redis 很黃很暴力（2018 版）

by

Triton Ho 

內容大綱

- 前言
- caching 常犯錯誤
- 很黃很暴力的延伸應用

  高流量下 redis 的奇技淫巧

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

ＣＡＰ定理

- Consistency
  - 資料一致性，多人試圖改動同一份資料是，是否會發生 race condition
    - RDBMS: multi-row multi-record consistency
    - HBase: single-row single-record consistency
    - Cassandra: No consistency
- Availability
  - 你的資料庫是否高可用
- Partition tolerance
  - 你的資料庫是否容許 multi-master

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

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

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 100ms 回傳資料 X ，那麼便有 100 個 request 進了主資料庫
  - 如果資料 X 需要複雜運算，需要 5000ms ，那便有 5000 個 request 進了主資料庫
- 後者，你的資料庫需要 25000 秒的 CPU time 來完成
  - 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

很黃很暴力的延伸應用

很黃很暴力的延伸應用

- 用作 locking server
- 用作 MQ server

用作 locking server

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

MQ server

- 專業應用請用 RabbitMQ
  - Redis 沒法做到 blocking
  - 沒法保證工作（成功跑了一次＋單一執行緒）
  - 更沒法保障 data loss
- 最簡單的版本：使用 List
  - 增加工作用 RPUSH
  - 拿工作出來用 LINDEX 然後 LPOP
- 進階版本：用 Set
  - 能讓高優先工作先執行
  - 增加工作用 SADD，
  - 拿工作出來用 SRANDMEMBER 然後 SREM

高流量下 redis 的奇技淫巧

高流量下前言

- ~~不管這世界有多少億正妹，老婆只能有一個~~
- 不管你的 redis cluster 有多少個 node ，一份資料只能存於一台 redis 上
- 如果這份資料有 C100K 的流量，存了那份資料的 redis 必當～
  - 我們實戰經驗，單台的 redis 大約能支持 C10K

高流量下 redis 的奇技淫巧

- caching 預熱
- 善用 local buffering / caching
- 善用 data sharding

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

案例分享

- 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１０個
  - 剩下的 9 個 token 留給未來 request 使用
  - 那麼 redis QPS 便會變成１／１０
- 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 / 10)
  - 跟系統流量沒關係

data sharding 案例

- 現在系統有一個排名榜，要顯示最高分的 10000 個物品
  - 物品的分數只會升不會跌的
- 物品改動的 QPS 很高
- Read QPS 也很高

data sharding 案例（續）

- 用 Redis Sorted Set 來存首一萬個物品分數，會讓單一台 redis CPU 太高
- 一台支持不了 Write ，就改用１６台啊～
- 我們改用１６個 hash
  - 每一個 hash 只負責 mod (itemId, 16) == n 的物品
  - 每個 hash 只存首一萬個物品
- 要得到首 10000 個物品，只需要把 16 個 hash 都拿出來，然後再跑一次 sorting 便好

End