Hadoop MR任务本地性

MR是Hadoop的计算框架，用于大规模数据集的并行计算，其中核心的概念是映射（Map）和规约（Reduce），Map把一组键值对映射成一组新的键值对，指定并发的Reduce函数，用于保证所有映射的键值对中每个共享相同的键组，Reduce对Map的输出结果进行合并。NativeTask是Hadoop MapReduce到的高效执行引擎实现，其替换原MR框架中的Map阶段部分模块。MapReduce的执行流程如下图所示：



Map任务提交到Yarn后，被ApplicationMaster启动，任务的形式是YarnChild，在其中会执行MapTask的run方法。在提交之前，JobClient会对数据源进行切片，切片信息由InputSplit对象封装，接口定义如下：

1. public interface InputFormat<K, V> {
2. InputSplit[] getSplits(JobConf job, int numSplits) throws IOException;
3. RecordReader<K, V> getRecordReader(InputSplit split,JobConf job,Reporter reporter) throws IOException;
4. }

JobClient通过getSplits方法来计算切片信息，切片默认大小和HDFS的块大小相同。JobSplitWriter将Splits信息（SplitMetaInfo），写入任务执行目录的文件中，SplitMetaInfo保存了该Split的数据大小及数据所在的位置。

数据的切片大致流程（FileInputFormat#getSplits），计算输入文件的大小，根据分片数（默认HDFS块大小，也可以配置Map数目）计算每个分片的大小，获取每个分片的BlockLocation，将位置信息写入SplitsMetaInfo中。

MapTask.run，根据splitIndex从SplitsMetaInfo文件中获取对应的InputSplit信息，然后创建RecordReader，从数据文件中获取Key及Value。然后调用Mapper.run，执行定义的操作。

# Map任务本地性

Map任务本地性主要是根据InputSplit中的元数据信息，将Task提交到对应的主机上（任务本地性）的核心是Task的host选择算法。

## 1.1 Map 处理数据元数据

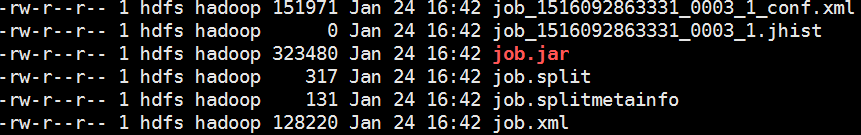
JobSplit的类图如下：



SplitMetaInfo类描述一个InputSplit的元数据，如下所示：

1. public static class SplitMetaInfo implements Writable {
2. private long startOffset;
3. private long inputDataLength;
4. private String[] locations;
5. }

所有的InputSplit对应的SplitMetaInfo都存在job.splitmetainfo文件中（该文件放置在.staging/job目录下），如下：



.split和.splitmetainfo中保存了要处理的文件及位置信息，示例如下所示：

FileSplit={file=hdfs://server1:9000/user/admin/in/yellow.txt,hosts= [server3, server2], length=67108864,start=0}

SplitMetainfo：

JobSplit$SplitMetaInfo ={data-size:67108864，start-offset:7，locations:[server3, server2]}

当JobImpl创建Task时，从SplitMetaInfo文件中读取内容生成TaskSplitMetaInfo，然如传入MapTaskImpl中，作为MapTask的数据输入文件，TaskSplitMetaInfo的数据结构示例如下：

*TaskSplitMetaInfo[0]={*

*inputDataLength=67108864,*

*locations=[server3, server2],*

*splitIndex=JobSplit$TaskSplitIndex{../.staging/job\_201404200521\_0001/job.split" ,*

*startOffset=7 }*

*}*

Task执行时从TaskSplitMetaInfo中获取要处理的文件信息，包括文件路径，处理长度和启动Offset等。至此任务本地性所需要的元数据已经生成。

## 1.2 Job setup

在HDFS中，文件是以block为单位的，所有一个大文件在HDFS上对应多个Block，分散到各个节点上，从文件切片算法上看，一个InputSplit也可能对应多个Block分布到不同节点，因此Hadoop中的任务不可能完全的数据本地性。在JobImpl中会创建Map Tasks，在该过程中根据TaskSplitMetaInfo信息生成

TaskAttemptImpl，其中配置数据所在的主机：

*protected Set<String> dataLocalHosts;*

*protected Set<String> dataLocalRacks;*

然后调用JobImpl#schedulerTask将其提交到YARN中。下一步是启动MRAppMaster后，从YARN RM中获取资源，然后调度Task的执行，也就是任务本地性的核心。

## 1.3 Host选择算法

在MRAppMaster中的AppContext中维护Attempt信息， 在MRAppMaser的createJob中将JobImpl信息添加到AppContext中，即TaskAttemptImpl添加到MRAppMaster中，JobImpl中维护Task的数据结构如下所示：

*volatile Map<TaskId, Task> tasks = new LinkedHashMap<TaskId, Task>();*

在MRAppMaster从RM中获取一个Container，并启动Attempt后，TaskAttempt会进行数据本地性的计算，代码如下：

*private void computeRackAndLocality() {*

*NodeId containerNodeId = container.getNodeId();*

*nodeRackName = RackResolver.resolve(containerNodeId.getHost()).getNetworkLocation();*

*locality = Locality.OFF\_SWITCH;*

*if (dataLocalHosts.size() > 0) {*

*String cHost = resolveHost(containerNodeId.getHost());*

*if (dataLocalHosts.contains(cHost)) {*

*locality = Locality.NODE\_LOCAL;*

*}}*

*if (locality == Locality.OFF\_SWITCH) {*

*if (dataLocalRacks.contains(nodeRackName)) {*

*locality = Locality.RACK\_LOCAL;*

*}}}*

MRAppMaster与RM交互获取的containers后，进行分配，如下所示：

*private void assignContainers(List<Container> allocatedContainers) {*

*Iterator<Container> it = allocatedContainers.iterator();*

*while (it.hasNext()) {*

*Container allocated = it.next();*

*ContainerRequest assigned = assignWithoutLocality(allocated);*

*if (assigned != null) {*

*containerAssigned(allocated, assigned);*

*it.remove();*

*}*

*}*

*assignMapsWithLocality(allocatedContainers);*

*}*

主要是分配算法是assignWithoutLocality和assignMapsWithLocality中执行。

其中assignWithoutLocality的分析如下：

*private ContainerRequest assignWithoutLocality(Container allocated) {*

*ContainerRequest assigned = null;*

*Priority priority = allocated.getPriority();*

*//优先级是PRIORITY\_FAST\_FAIL\_MAP，优先级为5，FAST\_FAIL\_MAP（失败的map过的优先执行）*

*//该map放在RMContainerAllocator的earlizerFailedMaps数据结构中，不会考虑数据本地性*

*if (PRIORITY\_FAST\_FAIL\_MAP.equals(priority)) {*

*LOG.info("Assigning container " + allocated + " to fast fail map");*

*assigned = assignToFailedMap(allocated);*

*} else if (PRIORITY\_REDUCE.equals(priority)) {*

*//REDUCE任务，优先级别比MAP高，不考虑数据本地性，*

*// 因为reduce从Map中获取中间数据，不存在数据本地性的因素*

*assigned = assignToReduce(allocated);*

*}*

*return assigned;*

*}*

以上经过FAST\_FAILED\_MAP及REDUCE的任务分配后，再进行MapTask的分配，流程图（核心算法）如下：



在RMContainerAllocator中维护了三个数据结构：

* mapsHostMapping：MAP<String,LinkedList<TaskAttemptId>>，维护了主机名与有数据本地性相关的TaskAttemptId的对应关系
* mapsRackMapping:Map<String,LinkedList<TaskAttemptId>>，维护了机架与有数据本地性相关的TaskAttemptId的对应关系
* maps:Map<TaskAttemptId，ContainerRequest>，维护待分配的TaskAttemptId

从流程图上看，map任务的数据本地性并没有什么复杂的算法，而且根据前期的TaskSplitMetaInfo信息构建上述的三种TaskAttemptId的数据列表，当有分配Container时，查询其对应的host，然后进行对应流程的分配，优先级分别为：host -> rack -> others。

# Reduce的任务本地性

源码如下：

*private ContainerRequest assignToReduce(Container allocated) {*

*ContainerRequest assigned = null;*

*//try to assign to reduces if present*

*if (assigned == null && reduces.size() > 0 && canAssignReduces()) {*

*TaskAttemptId tId = reduces.keySet().iterator().next();*

*assigned = reduces.remove(tId);*

*LOG.info("Assigned to reduce");*

*}*

*return assigned;}*

从代码中可以看到，reduce并没有数据本地性，因为reduce任务是从map中fetch中间数据，在map未运行之前是无法知道map运行在哪些机器上，而且map一般很多，reduce均需要从map中获取数据，因此没有数据本地性的需求。