Hadoop OS

“大数据胜于好算法”

计算机硬盘的发展趋势：寻址时间的提升远远不敌于传输速率的提升
–所以瓶颈是提升寻址时间或者通过其他方案来提升整体的寻址时间
–Hadoop就像H5，不是一个技术的描述，而是一组相关技术：Common、Avro、MR、HDFS、PIG、Hive、HBase、ZooKeeper、Sqoop、Oozie

MR

Hadoop将MR的输入数据进行分片（input split），Hadoop为每个分片构建一个map任务，并由该任务来运行用户订立的map函数，从而处理分片中的每条记录（**如何分片？**）
分片越小，越好负载均衡（好的机器执行更多分片）；但是太小，则会产生太多map，消耗过大，所以一般大小是HDFS块大小（默认64M，如果分片大于块大小，那需要的两块数据很大概率上其中一块位于其他机器，那就需要进行网络传输，降低效率，违背了‘使用本地数据计算’的规则），可以调节。

map的数据是写入本地（map产生的是中间数据，输出后即删除），如果map产生的数据传给reduce之前失败，则Hadoop将在另一个节点上重新运行map任务，以再次构建map中间结果
Reduce数据来源是map的输出汇总，按照key汇总给reduce，所以reduce数据一般是需要网络传输，结果存储在HDFS上，一份存在本地机器，其他存在其他机器或机架。

从MR的编程模型来看，reduce按照key来分配，如果key很多，那reduce的数量小于key，则需要reduce排队处理

Combiner可以将map的输出按照key来合并，减少map和reduce之间的交互。但是不是也增加了Map和Combiner这一层的交互，还增加了Combiner和Reduce之间的交互，有什么好处呢？Combiner是Map输出的处理，即每个Map的输出都会经过Combiner处理，减少的不是交互，而是传输的数据。例如求最小值，Combiner处理后，可能传给reduce就是这个Map中key对应的最小值，而不是这个key对应的所有制等着reduce执行。而reduce处理不同map之间该key的最小值。

Hadoop Streaming

Hadoop Streaming使用了Unix标准流作为hadoop和应用程序之间的接口。Map的输入数据通过标准输入流传递给map函数，并且一行一行的传输，最后将行写到标准输出。输出的键值对是以制表符为分隔符
Hadoop Streaming需要自己找到key的边界，这个看看代码就明白了。

Hadoop Pipes：MR的C++接口名称

HDFS

每个文件、目录和数据块的存储信息大约占150bytes，用户这个和文件数就能算出namenode耗费了多少内存（这个150bytes可能会不同版本有不同值）

磁盘块——文件系统块（OS的文件系统块大小一般是几K）
HDFS的块大小事64M，这么大的目的是为了最小化寻址开销。如果块太大，Map的任务通常一次只处理一个块中的数据，这样块太大就会减少map的数量，降低处理效率。
`hdfs fsck / -files -blocks` 查看各个文件由哪些块组成

HDFS高可用（HA）

每个namenode都运行着一个轻量级的故障转移控制器（failover_controller），监视宿主nn是否失效（通过心跳实现），并在nn失效时进行故障切换。（通过ZK和HA配合处理热切）
客户端的故障切换通过客户端类库实现透明处理，最简单的实现是通过客户端的配置文件实现故障切换的控制。HDFS URI使用一个逻辑主机名，该主机名映射到一对NN地址（配置文件），客户端类库会访问每一个nn地址，直至处理完成。
详见：
http://blog.csdn.net/tantexian/article/details/44964587

HDFS 访问

- 读取
  client调用FileSystem对象的open方法，通过本地库中的DistributeFileSystem类用RPC来调用namenode，以确定文件起始块的位置，对于每一个块，namenode返回datanode的地址。DistributeFileSystem类返回一个FSDataInputStream对象，该类管理着datanode和namenode的I/O。Client从流中读取数据时，块是按照打开DFSInputStream与datanode建立连接的顺序读取（块是串行读取）。而且一旦某个dn出错，则从其他dn读取（最近的一个），所以这个是在Client端的DFSInputStream完成的。
- 写入（是哪一步client从nn获得存储的dn和块信息列表的？如果一个dn出错，具体和nn交流是怎样的？）
  像上面一样，只不过，DistributeFileSystem返回的是DFSoutPutStream，用来负责nn和dn之间的通信。FSDataOutputStream封装了DFSoutPutStream类。客户端写入数据时，DFSOutputStream将它分成一个个的数据包，并写入内部队列，称为数据队列。DataStreamer处理数据队列，它的责任是根据datanode列表来要求namenode分配适合的新块来存储数据复本。这一组datanode构成一个管线，DataStreamer将数据包流式传输到管线中的第一个dn上，该dn存储数据包并发给第二个dn，以此类推。DFSOutputStream也维护着一个内部数据包队列等待datanode的确认回执，等收到管道内所有datanode的回执后将其删除。

如果写入期间，某个datanode故障，则先关闭管线，确认把队列中所有数据包添加回数据队列的前端，保证故障节点下游的datanode不会漏掉任何一个数据（但是不对啊，也只有管线的数据会影响啊），为存储在另一个正常的datanode的当前数据块指定一个新的标识，并将该标识返回给nn——这样做的目的是等到故障dn恢复，会自动删除故障时存储的该块的部分数据。从管线中删除故障数据节点（删除故障dn），并且将余下的块写入管线中其他正常的dn。只要写入了dfs.replication.min的复本数（默认为1），写操作就会成功，并且这个块可以在集群中异步复制，直到达到符合replication因子数。
`distcp` 集群间文件拷贝，通过并行产生很多map来达到并行拷贝的目的。

Namenode and Datanode

namenode-管理者；datanode-工作者

Namenode

namenode管理文件系统的命名空间，维护着文件系统树及整棵树内所有的文件和目录。这些信息以两个文件形式永久保存在本地磁盘上：命名空间镜像文件和编辑日志文件。
namenode也记录每个文件中各个块所在数据节点的信息，不永久保存块位置信息，这个信息是每次系统启动时由数据节点重建。
可以通过hadoop-HA和Zookeeper来一起实现namenode的自动切换。

联邦HDFS

由于namenode的内存可能限制集群大小（文件索引），所以namenode的横向扩展是一个解决方案。每个namenode单独维护自己的目录数据（namespace），这个namespace下的block组成一个block pool。namespace和block pool一起叫做namespace卷。每一个namespace是自己管理自己，删除某个namespace，这个namespace的block pool也就被删除了。datanode需要和所有的namenode注册到每个namenode，并且来自多个namespace的block pool中的数据块。
federation hdfs增加了一个NameServiceID，用来标识某个Namenode以及其Secondary、backup、checkpoint node等。
Client要访问Federation HDFS，需要使用ViewFS（View File System）
namenode format的时候需要指定clusterID（format第一个namenode的时候format，如果不指定，则自动产生，后续format使用的clusterID必须都一致——因为ViewFS访问的时候要用：viewsFS://clusterX）

DataNode

DN定期向Namenode发送自己存储的块列表

Hadoop I/O

完整性校验

datanode在读取、写入数据时都需要校验数据和校验码是否匹配，写数据时，由管线的最后一个dn来检查。
每个dn后台也运行一个线程——DataBlockScanner，定期验证存储在这个datanode上的块数据的校验和。
如果检测到某个块错误，则先通知namenode,namenode将这个块标识为已损坏，之后安排这个数据块的一个复本复制到另一个dn上
文件压缩，由于大部分压缩算法不支持切分（就是map可以按照文件块处理，处理bzip2，其他都不支持切分），所以map处理一个文件（大于块大小），则失去了本地处理优势，需要从其他地方将这个文件的块传过来处理。所以选择可切分的压缩格式可能会好点（bzip2相比其他可能压缩比较慢）
如果要在mr中压缩输出结果，则需要进行设置，`mapreduce.output.compress=true`
由于map本身处理完之后，结果就是要网络传输给reduce，所以讲map结果压缩传输可能会有比较好的效果
Avro：数据序列化系统

序列化
SequenceFile、MapFile（有索引键）
需要练习

MapReduce and Yarn

Hadoop任务调度

最开始就是FIFO，但不久之后，增加了设置作业优先级的功能，在默认的FIFO调度基础上，增加了公平调度器（Fair Scheduler）和容量调度器（Capacity Scheduler）

公平调度器

和Capacity Schedule一样，是可插拔的调度器（可作为插件配置），都是作为多用户共享资源使用。不同的是调度策略和内存约束管理。

容量调度器

适合多用户共享集群环境，安全可靠的使用集群资源。以队列划分资源，每个队列可以设置资源使用上限（每个队列分配一定的容量）。同时每个用户也可以设置资源使用上限。（队列可能源自用户任务集合）

Shuffle和排序

Reduce得到的输入都是按键排序的，这个排序过程称为Shuffle，现在看来，Shuffle是存在于Map和Reduce两端的。

Map端

Map有个缓冲区（默认100M），map的输出先写入该缓冲区，当缓冲区达到阈值（可以配置，默认80%）时，写入磁盘（此时map还可以继续往缓冲区写，直到缓冲区填满阻塞。但是这里有个问题，达到阈值后写入磁盘，是写多少？全部写入还是写一个固定大小？），达到缓冲区叫溢出。每次溢出创建一个溢出文件，任务完成之前会有几个溢出文件，如果大于3个，会进行合并（如果有combiner的话，小于3个，会觉得合并没必要）。在写磁盘之前，线程会根据最终要传的reducer把数据划分成相应的分区，每个分区中，后台线程按键进行内排序（shuffle），如果有combiner，就在排序后的输出上运行，如果有压缩，则进行压缩处理。

Reduce端

Reduce通过http获得map输出文件的分区，由于reduce会从多个map获得数据，而map的完成时间不同，所以一旦有map完成，则reduce就开始复制数据，这就是reduce的复制阶段。reduce有少量的复制线程，因此能够并行取的map的输出，默认值是5个线程（这个值可配置）

> reducer如何知道从哪些map读呢？当map运行完，tasktracker将map和自己的映射信息包含在给jobtracker的心跳中，reduce的一个线程会定期询问jobtracker，一遍获得map的输出位置，直到获得所有输出位置。在tasktracker将map输出位置以及自己的映射信息发送给jobtracker之后，不马上删除这个输出（因为reduce可能失败），而是等待jobtracker通知自己删除。

如果map的输出比较小，则被复制到reduce任务的JVM内存中（缓冲区大小可配置，指定此用途的堆空间的百分比），否则写入磁盘。一旦缓冲区达到阈值，则合并后溢出写入磁盘中，如果指定了combiner，则在合并期间运行它以降低写入磁盘的数据量（由于不同map之间可能有相同的键值，所以这个时候combiner还是有必要的）。随着写入磁盘的副本增多，后台线程会将他们合并为更大的、排好序的文件。为了合并，map中压缩的输出都必须在内存中被解压。
复制完所有map输出后，进行合并排序（虽然说map已经排序，但是不同map间不用排序？）。最后是reduce阶段，对已排序输出中的每个键调用reduce函数。

任务执行

推测执行

由于将一个job拆分成多个task执行，总有拖后腿的，当所有task执行一段时间后，发现有的比预期慢，可以再起一个task执行该task的任务，这两个task谁先执行完就听谁的，然后关闭另一个。推测执行默认是打开的，也可以关闭。可以针对map和reduce分别设置，reduce推荐关闭，因为reduce要复制数据。

OutputCommiters

例如创建_SUCCESS文件等

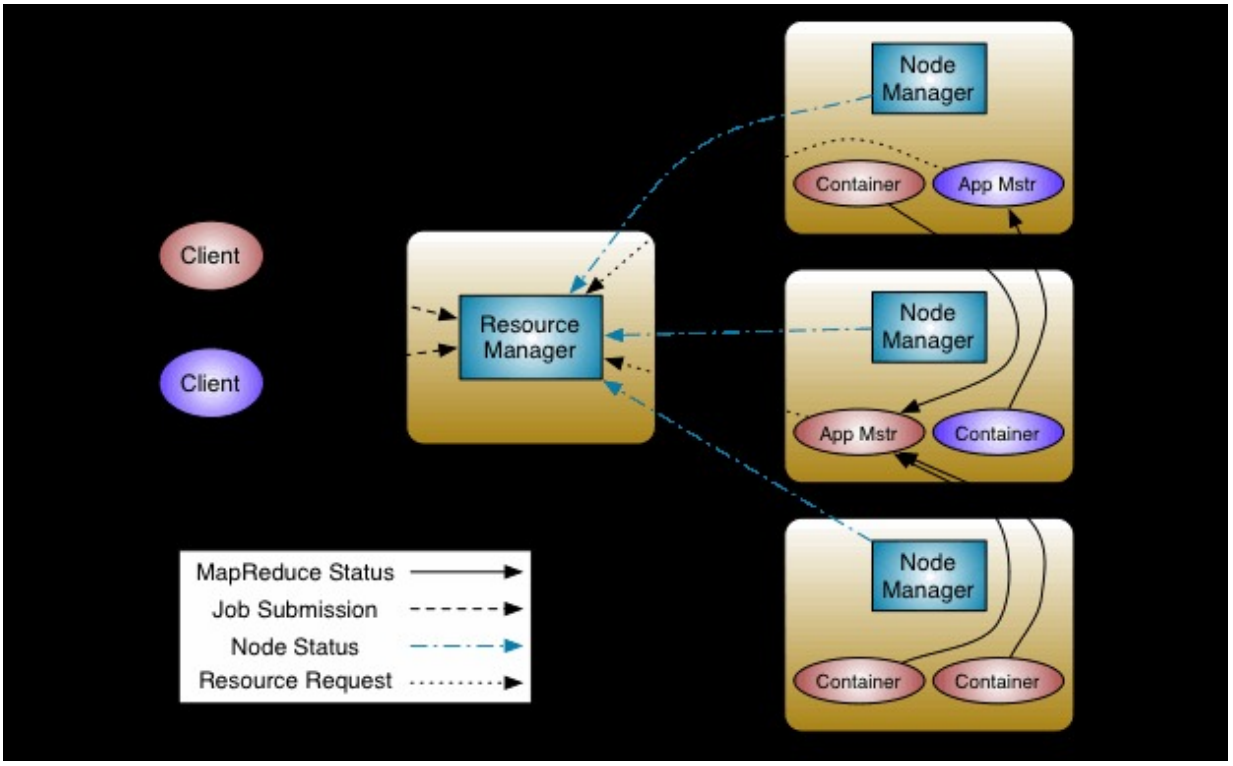
JVM重用：不同job的task肯定是在不同JVM中执行

Hadoop管理

块位置信息不是由nn来维护，而是以块列表的形式保存在datanode上，namenode的安全模式时，datanode发送位置信息给namenode

MR 应用

Ooize：作为服务器运行
Maven配置



需要练习