Redis, another step on the road

2015-05-15

Ant
yftzeng@gmail.com

Agenda

- ✔ Redis history
- ✔ Redis 3.0
- ✔ Redis features
- ✔ Redis and Memcached
- ✔ Redis and Aerospike
- ✔ Insight on the pit

議程

- ✔ Redis 歷史
- ✔ Redis 3.0
- ✔ Redis 特性
- ✔ Redis 與 Memcached
- ✔ Redis 與 Aerospike
- ✔ 坑裡的洞見

Agenda

- ✔ Redis history
- ✔ Redis 3.0
- ✔ Redis features
- ✔ Redis and Memcached
- ✔ Redis and Aerospike
- ✔ Insight on the pit

議程

- ✔ Redis 歷史
- ✔ Redis 3.0
- ✔ Redis 特性
- ✔ Redis 與 Memcached
- ✔ Redis 與 Aerospike
- ✔ 坑裡的洞見

Redis history

Redis 2.0

✔ Key-value store policy, key in memory, value in disk.

Redis 歷史

Redis 2.0

✔ Key-value 儲存策略， key 存記憶體， value 存硬碟。

Redis history

Redis 2.4

- ✔ Key-value store policy, both key & vlaue are in memory.
- ✔ Add 2 background threads,
  - ✔ fsync file descriptor.
  - ✔ close file descriptor.

Redis 歷史

Redis 2.4

- ✔ Key-value 儲存策略， key 及 value 存記憶體。
- ✔ 除了 main thread 外，引入了 2 個 background threads,
  - ✔ fsync file descriptor 。
  - ✔ close file descriptor 。

Redis history

Redis 2.6

- ✔ Server side Lua scripting.
- ✔ Virtual Memory removed.
- ✔ Milliseconds resolution expires.
- ✔ Removed hardcoded number of clients.

Redis 歷史

Redis 2.6

- ✔ 伺服器端支援 Lua 。
- ✔ 移除 Virtual Memory 。
- ✔ 「 Expires 」的毫秒精準度。
- ✔ 移除寫死的客戶端數量限制。

Redis history

Redis 2.8

- ✔ Saving synchronization resource.
  - ✔ Before 2.8, slave use SYNC with the master.
  - ✔ After 2.8, slave use PSYNC with the master.

Redis 歷史

Redis 2.8

- ✔ 節省同步資源。
  - ✔ 2.8 以前， slave 使用 SYNC 與 master 同步。
  - ✔ 2.8 以後，改用 PSYNC（偏移量同步）與 master 同步。

Agenda

- ✔ Redis history
- ✔ Redis 3.0
- ✔ Redis features
- ✔ Redis and Memcached
- ✔ Redis and Aerospike
- ✔ Insight on the pit

議程

- ✔ Redis 歷史
- ✔ Redis 3.0
- ✔ Redis 特性
- ✔ Redis 與 Memcached
- ✔ Redis 與 Aerospike
- ✔ 坑裡的洞見

Redis 3.0

Release date: 1 Apr 2015.

- ✔ Redis Cluster.
- ✔ New "embedded string" object.
- ✔ Improved LRU approximation algorithm.

Redis 3.0

2015 年 4 月 1 日正式釋出。

- ✔ Redis Cluster 。
- ✔ 新的 “ embedded string” 。
- ✔ LRU 演算法的改進。

Agenda

- ✔ Redis history
- ✔ Redis 3.0
- ✔ Redis features
- ✔ Redis and Memcached
- ✔ Redis and Aerospike
- ✔ Insight on the pit

議程

- ✔ Redis 歷史
- ✔ Redis 3.0
- ✔ Redis 特性
- ✔ Redis 與 Memcached
- ✔ Redis 與 Aerospike
- ✔ 坑裡的洞見

Redis features

- ✔ RDBMS
  - ✔ Oracle, DB2, PostgreSQL, MySQL, SQL Server, ...
- ✔ NoSQL
  - ✔ Cassandra, HBase, Memcached, MongoDB, Redis, ...
- ✔ NewSQL
  - ✔ Aerospike, FoundationDB, RethinkDB, ...

Redis 特性

- ✔ RDBMS
  - ✔ Oracle, DB2, PostgreSQL, MySQL, SQL Server, ...
- ✔ NoSQL
  - ✔ Cassandra, HBase, Memcached, MongoDB, Redis, ...
- ✔ NewSQL
  - ✔ Aerospike, FoundationDB, RethinkDB, ...

Redis features

- ✔ Key-value NoSQL
  - ✔ Memcached, Redis, ...
- ✔ Column family NoSQL
  - ✔ Cassandra, HBase, ...
- ✔ Documen NoSQL
  - ✔ MongoDB, ...
- ✔ Graph NoSQL
  - ✔ Neo4j, ...

Redis 特性

- ✔ Key-value NoSQL
  - ✔ Memcached, Redis, ...
- ✔ Column family NoSQL
  - ✔ Cassandra, HBase, ...
- ✔ Documen NoSQL
  - ✔ MongoDB, ...
- ✔ Graph NoSQL
  - ✔ Neo4j, ...

Redis features

Pure

- ✔ ANSI C.
- ✔ Lesser 3rd-party libraries.
  - ✔ Memcached depends on libevent.
  - ✔ Redis implement its own epoll event loop.
- ✔ KISS principle.
  - ✔ Data structure do what it should do.

Redis 特性

簡純

- ✔ ANSI C 撰寫。
- ✔ 幾乎不依賴第三方函式庫。
  - ✔ memcached 使用 libevent ，程式碼龐大。
  - ✔ Redis 參考 libevent 實現了自己的 epoll event loop 。
- ✔ KISS 原則。
  - ✔ 每個數據結構只負責自己應當做的。

Redis features

Simple

- ✔ No map-reduce.
- ✔ No indexes.
- ✔ No vector clocks.

Redis 特性

簡單

- ✔ No map-reduce.
- ✔ No indexes.
- ✔ No vector clocks.

Redis features

5 - We're against complexity. We believe designing systems is a fight against complexity. We'll accept to fight the complexity when it's worthwhile but we'll try hard to recognize when a small feature is not worth 1000s of lines of code. Most of the time the best way to fight complexity is by not creating it at all.

Ref: http://oldblog.antirez.com/post/redis-manifesto.html

Redis features

Single thread

✔ No thread context switch.
✔ No thread race condition.
✔ No other complicated condition.

Redis 特性

單執行緒

✔ No thread context switch.
✔ No thread race condition.
✔ No other complicated condition.

Redis features

In-memory but persistent on disk database

- ✔ Operation in memory.
- ✔ Persistent on disk.

Redis 特性

記憶體資料庫，但可永久儲存於硬碟中

- ✔ 記憶體操作資料。
- ✔ 資料可永久儲存於硬碟。

Redis features

Remote dictionary server

✔ No only a cache server.

Redis 特性

Remote dictionary server

✔ 不只是快取伺服器。

Ref: http://redis.io/topics/faq

Agenda

- ✔ Redis history
- ✔ Redis 3.0
- ✔ Redis features
- ✔ Redis and Memcached
- ✔ Redis and Aerospike
- ✔ Insight on the pit

議程

- ✔ Redis 歷史
- ✔ Redis 3.0
- ✔ Redis 特性
- ✔ Redis 與 Memcached
- ✔ Redis 與 Aerospike
- ✔ 坑裡的洞見

Redis and Memcached

- ✔ Redis is single thread IO multiplexing model.
  - ✔ Simple operations to archive high throughput.
  - ✔ Complicated (heavy) operations may block others.
  - ✔ One instance usually only use one CPU.

Redis 與 Memcached

- ✔ Redis 是單執行緒 IO 多路複用模式。
  - ✔ 簡單的操作可以達到高吞吐。
  - ✔ 複雜的操作容易阻塞其它的操作。
  - ✔ 一個 Redis 實例通常只會用到一顆 CPU 。

Redis and Memcached

- ✔ Memcached is multi-threaded, non-blocking IO multiplexing network model.
  - ✔ Multi-core architecture.
  - ✔ But got cache coherency & lock issues.

Redis 與 Memcached

- ✔ Memcached 是多執行緒非阻塞 IO 多路複用模式。
  - ✔ 多執行緒可善用多顆 CPU 。
  - ✔ 但會引入 cache coherency 及 lock 問題。

Redis and Memcached

- ✔ Redis can use jemalloc or tcmalloc to reduce memory fragmentation.
  - ✔ But it depends on the allocation patterns.
- ✔ Rarely use the Free-list and other ways to optimize memory allocation.
  - ✔ Redis is simple / pure / efficiency design.

Redis 與 Memcached

- ✔ Redis 使用 jemalloc 或 tcmalloc 降低記憶體碎片。
  - ✔ 但記憶體碎片的情形仍依賴於分配模式。
- ✔ 幾乎不用 Free-list 及其它方法來最佳化記憶體分配。
  - ✔ 符合 Redis 簡單 / 單純 / 效率的設計原則。

Ref: http://www.databaseskill.com/1256096/
Ref: http://stackoverflow.com/questions/18097670/why-the-memory-fragmentation-is-less-than-1-in-redis

Redis and Memcached

- ✔ Memcached use pre-allocated / slot memory pool.
  - ✔ slot and pool can reduce memory fragmentation.
  - ✔ But bring some wasted space. (memory overhead)

Redis 與 Memcached

- ✔ Memcached 使用預分配 slot 記憶體池。
  - ✔ slot 及池能有效降低某種程度的記憶體碎片。
  - ✔ 但會帶來一些空間浪費的問題。 (memory overhead)

Redis and Memcached

- ✔ Garbage Collection behavior: approximate LRU.
  - ✔ Redis 2.6
    - ✔ Random pick 3 samples, removed the oldest one, repeatedly until memory used less than 'maxmemory' limit.

Redis 與 Memcached

- ✔ 垃圾回收行為：近似 LRU 演算法。
  - ✔ Redis 2.6
    - ✔ 預設隨機取 3 個樣本，移除最舊的該筆，如此反覆，直到記憶體用量小於 maxmemory 的設定。

Ref: https://github.com/antirez/redis/blob/2.6/src/redis.c#L2464

Redis and Memcached

```
2462
2463            /* volatile-lru and allkeys-lru policy */
2464            else if (server.maxmemory_policy == REDIS_MAXMEMORY_ALLKEYS_LRU ||
2465                server.maxmemory_policy == REDIS_MAXMEMORY_VOLATILE_LRU)
2466            {
2467                for (k = 0; k < server.maxmemory_samples; k++) {
2468                    sds thiskey;
2469                    long thisval;
2470                    robj *o;
2471
2472                    de = dictGetRandomKey(dict);
2473                    thiskey = dictGetKey(de);
2474                    /* When policy is volatile-lru we need an additional lookup
2475                     * to locate the real key, as dict is set to db->expires. */
2476                    if (server.maxmemory_policy == REDIS_MAXMEMORY_VOLATILE_LRU)
2477                        de = dictFind(db->dict, thiskey);
2478                    o = dictGetVal(de);
2479                    thisval = estimateObjectIdleTime(o);
2480
2481                    /* Higher idle time is better candidate for deletion */
2482                    if (bestkey == NULL || thisval > bestval) {
2483                        bestkey = thiskey;
2484                        bestval = thisval;
2485                    }
2486                }
2487            }
```

Ref: https://github.com/antirez/redis/blob/2.6/src/redis.c#L2464

Redis and Memcached

- ✔ Garbage Collection behavior: approximate LRU.
  - ✔ Redis 3.0
    - ✔ Default random pick 5 samples, insert/sort into a pool, remove the best one, repeatedly until memory used less than 'maxmemory' limit.
      - ✔ 5 (now) is more than 3 (before) samples ;
      - ✔ The best one is more approximate global.

Redis 與 Memcached

- ✔ 垃圾回收行為：
  - ✔ Redis 3.0
    - ✔ 預設隨機取 5 個樣本，插入並排序至一個 pool ，移除最佳者，如此反覆，直到記憶體用量小於 maxmemory 的設定。
      - ✔ 樣本 5 比先前的 3 多；
      - ✔ 從局部最優趨向全局最優。

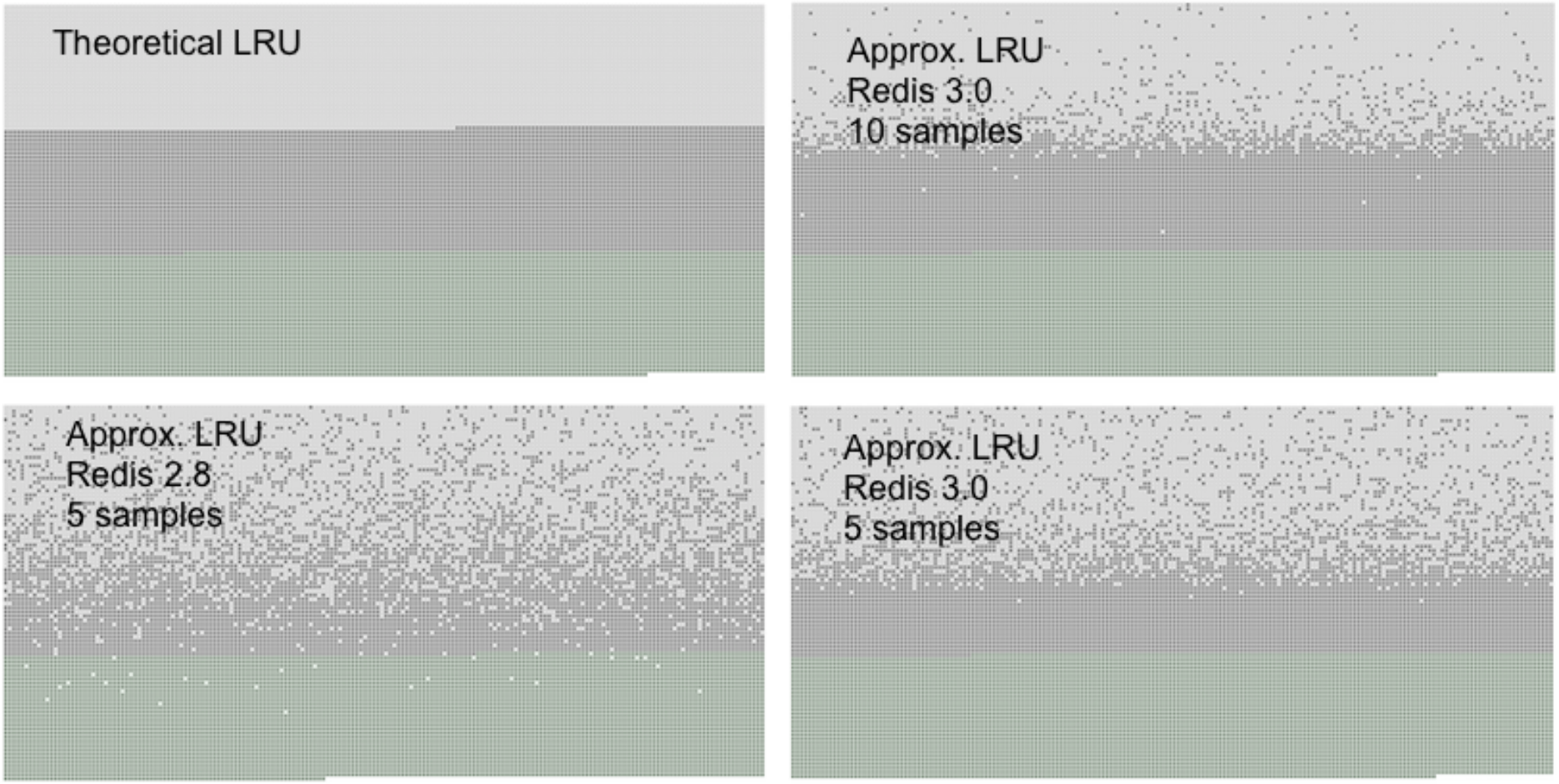
Ref: https://github.com/antirez/redis/blob/3.0/src/redis.c#L3251

Redis and Memcached

```
3250            /* volatile-lru and allkeys-lru policy */
3251            else if (server.maxmemory_policy == REDIS_MAXMEMORY_ALLKEYS_LRU ||
3252                server.maxmemory_policy == REDIS_MAXMEMORY_VOLATILE_LRU)
3253            {
3254                struct evictionPoolEntry *pool = db->eviction_pool;
3255
3256                while(bestkey == NULL) {
3257                    evictionPoolPopulate(dict, db->dict, db->eviction_pool);
3258                    /* Go backward from best to worst element to evict. */
3259                    for (k = REDIS_EVICTION_POOL_SIZE-1; k >= 0; k--) {
3260                        if (pool[k].key == NULL) continue;
3261                        de = dictFind(dict,pool[k].key);
3262
3263                        /* Remove the entry from the pool. */
3264                        sdsfree(pool[k].key);
3265                        /* Shift all elements on its right to left. */
3266                        memmove(pool+k,pool+k+1,
3267                            sizeof(pool[0])*(REDIS_EVICTION_POOL_SIZE-k-1));
3268                        /* Clear the element on the right which is empty
3269                         * since we shifted one position to the left.  */
3270                        pool[REDIS_EVICTION_POOL_SIZE-1].key = NULL;
3271                        pool[REDIS_EVICTION_POOL_SIZE-1].idle = 0;
3272
3273                        /* If the key exists, is our pick. Otherwise it is
3274                         * a ghost and we need to try the next element. */
3275                        if (de) {
3276                            bestkey = dictGetKey(de);
3277                            break;
3278                        } else {
3279                            /* Ghost... */
3280                            continue;
3281                        }
3282                    }
3283                }
3284            }
```

Ref: https://github.com/antirez/redis/blob/3.0/src/redis.c#L3251

Redis and Memcached



Ref: http://redis.io/images/redisdoc/lru_comparison.png

Redis and Memcached

- ✔ Strong sides of Redis.
  - ✔ Rich (data type) operations.
    - ✔ Hashs, Lists, Sets, Sorted Sets, HyperLogLog etc.
  - ✔ Bulit-in replication & cluster.
  - ✔ in-place update operations.
  - ✔ Support persistent on disk.
    - ✔ Avoid thundering herd.

Redis 與 Memcached

- ✔ Redis 的長處。
  - ✔ 豐富的 (資料型態) 操作。
    - ✔ Hashs, Lists, Sets, Sorted Sets, HyperLogLog 等。
  - ✔ 內建 replication 及 cluster 。
  - ✔ 就地更新 (in-place update) 操作。
  - ✔ 支援持久化 (硬碟) 。
    - ✔ 避免雪崩效應。

Redis and Memcached

- ✔ Strong sides of Memcached.
  - ✔ Multi-threaded.
    - ✔ Use almost all CPUs.
    - ✔ Fewer blocking operations.
      (And center locks don't scaled up to 5 threads)
  - ✔ Lower memory overhead.
  - ✔ Lower memory allocation pressure.
  - ✔ Maybe less memory fragmentation.

Redis 與 Memcached

- ✔ Memcached 的長處。
  - ✔ 多執行緒。
    - ✔ 善用多核 CPU 。（而 center locks 不隨 CPU 擴展）
    - ✔ 更少的阻塞操作。
  - ✔ 更少的記憶體開銷。
  - ✔ 更少的記憶體分配壓力。
  - ✔ 可能有更少的記憶體碎片。

Ref: https://github.com/memcached/memcached/blob/master/thread.c#L747

Redis and Memcached

```c
/*
 * Initializes the thread subsystem, creating various worker threads.
 *
 * nthreads  Number of worker event handler threads to spawn
 * main_base Event base for main thread
 */
void memcached_thread_init(int nthreads, struct event_base *main_base) {
    int         i;
    int         power;

    for (i = 0; i < POWER_LARGEST; i++) {
        pthread_mutex_init(&lru_locks[i], NULL);
    }
    pthread_mutex_init(&worker_hang_lock, NULL);

    pthread_mutex_init(&init_lock, NULL);
    pthread_cond_init(&init_cond, NULL);

    pthread_mutex_init(&cqi_freelist_lock, NULL);
    cqi_freelist = NULL;

    /* Want a wide lock table, but don't waste memory */
    if (nthreads < 3) {
        power = 10;
    } else if (nthreads < 4) {
        power = 11;
    } else if (nthreads < 5) {
        power = 12;
    } else {
        /* 8192 buckets, and central locks don't scale much past 5 threads */
        power = 13;
    }
```

Ref: https://github.com/memcached/memcached/blob/master/thread.c#L747

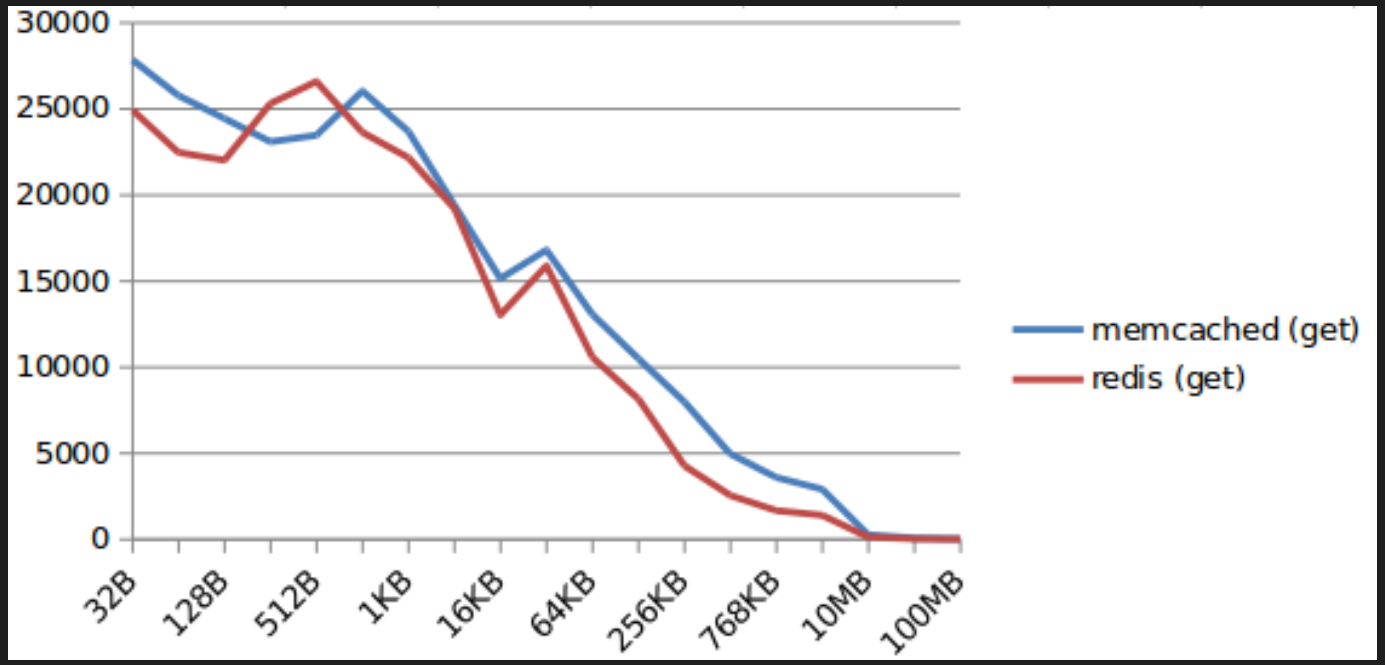
Redis and Memcached

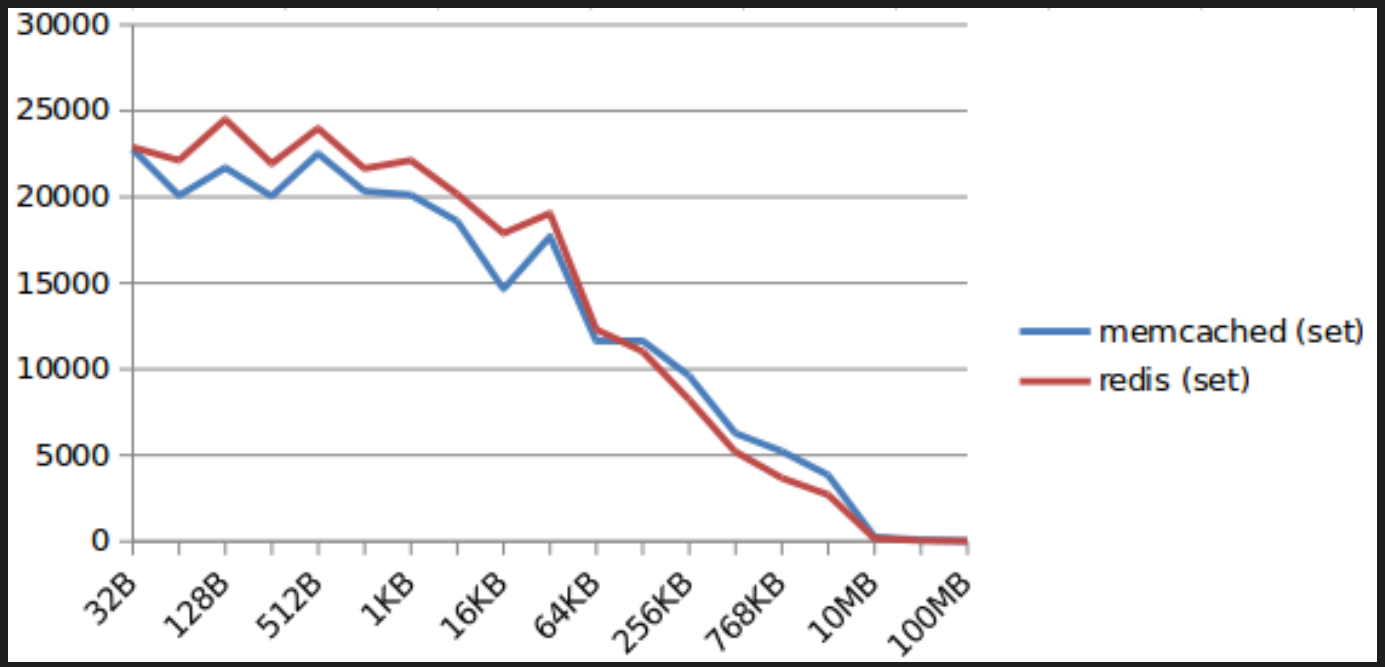
- ✔ My testbed (NO WARRANTY)
  - ✔ Get: Memcached is usually faster than Redis.
  - ✔ Set: Redis is usually faster than Memcached.
  - ✔ Size from 0 ~ 100KB is better for Redis.
  - ✔ Size from 100KB ~ 10MB is better for Memcached.
  - ✔ Size from 10M ~ is better for Redis.

Redis 與 Memcached

- ✔ 我的使用經驗（免責聲明）
  - ✔ Get 時， Memcached 比 Redis 快。
  - ✔ Set 時， Redis 比 Memcached 快。
  - ✔ 數據 0~100KB 時，適合 Redis。
  - ✔ 數據 100KB~10MB 時，適合 Memcached。
  - ✔ 數據 10M 以上時，適合 Redis。

Redis and Memcached





Agenda

- ✔ Redis history
- ✔ Redis 3.0
- ✔ Redis features
- ✔ Redis and Memcached
- ✔ Redis and Aerospike
- ✔ Insight on the pit

議程

- ✔ Redis 歷史
- ✔ Redis 3.0
- ✔ Redis 特性
- ✔ Redis 與 Memcached
- ✔ Redis 與 Aerospike
- ✔ 坑裡的洞見

Redis and Aerospike

AEROSPIKE

- ✔ Speed
- ✔ Scalable
- ✔ Flash-optimized
- ✔ In-memory NoSQL
- ✔ ACID Compliant

Ref: http://www.aerospike.com/

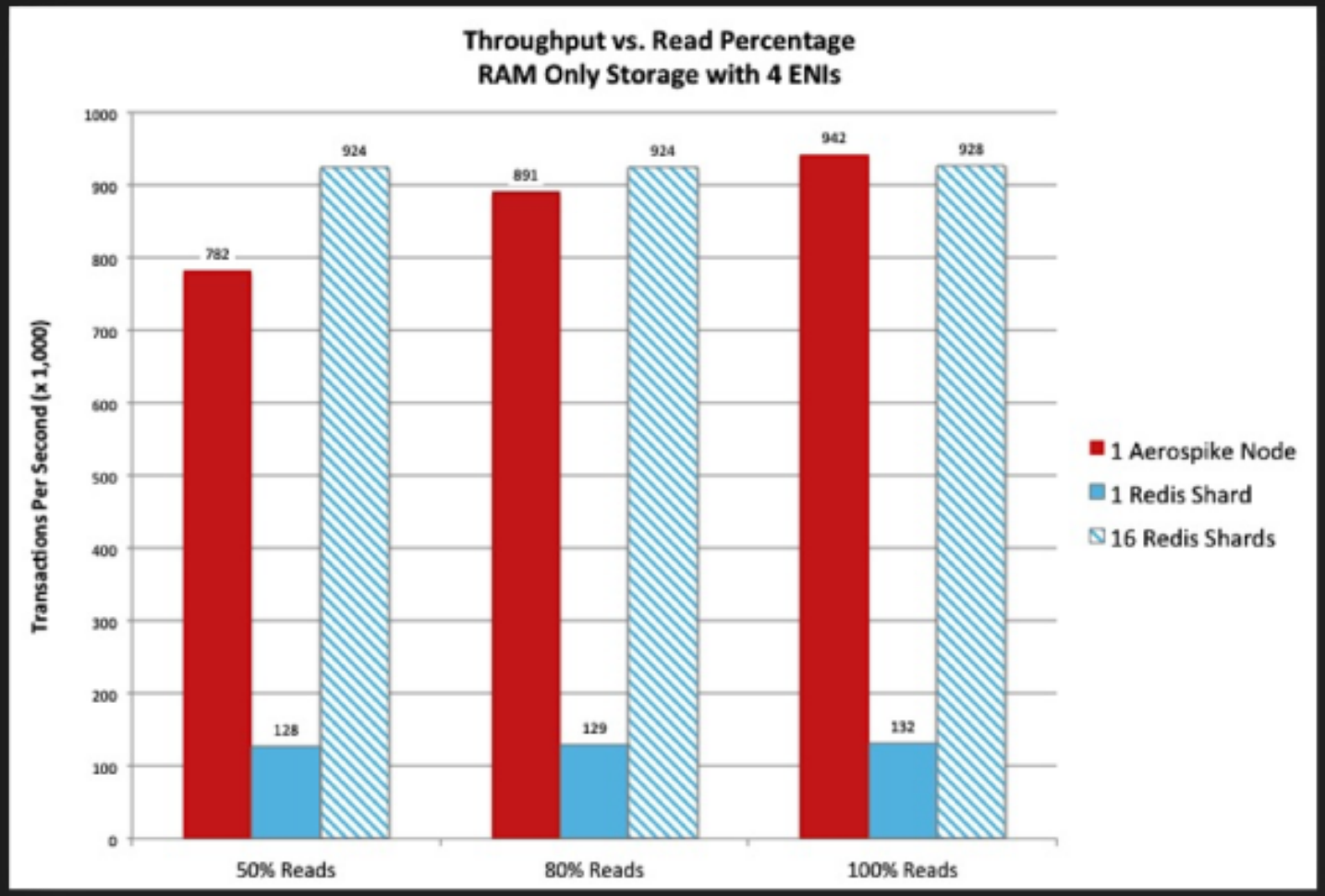
Redis and Aerospike

- ✔ Strong sides of Aerospike.
  - ✔ Auto node discovery (cluster).
  - ✔ ACID Compliant
  - ✔ Flash-optimized (Memory & Disk persistence).
  - ✔ Intelligent Client (Optimistic row locking etc.).
  - ✔ Cross data center replication.
  - ✔ Multi-core optimization.
  - ✔ No hotspots.

Redis 與 Aerospike

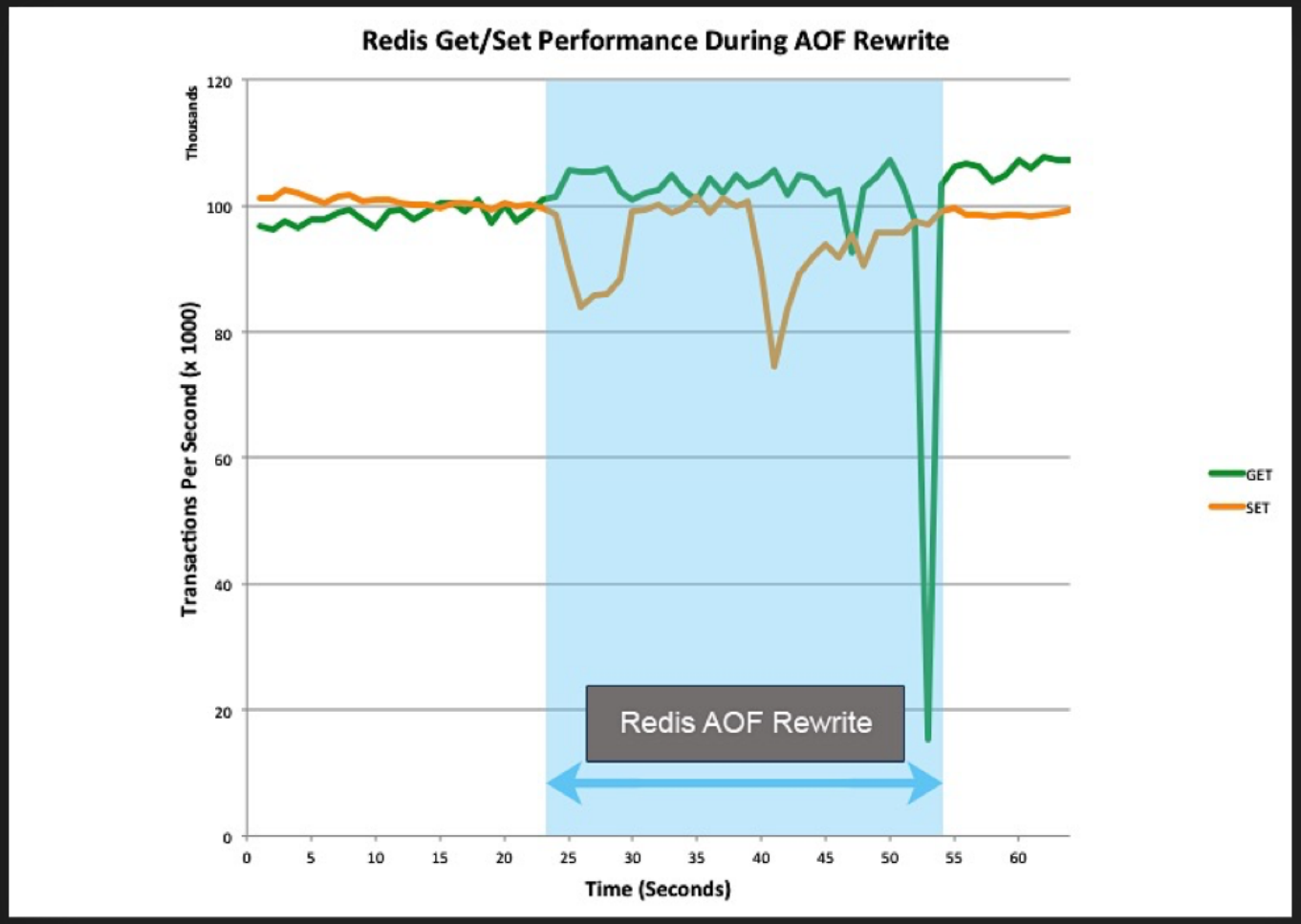
- ✔ Aerospike 的長處。
  - ✔ 自管理集群。
  - ✔ ACID 兼容。
  - ✔ 數據存儲最佳化 (Flash/SSD) 。
  - ✔ 智能客戶端 (Optimistic row locking 等) 。
  - ✔ 跨數據中心集群。
  - ✔ 多核最佳化。
  - ✔ 無熱點瓶頸。

Redis and Aerospike



Ref: http://lynnlangit.com/2015/01/28/lessons-learned-benchmarking-nosql-on-the-aws-cloud-aerospikedb-and-redis/

Redis and Aerospike



Ref: http://lynnlangit.com/2015/01/28/lessons-learned-benchmarking-nosql-on-the-aws-cloud-aerospikedb-and-redis/

Redis and Aerospike

- ✔ Itamar Haber (Redis Labs, Chief Developers Advocate)
  - ✔ Why didn't ... use ... pipelining and multi-key operations?
  - ✔ Missing piece is a 20%-80% read/write test and a 100% write test.
  - ✔ Totally unexplained by the fact that she used AOF.
  - ✔ Comparisons are as hard to do right as they are easy to do wrong.

Redis 與 Aerospike

- ✔ Itamar Haber (Redis Labs 公司的首席開發者推廣師)
  - ✔ 為什麼不用 Redis 推薦做法，如使用 piplining 和多鍵操作。
  - ✔ 沒有測試工作負載： 20%-80% 讀寫和 100% 寫的情境。
  - ✔ 對於 AOF ，一般都是建議非主 Redis 實例執行。
  - ✔ 最後，比較是一件很難做對卻很容易做錯的事。

Ref: https://redislabs.com/blog/the-lessons-missing-from-benchmarking-nosql-on-the-aws-cloud-aerospikedb-and-redis

Redis and Aerospike

- ✔ Salvatore Sanfilippo (antirez, the author of Redis)
  - ✔ GET/SET Benchmarks are not a great way to compare different database systems.
  - ✔ A better performance comparison is by use case.
  - ✔ Test with instance types most people are going to actually use, huge instance types can mask inefficiencies of certain database systems, and is anyway not what most people are going to use.

Redis 與 Aerospike

- ✔ Salvatore Sanfilippo (antirez, Redis 作者)
  - ✔ GET/SET 不能比較出資料庫間的效能差異。
  - ✔ 效能是需要依據業務場景而定。
  - ✔ 測試應當依據大多數用戶的實際案例，太多的案例會掩蓋某些資料庫的低效率，而且這樣的案例也不是大多數用戶會遇到的。

Ref: http://antirez.com/news/85

Redis and Aerospike

Aphyr Blog Photography Code

Call me maybe: Aerospike

2015/05/04
Databases Distributed Systems Jepsen Aerospike

Previously, on Jepsen, we reviewed Elasticsearch's progress in addressing data-loss bugs during network partitions. Today, we'll see Aerospike, an "ACID database", react violently to a basic partition.

Aerospike is a high-performance, distributed, schema-less, KV store, often deployed in caching, analytics, or ad tech environments. Its five-dimensional data model is similar to Bigtable or Cassandra: *namespaces* (databases) contain *sets* (tables) of records, where *keys* identify *records*. Each record is a map of *bin names* to *values*. Aerospike has put a good deal of work into performance across good-size (~100TB) datasets, and is repositioning itself as a general purpose datastore competitive with, say, MongoDB.

Data is sharded and balanced between servers using a Paxos-based membership algorithm. Stored procedures are available in Lua and allow for MapReduce-style parallel queries. There's a lot to like here. However, Aerospike makes a dangerous assumption for a distributed datastore: it assumes the network is reliable. In this post, we'll explore what happens in Aerospike 3.5.4 when the network is *not* reliable.

Ref: https://aphyr.com/posts/324-call-me-maybe-aerospike

Redis and Aerospike

- ✔ However, as the network shifts, ...By the time of the final read, about 10% of the increment operations have been lost.
- ✔ Just like the CaS register test, increment and read latencies will jump from ~1 millisecond to ~500 milliseconds when a partition occurs.
- ✔ Aerospike can service every request successfully, peaking at ~2 seconds.

Redis 與 Aerospike

- ✔ 當 Network partition 發生時， Aerospike 會在某個很短的時間內丟失操作。以每秒 500 次的 increment operations 測試，丟失約 10% 的寫入。
- ✔ 在 partition 完成後，會有幾秒很明顯的 latency 高峰出現。
- ✔ Aerospike 即使在已經執行已久的 partition 中，也會出現服務中斷的情形，中斷甚至長達 2 秒。

Ref: http://antirez.com/news/85

Redis and Aerospike

- ✔ In the summer of 2013 we faced exactly this problem: big-memory (192 GB RAM) server nodes were running out of memory and crashing again ... We were being bitten by fragmentation.

Redis 與 Aerospike

- ✔ 2013 年夏天， Aerospike 突然有一台 192 GB RAM 的伺服器因記憶體用盡而當機， ASMalloc 工具未查出 memory leak ，所以看來是因為記憶體碎片造成的。

Ref: http://highscalability.com/blog/2015/3/17/in-memory-computing-at-aerospike-scale-when-to-choose-and-ho.html

Agenda

- ✔ Redis history
- ✔ Redis 3.0
- ✔ Redis features
- ✔ Redis and Memcached
- ✔ Redis and Aerospike
- ✔ Insight on the pit

議程

- ✔ Redis 歷史
- ✔ Redis 3.0
- ✔ Redis 特性
- ✔ Redis 與 Memcached
- ✔ Redis 與 Aerospike
- ✔ 坑裡的洞見

Insight on the pit

Server-side sessions with Redis

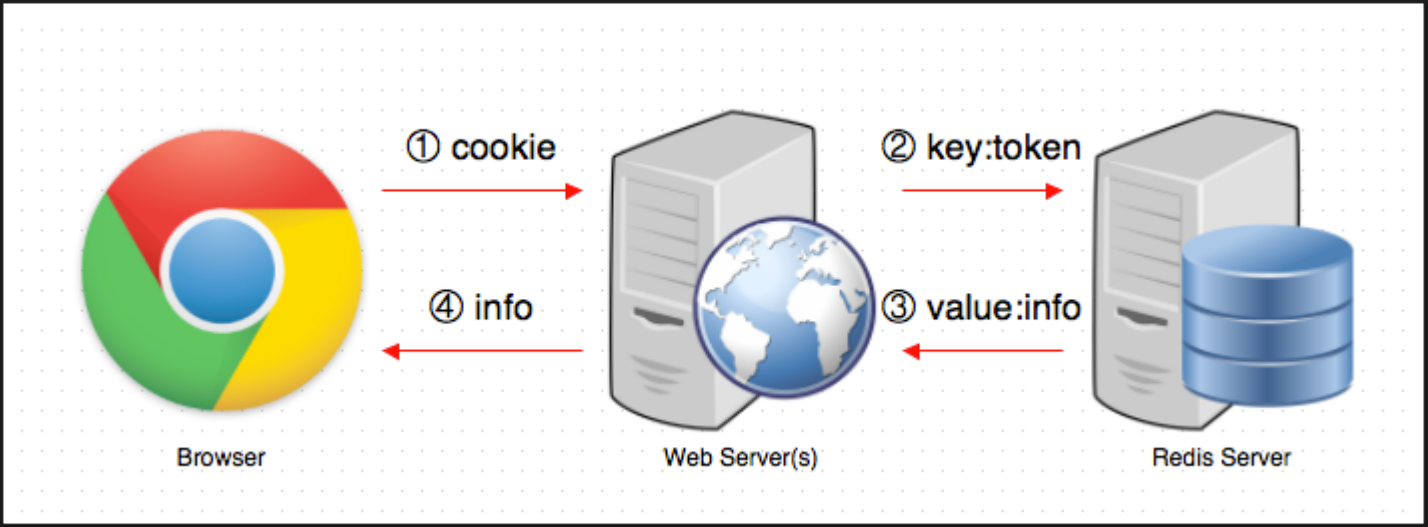
坑裡的洞見

使用 Redis 共享 Sessions

小心
先驅變先烈

朕知道了

Insight on the pit 【 Server-side sessions with Redis 】



Ref: http://vc2tea.com/redis-session/

Insight on the pit 【 Server-side sessions with Redis 】

- ✔ Redis has many eviction policies, but most of them are based on 'sampling'.
- ✔ This means eviction item is not global optimization, but local optimization.
  - ✔ When reach 'maxmemory', it may evict items not old enough.
    - ✔ Users get logged out early, and the worst is you won’t even notice it, until users start complaining.

坑裡的洞見【使用 Redis 共享 Sessions 】

- ✔ Redis 有很多種移除舊數據的策略，但大多基於「抽樣」。
- ✔ 這意謂移除舊數據不是全局最優而是局部最優。
  - ✔ 當達到 'maxmemory' 上限時，可能造成移除的數據「不夠舊」。
    - ✔ 使得使用者提前被登出。最糟的是，你可能都不會知道，直到使用者開始抱怨。

Ref: http://redis.io/topics/lru-cache

Insight on the pit 【 Server-side sessions with Redis 】

- ✔ Alternative solutions.
  - ✔ Use database as an another back-end.
    - ✔ 1. When write session, set both in Redis and database.
    - ✔ 2. When read session, Redis first, database second.
  - ✔ Redis 3.0.
    - ✔ More 'sampling'.

坑裡的洞見【使用 Redis 共享 Sessions 】

- ✔ 替代方案。
  - ✔ 使用資料庫為另一儲存後台。
    - ✔1. 寫入 Session 時，同時寫進 Redis 及資料庫。
    - ✔2. 讀出 Session 時， Redis 優先，資料庫其次。
  - ✔ 使用 Redis 3.0 。
    - ✔ 選擇較大的 'sampling'（抽樣數）。

Insight on the pit 【 Server-side sessions with Redis 】

- ✔ A better way is ... (thinking)

坑裡的洞見【使用 Redis 共享 Sessions 】

- ✔ 還有其它的解法 !!! (思考)

Insight on the pit

Maximize CPUs usage

坑裡的洞見

善用多核 CPU

偶爾任性是可愛，
一天到晚任性是妖孽。

朕知道了

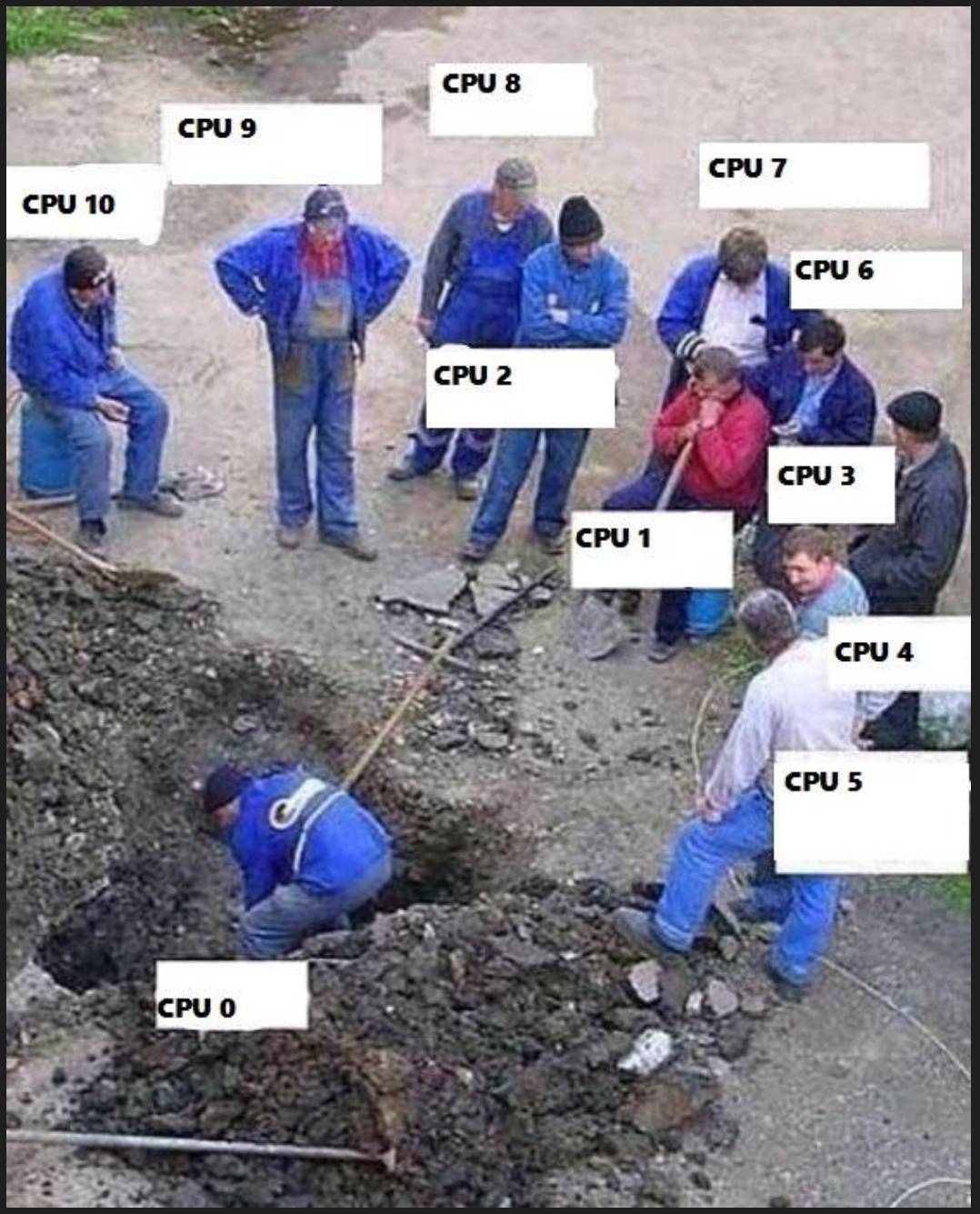
Insight on the pit 【 Maximize CPUs usage 】

- ✔ Redis is single thread.
  - ✔ One instance usually only use one CPU.
    - ✔ (background threads.)
    - ✔ (background tasks, such as BGSAVE, AOF rewrite.)

坑裡的洞見【善用多核 CPU 】

- ✔ Redis 是單執行緒。
  - ✔ 一個 Redis 實例通常只會用到一顆 CPU 。
    - ✔ (背景執行緒)
    - ✔ (背景工作，例如 BGSAVE 及 AOF rewrite)

Insight on the pit 【 Maximize CPUs usage 】



Insight on the pit 【 Maximize CPUs usage 】

- ✔ Maximize CPUs usage.
  - ✔ Redis instances is same as CPU cores.
  - ✔ But,
    - ✔ 1.Set 'maxmemory' for each instance carefully.
    - ✔ 2.Instance should have different 'dbfilename'.
    - ✔ 3.Instance should have different 'appendfilename'.

坑裡的洞見【善用多核 CPU 】

- ✔ 善用多核 CPU 。
  - ✔ 啟動的 Redis 實例與 CPU 核心數一樣多。
  - ✔ 但 ,
    - ✔ 1. 每個實例的 'maxmemory' 需要小心配置。
    - ✔ 2. 每個實例的 'dbfilename' 需要不一樣。
    - ✔ 3. 每個實例的 'appendfilename' 需要不一樣。

Insight on the pit

Memory optimization

坑裡的洞見

記憶體優化

前程四緊：

手頭緊、

眉頭緊、

衣服緊、

時間緊。

朕知道了

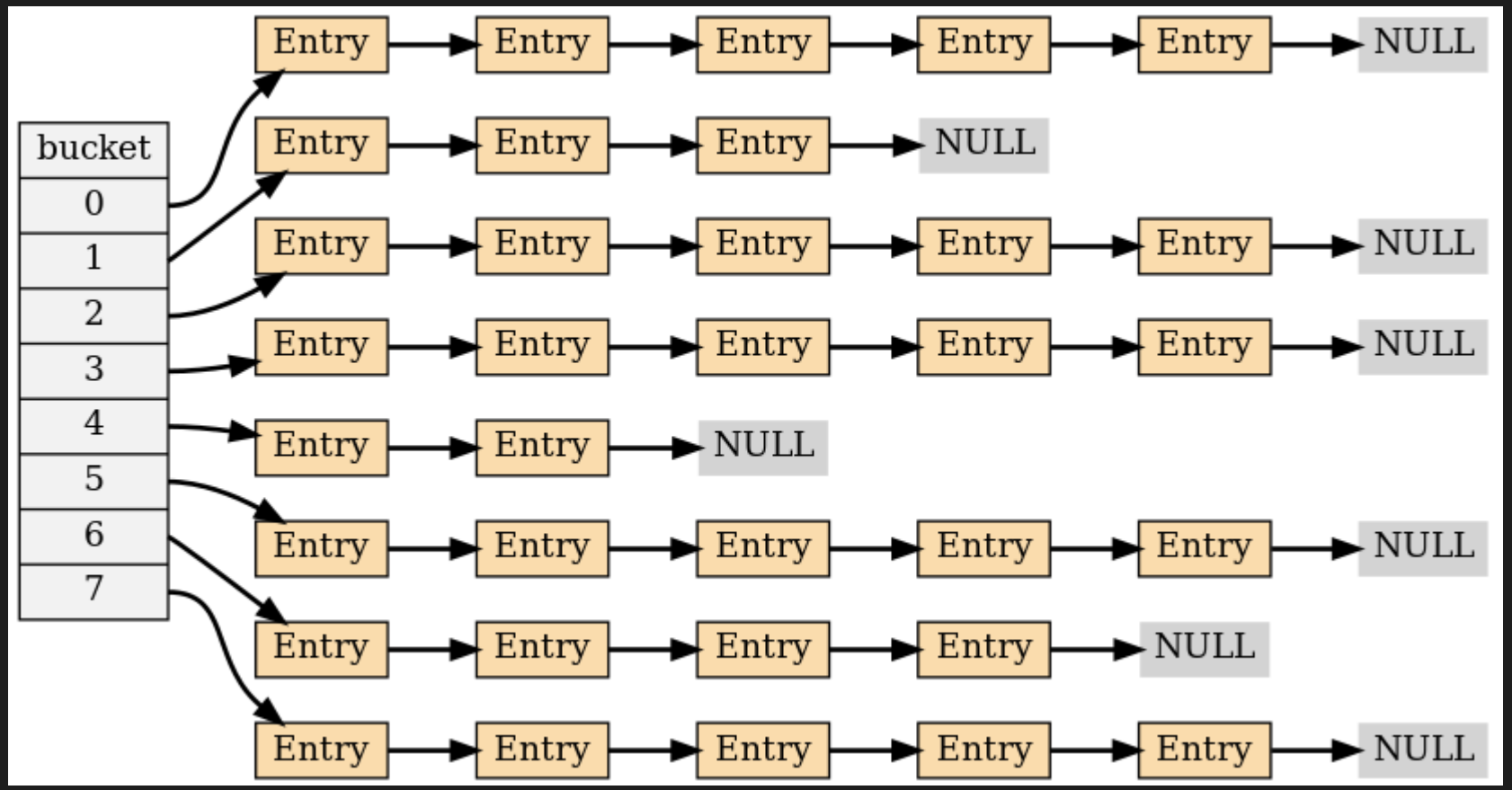
Insight on the pit 【 Memory optimization 】

- ✔ Memory fragmentation.
  - ✔ SET.
  - ✔ rehash.
    - ✔ When hash table needs to switch to a bigger or smaller table this happens incrementally.

坑裡的洞見【記憶體優化】

- ✔ 記憶體碎片。
  - ✔ SET。
  - ✔ rehash。
    - ✔ 當 dict 鍵值持續增加時，為保持良好的效能， dict 需要執行 rehash。

Insight on the pit 【 Memory optimization 】



Ref: http://redisbook.readthedocs.org/en/latest/internal-datastruct/dict.html

Insight on the pit 【 Memory optimization 】

- ✔ Key name length.
  - ✔ Shorter is better.
  - ✔ But also meaningful ones.
    - ✔ “product:user1:count” is better than “pu1c”.

坑裡的洞見【記憶體優化】

- ✔ Key 命名長度。
  - ✔ 長度愈短愈好。
  - ✔ 但還是要有意義。
    - ✔ “product:user1:count” 優於” pu1c” 。

Insight on the pit 【 Memory optimization 】

- ✔ Ziplist.
  - ✔ The ziplist is a specially encoded dually linked list that is designed to be very memory efficient.
    - ✔ Ziplist is space efficient.

坑裡的洞見【記憶體優化】

- ✔ Ziplist。
  - ✔ 符合某種設定下，資料結構會以 Ziplist 方式儲存。
    - ✔ 類似一維線性儲存，省去大量的指針開銷。

Ref: http://redis.io/topics/memory-optimization

Insight on the pit 【 Memory optimization 】

- ✔ Ziplist.
  - ✔ hash-max-ziplist-entries 64
  - ✔ hash-max-ziplist-value 512
  - ✔ list-max-ziplist-entries 512
  - ✔ list-max-ziplist-value 64
  - ✔ zset-max-ziplist-entries 128
  - ✔ zset-max-ziplist-value 64
  - ✔ set-max-intset-entries 512

坑裡的洞見【記憶體優化】

- ✔ Ziplist。
  - ✔ hash-max-ziplist-entries 64
  - ✔ hash-max-ziplist-value 512
  - ✔ list-max-ziplist-entries 512
  - ✔ list-max-ziplist-value 64
  - ✔ zset-max-ziplist-entries 128
  - ✔ zset-max-ziplist-value 64
  - ✔ set-max-intset-entries 512

Insight on the pit 【 Memory optimization 】

- ✔ Ziplist.
  - ✔ hash-max-ziplist-entries 64
  - ✔ hash-max-ziplist-value 512
    - ✔ Use ziplist if entries count ≦ 64 or every entry size ≦ 512.

坑裡的洞見【記憶體優化】

- ✔ Ziplist。
  - ✔ hash-max-ziplist-entries 64
  - ✔ hash-max-ziplist-value 512
    - ✔ 如果 Hash 的數量 ≦ 64 ，或其中一個 Hash 的值≦ 512 ，則使用 Ziplist。

Insight on the pit 【 Memory optimization 】

- ✔ Ziplist.
  - ✔ Twitter use case.
    - ✔ A Redis ziplist threshold is set to the max size of a Timeline. Never store a bigger Timeline than can be stored in a ziplist.

坑裡的洞見【記憶體優化】

- ✔ Ziplist。
  - ✔ Twitter 的案例。
    - ✔ Ziplist 的數量設定與 Timelines 的最大數量一致；
    - ✔ Timeline 的儲存大小也不會超過 Ziplist 的上限。

Ref: http://highscalability.com/blog/2014/9/8/how-twitter-uses-redis-to-scale-105tb-ram-39mm-qps-10000-ins.html

Insight on the pit 【 Memory optimization 】

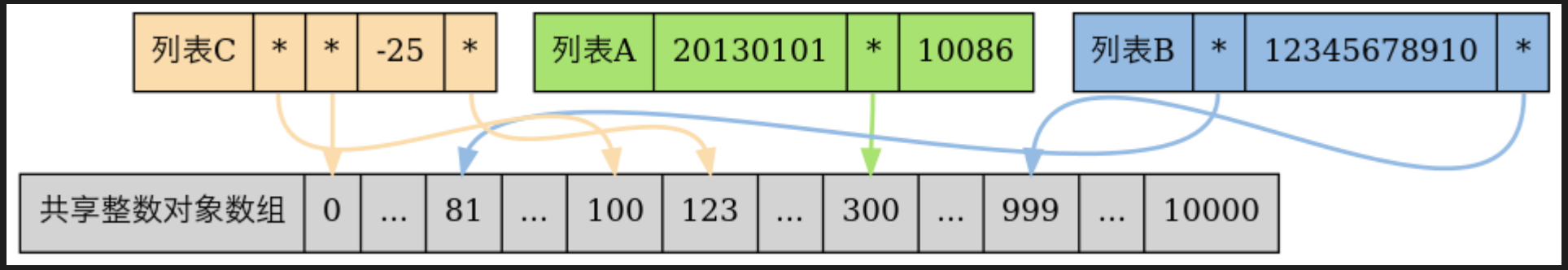
- ✔ REDIS_SHARED_INTEGERS.
  - ✔ Default is 10,000.
  - ✔ Integers can be stored in a shared memory pool, and don't have any memory overheads.

坑裡的洞見【記憶體優化】

- ✔ REDIS_SHARED_INTEGERS。
  - ✔ 預設是 10,000。
  - ✔ 整數 (包括 0) 可以預分配在共享池，避免重複分配而節省記憶體。

Insight on the pit 【 Memory optimization 】

Flyweight



src/redis.h

```
86 #define REDIS_DBCRON_DBS_PER_CALL 16
87 #define REDIS_MAX_WRITE_PER_EVENT (1024*64)
88 #define REDIS_SHARED_SELECT_CMDS 10
89 #define REDIS_SHARED_INTEGERS 10000
90 #define REDIS_SHARED_BULKHDR_LEN 32
91 #define REDIS_MAX_LOGMSG_LEN    1024 /* Default maximum length of syslog messages */
92 #define REDIS_AOF_REWRITE_PERC  100
```

Ref: http://redisbook.readthedocs.org/en/latest/datatype/object.html

Insight on the pit 【 Memory optimization 】

- ✔ Bitmaps.
- ✔ HyperLogLogs.

坑裡的洞見【記憶體優化】

- ✔ Bitmaps.
- ✔ HyperLogLogs.

Insight on the pit

Availability

坑裡的洞見

可用性

開發都想好自在，
客戶都要靠得住。

朕知道了

Insight on the pit 【 Availability 】

- ✔ Twemproxy (Twitter)
- ✔ Codis (豌豆荚)
- ✔ Redis Cluster (Official)
- ✔ Cerberus (HunanTV)

坑裡的洞見【可用性】

- ✔ Twemproxy (Twitter)
- ✔ Codis (豌豆荚)
- ✔ Redis Cluster (官方)
- ✔ Cerberus (芒果 TV)

Insight on the pit 【 Availability 】

- ✔ Twemproxy (Twitter)
  - ✔ Twemproxy is proxy-based solution.
  - ✔ Good parts
    - ✔ Stable, enterprise ready.

坑裡的洞見【可用性】

- ✔ Twemproxy (Twitter)
  - ✔ 代理分片機制。
  - ✔ 優點
    - ✔ 非常穩定，企業級方案。

Insight on the pit 【 Availability 】

- ✔ Twemproxy (Twitter)
  - ✔ Bad parts
    - ✔ SPOF (Single Point Of Failure)
      - ✔ Keepalived etc.
    - ✔ Smoothless on scale.
    - ✔ No dashboard.
    - ✔ Proxy-based, more route trip times, higher latency.
    - ✔ Single-threaded proxy model.

坑裡的洞見【可用性】

- ✔ Twemproxy (Twitter)
  - ✔ 缺點
    - ✔ 單點故障。
      - ✔ 需依賴第三方軟體，如 Keepalived 。
    - ✔ 無法平滑地橫向擴展。
    - ✔ 沒有後台介面。
    - ✔ 代理分片機制引入更多的來回次數並提高延遲。
    - ✔ 單核模式，無法充份利用多核，除非多實例。

Insight on the pit 【 Availability 】

- ✔ Twemproxy (Twitter)
  - ✔ Bad parts
    - ✔ Twemproxy is not used by Twitter internally.

坑裡的洞見【可用性】

- ✔ Twemproxy (Twitter)
  - ✔ 缺點
    - ✔ Twitter 官方內部不再繼續使用 Twemproxy 。

Ref: http://highscalability.com/blog/2014/9/8/how-twitter-uses-redis-to-scale-105tb-ram-39mm-qps-10000-ins.html

Insight on the pit 【 Availability 】

- ✔ Codis (豌豆莢)
  - ✔ Codis is proxy-based solution.
  - ✔ 豌豆莢 open source on Jan 2014.
  - ✔ Written in Go and C.

坑裡的洞見【可用性】

- ✔ Codis (豌豆莢)
  - ✔ 代理分片機制。
  - ✔ 豌豆莢於 2014 年 11 月開放源碼。
  - ✔ 基於 Go 與 C 開發。

Insight on the pit 【 Availability 】

- ✔ Codis (豌豆莢)
  - ✔ Good parts
    - ✔ Stable, enterprise ready.
    - ✔ Auto Rebalance.
    - ✔ High performance.
      - ✔ Simple testbed is faster 100% than Twemproxy.
    - ✔ Multi-threaded proxy model.

坑裡的洞見【可用性】

- ✔ Codis (豌豆莢)
  - ✔ 優點
    - ✔ 非常穩定，企業級方案。
    - ✔ 數據自動平衡。
    - ✔ 高效能。
      - ✔ 簡單的測試顯示較 Twemproxy 快一倍。
    - ✔ 善用多核 CPU 。

Insight on the pit 【 Availability 】

- ✔ Codis (豌豆莢)
  - ✔ Good parts
    - ✔ Simple
      - ✔ No paxos-like coordinators,
      - ✔ No master-slave replication.
    - ✔ Dashboard.

坑裡的洞見【可用性】

- ✔ Codis (豌豆莢)
  - ✔ 優點
    - ✔ 簡單。
      - ✔ 沒有 Paxos 類的協調機制。
      - ✔ 沒有主從複製。
    - ✔ 有後台介面。

Insight on the pit 【 Availability 】

- ✔ Codis (豌豆莢)
  - ✔ Bad parts
    - ✔ Proxy-based, more route trip times, higher latency.
    - ✔ Need 3rd-party coordinators
      - ✔ Zookeeper or Etcd.
    - ✔ No master-slave replication.

坑裡的洞見【可用性】

- ✔ Codis (豌豆莢)
  - ✔ 缺點
    - ✔ 代理分片機制引入更多的來回次數並提高延遲。
    - ✔ 需要第三方軟體支持協調機制。
      - ✔ 目前支援 Zookeeper 及 Etcd 。
    - ✔ 不支援主從複製，需要另外實作。

Insight on the pit 【 Availability 】

Codis 的设计与实现 Part 3

- ✔ Codis 采用了 Proxy 的方案，所以必然会带来单机性能的损失。
- ✔ 经测试，在不开 pipeline 的情况下，大概会损失 40% 左右的性能，但是 Redis 本身是一个快得吓人的东西，即使单机损失了 40% 仍然是一个很大的数字。

Ref: http://0xffff.me/blog/2014/11/11/codis-de-she-ji-yu-shi-xian-part-3/

Insight on the pit 【 Availability 】

- ✔ Redis Cluster (Official)
  - ✔ Official supports.
  - ✔ Requires Redis 3.0 or higher.

坑裡的洞見【可用性】

- ✔ Redis Cluster (官方)
  - ✔ 官方支援。
  - ✔ 需要 Redis 3.0 或更高版本。

Insight on the pit 【 Availability 】

- ✔ Redis Cluster (Official)
  - ✔ Good parts
    - ✔ Official supports.
    - ✔ Pear-to-pear Gossip distributed model.
      - ✔ Less route trip times, lower latency.
    - ✔ Automatically sharded across multiple Redis nodes.
    - ✔ Do not need 3rd-party coordinators

坑裡的洞見【可用性】

- ✔ Redis Cluster (官方)
  - ✔ 優點
    - ✔ 官方支援。
    - ✔ 無中心的 P2P Gossip 分散式模式。
      - ✔ 更少的來回次數並降低延遲。
    - ✔ 自動於多個 Redis 節點進行分片。
    - ✔ 不需要第三方軟體支持協調機制。

Insight on the pit 【 Availability 】

- ✔ Redis Cluster (Official)
  - ✔ Bad parts
    - ✔ Requires Redis 3.0 or higher.
    - ✔ Need time to prove its stability.
    - ✔ No dashboard.
    - ✔ Need smart client.
      - ✔ Redis client need to support for Redis Cluster.
    - ✔ More maintenance cost than Codis.

坑裡的洞見【可用性】

- ✔ Redis Cluster (官方)
  - ✔ 缺點。
    - ✔ 需要 Redis 3.0 或更高版本。
    - ✔ 需要時間驗證其穩定性。
    - ✔ 沒有後台介面。
    - ✔ 需要智能客戶端。
      - ✔ Redis 客戶端必須支援 Redis Cluster 設計。
    - ✔ 較 Codis 有更多的維護升級成本。

Insight on the pit 【 Availability 】

- ✔ Cerberus (HunanTV)
  - ✔ Good parts
    - ✔ Auto Rebalance.
    - ✔ Implement Redis's Smart Client.
    - ✔ Read-write split.

坑裡的洞見【可用性】

- ✔ Cerberus (芒果 TV)
  - ✔ 優點
    - ✔ 數據自動平衡。
    - ✔ 本身實現了 Redis 的 Smart Client 。
    - ✔ 支援讀寫分離。

Ref: https://github.com/HunanTV/redis-cerberus

Insight on the pit 【 Availability 】

- ✔ Cerberus (HunanTV)
  - ✔ Bad parts
    - ✔ Requires Redis 3.0 or higher.
    - ✔ Proxy-based, more route trip times, higher latency.
    - ✔ Need time to prove its stability.
    - ✔ No dashboard.

坑裡的洞見【可用性】

- ✔ Cerberus (芒果 TV)
  - ✔ 缺點
    - ✔ 需要 Redis 3.0 或更高版本。
    - ✔ 代理分片機制引入更多的來回次數並提高延遲。
    - ✔ 需要時間驗證其穩定性。
    - ✔ 沒有後台介面。

Insight on the pit

Stabilization

坑裡的洞見

穩定性

每一個穩定服務背後，
都有一個齷齪的實現。

朕知道了

Insight on the pit 【 Stabilization 】

- ✔ Performance fluctuation.
- ✔ Out of memory.
- ✔ Redis instances is same as CPU cores.
- ✔ Big Ziplist.
- ✔ Master-slave.

坑裡的洞見【穩定性】

- ✔ 效能抖動。
- ✔ 記憶體不足。
- ✔ 啟動的 Redis 實例與 CPU 核心數一樣多。
- ✔ Big Ziplist 。
- ✔ 主從模式。

Insight on the pit 【 Stabilization 】

- ✔ Performance fluctuation.
  - ✔ For production, stabilization is more important than average performance.
    - ✔ Easy to estimated, reduce the chances of an important moment occurred at lower point.
  - ✔ Redis is single thread.

坑裡的洞見【穩定性】

- ✔ 效能抖動。
  - ✔ 對於一個上線服務而言，穩定性遠大於平均效能。
    - ✔ 效能防抖動，好預估，降低重要時刻發生在低點的機率。
  - ✔ Redis 是單執行緒。

Insight on the pit 【 Stabilization 】

- ✔ Performance fluctuation.
  - ✔ Tips: Split heavy commands.
    - ✔ MGET
      - ✔ redis> MGET 1 2 3 ... 999
    - ✔ ZRANGE
      - ✔ redis> ZRANGE myset 0 -1
    - ✔ SORT / LREM / SUNION / SDIFF / SINTER
    - ✔ KEYS / SMEMBERS / HGETALL

坑裡的洞見【穩定性】

- ✔ 效能抖動。
  - ✔ 拆解「重」指令。
    - ✔ MGET。
      - ✔ redis> MGET 1 2 3 ... 999
    - ✔ ZRANGE。
      - ✔ redis> ZRANGE myset 0 -1
    - ✔ SORT / LREM / SUNION / SDIFF / SINTER。
    - ✔ KEYS / SMEMBERS / HGETALL。

Insight on the pit 【 Stabilization 】

- ✔ Performance fluctuation.
  - ✔ Tips: Rethink block commands.
    - ✔ BLPOP
    - ✔ BRPOPLPUSH
    - ✔ BRPOP
    - ✔ MULTI / EXEC

坑裡的洞見【穩定性】

- ✔ 效能抖動。
  - ✔ 「阻塞」指令。
    - ✔ BLPOP。
    - ✔ BRPOPLPUSH。
    - ✔ BRPOP。
    - ✔ MULTI / EXEC。

Insight on the pit 【 Stabilization 】

- ✔ Out of memory (OOM).
  - ✔ Be careful those commands will ask huge memory.
  - ✔ Reduce the chances of Redis to be killed by OOM.
  - ✔ SWAP, lose a little performance is better than crash.

坑裡的洞見【穩定性】

- ✔ 記憶體不足 (Out of memory, OOM) 。
  - ✔ 留意那些會大量耗用記憶體的指令。
  - ✔ 降低 Redis 強制被 Out of memory 關閉的機率。
  - ✔ 開啟 SWAP ，效能下降總比服務停用來得好。

Insight on the pit 【 Stabilization 】

- ✔ Out of memory (OOM).
  - ✔ maxmemory
  - ✔ overcommit_memory
  - ✔ SWAP
  - ✔ zone_reclaim_mode
  - ✔ oom_adj

坑裡的洞見【穩定性】

- ✔ 記憶體不足 (Out of memory, OOM) 。
  - ✔ maxmemory
  - ✔ overcommit_memory
  - ✔ SWAP
  - ✔ zone_reclaim_mode
  - ✔ oom_adj

Insight on the pit 【 Stabilization 】

- ✔ Out of memory (OOM).
  - ✔ maxmemory
    - ✔ A rule of thumbs is 50% of total memory.
      - ✔ BGSAVE.
      - ✔ AOF rewrite.

坑裡的洞見【穩定性】

- ✔ 記憶體不足 (Out of memory, OOM) 。
  - ✔ maxmemory
    - ✔ 經驗法則是設定為總記憶體的 50% 。
      - ✔ BGSAVE 。
      - ✔ AOF rewrite 。

Ref: http://redis.io/topics/admin

Insight on the pit 【 Stabilization 】

- ✔ Out of memory (OOM).
  - ✔ overcommit_memory
    - ✔ overcommit_memory = 1
      - ✔ Do overcommit.

坑裡的洞見【穩定性】

- ✔ 記憶體不足 (Out of memory, OOM) 。
  - ✔ maxmemory
    - ✔overcommit_memory = 1
      - ✔ 請求分配記憶體時，永遠假裝還有足夠的記憶體。

Insight on the pit 【 Stabilization 】

- ✔ Out of memory (OOM).
  - ✔ SWAP
    - ✔ Use SWAP, and same size of memory.

坑裡的洞見【穩定性】

- ✔ 記憶體不足 (Out of memory, OOM) 。
  - ✔ SWAP
    - ✔ 使用 SWAP ，並且與記憶體一樣大。

Insight on the pit 【 Stabilization 】

- ✔ Out of memory (OOM).
  - ✔ zone_reclaim_mode
    - ✔ zone_reclaim_mode = 0 (default)

坑裡的洞見【穩定性】

- ✔ 記憶體不足 (Out of memory, OOM) 。
  - ✔ zone_reclaim_mode
    - ✔ zone_reclaim_mode = 0 (預設)

Insight on the pit 【 Stabilization 】

- ✔ Out of memory (OOM).
  - ✔ If RHEL/CentOS ≧ 6.4 or Kernel ≧ 3.5-rc1.
    - ✔ (1) Prefer swap to OOM.
      - ✔ vm.swappiness = 1
    - ✔ (2) Prefer OOM to swap.
      - ✔ vm.swappiness = 0
  - ✔ Else
    - ✔ vm.swappiness = 0

坑裡的洞見【穩定性】

- ✔ 記憶體不足 (Out of memory, OOM) 。
  - ✔ 如果 RHEL/CentOS ≧ 6.4 或 Kernel ≧ 3.5-rc1 。
    - ✔ (1) 寧願 swap 也不要 OOM 。
      - ✔ vm.swappiness = 1
    - ✔ (2) 寧願 OOM 也不要 swap 。
      - ✔ vm.swappiness = 0
  - ✔ 否則
    - ✔ vm.swappiness = 0

Insight on the pit 【 Stabilization 】

- ✔ Out of memory (OOM).
  - ✔ If (1) then oom_adj.
    - ✔ echo -15 > /proc/`pidof redis-server`/oom_adj
      - ✔ Reduce the chances of redis to be killed.
    - ✔ Tips
      - ✔ for i in $(pidof redis-server); \
        do echo -15 | sudo tee /proc/$i/oom_adj ; done

坑裡的洞見【穩定性】

- ✔ 記憶體不足 (Out of memory, OOM) 。
  - ✔ 如果 (1) 則 oom_adj 。
    - ✔ echo -15 > /proc/`pidof redis-server`/oom_adj
      - ✔ 降低 Redis 強制被 Out of memory 關閉的機率。
    - ✔ Tips
      - ✔ for i in $(pidof redis-server); \
        do echo -15 | sudo tee /proc/$i/oom_adj ; done

Insight on the pit 【 Stabilization 】

- ✔ 過去 vm.swappiness 設定為 0 可以降低 swap 的發生率，但非完全禁止。所以預期會發生 swap 而不會 OOM。
- ✔ 在 Linux Kernel 3.5-RC1 及 RHEL/CentOS Kernel 2.6.32-303 (CentOS 6.4) 之後的版本，已經改變此行為。
- ✔ 設定為 0 時完全不會有任何 swap ，但非預期的記憶體壓力可能會造成 OOM 而關閉 Redis。

Insight on the pit 【 Stabilization 】

Linux Kernel 3.4 (mm/vmscan.c)

```
out:
        for_each_evictable_lru(lru) {
                int file = is_file_lru(lru);
                unsigned long scan;

                scan = zone_nr_lru_pages(mz, lru);
                if (priority || noswap) {
                        scan >>= priority;
                        if (!scan && force_scan)
                                scan = SWAP_CLUSTER_MAX;
                        scan = div64_u64(scan * fraction[file], denominator);
                }
                nr[lru] = scan;
        }
}
```

Linux Kernel 3.5.1 (mm/vmscan.c)

```
out:
        for_each_evictable_lru(lru) {
                int file = is_file_lru(lru);
                unsigned long scan;

                scan = get_lru_size(lruvec, lru);
                if (sc->priority || noswap || !vmscan_swappiness(sc)) {
                        scan >>= sc->priority;
                        if (!scan && force_scan)
                                scan = SWAP_CLUSTER_MAX;
                        scan = div64_u64(scan * fraction[file], denominator);
                }
                nr[lru] = scan;
        }
}
```

Ref: https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.4.tar.xz
Ref: https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.5.1.tar.gz

Insight on the pit 【 Stabilization 】

linux-2.6.32-504.12.2.el6 (CentOS 6.4, mm/vmscan.c)

```
1825
1826        for_each_evictable_lru(l) {
1827                int file = is_file_lru(l);
1828                unsigned long scan;
1829
1830                scan = zone_nr_lru_pages(mz, l);
1831                if (priority || noswap || !sc->swappiness) {
1832                        scan >>= priority;
1833                        scan = (scan * percent[file]) / 100;
1834                }
1835                nr[l] = nr_scan_try_batch(scan,
1836                                          &reclaim_stat->nr_saved_scan[l]);
1837        }
1838
```

Ref: http://rpm.pbone.net/index.php3/stat/3/srodzaj/2/search/kernel-2.6.32-504.12.2.el6.src.rpm

Insight on the pit 【 Stabilization 】

- ✔ Redis instances is same as CPU cores.
  - ✔ Redis have some background tasks.
    - ✔ fsync file descriptor.
    - ✔ close file descriptor.
    - ✔ BGSAVE.
    - ✔ AOF rewrite.
  - ✔ Preserved CPU to do those tasks.

坑裡的洞見【穩定性】

- ✔ 啟動的 Redis 實例與 CPU 核心數一樣多。
  - ✔ Redis 會執行一些 background tasks 。
    - ✔ fsync file descriptor 。
    - ✔ close file descriptor 。
    - ✔ BGSAVE 。
    - ✔ AOF rewrite 。
  - ✔ 預留一些 CPU 執行這些 tasks 。

Insight on the pit 【 Stabilization 】

- ✔ Redis instances is same as CPU cores.
  - ✔ Instance have its own synchronization.
    - ✔ Disable automatic on BGSAVE / BGREWRITEAOF, and use manual control instead.
    - ✔ Avoid execution at the same time.

坑裡的洞見【穩定性】

- ✔ 啟動的 Redis 實例與 CPU 核心數一樣多。
  - ✔ 每個實例都有自己的同步機制。
    - ✔ 關閉自動 BGSAVE / BGREWRITEAOF ，改為手動。
    - ✔ 避免各實例同時啟動，耗用大量資源。

Insight on the pit 【 Stabilization 】

- ✔ Master-slave.
  - ✔ Best practices.
    - ✔ N Redis nodes.
    - ✔ 1 master, 1 slave, N-2 slaves of slave.
  - ✔ Never restart all or multiple slave instances.
    - ✔ (Master) High CPU loading.
    - ✔ (Master) May out of memory.

坑裡的洞見【穩定性】

- ✔ 主從模式。
  - ✔ 最佳實踐。
    - ✔ N 台 Redis 。
    - ✔ 1 台主服務， 1 台從服務， N-2 台從服務的從服務。
  - ✔ 不要同時重啟所有或大量的 slave 實例。
    - ✔ 造成主服務 CPU 負載過高。
    - ✔ 造成主服務記憶體用量過高。

Insight on the pit 【 Stabilization 】

- ✔ String value.
  - ✔ String value can be at max 512 MB in length.
    - ✔ A rule of thumbs is no more than 5KB.

坑裡的洞見【穩定性】

- ✔ 字串值。
  - ✔ 字串值最大可以儲存 512MB 的長度。
    - ✔ 經驗上最好不要大於 5KB 。

Insight on the pit

Low latency

坑裡的洞見

低延遲

很多事都介於

「不說憋屈」

和

「說了矯情」

之間

朕知道了

Insight on the pit 【 Low latency 】

- ✔ Durability vs latency tradeoffs, from higher to lower latency.
  - ✔ AOF + fsync always.
  - ✔ AOF + fsync every second.
  - ✔ AOF + fsync every second + No-appendfsync-on-rewrite set to yes.
  - ✔ AOF + fsync nerver.
  - ✔ RDB.

坑裡的洞見【低延遲】

- ✔ 數據持久性 vs 延遲性的權衡，延遲性從高至低排列。
  - ✔ AOF + fsync always。
  - ✔ AOF + fsync every second。
  - ✔ AOF + fsync every second + No-appendfsync-on-rewrite set to yes。
  - ✔ AOF + fsync nerver。
  - ✔ RDB。

Ref: http://redis.io/topics/latency

Insight on the pit 【 Low latency 】

- ✔ Latency induced by network and communication.
  - ✔ Reduce the numbers of commands.
    - ✔ Pipelining.
    - ✔ MSET / MGET.

坑裡的洞見【低延遲】

- ✔ 網路造成的延遲性。
  - ✔ 減少指令的使用次數。
    - ✔ Pipelining 。
    - ✔ MSET / MGET 。

Ref: http://redis.io/topics/latency

Insight on the pit 【 Low latency 】

✔ Fork time in different systems.

Linux on physical machine (Xeon@2.27Ghz)	9 ms/GB
Linux VM on EC2 (Xen)	10 ms/GB
Linux beefy VM on VMware	12.8 ms/GB
Linux on physical machine (Unknown HW)	13.1 ms/GB
Linux VM on 6sync (KVM)	23.3 ms/GB
Linux VM on Linode (Xen)	424 ms/GB

坑裡的洞見【低延遲】

✔ 不同系統間的 Fork 時間。

Linux on physical machine (Xeon@2.27Ghz)	9 ms/GB
Linux VM on EC2 (Xen)	10 ms/GB
Linux beefy VM on VMware	12.8 ms/GB
Linux on physical machine (Unknown HW)	13.1 ms/GB
Linux VM on 6sync (KVM)	23.3 ms/GB
Linux VM on Linode (Xen)	424 ms/GB

Ref: http://redis.io/topics/latency

Insight on the pit 【 Low latency 】

- ✔ Never use Huge page.
  - ✔ echo never > /sys/kernel/mm/transparent_hugepage/enabled

坑裡的洞見【低延遲】

- ✔ 永不用 Huge page。
  - ✔ echo never > /sys/kernel/mm/transparent_hugepage/enabled

Insight on the pit 【 Low latency 】

- ✔ Do you really need Proxy-based solution (Codis) ?

坑裡的洞見【低延遲】

- ✔ 真的需要代理分片機制 (如 Codis) 嗎 ?

Insight on the pit 【 Low latency 】

简单的测试，单 redis+ 单 proxy ，默认参数

CONC	PIPELINE	CODIS-LATENCY	REDIS-LATENCY	
50	10	3.17	0.60	~428%
50	20	5.88	0.89	~561%
50	75	21.78	2.40	~937%
10	1	0.11	0.07	~57%
20	1	0.21	0.13	
50	1	0.50	0.32	
100	1	1.02	0.68	~50%
200	1	2.18	1.46	
500	1	5.87	3.42	
1000	1	12.95	8.62	~50%

Ref: https://github.com/wandoulabs/codis/issues/63

Insight on the pit 【 Low latency 】

- ✔ Codis.
  - ✔ Disable pipeline.
  - ✔ Less CPU cores.

坑裡的洞見【低延遲】

- ✔ Codis。
  - ✔ 停用 pipeline。
  - ✔ 少核 CPU。

Insight on the pit 【 Low latency 】

Disable pipeline.

- pipeline = Yes

NTHRD	Ops/sec	Latency (ms)	KB/sec	Proxy CPU	Bench CPU	Redis CPU	Total CPU
1	172016.32	21.82200	24068.44	398	98	140	636
2	186390.20	40.38600	26079.63	398	194	112	704
4	202224.24	74.66500	28295.12	398	380	92	870

- pipeline = No

NTHRD	Ops/sec	Latency (ms)	KB/sec	Proxy CPU	Bench CPU	Redis CPU	Total CPU
1	60970.10	0.81700	8530.91	366	99	102	567
2	110429.13	0.89700	15451.19	397	195	145	737
4	153360.21	1.30300	21458.09	398	339	100	837

Ref: https://github.com/wandoulabs/codis/blob/master/doc/benchmark_zh.md

Insight on the pit 【 Low latency 】

Less CPU cores.

4 cores

NTHRD	Ops/sec	Latency (ms)	KB/sec
1	60970.10	0.81700	8530.91
2	110429.13	0.89700	15451.19
4	153360.21	1.30300	21458.09

8 cores

NTHRD	Ops/sec	Latency (ms)	KB/sec
1	57244.61	0.87100	8009.64
2	97680.20	1.01200	13667.37
4	148428.94	1.32900	20768.11
8	128279.81	3.11400	4497.74

12 cores

NTHRD	Ops/sec	Latency (ms)	KB/sec
1	51946.31	0.96000	7268.30
2	82324.00	1.21300	11518.74
4	136064.57	1.46900	19038.09
8	114749.57	3.45000	4023.34

16 cores

NTHRD	Ops/sec	Latency (ms)	KB/sec
1	49071.19	1.01600	6866.02
2	79192.66	1.25600	11080.60
4	127074.67	1.57000	17780.23
8	114165.92	3.49500	4002.88

Ref: https://github.com/wandoulabs/codis/blob/master/doc/benchmark_zh.md

Insight on the pit 【 Low latency 】

- ✔ Big Ziplist.
  - ✔ Adding to and deleting from a ziplist is inefficient, especially with a very large list.
  - ✔ Deleting from a ziplist uses memmove to move data around, to make sure the list is still contiguous.
  - ✔ Adding to a ziplist requires a memory realloc call to make enough space for the new entry.

坑裡的洞見【低延遲】

- ✔ Big Ziplist。
  - ✔ 從 Ziplist 中新增或刪除都沒有效率，尤其是 Big Ziplist。
  - ✔ 從 Ziplist 刪除會利用 memmove 移動資料，以確保 list 還是連續的。
  - ✔ 在 Ziplist 中新增需要 memory realloc 以產出足夠的空間供新值儲存。

Ref: http://highscalability.com/blog/2014/9/8/how-twitter-uses-redis-to-scale-105tb-ram-39mm-qps-10000-ins.html

Insight on the pit 【 Low latency 】

- ✔ Big Ziplist.
  - ✔ Potential high latency for write operations due to timeline size.

坑裡的洞見【低延遲】

- ✔ Big Ziplist。
  - ✔ Ziplist 中的寫操作很可能會因 Big Ziplist 而帶來高延遲。

Ref: http://highscalability.com/blog/2014/9/8/how-twitter-uses-redis-to-scale-105tb-ram-39mm-qps-10000-ins.html

Insight on the pit 【 Low latency 】

- ✔ Redis client.
  - ✔ Connection pool.
  - ✔ Keep alive.

坑裡的洞見【低延遲】

- ✔ Redis 客戶端。
  - ✔ Connection pool 。
  - ✔ Keep alive 。

Insight on the pit 【 Low latency 】

- ✔ Redis 3.0 embedded string.
  - ✔ New "embedded string" object.
    - ✔ Reduce memory operations.
    - ✔ If string length ≦ 39 bytes.

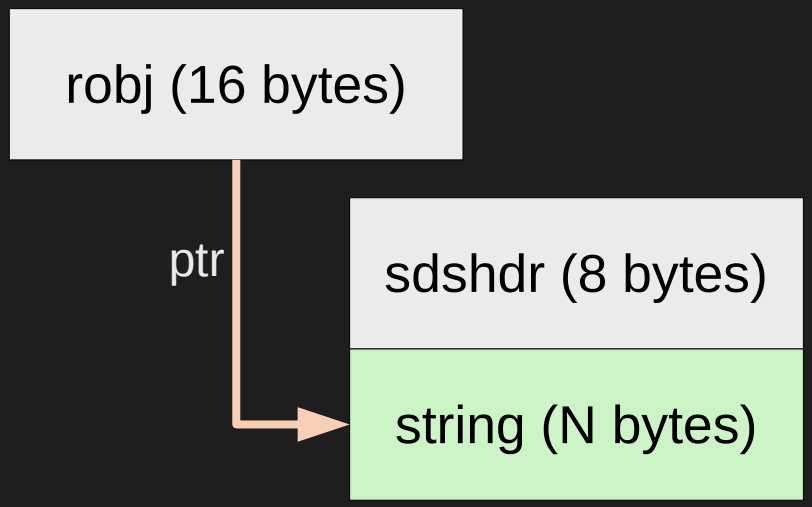
坑裡的洞見【低延遲】

- ✔ Redis 3.0 embedded string 。
  - ✔ 新的 "embedded string" 物件。
    - ✔ 減少記憶體操作次數。
    - ✔ 如果字串長度 ≦ 39 。

Insight on the pit 【 Low latency 】

Redis 2.8.20

```
/* The actual Redis Object */
#define REDIS_LRU_BITS 24
#define REDIS_LRU_CLOCK_MAX ((1<<REDIS_LRU_BITS)-1) /* Max value of obj->lru */
#define REDIS_LRU_CLOCK_RESOLUTION 1 /* LRU clock resolution in seconds */
typedef struct redisObject {
    unsigned type:4;
    unsigned encoding:4;
    unsigned lru:REDIS_LRU_BITS; /* lru time (relative to server.lruclock) */
    int refcount;
    void *ptr;
} robj;
```



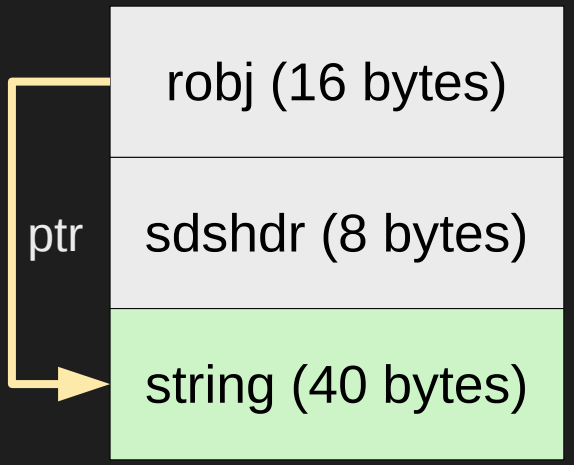
Insight on the pit 【 Low latency 】

Redis 3.0

```c
/* Create a string object with encoding REDIS_ENCODING_EMBSTR, that is
 * an object where the sds string is actually an unmodifiable string
 * allocated in the same chunk as the object itself. */
robj *createEmbeddedStringObject(const char *ptr, size_t len) {
    robj *o = zmalloc(sizeof(robj)+sizeof(struct sdshdr)+len+1);
    struct sdshdr *sh = (void*)(o+1);

    o->type = REDIS_STRING;
    o->encoding = REDIS_ENCODING_EMBSTR;
    o->ptr = sh+1;
    o->refcount = 1;
    o->lru = LRU_CLOCK();

    sh->len = len;
    sh->free = 0;
    if (ptr) {
        memcpy(sh->buf,ptr,len);
        sh->buf[len] = '\0';
    } else {
        memset(sh->buf,0,len+1);
    }
    return o;
}
```



In 64 bit system:
jemalloc arena may “64 byte-long”.

64 - 16 (robj) - 8 (sdshdr) = 40
40 - 1 (null term, \0) = 39

Feedback 勘誤回報

yftzeng@gmail.com

End 結語

Parvenu use Redis/Memcached (Much memory) ;
TRS use Aerospike (Memory / SSD mixed) ;
Mortals use SSDB (Disk only)。

暴發戶用 Redis/Memcached (記憶體多) ;
高富帥用 Aerospike (記憶體與 SSD 混用) ;
平民級用 SSDB (記憶體缺)。