redis 很黃很暴力 v2

by

Triton Ho 

內容大綱

- 前言
- caching 常犯錯誤
- 很黃很暴力的延伸應用
- 把 Redis 當成簡陋版 lock server
- 案例 1 ： ratelimiting
- 案例 2 ： data snapshotting

前言

前言

- 吃飯定理：
  - 便宜，好吃，不用排隊
  - 正常的餐廳最多只能滿足二項
  - 同時滿足三項者，最終一定虧本倒閉

在單一台機器時

- Persistence
  - 在當機後不會引發資料流失
- Low latency
  - 資料庫能用極短時間完成單一工作
- 以上二者你最多只能要一個
- Redis 是追求 Low latency ，想單用 Redis 做到 zero data loss ，跟在沙漠中吃生魚片一樣

Redis 的取向.....

- Redis 的預設，是每 10000 個 WRITE 才會寫入 Harddisk 的
  - 你可以改掉這設定，但是效能會大跌
- 如果 Redis 當掉，一定會有 data loss 的
- 所以，正常人使用 Redis……
  - 用作 caching ，資料同時存放於主資料庫
  - 儲存沒了也死不了人的 Hot Data

Single Key Consistency

- 不同的 KeyValue 會放到不同的 redis 機器
  - 除非你只跑 single-node
- 別在 lua script 內跑這種笨事
  - MGET Key1 Key2 Key3
- 要使用 Optimistic lock 時，無可避免要用上 Hash
  - Data 來存資料
  - LastUpdateTs 來存最後改動時間

Redis 是 Single thread

- 在 redis cluster ，一份資料只存放在一個 node 上
- 每一個 node 都是 single thread 的
- 好處
  - Redis 內部架構簡潔高效能，不用煩 ReadWrite Mutex
- 壞處
  - 跟 node.JS 一樣，只要一個工作要跑比較久，該 node 的 latency 立即衝破宇宙
  - 容易發生 hotspot ，沒法以增加硬體手段解決

caching 常犯錯誤

caching 常犯錯誤

- 只用 local cache
- 只用「一般」的 caching
- 沒使用 consistency hash
- 沒對 hot data 預熱
- 沒設定合理的 TTL

只用 Local cache

- local cache 是指 application server（簡稱ＡＳ）上的 local memory
  - 例子：很多 ORM 都有 caching 功能
- 缺點：
  - 改動沒法反映到全部的伺服器上，所以有機會用上舊的版本
  - ＡＳ的 local cache 不能隨流量加大，系統流量越大， cache miss 可能性便越高
  - 新開的ＡＳ其 local cache 是全空的

「一般」的 caching

- 這是一般的 caching 方法：
  1. 從 Redis 拿資料 X；如有則直接回傳
  2. 從主資料拿資料 X
  3. 把資料 X 放回 Redis
  4. 把資料 X 回傳
- 看起來很正常，很合理吧 ~
  - 結果我跟老婆約會時，要拿筆電來救火
  - ~~小聲：其實結婚當天也因為別的問題救火~~

「一般」的 caching 問題

- 如果資料 X 的 QPS 是 1000……
  - 如果主資料庫需要 80ms 回傳資料 X，那麼便有 80 個 request 進了主資料庫
  - 如果資料 X 需要複雜運算，需要 8000ms，那便有 8000 個 request 進了主資料庫
- 後者，你資料庫需要的 total CPU time =
  - 8 (sec per request) * 8000 request = 64000 sec
- 現實上，你的資料庫已經死了

高流量下的 caching

- 這是進階的 caching 方法：
  1. 從 Redis 拿資料 X；如有則直接回傳
  2. 拿到資料 X 的鎖（在離開時釋放）
  3. 再次從 Redis 拿資料 X；如有則直接回傳
  4. 從主資料拿資料 X
  5. 把資料 X 放回 Redis
  6. 把資料 X 回傳

沒使用 consistency hash

- 別使用 mod(md5(cacheKey), n) 來決定某一 keyValue 位置
  - n = 你的 redis server 總數
  - 嘛，都 2018 年了
- 用這方法，當你系統繁忙要加開 redis 時， n 的改動會讓你的 caching 全滅
- 請學習 consistent hash
  - 其實 redis cluster 內建了

caching 預熱

- 如果你的網頁首頁，某一排名榜需要 5 秒才能生產出來.....
- 一旦 cache miss ，大堆人便要等待這一份資料
  - 不管你的 locking 是用 Zookeeper 還是 redis ，還是吃了大量效能
  - 老闆看到網頁慢了，會找你談話.....
- 土法煉鋼但是有效的做法：
  - 寫一個 crontab ，在 cache miss 前到主資料庫拿資料放到 redis

沒設定合理的 TTL

- 有笨蛋會設 cache TTL = 一年
  - 不設 TTL 後果更嚴重，唉~
- 這種瘋狂的 TTL ，最終會讓 redis 存太多過期資料，觸發 cache eviction
- cache eviction 問題
  - 是在 peak hours 的寫入觸發
  - 必須先做 eviction 清出空間才能寫入，引發超高 latency
  - 有機會清掉近期資料（ Redis 不是純 LRU algorithm ）

把 Redis 當成簡陋版 lock server

用作 locking server

- 專業應用請用 Zookeeper / etcd
  - 沒有 blocking
- 指令：
  - SET <lockName> <pseudo-threadId> NX EX <lockTime>
- 如果不成功，則用 for-loop 重試
  - 關鍵字： exponential backoff ， jitter

Anti-pattern: Barrier

- 某人想每 5 分鐘就 refresh 一次 cacheX ，所以他寫了 crontab ，每一分鐘醒來一次
  - SET “BarrierX” <anything> NX EX 300sec
  - 如果 SETNX 結果是 0 ，則不工作直接結束
  - 否則，從主資料庫拿最近的資料並且寫到 cacheX
- 然後： Barrier pattern 用在 Redis 上會害你失掉 system robustness

Barrier vs Lock

- 雖然二者都是用 SETNX ，但是目的完全不同
- SETNX 回答 0 時
  - Barrier 是直接 return
  - Lock 是 sleeping 再重試
- SETNX 的 TTL
  - Barrier 是長時間的（資料更新的隔距）
  - Lock 是短時間的（ critical zone 的最大執行時間）

案例分享

案例 1 ： ratelimiting

- 案例分享：現在有 C100K QPS 想購買某商品，但是系統每一商品只能有 100 QPS 處理能力
- nginx 的 ratelimiting 是以 API 為單位的
  - 這其實對一般系統很夠用了，沒事別作死
- ratelimiting 常用算法： token bucket
  - 以 hash 存 remainingTokenCount 和 lastUpdateTime

善用 local buffering / caching

- 如果每一 Request 都要詢問 redis 一次， C50K 下系統必亡
- 如果 Application Server 每次不是拿一個 token ，而是 5 個
  - 剩下的 4 個 token 留給未來 request 使用
  - 那麼 redis QPS 便會變成 1 / 5
- 如果 Application Server 一個 token 也拿不到，便乾脆拒絕所有同類 Request 1 秒
- 如果你系統有 80 台伺服
  - 你的 redis max QPS 便是 80 + (100 / 5)
  - 跟系統流量沒關係

案例 2 ： data snapshotting

- 想像一下：你在看一個持續改動中的排名榜
- 如果沒有 snapshotting ，剛好第十名的物品變第十一，然後你拿 Page 2 的資料
  - 你就會看到重覆的物品
- RESTful endpoint:
  - GET /v1/leaderboard?ts=1561042015

Snapshotting with redis v1

- 建立 crontab ，每一分鐘從資料庫拿建立新的 snapshot 丟到 redis
- 在 redis 中：

  DataXXX-20190620:190000
  DataXXX-20190620:190100
  DataXXX-20190620:190200

- DataXXX-<ts> 是一個 hashSet
  - P1 是對應第 1 – 50 個 items
  - P2 是對應第 51 – 51 個 items

Snapshotting with redis v2

- 如果資料長期沒有改動， v1 只會建立大量重覆的 snapshots
- 如果一份 snapshots 跟之前相同，它只需把該 ts 存起來，而不用存相同的資料
  - 讀取該份 snapshot 時，先看看有沒有 ts field ；如有，就拿 ts 所指向的 snapshot
- Solution 2a
  - 建立 snapshot 時，跟之前一份先做對比
- Solution 2b
  - 改動資料時，把最近的 snapshot make as dirty

延伸： localcache on snapshot

- 一般的 localcache ，很可能是設定 500ms TTL
  - localcache 沒法知道別的 Application Server 改動
- snapshot 是永遠不會改動的資料
- 所以，其 localcache TTL 應該跟 Redis 相同的

End