# 淺談資料庫

By Triton Ho
Software Engineer of Mar's Potato

感謝各位台灣朋友的熱情幫助
沒有你們提供免費場地，住宿，活動統籌，便不會有這次活動

# 大綱

- 數據的成本
- 應該懂得跟老闆說的話
- 溫故知新： Relational database 的 ACID
- 新的思維： MVCC

# 數據的成本

# 數據的成本

- 老闆的腦殘度
- 數據新鮮度
- 數據正確度
- 數據生存時間
- 系統可用性
- 規模可伸縮性

# 老闆的腦殘度

- 「你聽我的指示便好，有什麼問題我負責 ~ 」
- 「我們的系統要支持一百萬人同時在線耶 ~ 」
- 「 Refactoring ？先把新功能做好了，之後有空再慢慢來吧。 」
- 「我們一秒也不能停機的，你去設計一個 24*7 運作的系統吧 ~ 」
- 「 facebook / google / instagram 這樣那樣如何如何，你去根據他們的架構來做吧 ~ 」

# 數據新鮮度

- 數據改動要即時反映出來嗎？
  - 即時？容許一定時間之後？
  - 只讓改動者「即時」看到？所有人都「即時」看到？

# 數據正確度

- 數據的先後次序是否關鍵？
- 同一個數據，會有多人同時試圖改動嗎？
  - ＬＯＬ尾刀問題
- 容許系統進入暫時性「錯誤」的狀態嗎？
- 容許數據出錯？
  - 限量搶購１００個商品問題
- 在系統某部份當掉時，容許小量數據流失？

# 數據生存時間

- 數據需要生存多久？
  - 請別說「永遠」
  - ~~每個男人都懂得說「我會永遠愛你～」~~

# 系統可用性

- 需要２４＊７系統可靠性嗎
- 系統可以關掉維修嗎？維修時間可以多久？
- 災難後，需要多久來復原系統？
- 災難時，能讓「部份」資料先行在線嗎？
- 在超高使用量時，能讓暫時拒絕部份用戶嗎？

# 規模可伸縮性

- 超高使用量時，需要能增加額外的資源？
- 增加時可以關機重開？
- 新增加資源，如何能快速有效使用？

# 應該懂得跟老闆說的話

# 老闆會說

- 多！快！好！省！
- ~~（你的工資能減一點更好！）~~

# 你應該回答

幹！

# 應該懂得跟老闆說

- ~~請增加我的工資~~
- 數據延遲是可以接受的
- 不是所有數據都是重要 / 必須正確的
- 勇敢地面對數據流失
- 24*7 不等於 0 秒離線
- 有限度服務的系統比全面癱瘓的系統好

# 數據延遲是可以接受的

- Facebook, whatsapp 是有延遲的
- Google Cloud Messaging 可以延遲超過 3 0 分鐘
- 除非發送者正在接收者面前，否則接收者不會知道數據的發送時間
  - 一般用戶：收到信息的時候便是發送時間，所以是「即時」收到的
- 在流動電話，一切延遲可以推給流動網絡 ~
- 有些延遲對用戶是無所謂，而且用戶也不能發現的
  （例子：討論區的「熱門話題」）

# 延遲的數據正確性

- 早期的魔獸世界，寄信／收信有一小時延遲
- 銀行也是先扣了你的錢，等一下才轉帳到目標帳戶
- 只要用戶可以接受，這樣的最終正確性能增加系統效能

# 不是所有數據都是重要

- 部份數據是（相對上）不重要的
  - LOL 的房間當掉沒有了，用戶罵完後只會開新遊戲繼續玩
- 在遊戲發行一百萬件虛寶，跟在一百萬零三十一件虛寶是接近無分別的
  - 越能把正確性「放鬆」，系統便會越快！

# 勇敢地面對數據流失

- Amazon S3 只保證資料流失機會率 < 10^-9
  - 這世界沒東西是永久的~~（包括愛情？）~~
- 軍事戰略： defense in depth（縱深防禦）
  - ~~好的後端工程師一定是遊戲迷~~
- 不要把所有氣力放在一條防線上，而是建立多層防線
- 公司運作要預先準確數據流失後的應變
- 越接近１００％可靠性，額外要付的錢便越高
  - 甚至比數據流失的應變費用更高
- 別讓公司把所有責任推給後端工程師

# 24*7 不等於 0 秒離線

- 主資料庫掛掉，需要一分鐘的時間做 failover
  - 主資料庫不回答可以是太忙，也可以是真掛掉了
- Google, Facebook, 也試過當掉了
- 只有數分鐘的停止服務，你可以推說是用戶網絡問題
- Maintenance windows（維修窗口）在 24*7 系統很正常的
  - 核電站也會關機保養
  - 魔獸世界逢星期四早上維修

# 有限度服務的系統

- 讓一萬個顧客全進一間十平方米的餐廳，只會讓所有人都吃不到東西
- 在超乎預計的用戶量時，系統要拒絕部份用戶
  - HTTP 503
  - LOL 和魔獸世界也要排隊啦
- 收下用戶的請求，先回應用戶＂請求已經收下＂
  - 這便是 asynchronous IO
  - 防止 HTTP timeout error
  - 防止用戶不停重按，在系統正在忙時增加系統的額外工作量

# 溫故知新： Relational database 的 ACID

# Relational database 的 ACID

- Atomicity
  同一個 TX(transaction) 內的資料改動必須全被執行 (committed) / 全不被執行 (rollbacked)
- Consistency
  所有數據更動都需要滿足 data types, constrains, triggers, cascades ，以保障資料的正確性
- Isolation
  同時執行的 TX 需要維持其獨立性。
  資料最終的執行結果必須是某一執行順序的結果
  即是說：不能發生 Race condition
- Durability
  已經 Committed 的 TX 的資料更動永不流失（除非儲存裝置故障）

# ACID，名詞介紹

- Dirty Read
  - 能讀取其他還未 committed 的 TX 的資料改動
- Non-repeatable read
  - 在同一個 TX ，同一筆資料在第一次讀取和第二次讀取出現不同結果
  - 另一個說法：讀取的資料包含其他已經 committed TX 的 update ，而這些 TX 的 commit 時間發生在本 TX 的開始之後
- Phantom read
  - 在同一 TX ，同一 Query 在第一次和第二次執行時，出現不同結果
  - 另一個說法：讀取的資料包含其他已經 committed TX 的 insert/update/delete ，而這些 TX 的 commit 時間發生在本 TX 的開始之後
- http://en.wikipedia.org/wiki/Isolation_(database_systems)

# RDBMS 的 Isolation level

- Read Uncommitted
- Read Committed
  - 能防止 Dirty Read
- Repeatable Reads
  - 能防止 Dirty Read, Non-repeatable read
- Serializable
  - 能防止 Dirty Read, Non-repeatable read, Phantom read

# 傳統的 Read Committed

- 對已經改動的 rows 加上 WRITE_LOCK ，直到本 TX 完結
- 其他試圖讀取 / 改動這些 rows 的 TX 將會等待（ blocked ），直到本 TX 完成
- Writes block Reads

# 傳統的 Repeatable reads

- 在 Read Commiited 的基礎上，所有讀取過的 rows 都加上 READ_LOCK
- 其他試圖更動這些 rows 的 TX 都會被 blocked
- Reads block Writes

# 傳統的 Serializable

- 先說一下： RDBMS 的 Serializable 不等於數學上的 Serializable
- predicate lock ：像是 (where last_update_time > now() - 1)
- 在 Repeatable Read 的基礎上，在執行 query 時，除了為會被讀取的 rows 加上 READ_LOCK 之外，還加上 predicate lock
- 其他 TX 的 insert/update/delete ，只要影響到的 rows 滿足 predicate lock 的範圍，那個 TX 也會上鎖
- Reads block Writes
- 極度吃 CPU 也極容易引起 deadlock ，一般情況下不建議使用

# 傳統 Isolation 的問題

- READ 會 block WRITE
- 容易發生 deadlock
- Deadlock detection 和 lock management 極吃 CPU
- Application programmer 沒法先拿舊的資料使用，必須等待目前 blocking TX 完成才能拿到最新的資料

# 新的思維： MVCC

# 新的思維： MVCC

- 全名： Multiversion concurrency control
- rows 會有多個版本，對 rows 的改動會為其增加一個新的版本
- 每個 TX 都彷彿在自己的 snapshot 內工作
  - 快照式事務隔離 (Snapshot Isolation)
- 沒有 READ_LOCK ，只有 WRITE_LOCK
- READ 永不被 BLOCKING
- READ 也永不引起 BLOCKING

# MVCC 的 Read Committed

- 所有的 Query ，只會考慮已經 committed 的最新版本
- 對目標 rows 進行 insert/update/delete 時，會為其加上 WRITE_LOCK ，直到 TX 完成
- 當試圖 insert/update/delete 的 rows 已經被其他 TX 加上 WRITE_LOCK ，本 TX 才會被 block
- 正常情況下，配合 conflict materialization + conflict promotion ，足以面對大部份情況而無需使用更高的 isolation level

# MVCC 的 Repeatable Reads

- 所有的 Query ，只會考慮已經 committed ，並且在本 TX 開始前已經存在的最新版本
- 別名： Shapshot（快照） Isolation
- 在 WRITE 時，額外檢查一下目標 rows 是否有在本 TX 建立後的新版本，如有則 raise exception 並且 rollback
- Oracle 聲稱的 Serializable ，實際上是 Repeatable Reads

# MVCC 的 Serializable

- 為每個 Query 的 Predicate 加上 Predicate monitering
- 當有新版本的 commited rows 滿足 predicate ，而其版本是在本 TX 開始的時間點後，則本 TX raise exception 並且 rollback
- 高 CPU 要求，不建議使用
- Oracle 沒有這個級別

# MVCC 的好處

- 沒有 READ_LOCK ，而且 READ 也不需要檢查 WRITE_LOCK ，系統中 LOCK 的總數量大減
  - LOCK Manager 的工作量大減
  - Deadlock detection 變得簡單，也工作量大減
- 先天性讓 deadlock 大幅減少
- Blocking 時間大大減少
- 在高流量時，系統性能遠遠比傳統 Isolation 更好

# MVCC 缺點

- 多個版本的 data 同時生存於 RDBMS 內部，RDBMS 需要管理舊版本的生命週期
- 每個 TX 比傳統 Isolation 用多了 CPU 和 Disk IO

第一節完
發問時間