# v0.1 ups到v0.2 ups的日志转发工具

## 1 日志转发工具的目的和功能

v0.2尚不具备同城多机房实时同步功能，我们可以做一个简单、独立的小工具， 把(v0.1的)UpdateServer上的commit log准实时地读出来，再发给其他的机群的UpdateServer机器上的类似的接收小工具， 写入其commit log文件，并由其UpdateServer进行重放？(这个小工具也可用来进行异地，例如杭州到青岛，的准实时同步)。 如果这样，v0.2版可以得到较多的测试，为v0.1版升级到0.2版打好基础。

上面说的是最原始的动机和解决思路，我们把这种解决思路称为日志转发。

为了便于讨论，我们把要解决的问题列出来：

1. 提供一种多机房准实时同步的方法。
2. 在线测试v0.2版 oceanbase。

这两个问题，特别是问题2，看似有一种更合理的解决方法，就是复制客户端的请求，将同一个请求发给v0.1的同时复制一份发给v0.2，但是只取v0.1的结果。对测试而言，这种方法毫无疑问更彻底。 但是有两点需要注意:

1. 请求复制的方案很难保证v0.1和v0.2的同步，因为客户端发送的请求被v0.1和v0.2处理的顺序可能不一样.
2. 客户端代码已经被部署在多个地方，不宜改动。

第二点使得请求复制成为不可取的方案，并且我们在此优先考虑多机房准实时同步的问题，所以放弃请求复制的方案。本文接下来讨论日志转发的实现方案。

### 1.1 交互过程

日志转发的过程涉及到三个实体, v0.1 ups, v0.2 ups及日志转发工具。 首先确定的是:

* v0.1 ups会与日志转发工具交互，从而工具可以获得v0.1的commit log， 不妨把这个过程称为fetch过程;
* 日志转发工具会与v0.2 ups交互，从而工具可以将日志发送给v0.2 ups， 不妨把这个过程成为push过程.
* v0.1 ups和v0.2 ups不需要交互

C:\Users\yuanqi.xhf\Desktop\notes_files\data-flow-simple.png

需要说明的是

1. v0.1是需要正常提供读写服务的，并且v0.1已经部署和运行，在进行日志转发的过程中不能要求重启v0.1，更不能要求更改v0.1的代码.
2. 尽量少改动v0.2的代码, 一方面为了减少工作量，另一方面也是测试的需求。现阶段v0.2 ups不能单独运行，为了完成每日合并的功能，至少需要rootserver和chunkserver的参与。

这两点告诉我们: v0.1 ups和v0.2 ups都是需要作为完整的ocean base instance中的一个组件存在的。 注意的是v0.1和v0.2的 ups采用HA实现高可用，对外表现为一个IP。

### 1.2限制

虽然不需要实时同步，但需要保证已同步的数据完全一致。这一点对日志转发过程做了如下限制:

1. v0.2 ob instance不能对外提供写服务，也不允许有多个日志转发过程同时对一个v0.2 ups进行。
2. 日志必须按v0.1 ups处理的顺序转发给v0.2 ups（比如插入和删除两个操作的顺序不同，最后的作用效果是不同的），这使得多线程转发难以实现。
3. 如果存在非幂等的操作(比如累加操作)，则要保证v0.2不能重复处理同一条v0.1的日志。 之所以同一条日志会被v0.2 ups多次收到，是因为我们假定转发过程会因为故障断掉，下次重发时不好确定上次v0.2 ups已经收到了多少日志。
4. 更一般地，如果操作依赖于已有的数据，比如条件更新，则要进一步保证v0.2和v0.1具有完全相同的初始数据。

前两条限制很容易满足，只要把日志转发工具做成单线程的，并且保证v0.2一次只接受一个日志转发工具发送的请求即可。 第三条限制比较麻烦，后面会详细讨论，为了简便，我们称其为断点续传的问题。第四条限制不用考虑(因为v0.1不支持条件更新)。

## 2 日志转发三方的交互过程及方案的选择

为了实现日志转发，先启动V0.1和V0.2的OB Instance, 再启动日志转发工具.

我们先讨论一下日志转发工具和v0.1 v0.2 ups的交互， 最后据此确定工具启动时需要的参数，

### 2.1 Fetch过程

v0.1和v0.2 OB Instance不要求实时同步， 日志转发工具与v0.1 ups的唯一交互就是工具会读取v0.1 ups commit log，v0.1 ups不知道工具的存在，不需要更改代码。 启动日志转发工具时需要指定v0.1 ups commit log的目录，为了简单起见：

1. 要求此目录可以直接被日志同步工具作为本地文件读取，最简单的实现方法是要求日志同步工具和v0.1 ups在同一台机器上。如果VIP漂移了会有问题吗？
2. 同步日志的点也由命令行给出，这里所谓同步日志的点是指需要转发的第一条日志的序号。人工怎么选定这个点? 断点续传时，这个点如何确定/是否应该指定？ 这个问题依赖于我们怎么实现push过程。留在后面讨论。

### 2.2 Push过程

日志转发工具和v0.2 ups通过网络通信，工具将commit log发送给v0.2 ups。 这里有个问题需要进一步确定: 工具发送给v0.2 ups的是commit log本身（类似于主备ups之间的日志同步），还是写请求(即作为客户端发送的ups mutator)？ 下面分别就这两种方案予以讨论：

1. ups已经有接受commit log并应用修改的代码了， 如果选择发送commit log本身, 首先一个优点是不用解析commit log的具体内容。 我们按照是否更改v0.2 ups代码考虑实现方案:
   1. 如果不改动v0.2 ups的代码:
      * 接受commit log的任务只能由slave ups处理，并且slave ups必须在向master注册成功后才能开始接受master ups发送的日志, 这即是说日志转发工具必需处理slave ups的注册请求。
      * 另一方面，只有master ups才会向root server报告生成了major version freeze, 并触发“每日合并"的过程, 这样日志转发工具必须参与"每日合并"。这样日志转发工具将变得很复杂。

第二点使得不改动v0.2 ups代码的方案变得不可行。

* 1. 如果选择修改v0.2 ups的代码, 我们需要给v0.2 ups增加一种新角色，不妨称这种新角色为passive master。
     + passive master需要提供master ups的所有功能，但有以下例外:
     + passive master拒绝客户端的写请求，可以提供读请求。
     + passive master允许接受一个日志转发工具发送日志的请求，并且只允许响应一个工具的请求。

这种代码改动是可以接受的。后面会详细讨论这种方案。

1. ups可以处理客户端发送的修改请求, 如果选择将日志转发工具以客户端的角色发送写请求，首先一个缺点是日志转发工具需要解析commit log的具体内容。 我们按照是否更改v0.2 ups的代码考虑实现方案, 在这之前明确一点：为了安全，目前ups不允许客户端用table id/column id的方式，指定需要修改的cell, 但commit log中保存的ups mutator都是用table id/column id指定cell的。
   1. 如果不修改v0.2 ups代码:
      * 日志转发工具需要schema信息将v0.1 ups commit log中的table id/columnd id转变为table name/column name. 这种方案对用户不友好。

初步决定放弃这种放案。

* 1. 如果修改v0.2 ups代码, 避开schema信息，有两种方法:
     + 允许v0.2 ups接受任何客户端发送的ups mutator中用table id/column id指定cell。这种方案对v0.2 ups代码的修改最小，但破坏了安全性。
     + 增加一种trusted client, v0.2 ups允许trusted client发送的ups mutator中用table id/column id指定cell， 让日志工具作为trusted client向v0.2 ups发送写请求。这种方案实现较难。

初步决定放弃这种方案。

例外虽然实现断点续传有多种方法，但不管哪种方法，我们都需要每条日志或每次修改有唯一标识，且有序，然而客户端发送的写请求没有唯一标识，且无序，所以日志转发工具不宜以客户端的角色发送写请求。

综上，为了实现push过程，初步决定修改v0.2 ups代码，给v0.2 ups增加新的passive master的角色。

### 2.3 工具预定义的命令行接口

启动日志转发工具之前，需要保证v0.1和v0.2 ob instance都已经启动，日志转发工具需要v0.1 ups commit log的目录及需要转发的第一条日志的序号， v0.2 ups的地址(ip和port)，不需要schema信息。

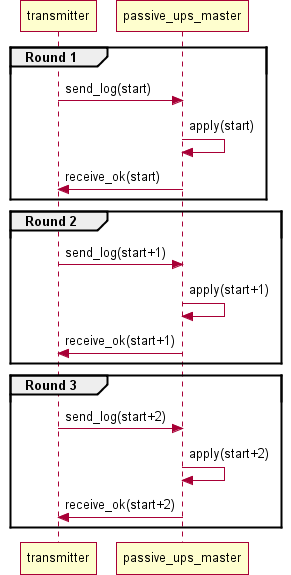
./ups\_log\_transmitter v0.1-ups-commit-log-dir start-log-id v0.2-ups-ip v0.2-ups-port

## 3 日志转发工具和Passive Ups Master的交互

在日志转发过程中: passive ups master先启动，是被动方; 日志转发工具后启动，是主动方。即总是日志转发工具主动将日志转发给passive ups master.

### 3.1 不考虑断点续传的问题

不考虑断点续传，过程非常简单。 一个可能的交互过程如下:



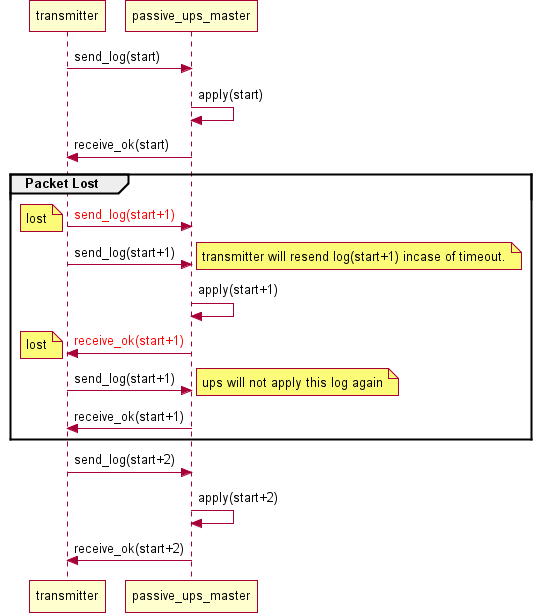
### 3.2 考虑断点续传

考虑断点续传，则我们要先明确造成“断点"的原因, 有以下几种可能:

1. 日志转发工具和passive ups master进程都活着，但是网络断了， 这时工具只需要在发出 send\_log(i) 请求超时后，重发即可。 这样passive ups master可能收到多条重复的 send\_log(i) ， 这种情况下passive ups master只要有一种方法保证不会重复应用这条log即可，换言之它需要记录已经收到并应用的所有日志的编号， 由于它收到且应用的日志编号连续且递增，所以只需要记录最大的一条日志编号即可。

值得注意的是由于v0.2 ups写日志使用了Direct IO. 所以日志中可能会添加额外的log作为padding，所以v0.1 ups的第 i 条日志可能对应于v0.2 ups的第 i+k 条日志。 所以不能以v0.2 ups commit log记录的最后一条日志的编号作为已收到并应用的最后一条日志的编号。

一个可能的交互过程如下:

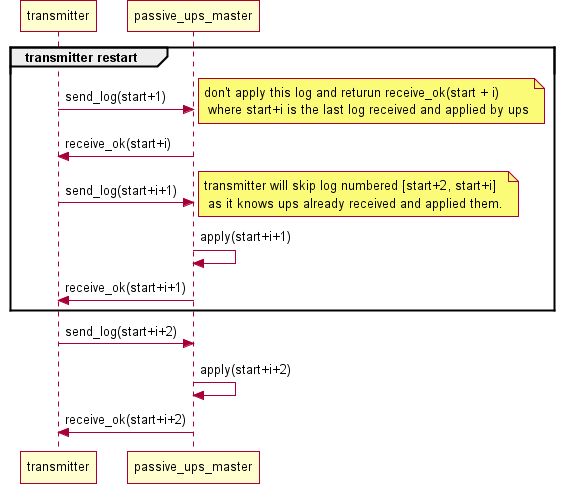


1. 日志转发工具因为bug死掉或被用户杀掉， 但是passive ups master一直活着。日志转发工具重启之后继续发送日志，这时从那条日志继续发送呢？ 由于passive ups master保证不会重复应用同一条日志，所以日志转发工具只需要从一个足够早的major freeze点开始发送就可以了。

为了优化这一过程，日志转发工具可以考虑转发之前询问一下passive ups master上次同步的最后一条日志编号是几。这样passive ups master需要处理一种新的查询同步点的请求。

实际上，在完成1的功能的基础上稍作变化：passive ups master在收到日志后总是返回已应用的最后一条日志的编号(实际上是确认了所有小于这个编号的日志都已被接受并被应用)，这样协议可以更简单一些。 日志转发工具收到passive ups master的响应后，会立即跳到该编号指定的日志的下一条日志。

一个可能的交互过程如下:



1. passive ups master死掉，需要用户介入，做如下操作:
   1. 杀掉日志转发工具
   2. 回滚到上一个major free点： 只需要清空passive ups master的commit log及sstable。
   3. 重启v0.2 ob instance
   4. 重启日志转发工具，通过命令行指定合适的日志同步点。

保证这种方案可行的一种简单方法是保证v0.1和v0.2的major freeze点是一致的，否则用户很难确定v0.2的最后一个major freeze点对应的v0.1 ups的日志编号。

1. v0.1 ups死掉，对日志转发过程没影响。

## 4 人工选择v0.1日志同步点的方法

在我们选择了上述的日志同步方法后，可以确定日志同步点的选择方法了:

1. 第一次启动工具时，指定从上一次major freeze点开始同步即可，不妨记这条日志编号为 start .
2. 如果重启日志转发工具，我们依然需要指定一个同步点，并且这个同步点的日志编号要足够小(只需要取 start 即可), 日志同步工具会从这个同步点开始同步，但是会立即跳过passive ups master已经收到并应用的日志。 需要注意编号为 start 的日志所在的文件可能已经被删除了，这种情况下，日志同步工具跳到现有的编号最小的日志(log\_min)，从它开始同步， 如果passive ups master已接受的日志编号都小于 log\_min-1 ， 这说明日志同步工具下线太久，有一部分日志因为被删除已经无法同步了。这种情况程序无法处理。

## 5 概要设计

按照上面的设计，日志转发过程中只涉及到两种包: send\_log(i) receive\_ok(i)， 假设 passive ups master 已经收到并应用的最大日志编号为 k , 日志转发工具准备发送的日志编号为 i :

1. 日志转发工具每次发送一个 send\_log(i) ， 必须等待 receive\_ok(k) （k>=i） 包之后才会接着发送 send\_log(k+1) 的包， 或者超时之后重发 send\_log(i)
2. passive ups master 收到 send\_ok(i) 的包之后，必须立即发送 receive\_ok(k) , 如果超时，不必重发.

### 5.1 网络发包的优化

使用TCP传输数据时，只要连接不断，应用级是不需要关心网络包丢失，超时，也不需要自己管理重发，上面提到的超时重发是应对TCP连接断掉之后的情况的。 这种情况是很少见的，所以，我们完全可以把它作为一种异常情况处理，正常情况下不需要 receive\_ok(k) 确认包已收到。

有以下的异常情况:

1. k > i 说明日志转发工具准备发送 ups 已经收到的日志, ups 应该通知日志转发工具跳过已经收到的日志；
2. k < i-1 说明日志转发工具准备跳过 ups 还未收到的日志， ups 应该通知日志转发工具重发还未收到的日志。

在异常情况下， ups 发送 receive\_log(k) 包给日志转发工具, 日志转发工具收到响应后，重设准备发送的日志编号为 k+1 即可。

### 5.2 日志转发工具的实现

启动两个线程，一个线程负责读取v0.1 ups commit log, 将其转发给v0.2 ups; 另一个线程负责接受v0.2 ups的"异常响应", 两个线程共享一个变量 i, 表示下一步准备转发的日志编号。

1. 当我们把所有已有的日志完全转发之后，我们可以用 inotify\_init()/inotify\_add\_watch() 系统调用监控v0.1 ups commit log的变化。
2. 必须确保要被转发，但还未被转发并且未被打开的日志文件不能被移动或删除。

### 5.3 v0.2 Ups的更改

添加一种新的角色， passive ups master, 这个角色可以在 ups master 的基础上改，具体修改如下:

1. passive master拒绝客户端的写请求，可以提供读请求。这意味着要关闭写线程。
2. passive master允许接受一个日志转发工具发送日志的请求,并且只允许响应一个工具的请求(这点不一定由程序逻辑保证)。接受日志后要应用修改，并且转发给 slave ups, 这一步可能改动大一些。
3. passive master不要主动生成major version.