noSQL 雜談

by

Triton Ho 

今天大綱

- 傳統 tablespace 問題
- 淺談 SSTable
- 回看 atomic check-and-set 案例
- Cassandra 架構
- Hbase 架構
- MongoDB 架構
- 小總結

RDBMS 的 tablespace

- 所有的 RDBMS ，其資料都是存放在 data page 內
  - Oracle, postgresql: 8KB
  - MySQL: 16KB

在 RDBMS 內改一筆資料

- 把資料所在的 data page 整個放到 memory
- 改動資料，並且把 data block mark as dirty
- 把改動 append 到 REDO log
- Background worker 把整個 data block 寫回去 storage

Data page 問題點

- 即使你只改動一點點，你還是要整個 data page 作 file IO
  - 而且是 Random Write IO
- 在 background worker 寫回 storage 前
  dirty page 會持續佔用你的 main memory
  - 別忘記： dirty page 內不是全部資料有改動耶～
- 你只能作 page level storage compression
  data page 才 16 KB ， compression 不會有很好效果
- 不良的 schema design 下，會發生 page level contention
- B+ tree 的 page splitting 和 merging 會引起廣域 page lock

SSTable 背景

- DB server 有多少 main memory
  - 2000 年： 16GB 算頂級
  - 2023 年：頂級 1024GB ，平民級 16GB ，窮人 4GB
- 20 年前，在 main memory 不足下
  DB 應該盡快把資料改動抄回去 storage ，讓多些 main memory 留作 caching
- 除了 RAM 便宜，今天的 storage 也像不用錢的～
  - Storage IO 很貴，但是 storage volume 很便宜

SSTable 架構

- SSTable 一旦建立後就永不改動（immutable）
- 有資料改動時， DB 會把新版本資料留在 main memory
  - REDO log 還是會寫的
- Memory buffer 快要滿時，再一口氣把全部新資料去建立新的 SSTable file
  - 所以沒有了 Random Write IO

SSTable 架構

- storage 中會存在多個 SSTable file
- 當你要讀取 Record X 時，你需要檢查每一個 SSTable
  - 如果 Record X 存在於多個 SSTable
    便使用最新的一個版本
- database 的 background worker ，會把多個 SSTable file 合成一個

SSTable 結構

- Meta data 部份
  - Primary Key 的 bloom filter
  - PK Index ，指向對應的 row data location
- Row Data 部份
  - 以 segment level 作 compression

SSTable 優點

- 因為 SSTable 永不改動，所以你不用像 RDBMS 中的 data page 要預留未來改動的空間
  - Postgresql 預留 data page 10% 用作未來改動
- 你的 PK index 不會有 B+ tree 的 splitting 和 merging
- 你能以 segment level 作 compression

SSTable 注意事項

- 你能用作 buffer 的 Main Memery 越少，便會觸發越多的 flushing ，建立更多的 SSTable file
- 在使用 SSTable 的 noSQL （像 Hbase, Cassandra），你應該先 scale up ，直到 server cost 會 non-linearly 增加時才 scale-out
- 太小的專案，你不應該用很小的機器
  他喵的你小流量就回去用 pg / mysql 啦！

atomic check-and-set 案例

- 我想建立一個簡單的戲院售票系統～

  table: seat

  - (event_id, seat_no) PK
  - status
  - reservation_user_id
  - reservation_ts

買票步驟 1

- 先去佔下一個空座位，然後在 5 分鐘內去 payment gateway 付費

  如果在 5 分鐘內沒法付費，則該座位開放其他人訂購

- update seat
  set status = ‘reserved’,
  reservation_ts = now(), user_id = :user_id
  where event_id = :event_id and seat_no = :seat_no
  and (reservation_ts is null or reservation_ts < now() - interval ‘5 minute’)
  and status != ‘sold’

買票步驟 2

- 付費後，改變座位 status
- update seat
  set status = ‘sold’,
  reservation_ts = null, user_id = null
  where event_id = :event_id and seat_no
  = :seat_no
  AND user_id = :user_id

Cassandra 架構

- 在 cluster 中，每一個 node 都是 master
  - REDO log 是存於 node local storage 上
- 一個 record ，會存到 n 個 node 上
  - 一般而言， n 是單數
  - 因為資料存於 multi-master ，所以不需要 master-slave replication
- Quorum Consistency
  - 不管 READ 和 WRITE ，都需要至少 (n/2)+1 台成功
  - 在 READ 時，如果二個 data node 返回不同版本時，使用較新版本
  - 保證拿到最新版本的 record
- Gossip
  - node 之間通訊
  - 交流 record 的最新版本
- 無法使用 Atomic CAS

增減 Cassandra node

- 根據 consistent hash ，在 N 個 node 內加減一個 node 時
  - 每一個 data node 需要搬動 1 / N 的資料量
- 在資料經過 gossip 轉移前，新的 node 是沒有該份資料的
- 但是，基於 Quorum Consistency 機制，少了一個 data node 是不影響的，所以對 end user 是 zero downtime 的

無法 Atomic CAS

- 整個過程都用 Quorum Consistency
- 開始時三個 data node A, B, C：
  - value = 100, ts = 10:00:00
- client 寫入 value = 101，現在成功跟 node A 和 node B 通訊
  - Node A, B: value = 101, ts = 10:00:04
  - Node C: value = 100, ts = 10:00:00
- client 發出 CAS: set value = 102 if value = 100
  - Node A, B: return fail, Node C: return OK
- 這時的最新狀態：
  - Node A, B: value = 101, ts = 10:00:04
  - Node C: value = 102, ts = 10:00:10
- 然後 gossip，Node C 的 ts 比較新，所以 value 102 變成所有 node 的 eventual state

Cassandra 適用場景

- Cassandra 是 AP 向的
  - 即是說：你一點也不介意 data consistency
  - 如果沒有 zookeeper / redis 作 locking server
    或是你走 event-based system 路線
    你單用 Cassandra 是寫不出售票系統的
- Cassandra 的 mult-master 概念，配上 DC awareness
  - 再一句：前提你需要一點也不介意 inconsistent data
  - 能大幅降低你的 latency
  - 你能做到整個 AWS / Azure region 趴掉時，還能維持服務
- 現實上，毫不介意 consistency 的商業應用很少的
  - 主要應用：不介意有錯的通訊系統／ social platform

HBase 架構

- HDFS = Hadoop file system
  - 所有 file 都是 immutable
  - 每一個 file 會存到 n 個 data node 來避免 data loss
- HBase = Database on top of HDFS
- HBase 的 node 本身不儲存 data，資料是存放在下層的 HDFS 內
  - REDO log 也是以放到 HDFS
  - 因為資料不是在 HBase node，所以 HBase node 死了也沒所謂
  - 所以不需要 master-slave replication
- 每一個 HBase node 只負責特定 PK Range 的 data operation
  - Record X 的 READ / WRITE，肯定會送到同一個 HBase node
  - 所以能有 strong consistency，也能做到 Atomic CAS

HBase coordinator

- 官方文件叫 master node，但跟 master-slave 毫無關係
- 本身不處理任何 data operation
- 儲存／管理 PK to node 的 mapping table
- 聆聽 HBase node 的 heartbeat
- 正常 cluster 中，只會有一個 coordinator(active)
  - 也許會有數個 coordinator(standby) 作 failover
- 因為 mapping table 只由 coordinator 一個來管理，這樣子保證同一 record 的所有 operation 都會送到同一個 HBase node 上
  - 所以 HBase 才能做到 Atomic CAS

增加 HBase node

- 再說一次：HBase node 本身不儲存任何資料的
- 新的 node 對 coordinator 發出 heartbeat
- coordinator 發現 cluster 內這個新的 node 沒有負責範圍，所以一點點的改動 PK mapping table
- 改動 PK mapping table ：
  - coordinator 告訴本來的 HBase node 改動
  - HBase node 把 memory buffer flush 到 HDFS 層
    然後回答 coordinator node OK
  - coordinator 把 mapping table 改到新的 HBase node
    然後把改動告知新的 HBase node ，讓新的 data node 負責該 PK range
- 重點：在 HBase node 轉移的，只是責任範圍而不是資料本身
  - 雖然受影響的資料不是完全的 zero downtime ，但是非常小

HBase 適用場景

- 你面對的流量是 single master postgresql / mysql 支持不了的
- HBase 算是 CA 向的，跟 RDBMS 一樣
- 你不能錯了也沒所謂，你需要 Atomic CAS
- HBase 強項： Range Scan with PK
  - 例子： Line 的對話歷史
- 注意： HBase 架構本身沒有 DC awareness

MongoDB 架構

- 一份資料，只會存於在某一 mongo data node 上（mongod）
  - 為了避免 data loss ， mongo data node 需要 replication
- 跟 HBase 和 Cassandra 不同， mongodb 是採用 B+tree variant 來存資料
  - 不是 SSTable
  - WiredTiger 嚴重缺乏說明文件
  - 早期的 MMapv3 engine 有 catastrophic data loss 風險
- 內建 2pc ，能一定程度上支持 multi-record atomicity

增減 mongodb data node

- 資料都存放在 data node 的 local storage 上
- 所以增減 data node 時，需要 data 的移動，很吃 resource
- 因為需要等待資料的搬動，所以 downtime 會不短
  - 所以，千萬別在 peak hours 時才加上 mongodb data node

mongodb 適用場景

- Decision maker 是笨蛋，而且公司很有錢想多花在 server cost 上
- 你希望同時擁有 noSQL 和 RDBMS 的缺點
- 你想公司發生 catastrophic data loss ，但不想自己動手

小總結

- 再次強調：你的資料量連 1TB 都沒到就喵的回去 RDBMS 吧
  - 雖然……個人對 Mysql 是否能支持 1TB data size 很有保留……
- Database layer 是系統中最難搬動的部份
  去挑選要用的 database 時，你最應該看的是其背後架構，那才是你真正限制你長遠發展的地方

完