Zcloud

etcd

王燕卫

2019-2-27

目 录

1 etcd 介绍

1.1 简介

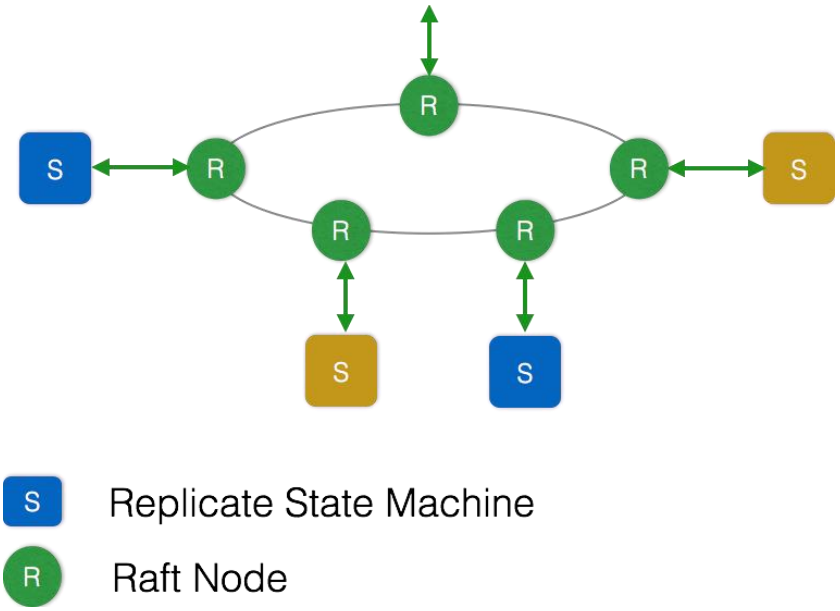
etcd 是一个可靠的分布式 kv 数据库，多用于配置共享和服务发现，特点：

- 简单：定义明确，面向用户的 API（gRPC）
- 安全：支持 SSL 证书验证（可选）
- 快速：基准测试 10,000 次/秒
- 可信：使用 Raft 算法保证多节点数据一致性

开发语言：go，coreos 公司开发，现在是 CNCF 的一个开源项目。

1.2 原理

ETCD 使用 Raft 协议来维护集群内各个节点状态的一致性。简单说，ETCD 集群是一个分布式系统，由多个节点相互通信构成整体对外服务，每个节点都存储了完整的数据，并且通过 Raft 协议保证每个节点维护的数据是一致的。



如图所示，每个 ETCD 节点都维护了一个状态机，并且，任意时刻至多存在一个有效的主节点。主节点处理所有来自客户端写操作，通过 Raft 协议保证写操作对状态机的改动会可靠的同步到其他节点。

ETCD 工作原理核心部分在于 Raft 协议。本节接下来将简要介绍 Raft 协议。
Raft 协议主要分为三个部分：选主，日志复制，安全性。

1.2.1 选主

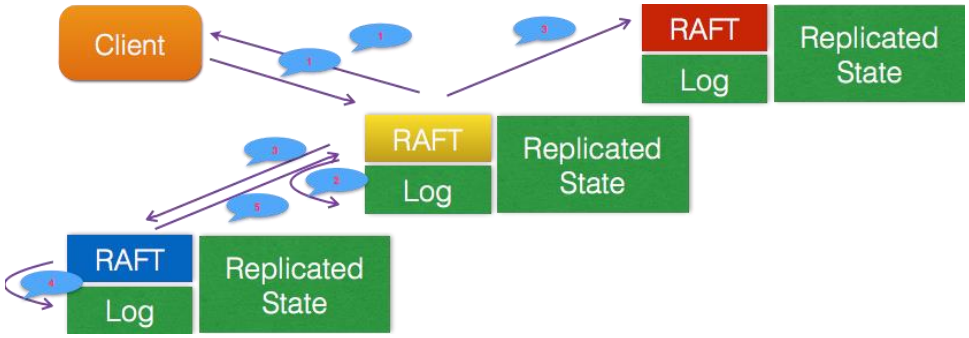
Raft 协议是用于维护一组服务节点数据一致性的协议。这一组服务节点构成一个集群，并且有一个主节点来对外提供服务。当集群初始化，或者主节点挂掉后，面临一个选主问题。集群中每个节点，任意时刻处于 Leader，Follower，Candidate 这三个角色之一。选举特点如下：

- 当集群初始化时候，每个节点都是 Follower 角色；
- 集群中存在至多 1 个有效的主节点，通过心跳与其他节点同步数据；
- 当 Follower 在一定时间内没有收到来自主节点的心跳，会将自己角色改变为 Candidate，并发起一次选主投票；当收到包括自己在内超过半数节点赞成后，选举成功；当收到票数不足半数选举失败，或者选举超时。若本轮未选出主节点，将进行下一轮选举（出现这种情况，是由于多个节点同时选举，所有节点均为获得过半选票）。
- Candidate 节点收到来自主节点的信息后，会立即终止选举过程，进入 Follower 角色。

  为了避免陷入选主失败循环，每个节点未收到心跳发起选举的时间是一定范围内的随机值，这样能够避免 2 个节点同时发起选主。

1.2.2 日志复制

所谓日志复制，是指主节点将每次操作形成日志条目，并持久化到本地磁盘，然后通过网络 IO 发送给其他节点。其他节点根据日志的逻辑时钟(TERM)和日志编号(INDEX)来判断是否将该日志记录持久化到本地。当主节点收到包括自己在内超过半数节点成功返回，那么认为该日志是可提交的(committed），并将日志输入到状态机，将结果返回给客户端。
这里需要注意的是，每次选主都会形成一个唯一的 TERM 编号，相当于逻辑时钟。每一条日志都有全局唯一的编号。



主节点通过网络 IO 向其他节点追加日志。若某节点收到日志追加的消息，首先判断该日志的 TERM 是否过期，以及该日志条目的 INDEX 是否比当前以及提交的日志

的 INDEX 跟早。若已过期，或者比提交的日志更早，那么就拒绝追加，并返回该节点当前的已提交的日志的编号。否则，将日志追加，并返回成功。
当主节点收到其他节点关于日志追加的回复后，若发现有拒绝，则根据该节点返回的已提交日志编号，发生其编号下一条日志。
主节点像其他节点同步日志，还作了拥塞控制。具体地说，主节点发现日志复制的目标节点拒绝了某次日志追加消息，将进入日志探测阶段，一条一条发送日志，直到目标节点接受日志，然后进入快速复制阶段，可进行批量日志追加。
按照日志复制的逻辑，我们可以看到，集群中慢节点不影响整个集群的性能。另外一个特点是，数据只从主节点复制到 Follower 节点，这样大大简化了逻辑流程。

1.2.3 安全性

截止此刻，选主以及日志复制并不能保证节点间数据一致。试想，当一个某个节点挂掉了，一段时间后再次重启，并当选为主节点。而在其挂掉这段时间内，集群若有超过半数节点存活，集群会正常工作，那么会有日志提交。这些提交的日志无法传递给挂掉的节点。当挂掉的节点再次当选主节点，它将缺失部分已提交的日志。在这样场景下，按 Raft 协议，它将自己日志复制给其他节点，会将集群已经提交的日志给覆盖掉。
这显然是不可接受的。
其他协议解决这个问题的办法是，新当选的主节点会询问其他节点，和自己数据对比，确定出集群已提交数据，然后将缺失的数据同步过来。这个方案有明显缺陷，增加了集群恢复服务的时间（集群在选举阶段不可服务），并且增加了协议的复杂度。
Raft 解决的办法是，在选主逻辑中，对能够成为主的节点加以限制，确保选出的节点已定包含了集群已经提交的所有日志。如果新选出的主节点已经包含了集群所有提交的日志，那就不需要从和其他节点比对数据了。简化了流程，缩短了集群恢复服务的时间。
这里存在一个问题，加以这样限制之后，还能否选出主呢？答案是：只要仍然有超过半数节点存活，这样的主一定能够选出。因为已经提交的日志必然被集群中超过半数节点持久化，显然前一个主节点提交的最后一条日志也被集群中大部分节点持久化。当主节点挂掉后，集群中仍有大部分节点存活，那这存活的节点中一定存在一个节点包含了已经提交的日志了。
至此，关于 Raft 协议的简介就全部结束了。

官方 demo：http://play.etcd.io/play

2 etcd 集群搭建

2.1 集群节点数量与网络分割

ETCD 使用 RAFT 协议保证各个节点之间的状态一致。根据 RAFT 算法原理，节点数目越多，会降低集群的写性能。这是因为每一次写操作，需要集群中大多数节点将日志落盘成功后，Leader 节点才能将修改内部状态机，并返回将结果返回给客户端。

也就是说在等同配置下，节点数越少，集群性能越好。显然，只部署 1 个节点是没什么意义的。通常，按照需求将集群节点部署为 3，5，7，9 个节点。
这里能选择偶数个节点吗？ 最好不要这样。原因有二：

- 偶数个节点集群不可用风险更高，表现在选主过程中，有较大概率或等额选票，从而触发下一轮选举。
- 偶数个节点集群在某些网络分割的场景下无法正常工作。试想，当网络分割发生后，将集群节点对半分割开。此时集群将无法工作。按照 RAFT 协议，此时集群写操作无法使得大多数节点同意，从而导致写失败，集群无法正常工作。

当网络分割后，ETCD 集群如何处理的呢?

- 当集群的 Leader 在多数节点这一侧时，集群仍可以正常工作。少数节点那一侧无法收到 Leader 心跳，也无法完成选举。
- 当集群的 Leader 在少数节点这一侧时，集群仍可以正常工 作，多数派的节点能够选出新的 Leader, 集群服务正常进行。

当网络分割恢复后，少数派的节点会接受集群 Leader 的日志，直到和其他节点状态一致。

2.2 etcd 参数说明

列举一些重要的参数，以及其用途。

- —data-dir 指定节点的数据存储目录，这些数据包括节点 ID，集群 ID，集群初始化配置，Snapshot 文件，若未指定—wal-dir，还会存储 WAL 文件；
- —wal-dir 指定节点的 was 文件的存储目录，若指定了该参数，wal 文件会和其他数据文件分开存储。
- —name 节点名称
- —initial-advertise-peer-urls 告知集群其他节点 url.
- — listen-peer-urls 监听 URL，用于与其他节点通讯
- — advertise-client-urls 告知客户端 url, 也就是服务的 url
- — initial-cluster-token 集群的 ID
- — initial-cluster 集群中所有节点

2.3 实验示例

1) 搭建一个 3 个节点的 etcd 集群，3 个节点的 ip 分别为：192.168.134.211、192.168.134.212、192.168.134.213

   安装 etcd 后，用以下脚本分别在各个节点上启动 etcd 进程

```
TOKEN=token-0
CLUSTER_STATE=new
NAME_1=etcd-node-1
NAME_2=etcd-node-2
NAME_3=etcd-node-3
HOST_1=192.168.134.211
HOST_2=192.168.134.212
HOST_3=192.168.134.213
CLUSTER=${NAME_1}=http://${HOST_1}:2380,${NAME_2}=http://${HOST_2}:2380,${NAME_3}=http://${HOST_3}:2380
THIS_NAME=${NAME_1}
THIS_IP=${HOST_1}
etcd --data-dir=data.etcd --name ${THIS_NAME} \
        --initial-advertise-peer-urls http://${THIS_IP}:2380 --listen-peer-urls http://${THIS_IP}:2380 \
        --advertise-client-urls http://${THIS_IP}:2379 --listen-client-urls http://${THIS_IP}:2379 \
        --initial-cluster ${CLUSTER} \
        --initial-cluster-state ${CLUSTER_STATE} --initial-cluster-token ${TOKEN}
~
```

注意不同节点需修改 THIS_NAME 和 THIS_HOST 变量

2) 查看集群节点和健康状态

```
$ ETCDCTL_API=3 etcdctl --endpoints 192.168.134.211:2379,192.168.134.212:2379,192.168.134.213:2379  member list
2cba3e56b0f15a2, started, etcd-node-3, http://192.168.134.213:2380, http://192.168.134.213:2379
33b15cfab58a64d2, started, etcd-node-2, http://192.168.134.212:2380, http://192.168.134.212:2379
82a0d3cb30d896bd, started, etcd-node-11, http://192.168.134.211:2380, http://192.168.134.211:2379
```

```
$ ETCDCTL_API=3 etcdctl --endpoints 192.168.134.211:2379,192.168.134.212:2379,192.168.134.213:2379  endpoint status
192.168.134.211:2379, 82a0d3cb30d896bd, 3.3.12, 20 kB, true, 36, 9
192.168.134.212:2379, 33b15cfab58a64d2, 3.3.12, 20 kB, false, 36, 9
192.168.134.213:2379, 2cba3e56b0f15a2, 3.3.12, 20 kB, false, 36, 9
```

3) 测试增删改查

```
$ ETCDCTL_API=3 etcdctl --endpoints 192.168.134.211:2379,192.168.134.212:2379,192.168.134.213:2379  put my-key "hello wenzi"
OK

# root @ wangyanwei_ubuntu in ~/zcloud [9:54:15]
$ ETCDCTL_API=3 etcdctl --endpoints 192.168.134.211:2379,192.168.134.212:2379,192.168.134.213:2379  get my-key
my-key
hello wenzi
```

```
$ ETCDCTL_API=3 etcdctl --endpoints 192.168.134.211:2379,192.168.134.212:2379,192.168.134.213:2379  watch my-key
PUT
my-key
hello wangyanwei
```

```
$ ETCDCTL_API=3 etcdctl --endpoints 192.168.134.211:2379,192.168.134.212:2379,192.168.134.213:2379  del my-key
1
```

4) 测试 leader 节点故障后，重新选举

```
$ grep leader nohup.out
2019-03-06 09:17:04.642613 I | raft: raft.node: 33b15cfab58a64d2 elected leader 82a0d3cb30d896bd at term 36
2019-03-06 10:23:36.349392 I | raft: raft.node: 33b15cfab58a64d2 lost leader 82a0d3cb30d896bd at term 37
2019-03-06 10:23:37.750475 I | raft: 33b15cfab58a64d2 became leader at term 38
2019-03-06 10:23:37.750496 I | raft: raft.node: 33b15cfab58a64d2 elected leader 33b15cfab58a64d2 at term 38
```

2.4 注意事项

1) Etcd 单次请求的大小限制，默认是 1.5mb，可通过参数进行修改
2) Etcd 容量有限制，默认 2GB，达到容量限制后，无法写入数据，也可以通过参数进行指定

3 etcd 运维

3.1 节点迁移

在生产环境中，不可避免遇到机器硬件故障。当遇到硬件故障发生的时候，我们需要快速恢复节点。ETCD 集群可以做到在不丢失数据的，并且不改变节点 ID 的情况下，迁移节点。
具体办法是：
1）停止待迁移节点上的 etc 进程；
2）将数据目录打包复制到新的节点；
3）更新该节点对应集群中 peer url，让其指向新的节点；
4）使用相同的配置，在新的节点上启动 etcd 进程

3.2 备份恢复

3.2.1 备份

#ETCDCTL_API=3 etcdctl --endpoints $ENDPOINT snapshot save snapshot.db
执行命令后会在—data-dir 目录下生成 snapshot.db 文件（etcd 集群数据快照）

3.2.2 恢复

#ETCDCTL_API=3 etcdctl --endpoints $ENDPOINT snapshot restore snapshot.db
执行命令后会将—data-dir 目录下的 snapshot.db 快照文件加载至 etcd 集群中

3.3 性能监控

3.3.1 Metrics 说明

Etcd 的 metrics 主要包括如下几类：

1) etcd namespace metrics：用于监控和告警，以 etcd 开头
2) server：描述 etcd server 状态，以 etcd_server 开头
3) disk：描述硬盘操作状态，以 etcd_disk 开头
4) network：描述网络状态，以 etcd_network 开头
5) gRPC requests：描述 gRPC 远程调用相关，包含 grpc_server 和 grpc_client 两类
6) etcd_debugging：etcd debug 信息，以 etcd_debugging 开头
7) snapshot：快照相关，以 snapshot 开头

详见 https://github.com/etcd-io/etcd/blob/master/Documentation/metrics.md

3.3.2 主要监控项 metric

1) leader_changes_seen_total：leader 变化总次数，如果 leader 变化很频繁说明 etcd 集群不稳定
2) proposals_committed_total：节点记录的已提交的 raft 共识总数，若某成员节点持续大幅滞后于 leader 节点，说明此节点不健康
3) proposals_applied_total：节点已应用的 raft 共识总数，与该节点的 proposals_committed_total 通常差异很小（几千），若差异持续增大，说明 etcd 过载
4) proposals_pending：表示提交的队列数量，如果该指标持续上升说明 etcd 负载过高或成员无法提交
5) wal_fsync_duration_seconds：应用之前记录 wal 日志的延迟
6) backend_commit_duration_seconds：后端 commit 延迟
7) server_is_leader：节点是否是 leader
8) mvcc_db_total_size：etcd db 大小
9) debugging_mvcc_keys_total：etcd keys 总数量

3.4 etcdctl 使用说明

etcd 官方命令行工具，常见使用方法如下：

3.4.1 全局 flags

```
ETCDCTL_DIAL_TIMEOUT=3s
ETCDCTL_CACERT=/tmp/ca.pem
ETCDCTL_CERT=/tmp/cert.pem
ETCDCTL_KEY=/tmp/key.pem
ETCDCTL_API=3
```

可以设置为环境变量使用，也可以在 etcdctl 命令最前携带全局 flag

3.4.2 常用操作

1) 增加、修改键值对

   #ETCDCTL_API=3 etcdctl put “test” “test”

2) 获取一个 key 的值

   #ETCDCTL_API=3 etcdctl get “test”

3) Watch 一个 key 的值变化

   #ETCDCTL_API=3 etcdctl watch “test”

4) 保存快照

#ETCDCTL_API=3 etcdctl snapshot save snapshot.db

5) 还原快照

#ETCDCTL_API=3 etcdctl snapshot restore snapshot.db

6) 查看集群健康状态和集群状态

#ETCDCTL_API=3 etcdctl endpoint health

#ETCDCTL_API=3 etcdctl endpoint status

7) 查看集群成员列表

#ETCDCTL_API=3 etcdctl member list

4 版本管理

etcd 官方会持续维护两个稳定版本，目前是 3.3 版本和 3.2 版本，kubernetes 最低要求的 etcd 版本：3.1+、3.2+、3.3+

4.1 3.2.24—>3.3.12

4.1.1 版本差异

1) 不兼容差异（important）

详见官方升级指导：https://github.com/etcd-io/etcd/blob/master/Documentation/upgrades/upgrade_3_3.md

2) 2019-02-07

Etcdctl：Strip out insecure endpoints from DNS SRV records when using discovery with etcdctl v2

3) 2019-01-11

gRPC Proxy：Fix memory leak in cache layer

Security, Authentication：Disable CommonName authentication for gRPC-gateway gRPC-gateway proxy requests to etcd server use the etcd client server TLS certificate. If that certificate contains CommonName we do not want to use that for authentication as it could lead to permission escalation.

4) 2018-10-10

Improved：

Improve "became inactive" warning log, which indicates message send to a peer failed.

Improve read index wait timeout warning log, which indicates that local node might have slow network.

Add gRPC interceptor for debugging logs; enable etcd --debug flag to see per-request debug information.

Add consistency check in snapshot status. If consistency check on snapshot file fails, snapshot status returns "snapshot file integrity check failed..." error

Metrics, Monitoring：add some Prometheus metrics

Clent v3：Fix logic on release lock key if cancelled in clientv3/concurrency package

2)-4)均为兼容特性增加或 bug 修复

4.1.2 升级方法

1) 按照官方升级指导，检查是否存在不兼容配置，若有，请提前准备升级后的配置文件，并在新版本中进行测试配置正确性
2) 备份集权数据
3) One by one 逐台升级集群各节点的 etcd 版本，可平滑升级，不影响业务

4.2 3.2.24—>3.2.26

4.2.1 版本差异

1) 2019-01-11

   gRPC Proxy：Fix memory leak in cache layer

   Security, Authentication：Disable CommonName authentication for gRPC-gateway gRPC-gateway proxy requests to etcd server use the etcd client server TLS certificate. If that certificate contains CommonName we do not want to use that for authentication as it could lead to permission escalation.

2) 2018-10-10

   Improved：

   Improve "became inactive" warning log, which indicates message send to a peer failed.

   Improve read index wait timeout warning log, which indicates that local node might have slow network.

   Add gRPC interceptor for debugging logs; enable etcd --debug flag to see per-request debug information.

   Add consistency check in snapshot status. If consistency check on snapshot file fails, snapshot status returns "snapshot file integrity check failed..." error

Metrics, Monitoring：add some Prometheus metrics

Clent v3：Fix logic on release lock key if cancelled in clientv3/concurrency package

2)-4)均为兼容特性增加或 bug 修复

4.2.2 升级方法

1) 备份集权数据

2) One by one 逐台升级集群各节点的 etcd 版本，可平滑升级，不影响业务