메모리 확인

Redis를 Standalone으로 Docker Container에 올리고 `docker stats`로 메모리를 확인해보았을 때 다음과 같았다.

```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT     MEM %     NET I/O         BLOCK I/O        PIDS
4829b0095f0f   fresh-trash-redis   0.37%     8.883MiB / 7.657GiB   0.11%     916B / 0B       0B / 0B          6
```

그리고 로그인 후 유저 정보를 캐싱하고 캐싱한 키와 값이 메모리를 얼마나 사용하고 있는지 확인해보았다.

```
127.0.0.1:6379> keys *
1) "Member"
2) "Member:1"
127.0.0.1:6379> memory usage Member:1
(integer) 1046
```

저장된 유저 정보마다 할당되는 크기가 다르겠지만 임의로 하나의 유저 정보가 사용하고 있는 메모리를 확인해보았을 때 `1046byte(1.046kb)` 만큼 사용하는 것을 확인할 수 있었다.

ElastiCache의 데모 옵션(0.5 GiB)을 기준으로 계산해보면 약 50만명의 유저 정보를 저장할 수 있다. 하지만 Redis의 메모리는 데이터를 저장하는 것 이외에도 다양한 기능을 통해 사용되기 때문에 동시 접속 사용자가 50만명보다 적더라도 충분히 OOM(Out Of Memory) 에러가 발생할 수 있다.

Redis의 Memory 설정

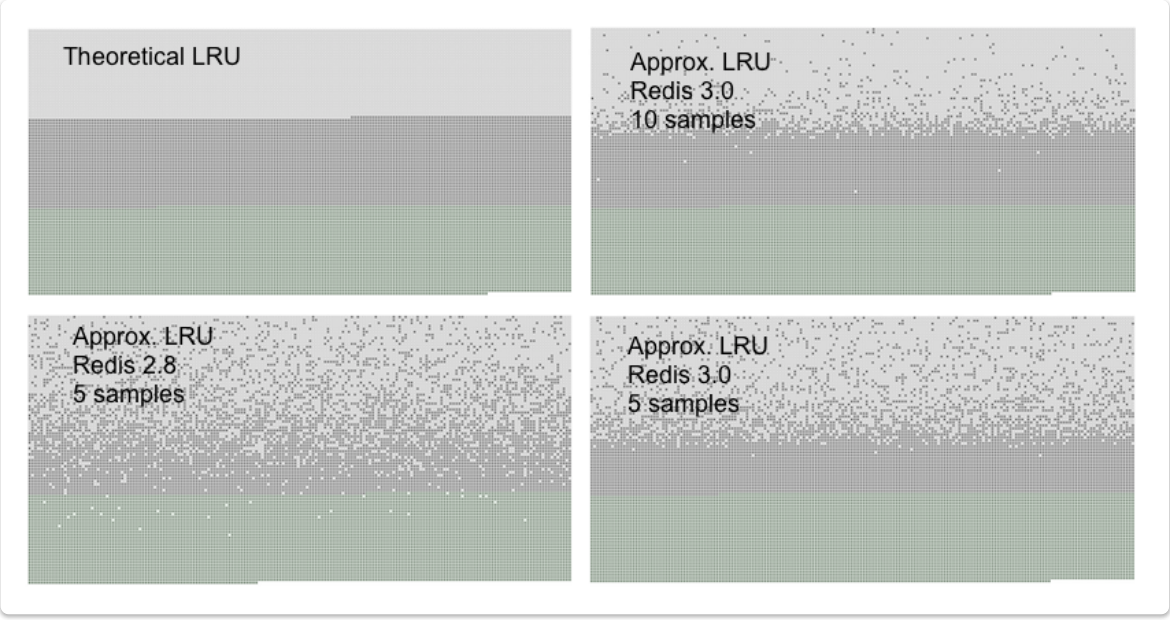
OOM 에러를 예방하기위해서 Redis의 `maxmemory`, `maxmemory-policy` 등을 설정해 줄 수 있다.

- `maxmemory <bytes>`
  - maxmemory 설정을 통해 메모리 사용량을 제한할 수 있음
  - maxmemory policy가 'noeviction'일 경우 설정할 필요없음
- `maxmemory-policy`
  - maxmemory에 도달할 때 redis가 어떻게 삭제할 것인지 policy를 선택
  - 기본 값은 'noeviction'
    - volatile-lru: TTL이 설정된 키에 LRU 적용
    - allkeys-lru: LRU 적용
    - volatile-lfu: TTL이 설정된 키에 LFU 적용
    - allkeys-lfu: LFU 적용
    - volatile-random: TTL이 설정된 키를 무작위로 삭제
    - allkeys-random: 무작위로 키를 삭제
    - volatile-ttl: 만료 시간이 가장 근접한 키를 삭제
    - noeviction: 아무것도안함(error return)

> **Note**

LRU: Least Recently Used (가장 오래된 캐시 삭제)
LFU: Least Frequently Used (가장 적게 사용된 캐시 삭제)

- `maxmemory-samples`
  - LRU, LFU, minimal TTL 알고리즘은 모두 정확히 같은 알고리즘이 아니지만 그와 근사한 알고리즘이다. 그래서 속도(speed) 또는 정확도(accuracy)를 튜닝할 수 있다.
  - 기본적으로 Redis는 5개의 key를 확인하고 최근 가장 적게 사용된 1개를 선택한다. 이때 sample size를 `maxmemory-samples` 에 설정해줄 수 있다.
  - 기본값 5로 설정하면 충분한 결과를 얻을 수 있다. 값을 10으로 설정하면 LRU와 거의 같아질 수 있지만 CPU 비용이 많이든다. 그리고 값을 3으로 설정하면 더 빨라지지만 정확도가 많이 떨어지게된다. 이때 최대 설정할 수 있는 값은 64이다.



- `lazyfree`
  - key를 삭제하기위한 두 가지 요소가 있다. 하나는 `DEL` 이라 불리고 blocking 방식으로 삭제한다. 다른 하나는 `UNLINK, FLUSHALL과 FLUSHDB 명령의 ASYNC 옵션` 과 같은 요소들을 사용하여 non-blocking 방식으로 background에서 삭제한다.
  - 아래와 같은 속성은 yes/no 값에 따라 non-blocking 방식으로 삭제할 것인지 blocking 방식으로 삭제할 것인지 설정할 수 있다.
    - `lazyfree-lazy-eviction` : eviction 수행 시 적용
    - `lazyfree-lazy-expire` : TTL과 연관된 key를 삭제 시 적용
    - `lazyfree-lazy-server-del` : 이미 존재하는 key에 data를 저장하는 명령어의 side effect에 적용한다. 예를 들어 `RENAME` 명령어는 이전 key의 content를 삭제하고 replace될 것이다.
    - `slave-lazy-flush` : 복제 중, replica가 master와 fully resynchronization을 수행할 때, 전체 데이터베이스를 삭제할 시 적용
    - `lazyfree-lazy-user-del` : `DEL` 명령어를 사용하면 내부적으로 `UNLINK` 를 기본으로 사용하도록 설정

언제 어떤 Eviction policy를 선택할까

- `allkeys-lru` policy는 요청의 인기도에 대해서 power-law distribution[1]이 예상될 경우 사용한다. 만약 무엇을 사용할지 확실하지 않다면 이 policy를 선택하는 것이 좋다 라고 한다.
- `allkeys-random` 은 지속적으로 모든 키를 ciclic access하거나 분포가 균일할 경우 사용한다.
- `volatile-ttl` 은 여러 TTL 값들을 사용할 경우 사용한다.

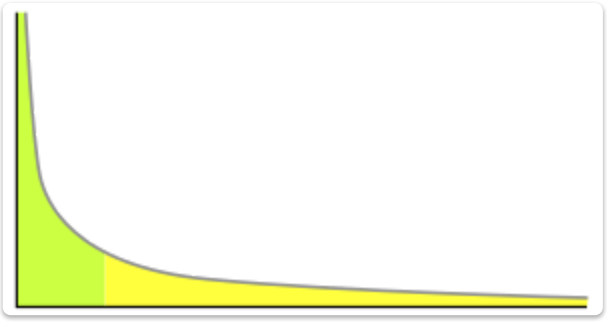
LFU mode

LFU mode는 `Morris counter`[2]라 불리는 확률 counter 알고리즘을 사용한다. LFU는 LRU와 마찬가지로 sampling하여 eviction을 진행하지만 튜닝하는 parameter에서 차이가 있다.

- lfu-log-factor: 값이 클수록 counter 값이 천천히 증가한다. 반대로 값이 작을수록 최댓값에 빠르게 도달한다. (default 10)
- lfu-decay-time: minute 단위로 값을 설정하며 설정한 시간마다 counter가 감소한다. (default 1)

Appendix

1. power-law distribution



위처럼 상위 몇 %만 대부분을 차지하고 나머지들이 long tail(오른쪽 부분)을 형성하는 모양의 분포이다.

2. Morris Counter

Morris Counter는 8-bit 레지스터를 사용한 probabilistic counter 알고리즘이다. (8bit이기 때문에 최댓값이 255가 나온다.)

수식으로 표현하면 다음과 같다.

$$X_n = log_2(1 + n)$$

여기서 n은 실제 event 개수를 의미한다. 로그 함수를 적용하기 때문에 n이 증가할 수록 X_n 값은 천천히 증가하고 값이 작을 수록 빠르게 증가하게된다.

Redis에서는 `lfu-log-factor`를 추가로 적용하는데 위 수식에 반영하면 아래와 같이 작성할 수 있다.

$$X_n = log_2(1 + n * factor)$$

위 설명을 바탕으로 factor 값이 클수록 X_n 이 천천히 증가하므로 더 많은 event를 필요하다는 것을 알 수 있다. factor 값에 따라 cache hit 개수에 대한 counter 값을 확인하면 다음과 같다.

```
+--------+------------+------------+------------+------------+------------
+
| factor | 100 hits   | 1000 hits  | 100K hits  | 1M hits    | 10M hits
|
+--------+------------+------------+------------+------------+------------
+
| 0      | 104        | 255        | 255        | 255        | 255
|
+--------+------------+------------+------------+------------+------------
+
| 1      | 18         | 49         | 255        | 255        | 255
|
+--------+------------+------------+------------+------------+------------
+
| 10     | 10         | 18         | 142        | 255        | 255
|
+--------+------------+------------+------------+------------+------------
+
| 100    | 8          | 11         | 49         | 143        | 255
|
+--------+------------+------------+------------+------------+------------
+
```

[reference]

- https://en.wikipedia.org/wiki/Power_law
- https://redis.io/docs/latest/develop/reference/eviction/
- https://redis.io/docs/latest/operate/oss_and_stack/management/config-file/
- https://arpitbhayani.me/blogs/morris-counter/
- https://arishs.medium.com/what-is-approximate-counting-can-approximate-counting-be-efficient-on-hardware-33d0fd7e1125
- https://gregorygundersen.com/blog/2019/11/11/morris-algorithm/
- http://redisgate.kr/redis/server/memory.php#stats