Redis Week3

Bulk loading

벌크 로딩은 이미 존재하는 대량의 데이터를 Redis에 로딩하는 과정이다.

일반적으로 레디스 클라이언트를 사용하여 bulk loading을 수행하는 것은 좋은 생각이 아닌데, 이는 하나의 명령을 차례로 전송하는 것은 모든 명령에 대한 왕복 시간이 존재하기 때문에 느리다. 파이프라이닝을 사용할 수 있지만 많은 레코드의 bulk loaging을 위해 가능한 빨리 삽입되었는지 확인하기 위해 동시에 응답을 읽어야 한다.

레디스로 대량의 데이터를 가져오는 방법은 필요한 데이터를 삽입하는 데 필요한 명령을 호출하기 위해 원시 형식의 레디스 프로토콜이 포함된 텍스트 파일을 생성하는 것이다. 를 들어 'keyN -> ValueN' 형식의 수십억 개의 키가 있는 대규모 데이터 세트를 생성해야 하는 경우 Redis 프로토콜 형식으로 다음 명령을 포함하는 파일을 생성한다.

```
SET Key0 Value0
SET Key1 Value1
...
SET KeyN ValueN
```

레디스 2.6 버전 이상에서는 redis-cli에서 대량 로딩을 수행하기 위해 설계된 redis-cli를 지원한다.

```
cat data.txt | redis-cli --pipe
```

그렇다면 다음과 같은 출력이 생성된다.

```
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 1000000
```

```
"""
    redis-mass.py
    ~~~~~~~~~~~~~
    Prepares a newline-separated file of Redis commands for mass insertion.
    from https://github.com/Squab/redis-mass-insertion
    :copyright: (c) 2015 by Tim Simmons.
    :license: BSD, see LICENSE for more details.
"""
import sys

def proto(line):
    result = "*%s\r\n$%s\r\n%s\r\n" % (str(len(line)), str(len(line[0])), line[0])
    for arg in line[1:]:
        result += "$%s\r\n%s\r\n" % (str(len(arg)), arg)
    return result

if __name__ == "__main__":
    try:
        filename = sys.argv[1]
        f = open(filename, 'r')
    except IndexError:
        f = sys.stdin.readlines()
```

```
    for line in f:
        print proto(line.rstrip().split(' ')),
```

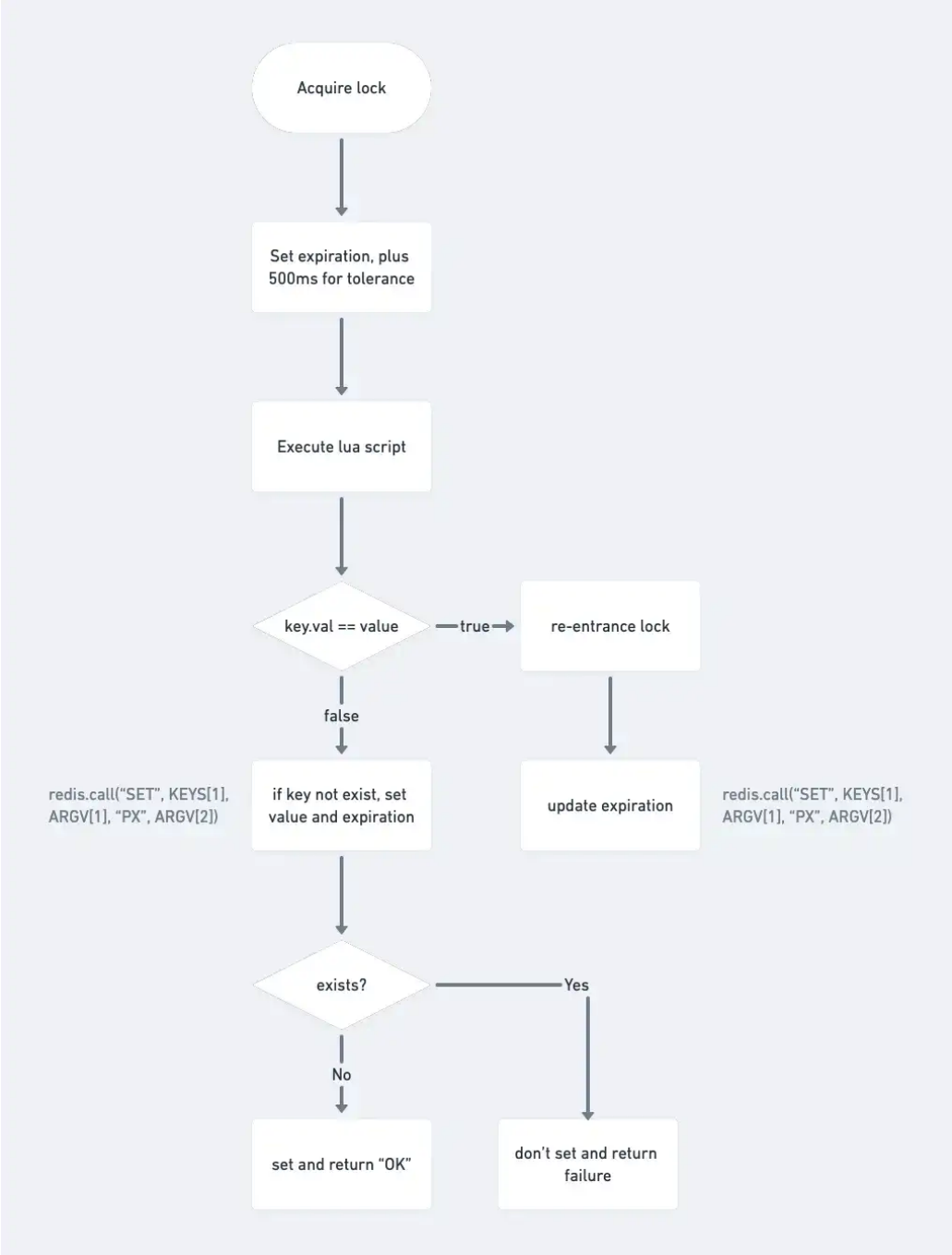
```
SET key value
SET key2 value2
SADD someset value
SADD someset value2
SADD someset value3 value4
```

```
python redis-mass.py input.txt | redis-cli --pipe
```

Distributed Locks with Redis

분산 락은 서로 다른 프로세스가 상호 배타적은 방식으로 공유 리소스와 함께 작동해야 하는 경우 사용할 수 있는 것이다. 공통된 저장소를 이용하여 자원이 사용 중인지 체크를 하고, 전체 서버에서 동기화된 처리가 가능해진다.

락을 획득하는 방법은 다음과 같다.




```
SET resource_name my_random_value NX PX 30000
```

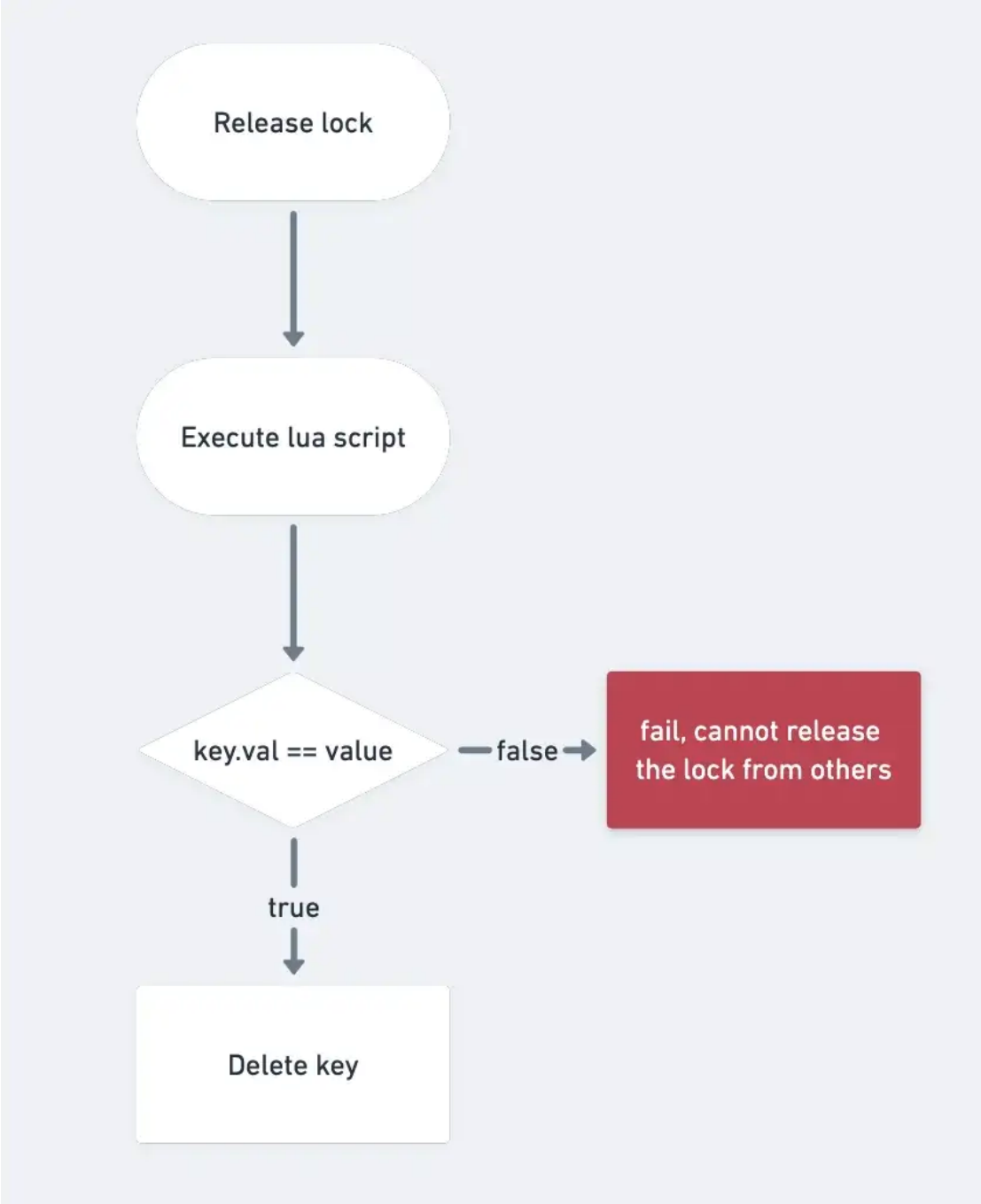
```
-- KEYS[1]: lock key
-- ARGV[1]: lock value, random string
-- ARGV[2]: expiration time
-- determine if the value held by the lock key is equal to the value passed in
-- If equal, it means that the lock is acquired again and the acquisition time is updated to prevent expiration on reentry
-- this means it is a "reentrant lock"
if redis.call("GET", KEYS[1]) == ARGV[1] then
    -- set
    redis.call("SET", KEYS[1], ARGV[1], "PX", ARGV[2])
    return "OK"
else
    -- If the lock key.value is not equal to the incoming value, it is the first time to get the lock.
    -- SET key value NX PX timeout : Set the value of the key only when the key does not exist
    -- Set success will automatically return "OK", set failure returns "NULL Bulk Reply"
```

```
    -- Why add "NX" here, because we need to prevent the lock from being overwritten by others
    return redis.call("SET", KEYS[1], ARGV[1], "NX", "PX", ARGV[2])
end
```

해당 명령은 이미 존재하지 않는 경우에만 키를 설정하고 30000밀리초 이후에 만료시킨다. 이 값은 모든 클라이언트와 락 요청에서 고유해야한다.

안전하게 락을 해제하기 위해서는 레디스에 Lua 스크립트로 함께 사용된다. 키가 존재하고 키에 저장된 값이 정확히 내가 기대하는 값인 경우에만 키를 제거하게 된다.

```
-- Release the lock
-- cannot release someone else's lock
if redis.call("GET", KEYS[1]) == ARGV[1] then
    -- return "1" for successful execution
    return redis.call("DEL", KEYS[1])
else
    return 0
end
```



레디스를 DLM(Distributed Lock Manager)으로 사용하기 위해 Redlock이라는 알고리즘을 사용한다.

Redis를 사용하여 분산 잠금을 구현하는 방법은 TTL(Time to Live)로 고유한 키를 설정하고 클라이언트가 리소스 사용을 완료하면 Redis에서 키를 삭제한다. 그리고 클라이언트 측에서 문제가 발생한 경우 Redis는 TTL을 기반으로 자동으로 잠금을 해제한다. Redlock은 단일 장애 지점을 방지하기 위해 독립적인 Redis 인스턴스가 있는 최소 3대의 시스템에서 작동하도록 설계되어있다.

Secondary indexing

redis는 인덱스를 포함하여 다양한 종류의 보조 인덱스를 생성해 인덱싱에 redis의 기능을 사용할 수 있다.

Sorted Sets as Indexes

ZSET은 고유한 멤버 셋으로 기본 레디스 데이터 유형이며 각 멤버는 score에 연결된다. 범위를 추가, 제거 및 검색하는 시간 복잡도가 상대적으로 낮기 때문에 자연스럽게 인덱스가 된다.

Using objects IDs as associated values

다른 곳에 저장된 객체의 일부 필드를 인덱싱 하기 원할 수 있다. 인덱스된 필드와 관련된 데이터를 저장하기 위해 정렬된 집합 값을 직접 사용하는 대신 객체의 ID만 저장할 수 있다.

예를 들어 사용자를 나타는 redis 해시가 있을 수 있다. 각 사용자는 ID로 직접 액세스 할 수 있는 단일 키로 표시 된다.

```
HMSET user:1 id 1 username antirez ctime 1444809424 age 38
HMSET user:2 id 2 username maria ctime 1444808132 age 42
HMSET user:3 id 3 username jballard ctime 1443246218 age 33
```

나이별로 사용자를 쿼리하기 위해 인덱스를 생성하려면 다음을 수행할 수 있다.

```
ZADD user.age.index 38 1
ZADD user.age.index 42 2
ZADD user.age.index 33 3
```

이번에는 정렬된 집합의 점수와 관련된 값이 개체의 ID이다. 따라서 인수를 사용하여 인덱스를 쿼리하면 필요한 정보 또는 유사한 명령을 검색 `ZRANGE` 해야 한다.