本人原创，转载请注明出处！ 本人QQ：530422429，欢迎大家指正、讨论。

欢迎访问： 西北工业大学 - 大数据与知识管理研究室 （Northwestern Polytechnical University - BigData and Knowledge Management Lab），链接：<http://wowbigdata.cn/>，<http://wowbigdata.net.cn/>，[http://wowbigdata.com.cn](http://wowbigdata.com.cn/)。

环境：在单机上（机器名：giraphx）启动了2个workers。

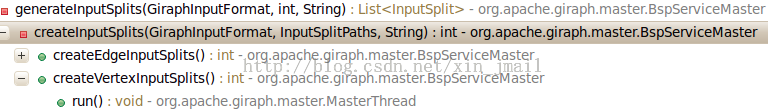
输入:SSSP文件夹，里面有1.txt和2.txt两个文件

1. 在Worker向Master汇报健康状况后，就开始等待Master创建InputSplit。方法：每个Worker通过检某个Znode节点是否存在，同时在此Znode上设置Watcher。若不存在，就通过BSPEvent的waitForever()方法释放当前线程的锁，陷入等待状态。一直等到master创建该znode。此步骤位于BSPServiceWorker类中的startSuperStep方法中，等待代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23687073)[在CODE上查看代码片](https://code.csdn.net/snippets/292200)

1. //Znode的路径
2. String addressesAndPartitionsPath =
3. getAddressesAndPartitionsPath(getApplicationAttempt(),
4. getSuperstep());
5. //把该znode的data读入到addressesAndPartitions中
6. AddressesAndPartitionsWritable addressesAndPartitions =
7. **new** AddressesAndPartitionsWritable(
8. workerGraphPartitioner.createPartitionOwner().getClass());
9. //当master创建该znode后，退出while循环
10. **while** (getZkExt().exists(addressesAndPartitionsPath, **true**) ==
11. **null**) {
12. //陷入等待状态
13. getAddressesAndPartitionsReadyChangedEvent().waitForever();
14. //当master创建该znode后，触发Watcher。调用process进而唤醒线程
15. getAddressesAndPartitionsReadyChangedEvent().reset();
16. }
17. //读入数据
18. WritableUtils.readFieldsFromZnode(
19. getZkExt(),
20. addressesAndPartitionsPath,
21. **false**,
22. **null**,
23. addressesAndPartitions);

2. Master调用createInputSplits()方法创建InputSplit。



在generateInputSplits()方法中，根据用户设定的VertexInputFormat获得InputSplits。代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23687073)[在CODE上查看代码片](https://code.csdn.net/snippets/292200)

1. List<InputSplit> splits=inputFormat.getSplits(getContext(), minSplitCountHint);

其中minSplitCountHint为创建split的最小数目，其值如下：

minSplitCountHint = Workers数目 \* NUM\_INPUT\_THREADS  
NUM\_INPUT\_THREADS表示 每个Input split loading的线程数目，默认值为1 。 经查证，在TextVertexValueInputFormat抽象类中的getSplits()方法中的minSplitCountHint参数被忽略。用户输入的VertexInputFormat继承TextVertexValueInputFormat抽象类。

**如果得到的splits.size小于minSplitCountHint，那么有些worker就没被用上。**

     得到split信息后，要把这些信息写到Zookeeper上，以便其他workers访问。上面得到的split信息如下：

[hdfs://giraphx:9000/user/root/SSSP/1.txt:0+66, hdfs://giraphx:9000/user/root/SSSP/2.txt:0+46]

遍历splits List，为每个split创建一个Znode，值为split的信息。如为split-0创建Znode，值为：hdfs://giraphx:9000/user/root/SSSP/1.txt:0+66

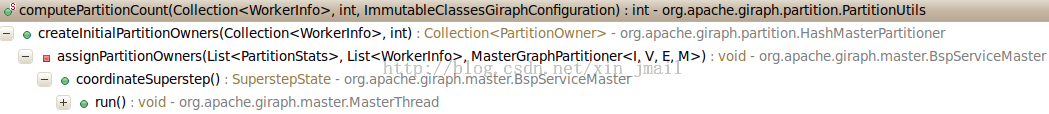
/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir/0

为split-1创建znode（如下），值为：hdfs://giraphx:9000/user/root/SSSP/2.txt:0+46

/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir/1

最后创建znode： /\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitsAllReady 表示所有splits都创建好了。

3. Master根据splits创建Partitions。首先确定partition的数目。



BSPServiceMaster中的MasterGraphPartitioner<I.V,E,M>对象默认为HashMasterPartitioner。它的createInitialPartitionOwners()方法如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23687073)[在CODE上查看代码片](https://code.csdn.net/snippets/292200)

1. @Override
2. **public** Collection<PartitionOwner> createInitialPartitionOwners(
3. Collection<WorkerInfo> availableWorkerInfos, **int** maxWorkers) {
4. //maxWorkers为Workers的最大数目，用户通过 -w 指定。实验时指定为 2
5. //availableWorkerInfos为健康的Workers列表,此处为：[Worker(hostname=giraphx, MRtaskID=1, port=30001), Worker(hostname=giraphx, MRtaskID=2, port=30002)]
6. **int** partitionCount = PartitionUtils.computePartitionCount(
7. availableWorkerInfos, maxWorkers, conf);
8. List<PartitionOwner> ownerList = **new** ArrayList<PartitionOwner>();
9. Iterator<WorkerInfo> workerIt = availableWorkerInfos.iterator();
10. //为每个Partition指定一个PartitionOwner，表示该Partition的元数据信息
11. **for** (**int** i = 0; i < partitionCount; ++i) {
12. PartitionOwner owner = **new** BasicPartitionOwner(i, workerIt.next());
13. //若遍历完availableWorkerInfos，则开始下一轮遍历。
14. **if** (!workerIt.hasNext()) {
15. workerIt = availableWorkerInfos.iterator();
16. }
17. ownerList.add(owner);
18. }
19. **this**.partitionOwnerList = ownerList;
20. **return** ownerList;
21. }

上面代码中是在工具类PartitionUtils计算Partition的数目，计算公式如下：

partitionCount=PARTITION\_COUNT\_MULTIPLIER \* availableWorkerInfos.size() \* availableWorkerInfos.size() ，其中PARTITION\_COUNT\_MULTIPLIER表示Multiplier for the current workers squared，默认值为1 。

可见，partitionCount值为4（1\*2\*2）。创建的partitionOwnerList信息如下：

[(id=0,cur=Worker(hostname=giraphx, MRtaskID=1, port=30001),prev=null,ckpt\_file=null),

(id=1,cur=Worker(hostname=giraphx, MRtaskID=2, port=30002),prev=null,ckpt\_file=null),

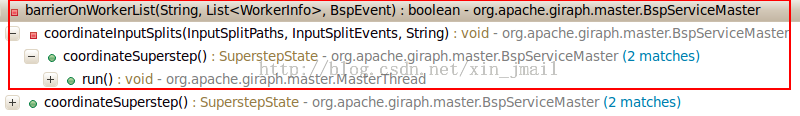
(id=2,cur=Worker(hostname=giraphx, MRtaskID=1, port=30001),prev=null,ckpt\_file=null),

(id=3,cur=Worker(hostname=giraphx, MRtaskID=2, port=30002),prev=null,ckpt\_file=null)]

4. Master创建Znode：/\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_partitionExchangeDir，用于后面的exchange partition。

5. Master最后在assignPartitionOwners()方法中，把masterinfo，chosenWorkerInfoList，partitionOwners等信息写入Znode中（作为Znode的data），该Znode的路径为：      /\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_addressesAndPartitions 。

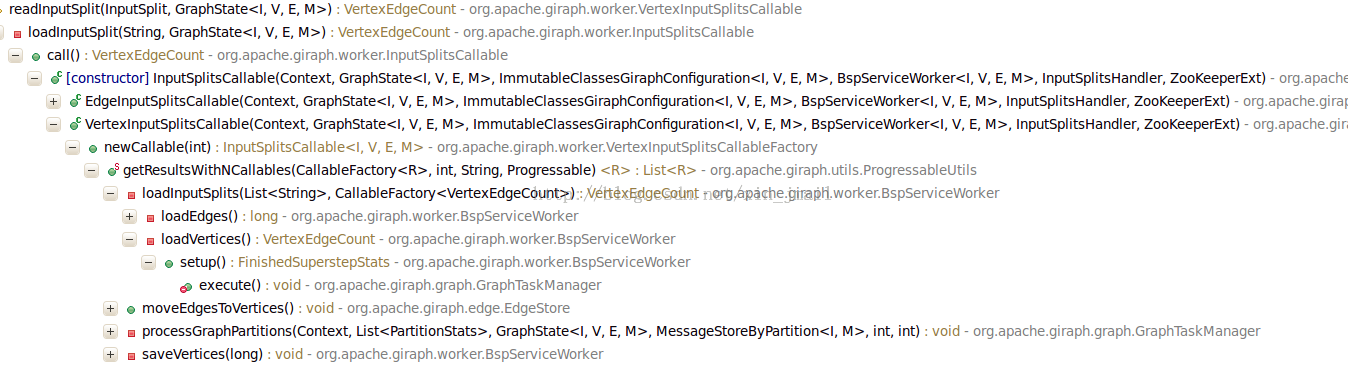
   Master调用barrierOnWorkerList()方法开始等待各个Worker完成数据加载。调用关系如下：



在barrierOnWorkerList中创建znode，path=/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDoneDir 。然后检查该znode的子节点数目是否等于workers的数目，若不等于，则线程陷入等待状态。后面某个worker完成数据加载后，会创建子node（如 /\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDoneDir/giraphx\_1）来激活该线程继续判断。

6. 当Master创建第5步的znode后，会激活worker。每个worker从znode上读出data，data包含masterInfo，WorkerInfoList和partitionOwnerList，然后各个worker开始加载数据。

   把partitionOwnerList复制给BSPServiceWorker类中的workerGraphPartitioner（默认为HashWorkerPartitioner类型）对象的partitionOwnerList变量，后续每个顶点把根据vertexID通过workerGraphPartitioner对象获取其对应的partitionOwner.



每个Worker从znode： /\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir获取子节点，得到inputSplitPathList，内容如下：

[/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir/1,  
/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir/0]

然后每个Worker创建N个InputsCallable线程读取数据。N=Min(NUM\_INPUT\_THREADS,maxInputSplitThread)，其中NUM\_INPUT\_THREADS默认值为1，maxInputSplitThread=（InputSplitSize-1/maxWorkers +1

那么，默认每个worker就是创建一个线程来加载数据。

在InputSplitsHandler类中的reserveInputSplit()方法中，每个worker都是遍历inputSplitPathList，通过创建znode来保留(标识要处理)的split。代码及注释如下：

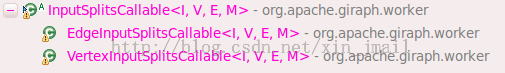
**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23687073)[在CODE上查看代码片](https://code.csdn.net/snippets/292200)

1. **public** String reserveInputSplit() {
2. String reservedInputSplitPath;
3. Stat reservedStat;
5. **while** (**true**) {
6. //currentIndex递增，要遍历完pathList
7. **int** splitToTry = currentIndex.getAndIncrement();
8. //遍历完pathList，说明所有的split都被处理了，退出while循环。
9. **if** (splitToTry >= pathList.size()) {
10. **return** **null**;
11. }
12. //得到split的znode path，如/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir/1
13. String nextSplitToClaim = pathList.get(splitToTry);
15. //构造znode路径，如： /\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir/1/\_vertexInputSplitReserved
16. String tmpInputSplitReservedPath = nextSplitToClaim + inputSplitReservedNode;
18. //检测znode是否存在。若存在，说明该split已经被其他worker处理了。设置watcher是为了容错，可暂时忽略。
19. reservedStat =
20. zooKeeper.exists(tmpInputSplitReservedPath, **this**);
21. //若不存在，说明该znode对应的split还没有被处理掉。但有可能其他worker也在申请处理当前znode，
22. //所以下面创建znode时,可能会出现KeeperException.NodeExistsException异常。
23. **if** (reservedStat == **null**) {
24. **try** {
25. // Attempt to reserve this InputSplit
26. //若成功创建，那么当前worker就出该split
27. zooKeeper.createExt(tmpInputSplitReservedPath,
28. **null**,
29. ZooDefs.Ids.OPEN\_ACL\_UNSAFE,
30. CreateMode.EPHEMERAL,
31. **false**);
32. reservedInputSplitPath = nextSplitToClaim;
33. **return** reservedInputSplitPath;
34. } **catch** (KeeperException.NodeExistsException e) {
35. LOG.info("reserveInputSplit: Couldn't reserve " +
36. "(already reserved) inputSplit" +
37. " at " + tmpInputSplitReservedPath);
38. }
39. }
40. }

当用reserveInputSplit()方法获取某个znode后，loadSplitsCallable类的loadInputSplit方法就开始通过该znode获取其HDFS的路径信息，然后读入数据、重分布数据。

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23687073)[在CODE上查看代码片](https://code.csdn.net/snippets/292200)

1. //inputSplitPath为znode的path，如： /\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir/1
2. **private** VertexEdgeCount loadInputSplit(
3. String inputSplitPath,
4. GraphState<I, V, E, M> graphState) {
5. //获取该znode对应的InputSplit信息，如得到：hdfs://giraphx:9000/user/root/SSSP/2.txt:0+46
6. InputSplit inputSplit = getInputSplit(inputSplitPath);
8. //从split中一行一行读入数据，把每行数据创建成一个vertex。
9. //然后根据vertexId把vertex发送到相应的partition上(数据重分布过程)
10. VertexEdgeCount vertexEdgeCount =
11. readInputSplit(inputSplit, graphState);
12. //处理完当前split后，创建结束znode标识该split已被处理掉。znode的path为： /\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDir/1/\_vertexInputSplitFinished
13. splitsHandler.markInputSplitPathFinished(inputSplitPath);
14. **return** vertexEdgeCount;
15. }



VertexInputSplitsCallable类的readInputSplit()方法如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23687073)[在CODE上查看代码片](https://code.csdn.net/snippets/292200)

1. **protected** VertexEdgeCount readInputSplit(
2. InputSplit inputSplit,
3. GraphState<I, V, E, M> graphState)
4. **throws** IOException, InterruptedException {
5. //获取用户输入的InputFormat类
6. VertexInputFormat<I, V, E> vertexInputFormat =
7. configuration.createVertexInputFormat();
8. VertexReader<I, V, E> vertexReader =
9. vertexInputFormat.createVertexReader(inputSplit, context);
10. vertexReader.setConf(
11. (ImmutableClassesGiraphConfiguration<I, V, E, Writable>) configuration);
12. vertexReader.initialize(inputSplit, context);
13. **long** inputSplitVerticesLoaded = 0;
14. **long** edgesSinceLastUpdate = 0;
15. **long** inputSplitEdgesLoaded = 0;
16. **while** (vertexReader.nextVertex()) {
17. //获取vertex
18. Vertex<I, V, E, M> readerVertex =
19. (Vertex<I, V, E, M>) vertexReader.getCurrentVertex();
20. **if** (readerVertex.getId() == **null**) {
21. **throw** **new** IllegalArgumentException(
22. "readInputSplit: Vertex reader returned a vertex " +
23. "without an id!  - " + readerVertex);
24. }
25. **if** (readerVertex.getValue() == **null**) {
26. readerVertex.setValue(configuration.createVertexValue());
27. }
28. readerVertex.setConf(configuration);
29. readerVertex.setGraphState(graphState);
30. //根据vertexID获取其partitionOwner
31. PartitionOwner partitionOwner =
32. bspServiceWorker.getVertexPartitionOwner(readerVertex.getId());
33. //把顶点发送到对应的partition上
34. graphState.getWorkerClientRequestProcessor().sendVertexRequest(
35. partitionOwner, readerVertex);
36. context.progress(); // do this before potential data transfer
37. ++inputSplitVerticesLoaded;
39. edgesSinceLastUpdate += readerVertex.getNumEdges();
40. }
41. vertexReader.close();
42. **return** **new** VertexEdgeCount(inputSplitVerticesLoaded,
43. inputSplitEdgesLoaded + edgesSinceLastUpdate);
44. }

7. 每个worker加载完数据后，调用waitForOtherWorkers()方法等待其他workers都处理完split。  
http://img.blog.csdn.net/20140415103359234?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQveGluX2ptYWls/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

策略如下，每个worker在/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDoneDir目录下创建子节点，后面追加自己的worker信息，如worker1、worker2创建的子节点分别如下：

/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDoneDir/giraphx\_1

/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDoneDir/giraphx\_2

，创建完后，然后等待master创建/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitsAllDone。

8.从第5步骤可知，若master发现/\_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitDoneDir下的子节点数目等于workers的总数目，就会在coordinateInputSplits()方法中创建

  \_hadoopBsp/job\_201404102333\_0013/\_vertexInputSplitsAllDone，告诉每个worker，所有的worker都处理完了split。

9. 最后就是就行全局同步。

  master创建znode，path=/\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_workerFinishedDir ，然后再调用barrierOnWorkerList方法检查该znode的子节点数目是否等于workers的数目，若不等于，则线程陷入等待状态。等待worker创建子节点来激活该线程继续判断。

  每个worker获取自身的Partition Stats，进入finishSuperStep方法中，等待所有的Request都被处理完；把自身的Aggregator信息发送给master；创建子节点，如/\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_workerFinishedDir/giraphx\_1，data为该worker的partitionStatsList和workerSentMessages统计量；

最后调用waitForOtherWorkers()方法等待master创建/\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_superstepFinished 节点。

  master发现/\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_workerFinishedDir的子节点数目等于workers数目后，根据/\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_workerFinishedDir子节点上的data收集每个worker发送的aggregator信息，汇总为globalStats。

Master若发现全局信息中（1）所有顶点都voteHalt且没有消息传递，或（2）达到最大迭代次数 时，设置 globalStats.setHaltComputation(true)。告诉works结束迭代。

master创建/\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_superstepFinished 节点，data为globalStats。告诉所有workers当前超级步结束。

每个Worker检测到master创建/\_hadoopBsp/job\_201404102333\_0013/\_applicationAttemptsDir/0/\_superstepDir/-1/\_superstepFinished 节点后，读出该znode的数据，即全局的统计信息。然后决定是否继续下一次迭代。

10. 同步之后开始下一个超级步。

11.master和workers同步过程总结。

     （1）master创建znode A，然后检测A的子节点数目是否等于workers数目，不等于就陷入等待。某个worker创建一个子节点后，就会唤醒master进行检测一次。

      （2）每个worker进行自己的工作，完成后，创建A的子节点A1。然后等待master创建znode B。

       （3）若master检测到A的子节点数目等于workers的数目时，创建Znode B

        （4）master创建B 节点后，会激活各个worker。同步结束，各个worker就可以开始下一个超步。

        本质是通过znode B来进行全局同步的。

本人原创，转载请注明出处！ 本人QQ：530422429，欢迎大家指正、讨论。