Redis 성능 튜닝

1 적절한 Eviction 정책 설정하기

Eviction 정책이란?

- 메모리가 한계에 도달했을 때 어떤 조치가 일어날지 결정
- 처음부터 메모리가 부족한 상황을 만들지 않는 것이 중요함
- 캐시로 사용할 때는 적절한 eviction policy가 사용될 수 있음

Redis의 메모리 관리

Memory 사용 한도 설정 => 지정하지 않으면 32bit에서는 3GB, 64bit에서는 0(무제한)으로 설정됨

```
maxmemory 100mb
```

maxmemory 도달한 경우 eviction 정책 설정

```
maxmemory-policy noeviction
```

maxmemory-policy 옵션

- noeviction: eviction 없음. 추가 데이터는 저장되지 않고 에러 발생(replication 사용시 master에 적용됨)
- allkeys-lru: 가장 최근에 사용된 키들을 남기고 나머지를 삭제(LRU: Least Recently Used)
- allkeys-lfu: 가장 빈번하게 사용된 키들을 남기고 나머지를 삭제(LFU: Least Frequently Used)
- volatile-lru: LRU를 사용하되 expire field가 true로 설정된 항목들 중에서만 삭제
- volatile-lfu: LFU를 사용하되 expire field가 true로 설정된 항목들 중에서만 삭제
- allkeys-random: 랜덤하게 삭제
- volatile-random: expire field가 true로 설정된 항목들 중에서 랜덤하게 삭제
- volatile-ttl: expire field가 true로 설정된 항목들 중에서 짧은 TTL 순으로 삭제

Redis 성능 튜닝

2 시스템 튜닝

Redis 성능 측정(redis-benchmark)

- redis-benchmark 유틸리티를 이용해 Redis의 성능을 측정할 수 있음.

```
# redis-benchmark [-h host] [-p port] [-c clients] [-n requests]
```

예제) redis-benchmark -c 100 -n 100 -t SET

```
~ % redis-benchmark -c 100 -n 100 -t SET
====== SET ======
  100 requests completed in 0.01 seconds
  100 parallel clients
  3 bytes payload
  keep alive: 1
  host configuration "save": 3600 1 300 100 60 10000
  host configuration "appendonly": no
  multi-thread: no

Latency by percentile distribution:
0.000% <= 2.463 milliseconds (cumulative count 1)
50.000% <= 7.743 milliseconds (cumulative count 50)
75.000% <= 9.663 milliseconds (cumulative count 75)
87.500% <= 9.935 milliseconds (cumulative count 88)
93.750% <= 10.159 milliseconds (cumulative count 94)
96.875% <= 10.223 milliseconds (cumulative count 97)
98.438% <= 10.567 milliseconds (cumulative count 99)
99.219% <= 10.679 milliseconds (cumulative count 100)
100.000% <= 10.679 milliseconds (cumulative count 100)

Cumulative distribution of latencies:
0.000% <= 0.103 milliseconds (cumulative count 0)
8.000% <= 3.103 milliseconds (cumulative count 8)
11.000% <= 4.103 milliseconds (cumulative count 11)
15.000% <= 5.103 milliseconds (cumulative count 15)
31.000% <= 6.103 milliseconds (cumulative count 31)
37.000% <= 7.103 milliseconds (cumulative count 37)
58.000% <= 8.103 milliseconds (cumulative count 58)
61.000% <= 9.103 milliseconds (cumulative count 61)
93.000% <= 10.103 milliseconds (cumulative count 93)
100.000% <= 11.103 milliseconds (cumulative count 100)

Summary:
  throughput summary: 9090.91 requests per second
  latency summary (msec):
          avg       min       p50       p95       p99       max
        7.514     2.456     7.743    10.183    10.567    10.679
```

Redis 성능에 영향을 미치는 요소들

- Network bandwidth & latency: Redis의 throughput은 주로 network에 의해 결정되는 경우가 많음. 운영 환경에 런치하기 전에 배포 환경의 network 대역폭과 실제 throughput을 체크하는 것이 좋음.
- CPU: 싱글 스레드로 동작하는 Redis 특성 상 CPU 성능이 중요. 코어 수보다는 큰 cache를 가진 빠른 CPU가 선호됨.
- RAM 속도 & 대역폭: 10KB 이하 데이터 항목들에 대해서는 큰 영향이 없음.
- 가상화 환경의 영향: VM에서 실행되는 경우 개별적인 영향이 있을 수 있음(non-local disk, 오래된 hypervisor의 느린 fork 구현 등)

성능에 영향을 미치는 Redis 설정

- rdbcompression <yes/no>: RDB 파일을 압축할지 여부로, CPU를 절약하고 싶은 경우 no 선택
- rdbchecksum <yes/no>: 사용시 RDB의 안정성을 높일 수 있으나 파일 저장/로드 시에 10% 정도의 성능 저하 있음
- save: RDB 파일 생성시 시스템 자원이 소모되므로 성능에 영향이 있음

Redis 성능 튜닝

3 SLOWLOG로 느린 쿼리 튜닝하기

SLOWLOG 설정

- 수행시간이 설정한 기준 시간 이상인 쿼리의 로그를 보여줌
- 측정 기준인 수행시간은 I/O 동작을 제외함

로깅되는 기준 시간(microseconds)

```
slowlog-log-slower-than 10000
```

로그 최대 길이

```
slowlog-max-len 128
```

SLOWLOG 명령어

slowlog 개수 확인

```
> slowlog len
```

slowlog 조회

```
slowlog get [count]
```

=> 일련번호, 시간, 소요시간, 명령어, 클라이언트 IP, 클라이언트 이름

```
127.0.0.1:6379> SLOWLOG LEN
(integer) 8
127.0.0.1:6379> slowlog reset
OK
127.0.0.1:6379> set aaa 123
OK
127.0.0.1:6379> slowlog get 1
1) 1) (integer) 10
   2) (integer) 1672082741
   3) (integer) 5
   4) 1) "set"
      2) "aaa"
      3) "123"
   5) "172.17.0.1:59964"
   6) ""
127.0.0.1:6379>
```