多机房同步协程版本开发

==========

作者: 刘绍宸

邮箱: liubaoan@mirrtalk.com

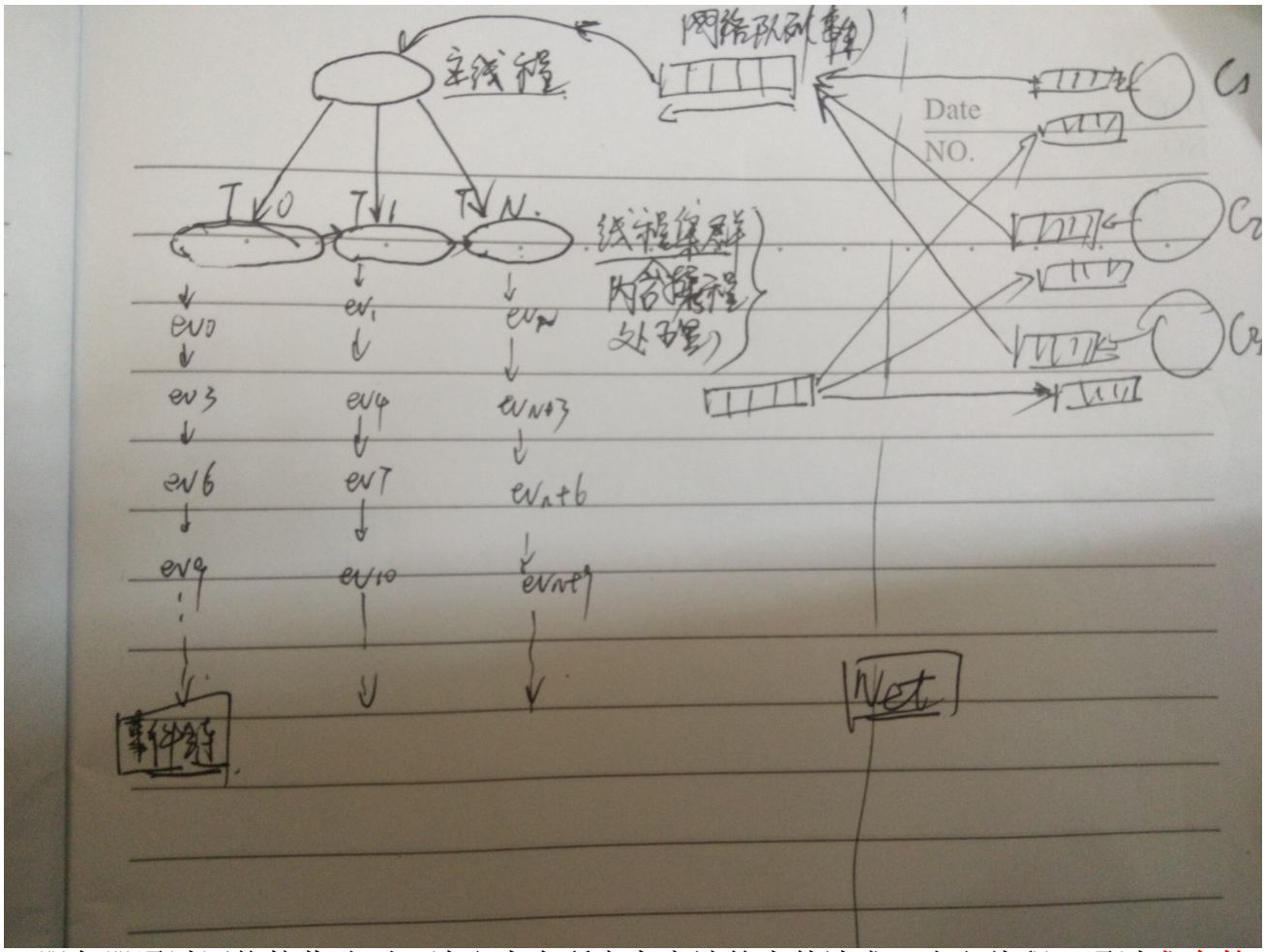
微信: 164776775

代码参考:

supex/programs/pole-M/src/event_handler.c

==========

【整体框架流程解析】



服务器通过网络接收队列，读取来自所有客户端的事件请求；由主线程，通过**求余算法**，将不同的客户端网络事件，分配到不同的子线程（由链表缓存，子线程的个数由配置文件决定）；

事件到子线程后，就进入了一个死循环中，不断的将链表中所有的事件请求,添加到协程集群中，由协程高效处理；（宏观上一个线程可以处理很多个客户端的请求）

最后，将处理的结果，发送到网络发送队列，并发送到客户端；

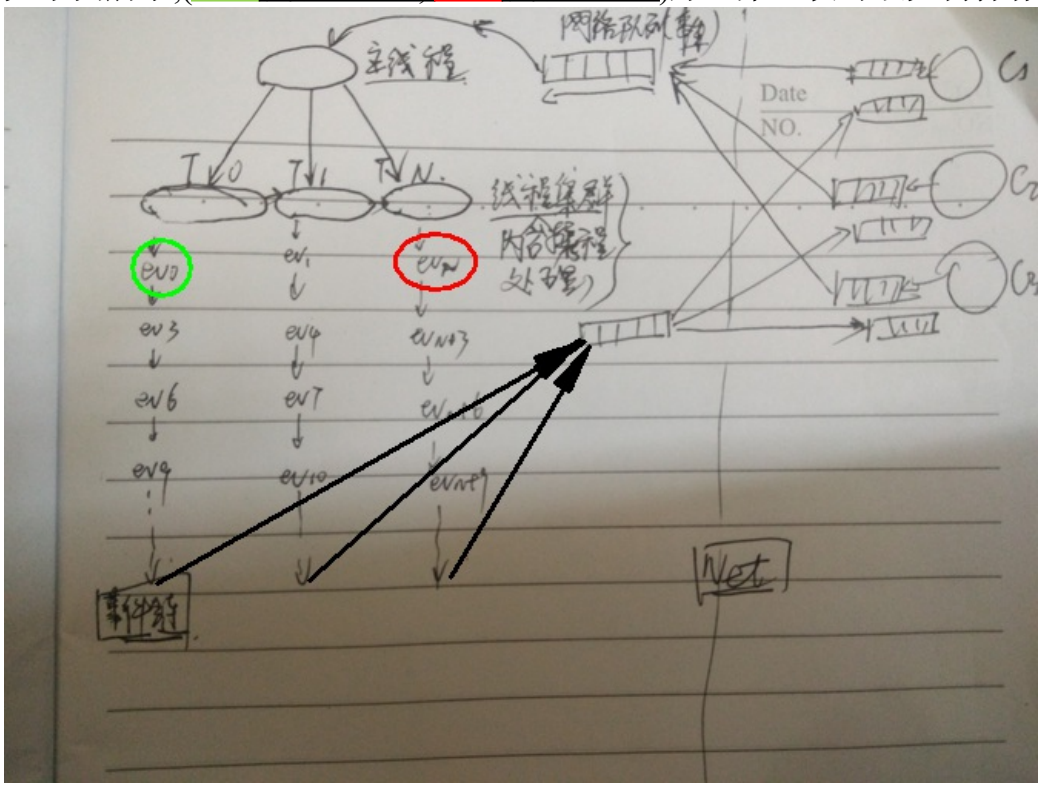
【流程解析】

【前提】

目前，做的这个要支持多机房同步，所以，首先得指定一个能够执行数据库拷贝的客户端标识；我们就设定为 **"master"**,第一次,master标识的客户端就直接执行增量同步;

1.master客户端,第一次就可以直接执行增量同步,并在服务器端创建状态数据;
2.其他客户端,第一次请求必须是DUMP操作,并且目标必须是已经在执行增量同步的客户端;
3.之后启动的客户端即可以向master请求DUMP,也可以向node1请求DUMP;

如下图所示,(绿色代表Node0, 红色代表master)那么第一次时该如何操作呢?

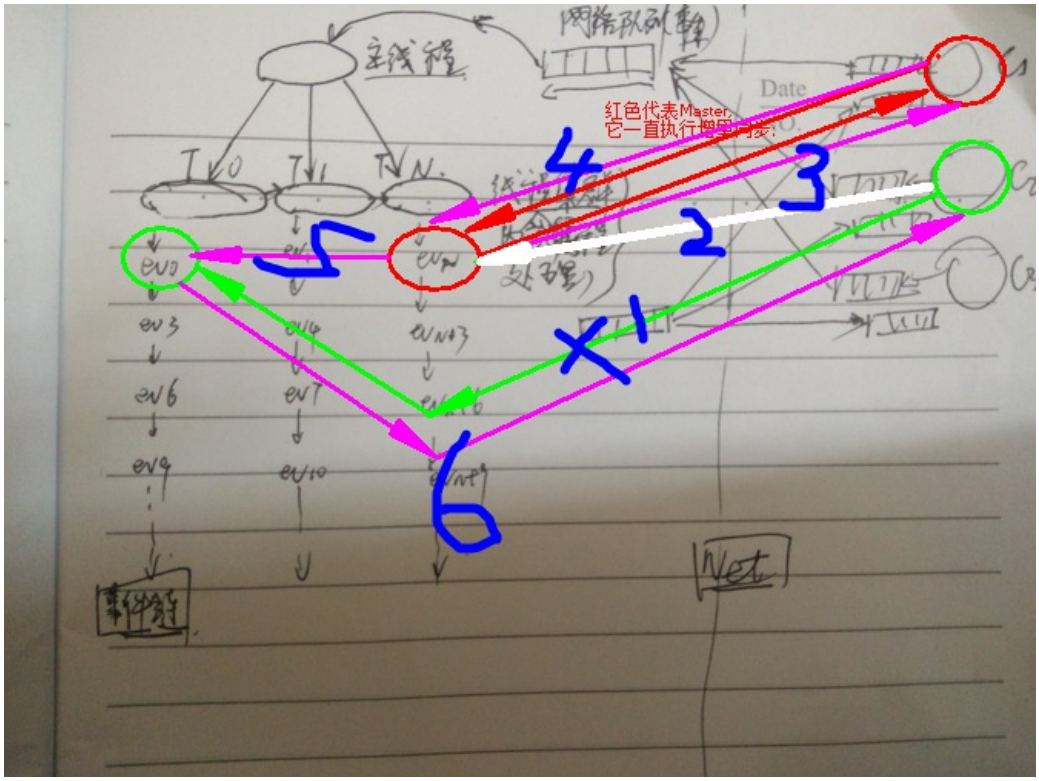


假设,目前就两个节点,node0在线程T(0), master在线程T(N).
那么node0 首次,肯定是向服务器请求DUMP操作(对象是master);

步骤: (如下图:**具体事件流向**)

如果事件类型是EV_DUMP,根据携带的目标DUMP ID(假如是:master),在事件入子线程之前,就调换id和ev_data的顺序,将事件分配给目标线程,并由其处理;

如图: 本来Node1(C2)绿色标注,{id=node2,ev_data=master} 要DUMP master的数据库,我们不是直接去ev0(**步骤1**),而是交换id和ev_data的值去evN(**步骤2**),由master执行DUMP(**步骤3**)之后,返回其结果(**步骤4**),再次调换位置并通过函数push_event_to_thread,将处理后的结果放入线程T0中(**步骤5**)此时的event,应该包含执行DUMP时对应的增量同步的序列(**sync_seq**),最后创建以ID=Node1的节点客户端状态,并返回(**步骤6**);
(**注**:每个新来的,执行DUMP操作的过程,都可以按照**图解步骤**操作;)



值得注意的是:

上面master本来在执行增量数据请求,此时,突然来了不束之客(DUMP数据库请求),可能情况如下:

1)增量数据已经发送出去;

2)增量数据还没发送出去;

如果是1,执行DUMP操作之前,需要等待其客户端的增量返回;并对返回值做判断!

如果是2,我们就要暂停增量数据往客户端发送,先发送DUMP操作的请求,等成功返回后,再发送2的增量数据;

接下来,node0会更新事件类型为INCR_REQ,并且再次发起请求;

步骤:

1.直接通过求余算法,进入到线程T0类,被协程调度执行;

2.它也可能被其他新的客户端指定为目标DUMP的操作者,此时,参考上面的DUMP过程;

3.其他方面就是错误类型判断和其对应客户端被意外KILL.

按照上面的图解流程;

那么具备执行增量同步的(客户端所在服务器端对应处理对象),只需要做两件事:

1) 对来自客户端增量请求的回应;

2) 对来自其他客户端的DUMP请求,调换id和ev_data后,发送到自己对端客户端,执行DUMP操作;

3) 等步骤2)操作完且返回DUMP请求成功后,我们就根据ev_data创建一个客户端状态信息对象,并添加到指定线程(根据求余算法获得),之后将DUMP的结果返回给ev_data所指定的客户端,之后就继续之前的增量同步;

4) 不管当前是执行增量响应,还是DUMP请求,上次都有可能执行同样的操作;

关于协程应该注意事项:

1.一个task就是一个协程任务;

2.一个线程内部,一般存在不同的ID的客户端处理任务(大家都执行增量同步时);

但也可能存在来自不同的客户端的ID向指定某个ID请求DUMP操作;(增量同时含有DUMP);

INCR_REQ(master&nil) -> INCR_REQ(Node4) -> DUMP_REQ(master&Node1) -> DUMP_REQ(master&Node2) -> INCR_REQ(Node8)

{这就是一个线程处理的多个协程任务,箭头就是链表的指针,Node1&Node2待执行DUMP}

3.每个task内部执行 evcoro_fastswitch(pthrd->evcoro_sched)时,就切换到其他协程任务;

4.当所有的task任务执行完成后,才可以进行下次协程集群任务(如果在task中存在异步等待的事件,那么我们需要将该事件从新放回到事件链表中,在下一轮循环中处理掉);

5.可以把所有可能存在的任务和条件都想象成一个线程内部处理所有的情况;

客户端不管是正常重启还是意外挂掉重启,都已解决!
服务器如果自己宕掉或被人KILL了,重启时也需要解决:(已解决)!
主要是在程序启动时,加载LevelDB的状态,知道当前有多少个客户端,分别同步到了哪个增量数据; XMQ内部实现,用了两个LevelDB库,一个存储状态,一个存储增量数据
(K:0000.....1) (V:具体的数据,可能是SQL语句,也可能是其他HTTP请求的数据);

开发建议:

1.对整体业务需求要心中有数,设计出总体的架构图,及值得注意的点;

2.对于执行流程比较复杂的程序,最好通过**GDB调试**结合**测试数据**进行开发,并**做好注解**;