zookeeper3.3.3源码分析(二)FastLeader选举算法 - xhh198781的专栏 - 博客频道 - CSDN.NET

zookeeper3.3.3源码分析(二)FastLeader选举算法

如何在zookeeper集群中选举出一个leader,zookeeper使用了三种算法,具体使用哪种算法,在配置文件中是可以配置的,对应的配置项是”electionAlg”,其中1对应的是LeaderElection算法,2对应的是AuthFastLeaderElection算法,3对应的是FastLeaderElection算法.默认使用FastLeaderElection算法.其他两种算法我没有研究过,就不多说了.

要理解这个算法,最好需要一些paxos算法的理论基础.

1) 数据恢复阶段

首先,每个在zookeeper服务器先读取当前保存在磁盘的数据,zookeeper中的每份数据,都有一个对应的id值,这个值是依次递增的,换言之,越新的数据,对应的ID值就越大.

2) 首次发送自己的投票值

在读取数据完毕之后,每个zookeeper服务器发送自己选举的leader,这个协议中包含了以下几部分的数据:

1)所选举leader的id(就是配置文件中写好的每个服务器的id) ,在初始阶段,每台服务器的这个值都是自己服务器的id,也就是它们都选举自己为leader.

2) 服务器最大数据的id,这个值大的服务器,说明存放了更新的数据.

3)逻辑时钟的值,这个值从0开始递增,每次选举对应一个值,也就是说:如果在同一次选举中,那么这个值应该是一致的 2)逻辑时钟值越大,说明这一次选举leader的进程更新.

4) 本机在当前选举过程中的状态,有以下几种:LOOKING,FOLLOWING,OBSERVING,LEADING,顾名思义不必解释了吧.

每台服务器将自己服务器的以上数据发送到集群中的其他服务器之后,同样的也需要接收来自其他服务器的数据,它将做以下的处理:

1) 如果所接收数据服务器的状态还是在选举阶段(LOOKING 状态),那么首先判断逻辑时钟值,又分为以下三种情况:

a) 如果发送过来的逻辑时钟大于目前的逻辑时钟,那么说明这是更新的一次选举,此时需要更新一下本机的逻辑时钟值,同时将之前收集到的来自其他服务器的选举清空,因为这些数据已经不再有效了.然后判断是否需要更新当前自己的选举情况.在这里是根据选举leader id,保存的最大数据id来进行判断的,这两种数据之间对这个选举结果的影响的权重关系是:首先看数据id,数据id大者胜出;其次再判断leader id,leader id大者胜出.然后再将自身最新的选举结果(也就是上面提到的三种数据广播给其他服务器).代码如下:

```java
if (n.epoch > logicalclock) {
logicalclock = n.epoch;
recvset.clear();
if(totalOrderPredicate(n.leader, n.zxid,getInitId(), getInitLastLoggedZxid()))
   updateProposal(n.leader, n.zxid);
else
updateProposal(getInitId(),getInitLastLoggedZxid());

sendNotifications();
```

其中的totalOrderPredicate函数就是根据发送过来的封包中的leader id,数据id来与本机保存的相应数据进行判断的函数,返回true说明需要更新数据,于是调用updateProposal函数更新数据

b) 发送过来数据的逻辑时钟小于本机的逻辑时钟

说明对方在一个相对较早的选举进程中,这里只需要将本机的数据发送过去就是了

c) 两边的逻辑时钟相同,此时也只是调用totalOrderPredicate函数判断是否需要更新本机的数据,如果更新了再将自己最新的选举结果广播出去就是了.

实际上，在处理选票之前，还有一个预处理的动作，它发生在刚刚接收到关于vote的message的时候，具体过程如下：

```java
<span style="font-family:FangSong_GB2312;">1.判断message的来源是不是observer，如果是，则告诉该observer我当前认为的Leader的信息，否则进入2
2.判断message是不是vote信息，是则进入3
3.根据message创建一张vote
4.如果当前server处理LOOKING状态，将vote放入自己的投票箱，而且如果vote源server处于LOOKING状态同时vote源server的选举时旧的，则当前server通知它新的一轮投票;
5如果当前server不处于LOOKING状态而vote源server处理LOOKING状态，则当前server告诉它当前的Leader信息。
</span>
```

三种情况的处理完毕之后,再处理两种情况:

1)服务器判断是不是已经收集到了所有服务器的选举状态,如果是那么根据选举结果设置自己的角色(FOLLOWING还是LEADER),然后退出选举过程就是了.

2)即使没有收集到所有服务器的选举状态,也可以判断一下根据以上过程之后最新的选举leader是不是得到了超过半数以上服务器的支持,如果是,那么尝试在200ms内接收一下数据,如果没有新的数据到来,说明大家都已经默认了这个结果,同样也设置角色退出选举过程.

代码如下:

```java
/*
* Only proceed if the vote comes from a replica in the
* voting view.
*/
if(self.getVotingView().containsKey(n.sid)){
recvset.put(n.sid, new Vote(n.leader, n.zxid, n.epoch));

//If have received from all nodes, then terminate
if((self.getVotingView().size() == recvset.size()) && (self.getQuorumVerifier().getWeight(proposedLead
{
self.setPeerState((proposedLeader == self.getId()) ? ServerState.LEADING: learningState());
```

```
leaveInstance();
return new Vote(proposedLeader, proposedZxid);

} else if (termPredicate(recvset,new Vote(proposedLeader, proposedZxid,logicalclock))) {

// Verify if there is any change in the proposed leader
while((n = recvqueue.poll(finalizeWait,TimeUnit.MILLISECONDS)) != null){
if(totalOrderPredicate(n.leader, n.zxid,proposedLeader, proposedZxid)){
   recvqueue.put(n);
   break;
}
}

/*
* This predicate is true once we don't read any new
* relevant message from the reception queue
*/
if (n == null) {
self.setPeerState((proposedLeader == self.getId()) ? ServerState.LEADING: learningState());
if(LOG.isDebugEnabled()){
LOG.debug("About to leave FLE instance: Leader= " + proposedLeader + ", Zxid = " + proposedZxid + ", My
}

leaveInstance();
return new Vote(proposedLeader,proposedZxid);
}
}
}
```

2）如果所接收服务器不在选举状态，也就是在FOLLOWING或者LEADING状态
做以下两个判断：
a）如果逻辑时钟相同，将该数据保存到recvset，如果所接收服务器宣称自己是leader，那么将判断是不是有半数以上的服务器选举它，如果是则设置选举状态退出选举过程
b）否则这是一条与当前逻辑时钟不符合的消息，那么说明在另一个选举过程中已经有了选举结果，于是将该选举结果加入到outofelection集合中，再根据outofelection来判断是否可以结束选举，如果可以也是保存逻辑时钟，设置选举状态，退出选举过程.
代码如下：

```
[java]

if(n.epoch == logicalclock){
recvset.put(n.sid, new Vote(n.leader, n.zxid, n.epoch));
if((n.state == ServerState.LEADING) || (termPredicate(recvset, new Vote(n.leader,n.zxid, n.epoch, n.st
{
self.setPeerState((n.leader == self.getId()) ?ServerState.LEADING: learningState());
leaveInstance();
return new Vote(n.leader, n.zxid);
}
}

outofelection.put(n.sid, new Vote(n.leader, n.zxid, n.epoch, n.state));

if(termPredicate(outofelection, new Vote(n.leader,n.zxid, n.epoch, n.state))&& checkLeader(outofelecti
   synchronized(this){
      logicalclock = n.epoch;
```

```
15.         self.setPeerState((n.leader == self.getId()) ? ServerState.LEADING: learningState());
16.     }
17.     leaveInstance();
18.     return new Vote(n.leader, n.zxid);
19. }
20. 
21. break;
22. }
23. }
```

以一个简单的例子来说明整个选举的过程.
假设有五台服务器组成的zookeeper集群,它们的id从1-5,同时它们都是最新启动的,也就是没有历史数据,在存放数据量这一点上,都是一样的.假设这些服务器依序启动,来看看会发生什么.

```
[java]
01. <span style="font-family:FangSong_GB2312;">1) 服务器1启动,此时只有它一台服务器启动了,它发出去的报没有任何响应,所以它的选举状态一直是LOOKING状态
02. 2) 服务器2启动,它与最开始启动的服务器1进行通信,互相交换自己的选举结果,由于两者都没有历史数据,所以id值较大的服务器2胜出,但是由于没有达到超过半数以上的服务器都同意选举它(这个例子中的半数以上是3),所以服务器1,2还是继续保持LOOKING状态.
03. 3) 服务器3启动,根据前面的理论分析,服务器3成为服务器1,2,3中的老大,而与上面不同的是,此时有三台服务器选举了它,所以它成为了这次选举的leader.
04. 4) 服务器4启动,根据前面的分析,理论上服务器4应该是服务器1,2,3,4中最大的,但是由于前面已经有半数以上的服务器选举了服务器3,所以它只能接收当小弟的命了.
05. 5) 服务器5启动,同4一样,当小弟.</span>
```


上一篇 zookeeper3.3.3源码分析(一)工作原理概述
下一篇 非阻塞通信

我的同类文章

ZooKeeper（10）

- Paxos算法与Zookeeper… 2013-09-02 阅读 21242
- ZooKeeper中的节点故障… 2011-10-12 阅读 1450
- ZooKeeper中的节点故障… 2011-10-11 阅读 1822
- Zookeeper中的FastLea… 2011-09-09 阅读 2430
- zookeeper3.3.3源码分析… 2011-07-17 阅读 7479
- ZooKeeper中的节点故障… 2011-10-12 阅读 1293
- ZooKeeper中的节点故障… 2011-10-12 阅读 2528
- ZooKeeper的使用 2011-09-13 阅读 4307
- ZooKeeper中的写操作细… 2011-09-08 阅读 3041