Daría un escenario más completo a la prueba y permitiría validar que el saldo de las cuentas es consistente respecto a lo procesado de forma automatizada.
- Generar un modelo más complejo y utilizar mayor cantidad de cuentas.
  El modelo utilizado tiene solamente diez cuentas con pocos campos y la base de datos no crece en tamaño. Sirve para validar conceptos pero no es un escenario realista. Se podría pensar un sistema con varias decenas o cientos de miles de cuentas en donde el tiempo de búsqueda y acceso pueda representar un problema. Se podría estar en la necesidad de crear índices para acceder de forma más eficiente. También se puede considerar como requerimiento del negocio una nueva colección que registre todas las transacciones recibidas por el sistema, por ejemplo, por motivos de auditoría. Esto último implica un crecimiento del tamaño de la base de datos, lo que puede derivar en una penalización del rendimiento. Para este punto se debe contar con un sistema de mayor capacidad.
- Implementar pruebas con parámetro `'j'` de *Write Concern*.
  Como se mencionó este parámetro puede afectar el rendimiento obligando a los nodos secundarios a un acceso a disco.

REFERENCIAS

[1] *Advantages of NoSQL Databases*. URL: www.mongodb.com/nosql-explained/advantages (visitado 01-07-2022).

[2] *What Is an OLTP Database? {Concepts & Examples}*. Knowledge Base by phoenixNAP. 12 de mayo de 2021. URL: https://phoenixnap.com/kb/oltp-database (visitado 01-07-2022).

[3] *What is OLTP?* URL: https://database.guide/what-is-oltp/ (visitado 01-07-2022).

[4] *Next Generation Databases*. URL: https://link.springer.com/book/10.1007/978-1-4842-1329-2 (visitado 01-07-2022).

[5] *MongoDB — Build Faster. Build Smarter.* en-us. URL: https://www.mongodb.com (visitado 03-07-2022).

[6] *MongoDB Drops ACID — MongoDB Blog*. MongoDB. URL: https://www.mongodb.com/blog/post/multi-document-transactions-in-mongodb (visitado 01-07-2022).

[7] Asya Kamsky. "Adapting TPC-C benchmark to measure performance of multi-document transactions in MongoDB". en. En: *Proceedings of the VLDB Endowment* 12.12 (ago. de 2019), págs. 2254-2262. ISSN: 2150-8097. DOI: 10.14778/3352063.3352140. URL: https://dl.acm.org/doi/10.14778/3352063.3352140 (visitado 03-07-2022).

[8] Scott T. Leutenegger y Daniel Dias. "A modeling study of the TPC-C benchmark". en. En: *ACM SIGMOD Record* 22.2 (jun. de 1993), págs. 22-31. ISSN: 0163-5808. DOI: 10.1145/170036.170042. URL: https://dl.acm.org/doi/10.1145/170036.170042 (visitado 03-07-2022).

[9] *Bank Transaction Data*. URL: https://www.kaggle.com/datasets/apoorvwatsky/bank-transaction-data (visitado 01-07-2022).

[10] *Multiprocessing — process-based parallelism — Python 3. 10. 5 Documentation*. URL: https://docs.python.org/3/library/multiprocessing.html (visitado 04-07-2022).

[11] *Transactions — MongoDB Manual*. URL: https://www.mongodb.com/docs/manual/core/transactions/ (visitado 01-07-2022).

[12] *MongoDB Atlas — multi-cloud developer data platform*. en-us. URL: https://www.mongodb.com/atlas (visitado 04-07-2022).

[13] *Atlas M0 (Free cluster), M2, and M5 limitations — Mongodb Atlas*. en. URL: https://www.mongodb.com/docs/atlas/reference/free-shared-limitations/ (visitado 04-07-2022).

[14] *Mongostat — MongoDB Database Tools*. en. URL: https://www.mongodb.com/docs/database-tools/mongostat/ (visitado 04-07-2022).

[15] *Write concern — mongodb manual*. en. URL: https://www.mongodb.com/docs/manual/reference/write-concern/ (visitado 04-07-2022).