

Redsync Algorithm - Distributed Lock Management using Redis

Smita Vijayakumar

Agenda

- Background
- Why DLM
- Examples of DLM
- Properties of DLM
- Redsync - Redis-Based Timed Mutual Exclusion Locks
- Criticism - Lack of Fencing Token
- Discussion - Is this Design without Fencing Correct?
- Conclusion - Is Redsync the answer?

Drive the Agenda!

More questions = More Answers!
(PS: It is lovelier to learn more than to
not learn at all!)

Background - Lock Manager

- Correctness - Manipulating Shared Resource
 - ★ Example - who will process an incoming event?
 - ★ Example - who writes to the DB?
 - ★ Example - who can concatenate a file part?
- Reducing Redundancy in task processing
- Distributed Resource
 - ★ Resource that is managed by several entities

Distributed Lock Manager (DLM)

- Eliminates single point failure - Lock Manager running over a cluster
- Every node holds information on locks (Will that always be complete?)
- Differences versus a centralised solution? (Hint: PACELC)

Examples of DLM

- Apache Zookeeper
- Apache Helix
- Google's Chubby
- ETCD
- Consul
- Redsync
- ...

Properties of DLM

Safe Lock

- Exactly one client holds lock
- Lock Released - By exactly the client that currently holds it

Properties of DLM

Deadlock-free Lock

- Client waiting on a lock eventually granted

Properties of DLM

Fault Tolerant

- Minimum of $N/2 + 1$ cluster nodes up
- Else ?

Redis

- In-memory data store
- Supports strings, lists, sets, hashes, range queries...

```
root@smita-virtual-machine#  
telnet localhost 15000  
Trying 127.0.0.1...  
Connected to localhost.  
Escape character is '^]'.  
GET str  
$0  
SET str new string  
+OK  
GEt str  
$9  
newstring  
SAVE  
+OK  
ZADD set 3 "three"  
:1  
ZADD set 2 "two"  
:1
```

```
ZADD set 4 "four"  
:1  
ZRANGE set 0 2 withscores  
*6  
$5  
"two"  
$1  
2  
$7  
"three"  
$1  
3  
$6  
"four"  
$1  
4  
SAVE  
+OK
```

Redis-Based Lock

SET Command's Optional Parameters

EX seconds -- Set the specified expire time, in seconds.

PX milliseconds -- Set the specified expire time, in milliseconds.

NX -- Only set the key if it does not already exist.

XX -- Only set the key if it already exist.

SET lockName signature NX PX 30000

Question - Why is a signature needed? (Hint: The job took longer than expected, and a new node acquired the lock assuming the task wasn't finished.)

Creating Redis Connection Pool

Define Redis Pools of Connection - using Redigo of pool <https://github.com/garyburd/redigo>

For each server in list do -

```
pool[i] = &redis.Pool{
    MaxIdle:    3,
    IdleTimeout: 240 * time.Second,
    Dial: func() (redis.Conn, error) {
        return redis.Dial("tcp", server[i].addr)
    },
}
```

Creating Mutex

Couple of ways -

For each shared resource do-

```
m, err := redsync.NewMutexWithGenericPool("SharedResource", pools)
```

where `pools := []redsync.Pool{`

and `N = len(pools)`

or,

```
m, err := NewMutex"SharedResource", redisAddrs)
```

redisAddrs- List of Redis node addresses

Quorum - $N/2 + 1$ (strictly greater than one half the size of pools)

Parameters

//How long the lock is valid in seconds

m.Expiry = 4 * time.Second

//How many times to try and acquire lock

m.Tries = 10

//Delay between two successive attempts to acquire lock (ms)

m.Delay = 500 * time.Millisecond

//Clock drift factor - How far apart are the clocks?

m.Factor = 0.001

Locking the Mutex

```
err := m.Lock()
```

```
/*
```

- * 1. For each Redis node*
 - * a. gets connection*
 - * b. sets signed lock for expiry time **only if not set***
 - * c. If set, increment count*
 - * 2. If Quorum reached within validity -> Success.*
 - * 3. Else, delete all locks set in 1.*
 - * 4. Decrement retries count*
 - * 5. Pause for delay between retries.*
 - * 6. If max tries reached -> Failure. Else goto 1.*
- ```
*/
```



# Reset timeout of Mutex - Touch

```
err := m.Touch()
```

```
/*
```

```
* 1. For each Redis node
```

```
* a. gets connection
```

```
* b. sets signed lock for expiry time only if set with value
```

```
* c. If set, increment count
```

```
* 2. If Quorum reached -> Success.
```

```
* 3. Else Failure.
```

```
*/
```

# Unlocking the Mutex

```
err := m.Unlock()
```

```
/*
```

```
* 1. For each Redis node
```

```
* a. gets connection
```

```
* b. deletes signed lock only if set with value
```

```
* c. If set, increment count
```

```
* 2. If Quorum reached -> Success.
```

```
* 3. Else Failure.
```

```
*/
```

# Discussion

- Are there scenarios where safety is not guaranteed?
- What is the complexity of this solution?

# Suggested Reading and References

- Distributed locks with Redis
- How to do distributed locking
- Is Redlock safe?

Introduce Yourself!

# Tell Us...

- Your Name, where you work and background
- Your passion
- Most inspiring speaker from Go community or otherwise
- Would you like to lead the next session?