**Goulette 分布式消息队列设计文档**

**1. 系统概述**

Goulette 是一个基于 Go 语言开发的高性能分布式消息队列系统，采用 gRPC 进行组件间通信。系统设计目标：

* **高吞吐低延迟**：支持大规模消息处理
* **持久化保证**：确保消息不丢失
* **高可用性**：自动故障转移和恢复
* **水平扩展**：支持动态扩容
* **负载均衡**：智能分配消息处理任务

**2. 系统架构**

text

+------------+ +-----------------+ +----------------+

| 客户端 |<---->| 哨兵集群 |<---->| Broker集群 |

| (生产者/消费者)| | (服务发现/负载均衡) | | (消息存储/处理) |

+------------+ +-----------------+ +----------------+

↑ ↑ ↑

| | |

+------------+ +-----------------+ +----------------+

| 消息滑动窗口 | | 一致性哈希环 |<---->| 持久化存储引擎 |

+------------+ +-----------------+ +----------------+

**3. 模块设计**

**3.1 客户端模块**

**功能**：

* 提供生产者(Producer)和消费者(Consumer)接口
* 管理消息发送/接收的滑动窗口
* 与哨兵和Broker集群交互

**关键流程**：

1. **初始化连接**：
   * 向哨兵集群发送RPC请求，获取Topic对应的Broker地址列表
   * 与相关Broker建立gRPC双向流连接
   * 发送元信息声明Topic
2. **生产者消息发送**：
   * 为每条消息分配唯一递增ID
   * 通过滑动窗口控制并发发送数量
   * 等待Broker返回ACK确认
   * 超时或失败时自动重试
3. **消费者消息拉取**：
   * 从多个Broker并行拉取消息
   * 使用滑动窗口管理未确认消息数量
   * 支持消息处理完成后发送确认

**3.2 Broker模块**

**功能**：

* 接收/发送消息
* 持久化存储消息
* 管理内存缓存加速访问
* 处理副本间数据同步
* 执行数据复制任务

**3.2.1 持久化存储引擎**

**设计方案**：

* **逻辑结构**：
  + Topic划分为多个Fragment(片段)
  + 每个Fragment是一个文件，包含多条消息
  + Fragment按时间顺序分配连续递增ID
* **写入流程**：
  1. 消息先写入WAL(预写日志)
  2. 定期刷盘(如每2秒)
  3. 按Topic将WAL内容写入对应Fragment
  4. 达到大小或数量限制时创建新Fragment
* **读取流程**：
  + 基于副本索引分配读取责任
  + 例如3副本场景：
    - Broker1负责fragment 0,3,6...
    - Broker2负责fragment 1,4,7...
    - Broker3负责fragment 2,5,8...
  + 消费者从所有Broker并行拉取其负责的Fragment

**改进方案**：

1. **WAL优化**：
   * 实现分段WAL（每64MB创建新文件）
   * 双缓冲区设计：活跃缓冲区接收写入，后台缓冲区刷盘
   * 支持同步写选项（重要消息强制刷盘）
2. **副本同步协议**：
   * 采用类Raft协议选举Leader
   * Producer只向Leader写入
   * Leader同步日志到Follower
   * 多数派确认后提交消息

**3.2.2 服务接口**

**提供的RPC服务**：

* **SendMessage**：接收生产者消息（双向流）
* **PullMessage**：提供消费者拉取消息（流式）
* **ReplicateFragment**：传输Fragment文件（副本恢复）
* **SyncMessage**：副本间实时消息同步

**3.3 哨兵模块**

**功能**：

* 服务发现与注册
* 一致性哈希环维护
* 故障检测与恢复
* 元数据缓存管理

**3.3.1 一致性哈希环**

**设计方案**：

* **Broker注册**：
  + 每个Broker映射为多个虚拟节点
  + 基于Broker地址生成虚拟节点标识
  + 计算哈希值插入哈希环
* **Broker查找**：
  + 计算Topic哈希值
  + 在环上查找后续N个不同Broker
  + N为副本数量
* **故障转移**：
  + 检测到Broker宕机时移除节点
  + 重新分配受影响Topic的副本
  + 触发数据复制流程

**改进方案**：

* **智能缓存策略**：

plaintext

BrokerCache {

主缓存: Topic -> [Broker列表]

健康状态: Broker -> 状态(健康/可疑/宕机)

备用队列: 历史可用Broker列表

}

* + 返回结果前过滤不健康节点
  + 定期健康检查更新状态
  + 备用队列提供快速回退

**3.3.2 操作日志同步**

**设计方案**：

* **状态同步机制**：
  + 所有状态变更记录为操作日志
  + 日志存储在嵌入式etcd中
  + 哨兵节点监听日志变化
  + 应用日志到本地哈希环
* **新节点加入**：
  1. 从etcd获取完整操作历史
  2. 重放所有操作重建状态
  3. 开始监听新日志变更

**容错机制**：

* 使用etcd事务保证日志一致性
* 幂等操作设计避免重复执行
* 定期创建检查点加速恢复

**3.3.3 服务接口**

**提供的RPC服务**：

* **KeepAlive**：Broker心跳维持
* **GetBrokers**：查询Topic对应Broker
* **RegisterBroker**：新Broker注册
* **UnregisterBroker**：Broker下线

**4. 关键流程设计**

**4.1 消息生产流程**

1. Producer向Sentinel查询Topic的Broker列表
2. Sentinel返回健康Broker列表
3. Producer与Broker Leader建立gRPC流
4. 发送元信息(Topic)
5. 通过滑动窗口发送消息（附带递增ID）
6. Broker：
   * 写入WAL
   * 复制到Follower
   * 多数派确认后提交
7. 返回ACK给Producer
8. Producer滑动窗口前移

**4.2 消息消费流程**

1. Consumer向Sentinel查询Topic的Broker列表
2. Sentinel返回所有副本Broker
3. Consumer连接所有相关Broker
4. 发送元信息(Topic)
5. Broker根据副本索引确定负责的Fragment
6. 从内存缓存或磁盘读取消息
7. 流式返回给Consumer
8. Consumer滑动窗口管理处理速率

**4.3 故障恢复流程**

1. Sentinel检测Broker宕机（心跳超时）
2. 从哈希环移除故障节点
3. 更新缓存标记节点不可用
4. 客户端请求触发副本重建：
   * 从健康副本获取数据分片
   * 并行传输到新Broker
   * 校验数据完整性
5. 新Broker加入哈希环
6. 客户端重连到新Broker

**4.4 数据复制流程**

1. 识别需要复制的Fragment
2. 将Fragment分成1MB大小的分片
3. 从多个副本并行拉取不同分片
4. 新Broker接收并重组分片
5. 校验哈希值确保完整性
6. 增量复制只传输差异部分

**5. 改进建议总结**

**5.1 持久化优化**

* **WAL分段管理**：避免单个文件过大
* **双缓冲刷盘**：活跃缓冲区接收写入，后台缓冲区异步刷盘
* **同步写选项**：关键消息支持同步刷盘保证持久化

**5.2 副本一致性**

* **Raft协议实现**：Leader选举+日志复制
* **多数派提交**：确保数据安全
* **Follower读优化**：支持只读查询分散负载

**5.3 缓存策略**

* **多级健康检查**：
  + 快速PING检测（秒级）
  + 深度健康检查（分钟级）
* **智能降级**：部分节点不可用时仍提供服务
* **缓存预热**：新节点加入时预加载热点数据

**5.4 数据复制**

* **分片并行传输**：加速大数据迁移
* **增量复制**：仅同步差异数据
* **校验和验证**：确保数据完整性

**6. 部署建议**

**6.1 集群配置**

| **组件** | **数量** | **说明** |
| --- | --- | --- |
| Sentinel | 3-5 | 奇数节点组成集群 |
| Broker | N+1 | 根据负载动态扩展 |
| etcd | 3-5 | 与Sentinel同机部署 |

**7. 监控指标**

**7.1 关键指标**

* **Broker**：
  + 消息吞吐量（入站/出站）
  + 持久化延迟（WAL到磁盘）
  + Fragment使用率
  + 副本同步延迟
* **Sentinel**：
  + 请求响应时间
  + 缓存命中率
  + 哈希环变更频率
* **客户端**：
  + 消息端到端延迟
  + ACK等待时间
  + 滑动窗口利用率

**8. 容错设计**

**8.1 故障场景处理**

1. **Broker故障**：
   * 自动从副本组移除
   * 触发数据重新平衡
   * 新节点自动补充
2. **Sentinel故障**：
   * etcd保证操作日志不丢失
   * 客户端重试其他节点
   * 新节点自动同步状态
3. **网络分区**：
   * 多数派原则维持服务
   * 分区恢复后自动合并
   * 基于时间戳解决冲突

**8.2 数据安全**

* **多副本存储**：至少3副本
* **定期快照**：每小时全量快照
* **跨机架分布**：副本分布在不同机架
* **端到端校验**：消息完整性验证