etcd：基本介绍

李嘉睿

中间件团队

2020.9

主要内容

- 什么是 etcd
- 为什么使用 etcd
- etcd 一些实现细节
- 如何使用 etcd
- 一些仍需解决的问题

什么是 etcd

etcd 是一个强一致的分布式键值存储。

强一致

- 基于 raft 共识算法，单一 raft 备份组，无分片
- 线性读写: leader 分配所有备份节点的全局读写顺序，同时保证读总是能读到最新的值

数据模型：键值存储 + 版本

扁平的键空间模拟文件目录结构。

- 持久化: 键值对存储在持久化 b+ 树中，支持对键的范围查找
- 内存 b 树缓存指向键值对的指针
- MVCC: 每次数据版本更新添加数据增量部分

为什么使用 etcd

元数据存储

单一 raft 备份组，无分片。

- 无分片，水平扩展能力较差，支持数 GB 数据 (默认 2GB，最大 8GB)，因此海量数据存储需要使用 NewSQL 数据库。
- 无分片，无需分片备份组之间两阶段提交以及分布式锁的开销，性能更好。
- etcd 的使用举例：k8s 利用 etcd 存储元数据，tikv 的 PD 利用 etcd 存储 region 位置等元数据信息。

分布式协调

开箱即用的分布式协调原语。

- 提供监视器、租约、leader 选举和分布式锁的支持。
- 简单易用，支持 Restful 接口，可以从命令行使用分布式协调服务。
- 使用 gRPC 框架，已经有多种语言的 API 支持或客户端实现。

为什么使用 etcd

本质上，etcd 和 zookeeper 解决了相同的问题。其比较如下：

比较

- 数据模型：etcd 使用键值对，支持范围查找，使用 role-based 访问控制。zk 使用树形 znode 结构，其内包含 ACL 访问控制列表。
- 存储限制：均最多支持几 GB 数据的存储。
- 并发原语：etcd 内置并发原语，zk 使用外部客户端库 curator。
- 读操作：zk 不支持线性化读，读操作可能读到过时数据。
- MVCC：etcd 支持 MVCC，zk 不支持。
- 监视器通知：etcd 支持范围键值的监视器。
- API 支持：etcd 支持 HTTP/JSON API，zk 不支持。
- RPC 框架：etcd 使用 grpc 框架，zk 使用自己定制的 rpc 协议。

为什么使用 etcd

etcd 和 NewSQL 数据库的比较 (以 TiKV 为例)。虽然 TiKV 最初的 Raft 实现是对 etcd 的 Raft 实现的 Rust 版重写，但是这两种 kv 存储的应用场景有较大差异。其比较如下：

比较

- 实现动机：TiKV 为解决传统单机数据库难以（高效）支持分布式（事务）场景的问题，面向分布式数据库存储。etcd 专注集群元数据存储以及分布式协调。
- 数据模型：相同，均为分布式有序 Map。etcd 单机存储引擎为 boltDB，TiKV 单机存储引擎为 RocksDB，均和应用程序直接绑定，boltDB 使用 LSM 树作为存储结构而 RocksDB 使用 b+ 树作为存储结构。
- 存储模型：TiKV 对数据分片（默认 64MB 数据为一个 Region），一个 Region 通常有 3 个或 5 个备份，存在于一个 Raft 组中，因此 TiKV 支持的是 multi-raft，需要独立的节点管理组件。
- 存储限制：etcd 最多 8GB，TiKV 理论上可以无限水平扩容。

etcd 其它细节：raft 与 etcd 的 raft 实现

raft 的 CP 保证

- Leader 选举：利用 quorum 解决 split-brain 问题
- 日志备份：每一个 term 内，Leader、followers 数据的一致性
- Leader 选举约束：新旧 Leader 间数据的一致性

etcd 其它细节：raft 与 etcd 的 raft 实现

etcd 的 raft 实现：引入 learner 提升 A(vailablity)

首先看一下 raft 的 A 的问题

- 新成员加入使得 leader 处于过载
- 网络分区，当 leader 发现失去和 quorum 的连接，反转回 follower
- 情形
  1. 集群 2+2 分裂
  2. 2+1 分区时加入新节点
- 集群错误配置，加入一个错误的 url，并未添加对应的节点，造成节点成员身份改变

引入 learner 的好处，所谓 learner 即不参与投票的集群成员（当 learner 和 leader 的日志同步后会转为 follower）

- learner 不处理读写，即不路由客户端请求到 leader
- learner 日志和 leader 一致之后，向 leader 发出 member promote 请求

etcd 其它细节：客户端实现

grpc1.0

客户端为每个 endpoint 维护一个 TCP 连接，第一个成功建立连接的 endpoint 作为"pinned address". 多个 TCP 连接有利于更快的故障恢复，但是需要更多资源。

grpc1.7

首先尝试连接所有 endpoints，维护第一个成功连接的 TCP 连接。遇到错误时，由客户端的错误处理器 (error handler) 决定重连或选择新的服务器地址。需要维护 endpoints 状态列表，其中不健康状态的判定存在误判的可能，即被标记为不健康的节点可能在之后恢复健康但客户端无法获取该信息。

grpc1.23

客户端为每个 endpoint 维护一个 TCP 连接，通过轮转 (round robin) 负载均衡，通过 gRPC 链式拦截器实现重连. 仍未解决:网络分区情况下阻塞，缺少集群健康情况查询服务。

思考中的问题

数据迁移：

- 添加新节点，让一个旧的副本节点 (可以是 leader，raft 有直接的 LeaderTransfer 命令支持) 脱离集群。
- 利用 learner 节点，在不影响（参与投票的）总副本数的情况下创建副本。例：TiKV 依靠 learner 节点，在日志完成同步后，调度集群（PD）发出 PromoteLearner 和 RemoveNode 命令，让副本节点脱离集群。
- mirror maker: etcdctl 内置特性，利用 watch 实现，数据实时同步，低延迟，但因为不是集群的一个节点，因此不能保证数据修改顺序
- tradeoff：使用 mirror 低延迟跨数据中心备份数据，使用 learner 保持所有历史数据以及修改顺序。

思考中的问题

集群监控：

- etcd 原生导出 metrics 接口，使用 Prometheus 进行监控，Grafana 进行监控数据可视化。

思考中的问题

利用 etcd 实现故障时主从服务的切换：

- 基于 etcd 提供的分布式锁和租约实现。只有持有锁的节点能够提供服务，同时设定锁的租约持有时间。正常情况下主服务器持有锁并周期性续约，在出现故障且租约超时时，备服务器获取锁成为服务的提供者。

下一步仍需解决的一些问题

- 数据增量添加的意义和 b+ 树的联系
- 软件事务内存
- 基于角色的鉴权
- API 原语的实际使用
- MVCC 的源码级别实现及应用
- 性能指标
- 数据迁移
- 双活
- 注册中心
- 配置中心

欠缺的一些能力

- 对 Golang 的理解和使用，一些常用的 go 并发模式
- 源码阅读能力
- 性能调优方法（go pprof，Vtune）
- 其它分布式知识（分布式事务，分布式协调，paxos）
- 分布式系统理论和实践的结合
- 其它分布式系统（ZK，TiKV，k8s、Docker）

下一步的规划

- golang 基础
- 学习一些测试方法（go-tpc，etcd 自带的 benchmark 工具）
- 学习性能调优
- 希望尽快具备解决一些问题的能力

谢谢

ppt 和做的一些笔记已经上传到 github 仓库中。
`https://github.com/Willendless/internship`
欢迎随时给出修改建议。