PostgreSQL 筆記
資料可靠性及WAL

陳彥文

Write-Ahead Logging (WAL)

- 預寫日誌
- 中心思維：改變前先記錄
  - 預先將一連串的Transaction先行寫入到Log, 來預防系統潰散時的損失
  - 固定時間或是緩衝用完時，把所有Transaction一口氣flush回硬碟
- 優勢
  - 減少硬碟的IO次數 (使用fsync()將Transaction一口氣**依序**寫入硬碟)
    - 非隨機寫入，所以性能很好
  - 確保資料完整性 (以往的方式無法保證此事)
  - 易於資料庫在運作時，也可執行備份及回復 (利用pg_dump來取得定時資料)
- PostgreSQL 7.1+

WAL 特性

- Transaction 特性 (Gray & Reuter)
  - **Atomicity (transactions are all-or-nothing)**
  - Consistency (data typing, referential integrity)
  - Isolation (concurrent execution appears serial)
  - **Durability (transactions can't come uncommitted)**
- 高度可用性 (High Availability)
  - 如果Server掛了，我們希望會有一台擁有相同資料，而且是最新資料的機器
- 時間點回復 (PITR, Point-in-Time Recovery)
  - 如果我們在 4:03am 刪除了一張Table，我們可以透過PITR回復到還沒刪除Table前的狀態 (ex: 4:02am)

Crash

- Many database actions involve modifying more than one page.
    - Example: Splitting an index page.
- Writing two data pages A and B is not atomic.
- One write must happen before the other.
    - We need to give the operating system and disk controller freedom to reorder the writes; we don't even know which one will happen first.
- We could crash in between.

- Writing one data page isn't even atomic.
- Pages are 8kB, but the disk writes in chunks as small as 512 bytes.
- We could crash in the middle (“torn page”).

Durability (持久性)

- 我們會將所有的WAL寫入到磁碟後，才告知Client，Transaction已完成
- 如果有錯誤的情況發生，所有的資料都會被復原
- 他的行為就跟一次將所有Block寫入到磁碟的效果一樣好，但是成本更便宜
  - I/O 的行為是依序的，並非隨機
  - 比起寫入整個Page，WAL的entries很小
  - Page 可能會更新許多次，但是只會寫入一次

Replication (副本)

- 可進行hot backup (不停機備份)到其他的幾次
  - 需要使用pg_start_backup/pg_stop_backip
  - 在PostgreSQL 9.1之後，提供pg_basebackup工具可使用
- 可將每個WAL entry送到網路上，並且在遠端的機器還原 (Archive Recovery)
- 每個更新將會在本地執行，也將會在遠端發生 – 兩個資料庫將會同時鎖住
- 容錯轉移(Failover)：若是Master掛了，則可轉移到其它可用機器
- 在PostgreSQL 9.0之後，你甚至可以對Slave進行”唯讀”的查詢

WAL 概念

- Log sequence number (LSN).
- WAL Segment.
- Checkpoint.
- Restartpoint.
- Full page write.
- Consistency.
- Minimum recovery point.

Log sequence number

- 我們在WAL Stream中，透過byte的位置來分辨出特定的WAL Entry
- 當資料庫Cluster第一次啟動時，啟動了WAL Stream，放置在第一個位置
- LSN為64bit的整數，所以用不完
- 每個檔案分頁，都包含著LSN，緊接著下一個WAL Entry就記錄著如何修改這個分頁
- 在將分頁寫出之前，我們必須flush所有的WAL Entry到該分頁

WAL Segment

- WAL entry所寫入的檔案，每個檔案固定在16MB
- 該檔案就被稱為WAL Segment

Checkpoints

- 儘管LSN的數值會持續往上增加，但我們並不希望永遠保存WAL
- 所以我們需要Checkpoints
- 所有修改過的buffer(又稱Dirty buffer)將會被寫入(flush)到磁碟
- 在Server啟動時，我們會需要重新載入最後一個checkpoint後所有的WAL
- 在Checkpoint之前，所產生的WAL segment都會被回收(通常是複寫，而非建立新的檔案)

Restartpoints

- Replication是持續不斷的
- 很多個WAL segments也會不停的累積
- Restarpoints基本上等同於Checkpoints，但僅發生在Recovery時
- Restartpoints只會發生在系統執行Checkpoints時
- 如果線上系統發生crash，則可以從Restartpoint繼續執行

Full Page Write

- Physical Logging
    - 不論何時改變，都會將整個page寫入
    - 簡單，但是很沒效率
- Logical Logging
    - 依序將 Insert/Update/Delete 儲存下來
    - Redo困難
- Physiological Logging
    - 儲存每一個Page所修改的地方
    - 必須依賴現存的Page，所以無法處理撕裂後的Page (Torn Page)
        - Torn Page，可能發生在I/O Error時
- 解法 —> Full Page Write
    - 在最新的checkpoint之後，使用Physical Logging將整個Page寫入
    - 之後所有的更動，則使用 Physiological Logging 來實作
        - 如果因I/O error產生Torn Page，Physical Logging則會重新覆寫整個Page

Consistency

- 無法載入WAL造成的結果
  - Torn Page
  - Index損毀
  - 部分或是所有的更動將會遺失
- 如果系統在一般運作情況下發生Crash，我們必須重現在Crash發生之前所有的WAL entry
- 但是在Archive Recovery(Replication, PITR等)，這個情況會更加複雜
  - 當開始第一次Archive Recovery，我們必須重新載入在hot backup期間，所有產生的WAL entry
  - 如果Archive Recovery因為Crash或是Restart而必須重新啟動，我們必須重新載入Restartpoint之後的WAL Entry
  - 如果我們已經載完了WAL entry，並且修改了Page X，如果Page X還沒寫入磁碟前就發生系統Crash，我們不需要再載入WAL entry了

Minimum Recovery Point

- 第一筆WAL記錄的LSN不需要被重新載入，就可以達到一致性

- 在Recovery時，我們會block寫入時，更新Minima Recovery Point

- Minimum Recovery Point儲存在Control File

- 如果寫入的block，它的LSN比Minimum Recovery Point還大，我們則會把Minimum Recovery Point指向該Block的LSN

減少WAL的overhead

- WAL bypass
- wal_level
- wal_buffers
- Checkpoint parameters
- synchronous_commit
- fsync
- Temporary tables
- Unlogged tables (in 9.1)

WAL bypass

- 如果你的系統沒有Replication …
- 而且你在這個Transaction期間，建立及移除Table …
- 那麼針對該張Table的行為將不會被WAL記錄下來
- 如果系統崩潰了，Transaction將被視為**中止**，整個**檔案**將會被移除 …事實上是部分的資料會被遺棄，無所謂

wal_level

- 預設為wal_level=minimal
- 如果要採用Warm Standby，則需設定為 wal_level=archive
  - 這個設定會取消WAL bypass
  - Warm Standby: 意指架構在第二台的Server，已經預裝好軟體，當Master掛點時，可以隨時轉移到此主機，而Master也會將資料mirror到這台機器。需要花費Recovery Time
- 如果要採用Hot Standby，則需要設 wal_level=hot_standby
  - 這個設定也會同時取消WAL bypass
  - 它同時會記錄額外的資訊，使新的機器可以透過有效的MVCC snapshot建構資料
  - Hot Standby: 意指同時有兩台Server，並且做及時備份，當一台掛點時，另外一台可以及時on起來

wal_buffers

- 在PostgreSQL 9.0以上，wal_buffers預設為64K
- 這個數字太小，容易引發頻繁的flushing
- 從PostgreSQL 9.1開始，預設為3%的shared_buffer，最大為16MB，儘管不大，但是已經比原始的預設值好上許多

配置 - Checkpoints

- Checkpoints (檢查點)
  - 每固定時間的Transaction中會插入的點
  - 在這個點之前的Transaction都會保證已寫入磁碟
  - 若寫入失敗，則會roll-back回上一個checkpoint
  - 檢查點越頻繁，資料越安全，但是效能越差。反之亦然
- 相關參數
  - checkpoint_segments
    - 設置在LogFile中，多少個Segments要放一個檢查點 (每個Segments預設16mb)。預設為3個Segments
    - 3可能太小，建議可達30
  - checkpoint_timeout
    - 設置幾分鐘做一次檢查點，預設為5分鐘
    - 5min是合理的數值，但是若能更高，則可能可以更減少I/O的消耗

配置 - Checkpoints

- checkpoint_completion_target (0~1, Float)
  - 執行檢查點完成所需的時間，計算方式為 checkpoint_timeout * checkpoint_completion_target
  - 執行下一次檢查點之前，必須完成上一個檢查點，所以設置這個數值相當需要經驗
  - 預設值為0.5，建議可以提升到0.9
- checkpoint_warning
  - 如果兩次檢查點中的間隔時間(完成->開始)小於這個數值，則在Log中出現警示訊息

重點實作機制

- 內部使用XLogInsert及XLogFlush來達成整個WAL
- XLogInsert
  - 當一筆新的資料進來，系統會使用XLogInsert在WAL Buffer配給空間，再將資料寫入，XLogInsert影響到資料更動的整體性能
    - 引響性能因素
      - lock-row等排他性問題
      - 建立新的Segment區段
- XLogFlush
  - 如果WAL buffer已滿，則採用XLogFlush來將資料flush進硬碟
- 可使用pg_test_fsync來測試寫入性能

配置 - WAL參數

- full_page_writes
    - 可使每個Segment寫入磁碟時，是以Page的方式，以防寫入過程中如果發生錯誤，可能導致DB損毀 (寫入中發生錯誤，可以Page的方式直接回復到之前的狀態)
- wal_buffers
    - 在共享記憶體內，WAL緩衝區的大小
    - 加大這個數值可以減緩full_page_write使系統太忙碌時，仍然可以確保回應給client的效能
    - 預設為 -1，系統會自動調整為共享緩衝區的3% (但介於64k ~ 16M)
- commit_delay
    - 定義了當XLogFlush之前，Committer必須停工多少時間
    - 此項設定是為了避免Commit不完全所設立
    - 若沒有打開fsync或是當前連線數少於commit_siblings時，則不會停工
    - 預設為0
- commit_siblings
    - 在執行commit_dalay之前，最少的transaction請求數量

配置 - WAL參數

- wal_sync_method
  - 決定PostgreSQL以什麼樣子的形式讓系統將Transaction寫入磁碟
    - open_datasync (write WAL files with open() option O_DSYNC)
    - fdatasync (call fdatasync() at each commit)
    - fsync (call fsync() at each commit)
    - fsync_writethrough (call fsync() at each commit, forcing write-through of any disk write cache)
    - open_sync (write WAL files with open() option O_SYNC)
  - 一般採用系統預設，官方文件建議使用pg_test_fsync來測試寫入性能之後，在決定採用的模式
- wal_debug
  - 每個XLogInsert及XFLogFlush的行為將寫入到Server Log內
  - 未來將提供更好的機制來取代這個功能

配置 - Archive

- Archive (打包, 又名歸檔)
    - 由於WAL相當肥大，所以可將使用過的Log打包起來節省空間
    - 也同時可作為備份歸檔用途
    - 利用Archive做熱備援可參考： http://kaien.kaienroid.com/blog/?p=334
- 相關參數
    - archive_mode (Boolean)
        - 是否啟動打包模式
    - archive_command (String)
        - 使用系統指令進行打包
        - Any %p in the string is replaced by the path name of the file to archive, and any %f is replaced by only the file name. (The path name is relative to the working directory of the server, i.e., the cluster's data directory.) Use %% to embed an actual % character in the command. It is important for the command to return a zero exit status only if it succeeds.
    - archive_timeout
        - 強制Server在固定時間進行打包

Asynchronous Commit

- 非同步認可模式
- 用於多Database Server架構
  - 同步Commit會即時的將所有Transaction寫入底下所有子Server，才會返回給Client (**預設**)
  - 非同步Commit則不保證所有Transaction都會即時更新到子Server，改採固定時間進行同步
- 優勢：在非同步化的模式底下，可擁有更好的寫入效能
- **注意**：並非所有情形都適用非同步化Commit，部分指令也不支援非同步

配置

- synchronous_commit
  - 關閉這個選項，會使當Transaction結束時，flush的行為將會在背景實現，而非前景
    - 如果你的系統在flush進磁碟前就crash，可能會導致資料的流失
- fsync
  - 關閉這個選項將會使系統不再注意資料寫入是否正確
  - 如果你的系統crash了，祝你好運

Temporary & Unlogged Tables

- WAL不需要暫存的Table
- 如果系統崩潰，我們只需要把損毀的資料移除，如果資料正確則不需要在意
- Backend-local.
- 在PostgreSQL 9.1，你可以建立一個Unlogged Table，這張Table不會有任何的WAL logged，如果正常關閉Server，則這張Table會存活，如果系統Crash，這張Table則會被刪除

参考資料

- Introduction to Write-Ahead Logging by Robert Haas, EnterpriseDB company.

- PostgreSQL 9.3 manual