Redis Memory Optimization

WORK IN PROGRESS

Redis keeps all its data in memory, and so it is important to optimize memory usage. This document explains several strategies to reduce memory overheads. This document builds up on the official memory optimization notes on redis.io, so you should read that document first.

Special Encoding for Small Aggregate Objects

Redis uses a special, memory-efficient encoding if a list, sorted set or hash contains meets the following criteria -

1. Number of elements is less than the setting `<datatype>-max-ziplist-entries`
2. Size (in bytes) of each and every element in the list/set/zset is less than the setting `<datatype>-max-ziplist-value`

The default settings in redis.conf are a good start, but it is possible to achieve greater memory savings by adjusting them according to your data. The memory analyzer helps you find the best value for these parameters based on your data. Run the analyzer on your dump.rdb file, and then open the generated csv file.

Start by filtering on type=list and encoding=linkedlist. If there are no matching rows, or the number of filtered rows is very small, you already are using memory efficiently.

Otherwise, look at the columns `num_elements` and `len_largest_element`, and compare them to the settings `list-max-ziplist-entries` and `list-max-ziplist-value`. Try to find a value for these two settings such that most linkedlists are converted to ziplists.
But remember, this is a balancing act - you don't want to set an extremely large value, because then you would start consuming a lot of CPU cycles.

Repeat the steps for hashmaps and sorted sets.

Use Integer IDs

Always use integer ids for objects. Avoid GUIDs or other unique strings to identify an object.

IDs for objects are likely to be used in other data structures such as lists, sets and sorted sets. When you use integers, Redis can use a compact representation to save memory. Some examples of how Redis optimizes :

1. Sets have a special encoding called Integer Sets. If your set contains only integers, an Intset will save you a lot of memory. See section on Integer Sets below.
2. In Zip Lists, integers are encoded using a variable number of bytes. In other words, small integers use less memory.
3. Redis has a shared pool of integers less than 10000. This shared pool of objects is used whenever the key or value in a hashtable is an integer less than 10000.
4. Even if the integer is greater than 10000, the hashtable will avoid a separate string + pointer overhead, so you are still better off than using a string.

In short, use Integers whenever you can. Also, see : http://stackoverflow.com/questions/10109921/username-or-id-for-keys-in-redis/10110407#10110407

Use Int Sets instead of Hashtable based Sets

IntSet is essentially a sorted array of numbers. Inserting or retrieving a number takes O(log N) operations - its basically a binary search. But because this is only used for small sets, the amortized cost of the operation is still O(1)

Int Sets come in 3 flavours - 16bits, 32 bits or 64 bits. If you only use 16 bit numbers, the 16 bit version will be used. If one of your elements in greater than 2^15, Redis will automatically use 32 bits for each number. Similarly, when an element exceeds 2^31, Redis will convert the array to a 64 bit array.

In contrast, the regular implementation of Set uses a Hashtable. This has a lot of memory overhead.

To save memory, store integers in your sets. That way, Redis will automatically use the most memory efficient data structure.

Avoid Small Strings

String data type has an overhead of about about 90 bytes on a 64 bit machine. In other words, calling `set foo bar` uses about 96 bytes, of which 90 bytes is overhead.
You should use the String data type only if :

1. The value is at least greater than 100 bytes

2. You are storing encoded data in the string - JSON encoded or Protocol buffer
3. You are using the string data type as an array or a bitset

If you are not doing any of the above, use hashes instead. See this Instagram article on how they stored hundreds of millions of key value pairs in Redis.

Use lists instead of dictionaries for small, consistent objects

Lets say you want to store user details in Redis. The logical data structure is a Hash. For example, you would do this - `hmset user:123 id 123 firstname Sripathi lastname Krishnan location Mumbai twitter srithedabbler`.
Now, Redis 2.6 will store this internally as a Zip List; you can confirm by running `debug object user:123` and look at the encoding field. In this encoding, key value pairs are stored sequentially, so the user object we created above would roughly look like this `["firstname", "Sripathi", "lastname", "Krishnan", "location", "Mumbai", "twitter", "srithedabbler"]`
Now, if you create a second user, the keys will be duplicated. If you have a million users, well, its a big waste repeating the keys again.

To get around this, we can borrow a concept from Python - NamedTuples. A NamedTuple is simply a read-only list, but with some magic to make that list look like a dictionary.

Your application needs to maintain a mapping from field names to indexes. So, "firstname" => 0, "lastname" => 1 and so on. Then, you simply create a list instead of a hash, like this - `lpush user:123 Sripathi Krishnan Mumbai srithedabbler`. With the right abstractions in your application, you can save significant memory.
Don't use this technique if -

1. You have less than 50,000 objects
2. Your objects are not regular i.e. some users have lots of information, others very little.

The memory analyzer output has a column "bytes_saved_if_converted_to_list" against every hash. Sum up this column and see how much memory you'd save if you make that change. If it is significant and worth the added complexity, go ahead and make that change.