测试1

（在虚拟机进行）

2client服务器+随机执行未指定unioin和split配比的20条随机指令

ERROR：发现过程中由于有用户到client服务器的连接断开，导致server服务器会主动断开和该client服务器的连接，所以在重新连到服务器之前另一个client服务器的指令就无法再传到该服务器。

我们的实现是假设网络稳定，所以对这种状况并未做出处理。

在发生该错误之前，两个服务器以不同序分别执行相同集合的操作到达的状态相同

即使再做重同步也不对吧，因为这个时候client服务器已经执行了很多命令，如果从这个状态开始执行断开之间丢失的命令，也不对，除非将那个失联的当时状态保留，

直觉这不是实现的错误，网络socket连接断开。搞了半天，然后正好再一次测试中发现当master发现slave在repl\_timeout里没有收到它的replconf ack，就会断开和这个slave的连接。repl\_timeout是一个在配置文件可以修改的选项，想到可能是配置选项对连接进行了一些限制。

1. repl\_timeout: 因为命令自动产生执行的速度很快，很可能这些命令在outbuffer中总是抢先了replconf ack，导致在短时间里ack不能被发送给master。默认是1min，可以改大一些
2. client-output-buffer-limit: replication buffer的大小，如果太小也可能造成连接断开，现在设置为512mb client-output-buffer-limit slave 512mb 512mb 0

当修改了上述两个配置选项之后，在进行测试，发现就不会再发生服务器断连了

ERROR: Out Of Memory allocating

**两个client服务器，500条随机命令，设置延时3s（union）和2s（split），master服务器需要9.2GB**

在一个服务器执行了577条命令后，服务器crash，保上述错误，这时由于我们的2D状态空间是占用服务器内存的，大量的命令所需的空间也会比较大，所以可能会造成内存空间爆掉。

1. Maxmemory: 在配置文件中有该选项，这个选项是用来限定服务器的内存最大值，根据服务器的内存大小进行设定，假设这里我们的虚拟机的最大是4g，那可以设为3g
2. 另外，对2D空间上内容存储的类型进行调整。
3. 所有的端口号从6380开始，所以对oid从port\_opid变为（port-6380）\_opid
4. 每条边上存储的op的信息中，操作类型从共享变量变为整形的redisobject，宏定义1/0分别表示unionot/splitot，然后在创建时调用createStringObjectFromLongLong函数。不过，如果修改了这里，那么好多判断类型的地方都需要进行修改。（暂且不改动）

内存占用过大：

应该是状态空间里每个点上存放的oids在命令越来越多的时候，就会越来越大，其实，这个oids应该是从顶点开始沿着往下一路拼接而成，可能可以考虑根本不用存这样一个信息，每次需要用的时候，做一次拼接：

* Local processing：locatevetex不需要用到oids
* Remote processing：locatevertex 需要用到，这个时候可以用一个sds临时变量，从initial开始，沿着路径逐个将边上的oid拼接到initial后形成oids，

修改locateverex的过程，变成类似二叉树搜索目标节点，但这样的话会导致在2D空间中定位某个点的时间开销变大。

对参数的个数的判断需要修改，原来的太简单了，只能对10以内的数字或者是1bit的字符进行处理，因为我用了看每个字符的长度，现在需要进行修改，变为看这个参数是否包含了”,”号，是则该参数内容不止一个，如果不是，则参数个数为1。

Find和Umembers命令不涉及，无需修改

Unionot和splitot以及ot函数需要。 改了ot，ufs，sds文件

2 测试client服务器的操作的并发度

统计master服务器的log文件中ot process的个数

操作序列长度为500 each client

* 在设置延时分别为union(2s) split(1s)的时候，500的操作序列并发度50左右
* 不设置延时，并发度达到115，c1的417，c50，这样的话，导致c1的很多条都需要ot，内存占用很大，maste服务器内存爆掉

注意，如果将sleep的时间设置的非常小，要注意很可能有client服务器无法执行第一条umembers指令获取并查集的内容，此时就会造成错误。

对locatevertice函数进行修改，

1. 如果是local processing，用原来的locateVertice函数，只要找到一个点的ledge和redge皆为NULL即可。
2. 如果是remote processing和server processing，则需要使用新的定位函数，以顶点和目标上下文为函数参数，递归查找。

去掉vertice的oids域，这样的话需要修改

1. Uinit命令，初始顶点“init”不设置
2. Local processing：定位过程也要改成递归，知道发现顶点的ledge和redge皆为NULL，这样在过程中将oids构建用于后面修改命令参数转发给master
3. Remote processing：同样在locatevertice的过程中需要构建号oids，这个后面会用到

Oids只有在我需要在状态空间里定位点的时候用到，然后在local processing的最后将客户端的命令进行添加oid和ctx参数时需要，其他地方用不到，所以vertice不需要保存这个信息

量级：

两个client服务器

每个服务器100条：

1. 使用oids，76.4mb
2. 不使用oids，75.9mb

每个服务器500条，client服务器内存占到4.5g，太大了，

**在所有命令执行完之后，记得info 一下，保存相关的Commandstats和memory的使用信息**

**空间优化**

op（int+char\*+char\*）来表示储，节点存储的ufs状态也用char\*存储，这样每个字符串就不再需要sds的头部的9个字节。

ERROR：会出现莫名的字符串内容被篡改，上一步还是新创建的字符串，但是紧接着下一步输出看看内容发现已经被修改？？？

修改：发现用了sdsnew创建一个新的char\*，这个字符串的内容就可能被修改，但是先创建一个空的sdsempty()，在用sdscat赋值，就不会，直接使用提供的sdssplitlen函数也出现问题，现在改为用strstok来实现相同功能。

因为上述问题导致程序执行失败主要涉及到两个地方，一个是修改命令参数准备发送到其他服务器以及更新节点的ufs状态信息。

目前还是可能会发生错误。。。。。。

修改了findSds函数，目前测试命令的处理没有错误。。

但是又遇到了之前服务器断连的情况，之前是通过增大repl\_timeout的值来增大服务器等待连接确认ack的时间，但是这个超时也有可能是因为通信时写outbuf超时，导致断连，所以应该不能简单地通过修改repl\_timeout解决断连问题。

解决：主要是在server收到client的命令广播命令到其他服务器以及client向server发送客户端命令的过程中，如果写失败，需要写服务器的back\_log。（因为刚开始编程，假定网络稳定，所以对这块并没有处理）

改了相关的backlog的写后，测试500条，master在过程中还是发生断连，但是并没有在重新连接后将backlog的内容发送到断开的slave

断连之后，需要处理两个部分的内容：

1. Client的backlog要写到server
2. Server的backlog要写到client

原来的redis因为slave不处理写命令，所以假设断连，只需要将处理2即可。

现在client服务器接收处理写命令，那么断连后，如果client服务器继续处理客户端的命令，在重新和server连接之后，不是先从backlog的命令处理，而是重新接收server服务器或者客户端发来的新的命令，这样的话，导致backlog中的命令并没有在client服务器执行之后就处理了更新的命令，那么这些新的而命令就无法在2D空间中找到对应的插入位置。

如果需要解决上述问题，client服务器发现connection with master lost，即需要暂停处理客户端的请求，等到重同步完成以后执行完backlog中的命令之后才能继续处理客户端请求。

现在先简单尝试在自动执行命令的程序中，没执行完一条命令，等50ms保证repl conf能够发送出去。等待的事件应该根据设置的repl-timeout的事件决定，

ERROR: 两个split操作也可能需要ot

Example: 47,225,12,30,22,80,890/34 s(12) s(225,12)

|  |  |  |
| --- | --- | --- |
| Case（s(a), s(b)） |  |  |
| ab= | S(a) | S(b) |
| a b | S(b-a) | S(a) |
| b a | S(a-b) | S(b) |
| ab=a-b≠ | U(c,d) ca cb | U(c,d) |

2，阿里云

1. 部署在阿里云，进行测试发现用多线程为每个client服务器随机生成300条操作时，命令的执行时间会变得很长，应该是线程开始产生命令的起始时间并不相同，导致有些服务器上产生的指令在后面执行的时候，每次ot的过程会非常长相当于一个指令需要和另外一个方向上的非常多的指令进行ot，这样的话，导致平均事件就会变得很长(已经将所有服务器的线程改为在create所有thread后再继续执行)