1. 由前文知道每个BSPServiceWorker有一个WorkerServer对象，WorkerServer对象里面又有ServerData对象，作为数据实体。ServerData中包含该Worker的partitionStore、edgeStore、incomingMessageStore、currentMessageStore、聚集值等。

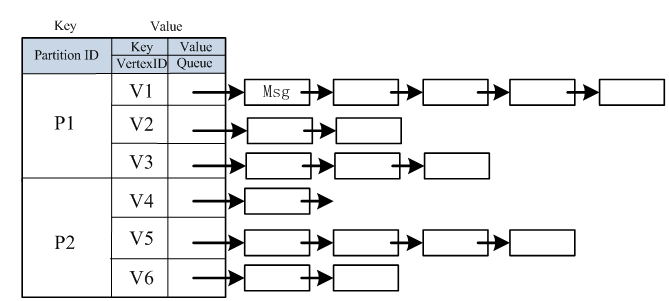
   其中incomingMessageStore对象为MessageStoreByPartition(接口)类型，也就是说消息时按照分区来存储的。MessageStoreByPartition接口的关系图如下：



 如果用户的程序使用Combiner， incomingMessageStore对象实际为OneMessagePerVertexStore ；否则incomingMessageStore对象为ByteArrayMessagesPerVertexStore类型。

  下面先分析ByteArrayMessagesPerVertexStore类型。对同一个vertex的multiple消息存储为一个byte arrays。

   在SimpleMessageStore抽象类中，有一个ConcurrentMap<Integer,ConcurrentMap<I,T>>类型的变量map，用来存储消息。第一层是pairtitionID到发送到该partition消息的映射；第二层是VertexID 到发送给该Vertex的消息队列。下图参考《Giraph通信模块分析》：<http://my.oschina.net/skyaugust/blog/95182> 。在此表示感谢！



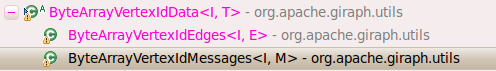
   每个顶点的消息列表具体为ExtendedDataOutput类型，它继承DataOutput接口，增加了几个方法而已。每个消息是以字节形式写入到ExtendedDataOutput对象中的。

2. 发送消息时，采用**异步式通信**。图顶点的计算处理与消息通信并发执行，在计算过程中就可以发送消息，将大规模消息发送分散在不同的时间段，避免瞬时网络通信阻塞，但是接受端需要额外的空间，存储临时接收到的消息，相当于空间换时间。而集中式通信，图顶点的计算处理与消息通信串行进行，在计算完毕后，统一发送消息，控制和实现方式简单，可在发送端对消息进行最大程度优化，但容易造成瞬时间的网络通信阻塞以及增加发送端的消息存储开销。

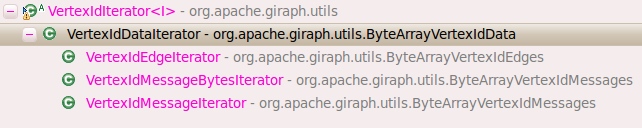
   不同Worker间的消息通信使用RPC方式，具体为Netty。同**一Worker内，连续两次迭代的消息直接通过内存操作，把要发送的消息直接复制到Worker的incomingMessageStore中**。下面详述消息的存储格式和发送机制。

3. Giraph使用Cache来缓存消息，当消息达到一定阈值后，一次性发送。既按照bulk模式进行，不会一条一条信息发送。向某个顶点发送的消息是按照<destVertexId,Message> pair存储在ByteArrayVertexIdData<I,T>中（**实际为ByteArrayVertexIdMessages<I,M>**类型）。介绍如下： org.apache.giraph.utils.ByteArrayVertexIdData<I,T>

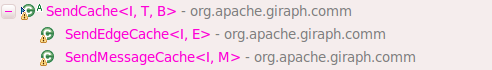
    功能：把<顶点ID，data> Pair 存储在一个 byte数组中。里面有 ExtendedDataOutput对象用来存储数据。



    该类中还有一个内部类：VertexIdDataIterator，该内部类继承 VertexIdIterator类。



    org.apache.giraph.comm.SendCache用来缓存发送的信息，然后以“Bulk”模式发送。**在Giraph中，每个Worker上可以对应多个分区**。消息缓存的阈值是以Worker为单位计算，而不是Partition。



    SendCache中有ByteArrayVertexIdData<I,T>[ ]  dataCache数组用来存储发送给每个Partition的消息；有int[ ] dataSizes数组用于记录向每个Worker发送的消息大小，若大于MAX\_MSG\_REQUEST\_SIZE（默认为512KB）就把此Worker上的所有Partition缓存的消息发送到给该Worker，同一Worker内消息也是如此缓存；有int[ ] initBufferSizes数组用于记录每个Worker上的每个Partition的初始化ByteArrayVertexIdData中ExtendedDataOutput对象的大小，同一Worker上的所有Partition初始值相同，该值为平均值。记MAX\_MSG\_REQUEST\_SIZE（message request size）值为M， 该Worker上有P个 partitions，ADDTITIONNAL\_MSG\_REQUEST\_SIZE（比平均值大的因子）默认为0.2f，记为A。则每个Partition的初始大小为：M\*(1+A) / P .

    由前文知道，每个Worker都有一个NettyWorkerClientRequestProcessor<I,V,E,M>用来发送消息。该类中有SendMessageCache对象用来缓存向外发送的信息。NettyWorkerClientRequestProcessor类中的sendMessageRequest(I,M)

方法如下，用于向某个顶点destVertexId发送消息message。

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23187831)[在CODE上查看代码片](https://code.csdn.net/snippets/284446)

1. @Override
2. **public** **boolean** sendMessageRequest(I destVertexId, M message) {
3. PartitionOwner owner =
4. serviceWorker.getVertexPartitionOwner(destVertexId);
5. WorkerInfo workerInfo = owner.getWorkerInfo();
6. **final** **int** partitionId = owner.getPartitionId();
7. ++totalMsgsSentInSuperstep;
9. // Add the message to the cache
10. **int** workerMessageSize = sendMessageCache.addMessage(
11. workerInfo, partitionId, destVertexId, message);
13. // Send a request if the cache of outgoing message to
14. // the remote worker 'workerInfo' is full enough to be flushed
15. **if** (workerMessageSize >= maxMessagesSizePerWorker) {
16. PairList<Integer, ByteArrayVertexIdMessages<I, M>>
17. workerMessages =
18. sendMessageCache.removeWorkerMessages(workerInfo);
19. WritableRequest writableRequest =
20. **new** SendWorkerMessagesRequest<I, M>(workerMessages);
21. doRequest(workerInfo, writableRequest);
22. **return** **true**;
23. }
24. **return** **false**;
25. }

方法解释：首先根据destVertexId得到对应的partitionId和WorkerInfo，然后把消息add到SendMessageCache中，并返回向该顶点所属Worker发送的消息大小workerMessageSize。若该值大于默认值512KB，则把此Worker对应的所有Partition消息从SendMessageCache中删除，把删除的消息赋值给workerMessages，其类型为PairList<Integer,ByteArrayVertexIdMessages<I,M>> ，key为partitionId，value为发送给该partition的消息列表，最后调用doRequest()方法发送信息。doRequest()方法如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23187831)[在CODE上查看代码片](https://code.csdn.net/snippets/284446)

1. **private** **void** doRequest(WorkerInfo workerInfo,
2. WritableRequest writableRequest) {
3. // If this is local, execute locally
4. **if** (serviceWorker.getWorkerInfo().getTaskId() ==
5. workerInfo.getTaskId()) {
6. ((WorkerRequest) writableRequest).doRequest(serverData);
7. localRequests.inc();
8. } **else** {
9. workerClient.sendWritableRequest(
10. workerInfo.getTaskId(), writableRequest);
11. remoteRequests.inc();
12. }
13. }

可以看到在发送消息时，先判断是否在同一Worker上。如果是的话，调用SendWorkerMessagesRequest<T,M>的doRequest发送消息；否则使用WorkerClient（底层使用Netty）进行消息发送。下面着重讨论同一Worker内的机制。

org.apache.giraph.comm.requests.SendWorkerMessagesRequest类中的doRequest方法如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23187831)[在CODE上查看代码片](https://code.csdn.net/snippets/284446)

1. @Override
2. **public** **void** doRequest(ServerData serverData) {
3. PairList<Integer, ByteArrayVertexIdMessages<I, M>>.Iterator
4. iterator = partitionVertexData.getIterator();
5. **while** (iterator.hasNext()) {
6. iterator.next();
7. **try** {
8. serverData.getIncomingMessageStore().
9. addPartitionMessages(iterator.getCurrentFirst(),
10. iterator.getCurrentSecond());
11. } **catch** (IOException e) {
12. **throw** **new** RuntimeException("doRequest: Got IOException ", e);
13. }
14. }
15. }

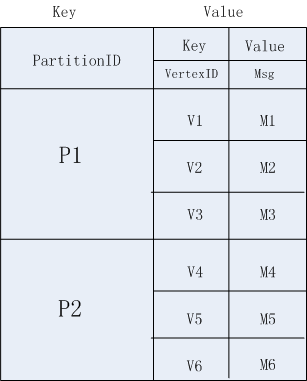
参数为该Worker的ServerData，代码中的partitionVertexData实际为PairList<Integer,ByteArrayVertexIdMessages<I,M>>workerMessages。  遍历<partitionID，对应的消息列表>来添加到ServerData中的incomingMessageStore中。

ByteArrayMessagesPerVertexStore类中的addPartitionMessages()方法如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23187831)[在CODE上查看代码片](https://code.csdn.net/snippets/284446)

1. @Override
2. **public** **void** addPartitionMessages(
3. **int** partitionId,
4. ByteArrayVertexIdMessages<I, M> messages) **throws** IOException {
5. ConcurrentMap<I, ExtendedDataOutput> partitionMap =
6. getOrCreatePartitionMap(partitionId);
7. ……
8. ByteArrayVertexIdMessages<I, M>.VertexIdMessageIterator
9. vertexIdMessageIterator = messages.getVertexIdMessageIterator();
10. // 遍历<destVertexId,Msg> Pair
11. **while** (vertexIdMessageIterator.hasNext()) {
12. vertexIdMessageIterator.next();
13. // 找到顶点对应的队列，不存在则新建
14. ExtendedDataOutput extendedDataOutput =
15. getExtendedDataOutput(partitionMap, vertexIdMessageIterator);
16. **synchronized** (extendedDataOutput) {
17. // 把Msg添加到队列中
18. vertexIdMessageIterator.getCurrentMessage().write(
19. extendedDataOutput);
20. }
21. }
22. }

4.  当用户使用了Combiner，incomingMessageStore对应的类型则为OneMessagePerVertexStore，该类**为每个顶点只存储一个消息，而非消息队列**。结构如下图：



当添加一条消息时，会把顶点已对应的消息和要添加的消息调用combine()方法进行合并，然后存储在上述结构图中。addPartitionMessages()方法如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23187831)[在CODE上查看代码片](https://code.csdn.net/snippets/284446)

1. @Override
2. **public** **void** addPartitionMessages(
3. **int** partitionId,
4. ByteArrayVertexIdMessages<I, M> messages) **throws** IOException {
5. ConcurrentMap<I, M> partitionMap =
6. getOrCreatePartitionMap(partitionId);
7. ByteArrayVertexIdMessages<I, M>.VertexIdMessageIterator
8. vertexIdMessageIterator = messages.getVertexIdMessageIterator();
9. // This loop is a little complicated as it is optimized to only create
10. // the minimal amount of vertex id and message objects as possible.
11. //遍历<destVertexID,Msg> Pair
12. **while** (vertexIdMessageIterator.hasNext()) {
13. vertexIdMessageIterator.next();
14. I vertexId = vertexIdMessageIterator.getCurrentVertexId();
15. M currentMessage =
16. partitionMap.get(vertexIdMessageIterator.getCurrentVertexId());
17. **if** (currentMessage == **null**) {
18. M newMessage = combiner.createInitialMessage();
19. currentMessage = partitionMap.putIfAbsent(
20. vertexIdMessageIterator.releaseCurrentVertexId(), newMessage);
21. **if** (currentMessage == **null**) {
22. currentMessage = newMessage;
23. }
24. }
25. **synchronized** (currentMessage) {
26. // 调用combine()方法进行合并，并赋值给currentMessage
27. combiner.combine(vertexId, currentMessage,
28. vertexIdMessageIterator.getCurrentMessage());
29. }
30. }
31. }

5. 在ComputeCallable中的call()方法调用computePartition(Partition)计算完所有Partition上的顶点后，调用WorkerClientRequestProcessor.flush()方法把所有剩余的消息发送出去。

**[java]** [view plaincopyprint?](http://blog.csdn.net/xin_jmail/article/details/23187831)[在CODE上查看代码片](https://code.csdn.net/snippets/284446)

1. @Override
2. **public** **void** flush() **throws** IOException {
3. ……
4. // Execute the remaining sends messages (if any)
5. PairList<WorkerInfo, PairList<Integer,
6. ByteArrayVertexIdMessages<I, M>>>
7. remainingMessageCache = sendMessageCache.removeAllMessages();
8. PairList<WorkerInfo,
9. PairList<Integer, ByteArrayVertexIdMessages<I, M>>>.Iterator
10. iterator = remainingMessageCache.getIterator();
11. **while** (iterator.hasNext()) {
12. iterator.next();
13. WritableRequest writableRequest =
14. **new** SendWorkerMessagesRequest<I, M>(
15. iterator.getCurrentSecond());
16. doRequest(iterator.getCurrentFirst(), writableRequest);
17. }
18. …
19. }