Redis Sentinel 架构优化改造

赵景波

目录

Redis Sentinel 简介

01

Sentinel 简介

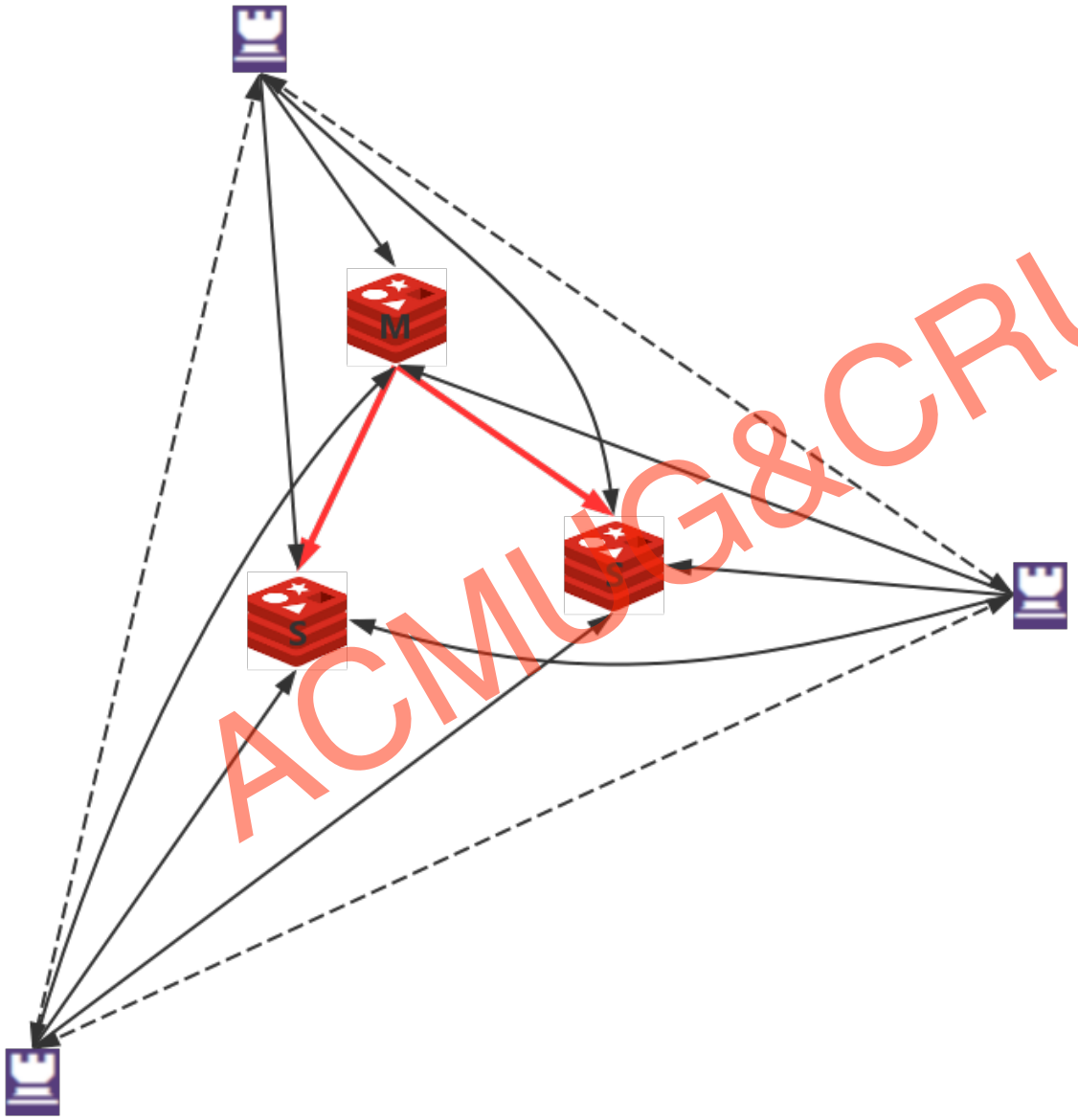
Redis Sentinel provides high availability for Redis. In practical terms this means that using Sentinel you can create a Redis deployment that resists without human intervention to certain kind of failures.

Redis Sentinel also provides other collateral tasks such as monitoring, notifications and acts as a configuration provider for clients.

This is the full list of Sentinel capabilities at a macroscopical level (i.e. the *big picture*):

- **Monitoring**. Sentinel constantly checks if your master and slave instances are working as expected.
- **Notification**. Sentinel can notify the system administrator, another computer programs, via an API, that something is wrong with one of the monitored Redis instances.
- **Automatic failover**. If a master is not working as expected, Sentinel can start a failover process where a slave is promoted to master, the other additional slaves are reconfigured to use the new master, and the applications using the Redis server informed about the new address to use when connecting.
- **Configuration provider**. Sentinel acts as a source of authority for clients service discovery: clients connect to Sentinels in order to ask for the address of the current Redis master responsible for a given service. If a failover occurs, Sentinels will **report the new address.**

Redis Sentinel 总体架构



Hello Message（pub/sub）：

- chanel:“__sentinel__:hello”
- message:"10.10.10.10,26380,b0985e88129bd106ad8b51532c64d7be02727e58,0,master23758,10.10.10.11,23758,0
- message:sentinel host,sentinel port,sentinel runid,sentinel epoch,master name,master ip,master port,master config epoch

Redis Sentinel 核心逻辑

02

切换流程

- 判断SDOWN、ODOWN
- 触发切换
- 选取Sentinel Leader
- 选取新的主库
- 其他从库指向新主库
- 完成切换
- 捕获Sentinel行为进行变更

从库选取流程

➢ 从库过滤

1、遍历老主服务器中的从库

2、忽略所有SDOWN、ODWON或者已经断线的从库

3、忽略从库优先级（slave_priority）为0的从库

4、忽略INFO回复已过期的从库

5、忽略下线的时间过长的从库

➢ 从库选取

1、Slave priority更小的从库优先

2、复制偏移量较大的从库优先

3、运行ID 较小的从库优先

4、如果运行ID相同，那么执行命令数量较多的那个从服务器会被选中。

捕获Sentinel行为

订阅sentinel 频道（hello）：

+odown

+switch-master

-failover-abort-no-good-slave

……

配置SCRIPTS（不建议）：

SCRIPTS EXECUTION

NOTIFICATION SCRIPT

CLIENTS RECONFIGURATION SCRIPT

Redis Sentinel 优化改造-上篇

03

部署架构（改造前）

- 一个sentinel集群
- 17 sentinel节点
- 南北共9个数据中心
- 监控1500+实例

IDC	数量
北方IDC1	2
北方IDC2	1
北方IDC3	2
北方IDC4	3
北方IDC5	3
北方IDC6	1
北方IDC7	1
南方IDC1	2
南方IDC2	2

现场

- Sentinel 连接失败
- Sentinel 无法进行主库切换

错误日志：

Error registering fd event for the new client: Numerical result out of range (fd=10135)

具体原因

Sentinel 连接数打满：

- 大量无用连接未释放
- Sentinel本身连接数过高

Sentinel在执行核心逻辑中会遍历所有监控的主库，每次遍历都会取出监控该主库的Sentinel进行依次发送ping等命令，目前sentinel监控了420个端口，总共17个sentinel节点，所以每个sentinel上应该有的连接数为：

420*(17-1)= 6720

Sentinel 默认连接数在10000左右

连接数优化


```
================================================================================
Redis 3.2.1      Released Fri Jun 17 15:01:56 CEST 2016
================================================================================

Upgrade urgency HIGH: Critical fix to Redis Sentinel, due to 3.2.0 regression
                      compared to 3.0.

Hey, this is Redis 3.2.1, and this release should bring some grain of
maturity to Redis 3.2. The list of commits following this note will tell

Upgrade urgency HIGH: Critical fix to Redis Sentinel, due to 3.2.0 regression
                      compared to 3.0.

Hey, this is Redis 3.2.1, and this release should bring some grain of
maturity to Redis 3.2. The list of commits following this note will tell
you the details, but the main things addressed in this release are the
following:

1. A critical bug in Sentinel was hopefully fixed. During the big 3.2
   refactoring of Redis Sentinel, in order to implement connection sharing
   to make Sentinel able to scale better (few Sentinels to monitor many
   masters), a bug was introduced that mis-counted the number of pending
   commands in the Redis link. This in turn resulted into an inability to talk
   with certain Redis instances. A common result of this bug was the inability
   of Redis Sentinel to reconfigure back the old master, after a failover,
   when it is reachable again, as the slave of the new master. This was due
   to the inability to talk with the old master at all.
```

连接数优化

优化前：

单个Sentinel连接总数量 = (总Sentinel数量-1) * 监控的Master 数量

优化后：

单个Sentinel连接总数量 = 总Sentinel数量 -1

针对两个Sentinel之间的连接，之前是采用以监控Master的维度建立，目前是只建立一个连接所有的Master共享。

优化改造

- 设置timeout参数
- 设置maxclients参数
- 升级Sentinel版本至3.2.8版本

Redis Sentinel 优化改造-下篇

04

问题现场

- Sentinel 多次切换失败
- Sentinel 选取Sentinel Leader 脑裂
- Sentinel 频繁进入TITL模式

部署架构改造

改造前：

IDC	Sentinel 数量
北方IDC1	2
北方IDC2	1
北方IDC3	2
北方IDC4	3
北方IDC5	3
北方IDC6	1
北方IDC7	1
南方IDC1	2
南方IDC2	2

改造后：

IDC	Sentinel 数量
北方IDC1	1
北方IDC2	1
北方IDC3	2
北方IDC4	2
北方IDC5	3
北方IDC6	0
北方IDC7	0
南方IDC1	0
南方IDC2	0

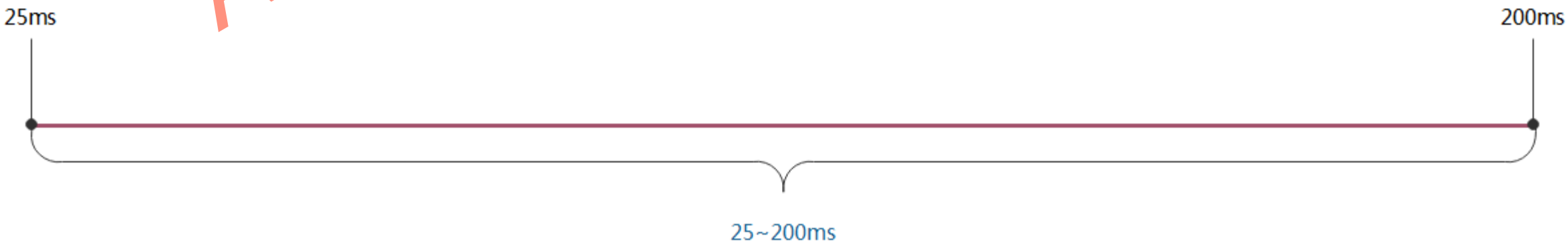
目前集群总共监控1000+ 端口，3000+实例。

逻辑优化

调整核心逻辑执行频率间隔：

1s/(10~20)HZ

1s/(5~40)HZ



新增参数：

sentinel.config_min_hz = 5;
sentinel.config_max_hz = 31;

频率变更：

变更前：
server.hz = REDIS_DEFAULT_HZ + rand() % REDIS_DEFAULT_HZ;
变更后：
server.hz = sentinel.config_min_hz + rand() % sentinel.config_max_hz;

逻辑优化

调整 sentinel 在选主的时候发现元数据中其他的sentinel没有选主时就给自己投一票，这里修改为不投给自己，等待下一轮核心逻辑执行：

```
...  @@ -3728,8 +3728,10 @@ char *sentinelGetLeader(sentinelRedisInstance *master, uint64_t epoch) {
3728          * common voted sentinel, or for itself if no vote exists at all. */
3729         if (winner)
3730             myvote = sentinelVoteLeader(master,epoch,winner,&leader_epoch);
3731 +           /*
3732         else
3733             myvote = sentinelVoteLeader(master,epoch,sentinel.myid,&leader_epoch);
3734 +           */
3735
```

逻辑优化

脑裂之后由获得投票数最多的节点快速发起下一轮投票：

sentinelFailoverWaitStart function

```
4036            if (mstime() - ri->failover_start_time > election_timeout) {
4037                sentinelEvent(LL_WARNING,"-failover-abort-not-elected",ri,"%@");
4038                sentinelAbortFailover(ri);
4039    +           /* if the sentinel is the winner which get the most votes,
4040    +            * it will retry next failover quickly */
4041    +           char *winner = sentinelGetWinner(ri, ri->failover_epoch);
4042    +           if (winner && strcasecmp(winner, sentinel.myid) == 0) {
4043    +               ri->failover_fast_retry_flag = true;
4044    +           }
4045    +           sdsfree(winner);
4046            }
4047            return;
4048        }
```

sentinelStartFailoverIfNeeded function

```
3885        if (master->flags & SRI_FAILOVER_IN_PROGRESS) return 0;
3886
3887        /* Last failover attempt started too little time ago? */
3888    +   if (!master->failover_fast_retry_flag &&
3889    +           mstime() - master->failover_start_time <
3890    +           master->failover_timeout*2)
3891        {
3892            if (master->failover_delay_logged != master->failover_start_time) {
3893                time_t clock = (master->failover_start_time +
```

https://github.com/antirez/redis/issues/4334

Redis Sentinel 使用建议

04

使用建议

- 单个IDC的数量不建议超过(Sentinel数量-quorum)
- 客户端长连接问题
- 可以通过设置slave-priority 控制选举（跨南北部署）
- 防止误切，切换灵敏度控制（quorum、down-after-milliseconds、failover-timeout）
- 设置maxclient、timeout参数
- 建议采用Sentinel 3.2.8 及以上版本
- 单套Sentinel集群监控Redis实例数量建议少于1500

Operations

sina 新浪