資料庫實戰技巧

By Triton Ho
Software Engineer of Mar's Potato

大綱

- 常犯錯誤
- 面對高流量技巧
- Async IO
- 小薯塊 (small potato) 成長之路

常犯錯誤（一般方面）

- 用 RDBMS 作為 Inter-process communication
  - RDBMS 內有 REDO LOG, UNDO LOG, Lock Manager, 這些都是為了維護資料的完整性和正確性。
  - 單純用來暫存 IPC 數據 , 會浪費 CPU 和 Storage IO 的
  - 最簡單的 IPC ，可以使用 Redis 中的 List ，進階的請用 rabbitMQ
- 用 RDBMS 作為 logging server
  - log = 建立了便永不改動的數據，而且這些數據極少被單獨使用，一般都會經過 aggregation 用來產生報告
  - txt 檔案是 log 的理想存放位置
- 在 Master 上產生報告
  - 應該在 read-only slave 上進行的
  - 產生報告時會吃掉 Master 寶貴的 Storage IO ，而且佔用寶貴的 Main Memory
  - 更大型系統，應該有專門作 Reporting 的資料庫（像是 Data Warehouse ）

常犯錯誤（編程方面１）

- SQL statement 超過了１００行
  - ~~會有報應的，請相信我~~
  - 太長的 statement ，會讓 Query optimizer 沒法有效工作
  - 應該把太長的 statement 改成多個步驟，每個步驟的結果用 temporary table 去暫存
- 把系統邏輯放到 RDBMS 的 trigger
  - （除非正在 database refactoring ，利用 trigger 作為一種暫時性手段）
  - 你應該把系統邏輯放到 application tier
  - 當出現了 bug 而引起資料錯誤，你要在十分鐘內作緊急的資料更正，這些 trigger 會妨礙你的
- 把所有的東西都寫在 application tier
  - 可以考慮 data intensive / multiple steps 工作用 stored procedure 來做

常犯錯誤（編程方面 2 ）

- 在 stored procedure 中使用 loop
  - SQL 本身是 set-based 語言，請盡量使用 SQL 作 set operation
- 在 stored procedure 中使用 recursion
  - 請考慮把工作放在 application tier

常犯錯誤（ schema 設計 1 ）

- 使用 auto-increment 作為 PK
  - 別害怕 OR/M 層的麻煩，系統發生 blocking 時要花的氣力遠比 OR/M 的麻煩要多
  - 如果有 natural key ，請優先使用， natural PK 有機會能用作 index skip scan
  - 真的不能用 natural key 時，請用 UUID
- 為了報表而建立 index
  - index 會大幅影響 master 的 WRITE 效能
  - Oracle 說：每個額外的 index 便多用 3 倍時間
  - 報表能在 slave node 慢慢產生
  - 報表很多時候需要 range scan ， secondary B+ tree index 幫助不大
- 在一般系統中使用 bitmap index
  - bitmap index 是針對報表的
  - update bitmap index 是單線程的

常犯錯誤（ schema 設計 2 ）

- 把不同種類的 row 放到同一 table 上
  - 特徵 1 ：會有一個叫 row_type 的 column ， application tier 需要先知道 row_type 是什麼才能決定怎去處理這個 row
  - 特徵 2 ：會有大量的 rows 在特定 column(s) 是 null value
  - 例子：把〔狗〕{主人，體重，高度} 和〔貓〕{主人，體重，毛色，可愛度} 都放到一個叫〔寵物〕{主人，種類，體重，高度，毛色，可愛度} 的資料表中

面對高流量技巧

- 盡量短的 TX
- 把多個 SQL statement 用一個 stored procedure 代替
- Conflict promotion
- Conflict Materialization
- 優化 data update 次序
- Aysnc update
- 用獨立 worker 跟第三方系統溝通
- 認清系統的 peak hours 和 idle time
- 小心選擇 PK

盡量短的 TX

- TX 越短， LOCKING 時間也越短，對應的 Redo Log 和 Undo Log 也越短生命
- Undo Log 過大的機會也越少
- LOCKING 時間也越短 ，發生 Blocking 的可能性也越短

把多個 SQL statement 用一個 stored procedure 代替

- 一般的 WRITE 之前都有 checking 的，像是轉帳前要先檢查這個用戶是否有足夠的錢，這樣多個的 statement 能用一個 stored procedure 代替
- 可以省下 application tier 和 database 的一次 data 來回
- 短一些的 TX = 少一些的 Blocking

Conflict promotion

- 以下是人們從銀行提款時的情況

  begin TX

  select balance from account where account_id = ‘AccountX’;

  if balance < withdrawal_amount then rollback TX and then raise exception

  update account set balance = balance - withdrawal_amount where id = ‘AccountX’;

  commit TX

- 這個 TX 需要 Repeatable Read 才不會發生 concurrency 問題
- 可是，如果改成 select … for update ，便會對 rows 產生 WRITE LOCK
- 這樣便能用低階的 Read Committed 也不會發生問題

Conflict Materialization

- 有些checking，是需要aggregation data的。比如說提款時，某銀行容許個別戶口為負值，但同一用戶所有戶口必須>= 0
- 這個時候，單純的select … for update沒法保證在concurrency的正確性，因為這樣無法防止用戶把所有錢提光，並且在同一時間建立一個負值balance的新戶口
  - begin TX

    select sum(balance) as total_balance from account where user_id = ‘UserX’;

    if total_balance < 10000 then rollback TX and then raise exception

    update account set balance = balance - withdrawal_amount where id = ‘AccountX’;

    commit TX
- 用上Serializable Isolation，Predicate locking / monitering能解決問題，只是會用大量的CPU
- 可是，如果額外再建立一個叫user_total_balance的table，然後首先執行select … from user_total_balance … for update，便能改用Read Committed了

優化 data update 次序

- 以下是轉帳的程式碼：
  1. 開啟 TX
  2. 從轉出用戶扣錢
  3. 把錢加到目標用戶上
  4. 結束 TX
- 看起來沒問題吧．可是如果同時發生：A 轉錢給 B ， B 轉錢給 C ， C 轉錢給 A , 便會發生 deadlock 了
- 為了迴避 deadlock ， application 應該讓 Locking 次序是基於 data ，而不是基於行為次序
- 最簡單來說，如果在 application 加入規則：首先 update user_id 比較小的 rows ，那麼 C 轉錢給 A 的第二步和第三步便會倒過來，然後「 A 轉錢給 B 」和「 C 轉錢給 A 」便不能同時執行，不會發生 deadlock

用獨立 worker 跟第三方系統溝通

- 你不知道第三方糸統什麼時候當掉（ Google GCM 也試過當掉）
- 請不要「有難同當」，別人系統當掉時自已也一起當
- 如果程式沒寫好，沒好好處理 exception ，隨時讓 TX 沒有結束留下來

認清系統的 peak hours 和 idle time

- 把維護性工作集中在 idle time 解決
- 部份 replication slave node / caching tier 在 idle time 時關掉以節省金錢

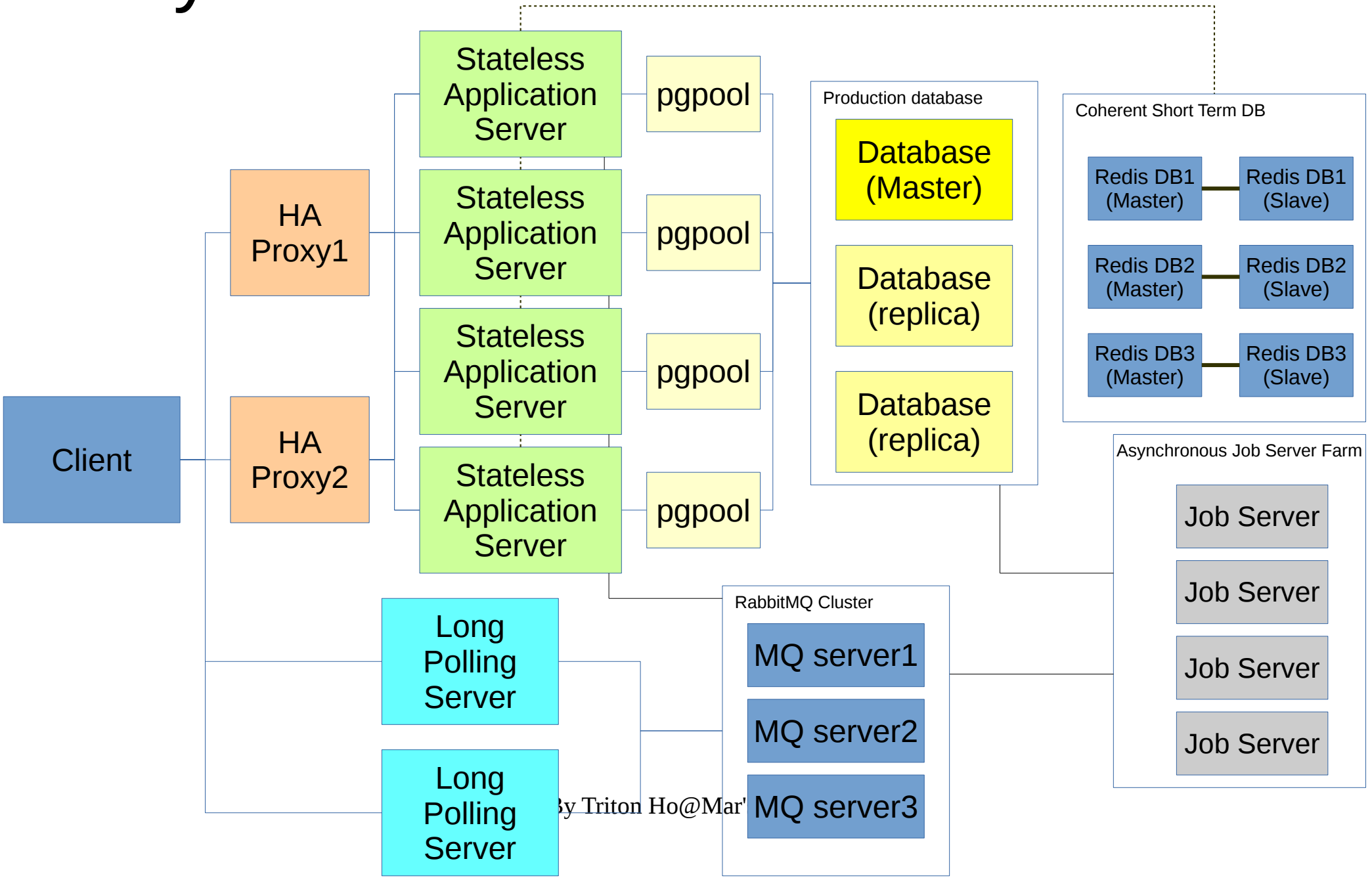
小心選擇 PK

- 再說一次：別用 auto-increment
- natural key 有可能有 pattern 的
  - 像是年紀相近的人會有相近的香港身份證ＩＤ
- 可以對 natural key 進行 2-way bit shuffling ，讓本來相鄰的 PK 被 shuffle 成為不相鄰

Async IO

- Peak hours 平均每秒 1000 個 req/sec ，不等於最高峰時每秒 1000 個 req/sec
  - Happy new year problem
- 系統繁忙時間容易知道，但突發性的 peak 卻不能預測
  - ９１１的網絡繁忙
- 讓所有用戶都在等待，只會讓用戶按下 Refresh ，然後便是全部人都用不了
- 應該先告訴用戶：你的請求收到了，工作做好了我會告訴你的
  - 這需要 UX 上作出配合
  - 這需要系統架構上作出大幅更動配合

Asynchronous API and Server Push



第三節完
發問時間