分布式计算开源框架 Hadoop 入门实践

作者介绍：*岑文初，就职于阿里软件公司研发中心平台一部，任架构师。当前主要工作涉及阿里软件开发平台服务框架（ASF）设计与实现，服务集成平台（SIP）设计与实现。没有什么擅长或者精通，工作到现在唯一提升的就是学习能力和速度。*

一、分布式计算开源框架 Hadoop 实践

在 SIP 项目设计的过程中，对于它庞大的日志在开始时就考虑使用任务分解的多线程处理模式来分析统计，在我从前写的文章《Tiger Concurrent Practice --日志分析并行分解设计与实现》中有所提到。但是由于统计的内容暂时还是十分简单，所以就采用 Memcache 作为计数器，结合 MySQL 就完成了访问控制以及统计的工作。然而未来，对于海量日志分析的工作，还是需要有所准备。现在最火的技术词汇莫过于"云计算"，在 Open API 日益盛行的今天，互联网应用的数据将会越来越有价值，如何去分析这些数据，挖掘其内在价值，就需要分布式计算来支撑海量数据的分析工作。

回过头来看，早先那种多线程，多任务分解的日志分析设计，其实是分布式计算的一个单机版缩略，如何将这种单机的工作进行分拆，变成协同工作的集群，其实就是分布式计算框架设计所涉及的。在去年参加 BEA 大会的时候，BEA 和 VMWare 合作采用虚拟机来构建集群，无非就是希望使得计算机硬件能够类似于应用程序中资源池的资源，使用者无需关心资源的分配情况，从而最大化了硬件资源的使用价值。分布式计算也是如此，具体的计算任务交由哪一台机器执行，执行后由谁来汇总，这都由分布式框架的 Master 来抉择，而使用者只需简单地将待分析内容提供给分布式计算系统作为输入，就可以得到分布式计算后的结果。

Hadoop 是 Apache 开源组织的一个分布式计算开源框架，在很多大型网站上都已经得到了应用，如亚马逊、Facebook 和 Yahoo 等等。 对于我来说，最近的一个使用点就是服务集成平台的日志分析。服务集成平台的日志量将会很大，而这也正好符合了分布式计算的适用场景（日志分析和索引建立就 是两大应用场景）。

当前没有正式确定使用，所以也是自己业余摸索，后续所写的相关内容，都是一个新手的学习过程，难免会有一些错误，只是希望记录下来可以分享给更多志同道合的朋友。

什么是 Hadoop？

搞什么东西之前，第一步是要知道 What（是什么），然后是 Why（为什么），最后才是 How（怎么做）。但很多开发的朋友在做了多年项目以后，都习惯是先 How，然后 What，最后才是 Why，这样只会让自己变得浮躁，同时往往会将技术误用于不适合的场景。

Hadoop 框架中最核心的设计就是：MapReduce 和 HDFS。MapReduce 的思想是由 Google 的一篇论文所提及而被广为流传的， 简单的一句话解释 MapReduce 就是"任务的分解与结果的汇总"。HDFS 是 Hadoop 分布式文件系统（Hadoop Distributed File System）的缩写，为分布式计算存储提供了底层支持。

MapReduce 从它名字上来看就大致可以看出个缘由，两个动词 Map 和 Reduce，“Map（展开）”就是将一个任务分解成为多个任 务，“Reduce”就是将分解后多任务处理的结果汇总起来，得出最后的分析结果。这不是什么新思想，其实在前面提到的多线程，多任务的设计就可以找到这 种思想的影子。不论是现实社会，还是在程序设计中，一项工作往往可以被拆分成为多个任务，任务之间的关系可以分为两种：一种是不相关的任务，可以并行执 行；另一种是任务之间有相互的依赖，先后顺序不能够颠倒，这类任务是无法并行处理的。回到大学时期，教授上课时让大家去分析关键路径，无非就是找最省时的 任务分解执行方式。在分布式系统中，机器集群就可以看作硬件资源池，将并行的任务拆分，然后交由每一个空闲机器资源去处理，能够极大地提高计算效率，同时 这种资源无关性，对于计算集群的扩展无疑提供了最好的设计保证。（其实我一直认为 Hadoop 的卡通图标不应该是一个小象，应该是蚂蚁，分布式计算就好比 蚂蚁吃大象，廉价的机器群可以匹敌任何高性能的计算机，纵向扩展的曲线始终敌不过横向扩展的斜线）。任务分解处理以后，那就需要将处理以后的结果再汇总起 来，这就是 Reduce 要做的工作。

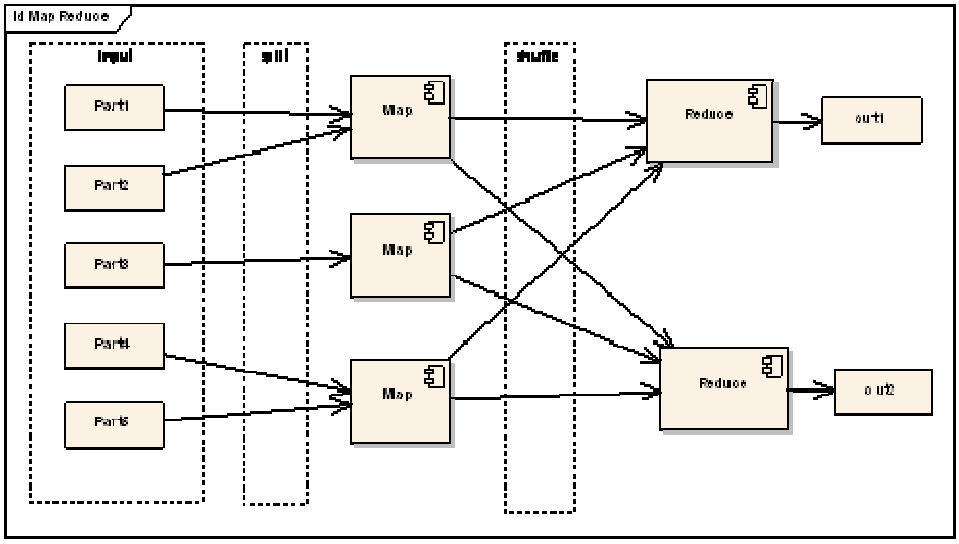


图 1：MapReduce 结构示意图

上图就是 MapReduce 大致的结构图，在 Map 前还可能会对输入的数据有 Split（分割）的过程，保证任务并行效率，在 Map 之后还会有 Shuffle（混合）的过程，对于提高 Reduce 的效率以及减小数据传输的压力有很大的帮助。后面会具体提及这些部分的细节。

HDFS 是分布式计算的存储基石，Hadoop 的分布式文件系统和其他分布式文件系统有很多类似的特质。分布式文件系统基本的几个特点：

1. 对于整个集群有单一的命名空间。
2. 数据一致性。适合一次写入多次读取的模型，客户端在文件没有被成功创建之前无法看到文件存在。
3. 文件会被分割成多个文件块，每个文件块被分配存储到数据节点上，而且根据配置会由复制文件块来保证数据的安全性。

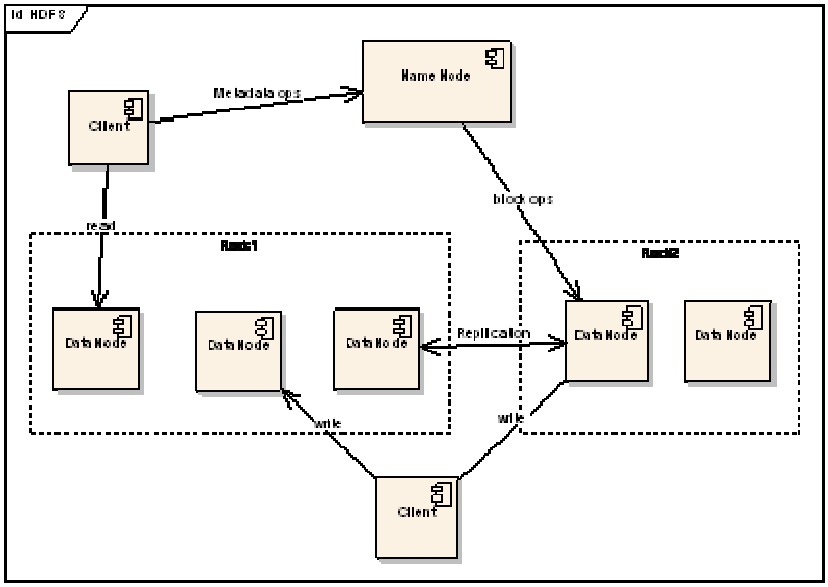


图 2：HDFS 结构示意图

上图中展现了整个 HDFS 三个重要角色：NameNode、DataNode 和 Client。NameNode 可以看作是分布式文件系统中的管理 者，主要负责管理文件系统的命名空间、集群配置信息和存储块的复制等。NameNode 会将文件系统的 Meta-data 存储在内存中，这些信息主要包括 了文件信息、每一个文件对应的文件块的信息和每一个文件块在 DataNode 的信息等。DataNode 是文件存储的基本单元，它将 Block 存储在本地 文件系统中，保存了 Block 的 Meta-data，同时周期性地将所有存在的 Block 信息发送给 NameNode。Client 就是需要获取分布式文 件系统文件的应用程序。这里通过三个操作来说明他们之间的交互关系。

文件写入：

1. Client 向 NameNode 发起文件写入的请求。
2. NameNode 根据文件大小和文件块配置情况，返回给 Client 它所管理部分 DataNode 的信息。
3. Client 将文件划分为多个 Block，根据 DataNode 的地址信息，按顺序写入到每一个 DataNode 块中。

文件读取：

1. Client 向 NameNode 发起文件读取的请求。
2. NameNode 返回文件存储的 DataNode 的信息。
3. Client 读取文件信息。

文件 Block 复制：

1. NameNode 发现部分文件的 Block 不符合最小复制数或者部分 DataNode 失效。
2. 通知 DataNode 相互复制 Block。
3. DataNode 开始直接相互复制。

最后再说一下 HDFS 的几个设计特点（对于框架设计值得借鉴）：

1. Block 的放置：默认不配置。一个 Block 会有三份备份，一份放在 NameNode 指定的 DataNode，另一份放在 与指定 DataNode 非同一 Rack 上的 DataNode，最后一份放在与指定 DataNode 同一 Rack 上的 DataNode 上。备份无非就是为了 数据安全，考虑同一 Rack 的失败情况以及不同 Rack 之间数据拷贝性能问题就采用这种配置方式。
2. 心跳检测 DataNode 的健康状况，如果发现问题就采取数据备份的方式来保证数据的安全性。
3. 数 据复制（场景为 DataNode 失败、需要平衡 DataNode 的存储利用率和需要平衡 DataNode 数据交互压力等情况）：这里先说一下，使用 HDFS 的 balancer 命令，可以配置一个 Threshold 来平衡每一个 DataNode 磁盘利用率。例如设置了 Threshold 为 10%，那么 执行 balancer 命令的时候，首先统计所有 DataNode 的磁盘利用率的均值，然后判断如果某一个 DataNode 的磁盘利用率超过这个均值 Threshold 以上，那么将会把这个 DataNode 的 block 转移到磁盘利用率低的 DataNode，这对于新节点的加入来说十分有用。
4. 数据交验：采用 CRC32 作数据交验。在文件 Block 写入的时候除了写入数据还会写入交验信息，在读取的时候需要交验后再读入。
5. NameNode 是单点：如果失败的话，任务处理信息将会纪录在本地文件系统和远端的文件系统中。
6. 数 据管道性的写入：当客户端要写入文件到 DataNode 上，首先客户端读取一个 Block 然后写到第一个 DataNode 上，然后由第一个 DataNode 传递到备份的 DataNode 上，一直到所有需要写入这个 Block 的 NataNode 都成功写入，客户端才会继续开始写下一个 Block。
7. 安全模式：在分布式文件系统启动的时候，开始的时候会有安全模式，当分布式文件系统处于安全模式的情况下，文 件系统中的内容不允许修改也不允许删除，直到安全模式结束。安全模式主要是为了系统启动的时候检查各个 DataNode 上数据块的有效性，同时根据策略必 要的复制或者删除部分数据块。运行期通过命令也可以进入安全模式。在实践过程中，系统启动的时候去修改和删除文件也会有安全模式不允许修改的出错提示，只需要等待一会儿即可。

下面综合 MapReduce 和 HDFS 来看 Hadoop 的结构：

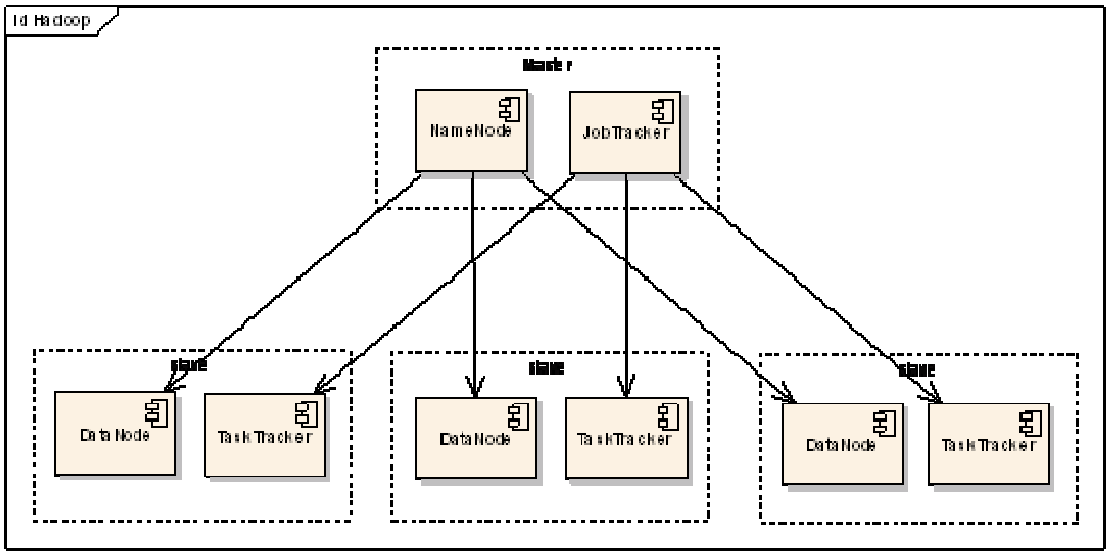


图 3：Hadoop 结构示意图

在 Hadoop 的系统中，会有一台 Master，主要负责 NameNode 的工作以及 JobTracker 的工作。JobTracker 的主要职责 就是启动、跟踪和调度各个 Slave 的任务执行。还会有多台 Slave，每一台 Slave 通常具有 DataNode 的功能并负责 TaskTracker 的 工作。TaskTracker 根据应用要求来结合本地数据执行 Map 任务以及 Reduce 任务。

说到这里，就要提到分布式计算最重要的一个设计点：Moving Computation is Cheaper than Moving Data。就是在分布式处理中，移动数据的代价总是高于转移计算的代价。简单来说就是分而治之的工作，需要将数据也分而存储，本地任务处理本地数据然后归 总，这样才会保证分布式计算的高效性。

为什么要选择 Hadoop？

说完了 What，简单地说一下 Why。官方网站已经给了很多的说明，这里就大致说一下其优点及使用的场景（没有不好的工具，只用不适用的工具，因此选择好场景才能够真正发挥分布式计算的作用）：

1. 可扩展：不论是存储的可扩展还是计算的可扩展都是 Hadoop 的设计根本。
2. 经济：框架可以运行在任何普通的 PC 上。
3. 可靠：分布式文件系统的备份恢复机制以及 MapReduce 的任务监控保证了分布式处理的可靠性。
4. 高效：分布式文件系统的高效数据交互实现以及 MapReduce 结合 Local Data 处理的模式，为高效处理海量的信息作了基础准备。

使用场景：个人觉得最适合的就是海量数据的分析，其实 Google 最早提出 MapReduce 也就是为了海量数据分析。同时 HDFS 最早是为了搜索引擎实现而开发的，后来才被用于分布式计算框架中。海量数据被分割于多个节点，然后由每一个节点并行计算，将得出的结 果归并到输出。同时第一阶段的输出又可以作为下一阶段计算的输入，因此可以想象到一个树

状结构的分布式计算图，在不同阶段都有不同产出，同时并行和串行结 合的计算也可以很好地在分布式集群的资源下得以高效的处理。

二、Hadoop 中的集群配置和使用技巧

其实参看 Hadoop 官方文档已经能够很容易配置分布式框架运行环境了，不过这里既然写了就再多写一点，同时有一些细节需要注意的也说明一下，其实 也就是这些细节会让人摸索半天。Hadoop 可以单机跑，也可以配置集群跑，单机跑就不需要多说了，只需要按照 Demo 的运行说明直接执行命令即可。这里 主要重点说一下集群配置运行的过程。

环境

7 台普通的机器，操作系统都是 Linux。内存和 CPU 就不说了，反正 Hadoop 一大特点就是机器在多不在精。JDK 必须是 1.5 以上的，这个切记。7 台机器的机器名务必不同，后续会谈到机器名对于 MapReduce 有很大的影响。

部署考虑

正如上面我描述的，对于 Hadoop 的集群来说，可以分成两大类角色：Master 和 Slave，前者主要配置 NameNode 和 JobTracker 的角色，负责总管分布式数据和分解任务的执行，后者配置 DataNode 和 TaskTracker 的角色，负责分布式数据存储以及任 务的执行。本来我打算看看一台机器是否可以配置成 Master，同时也作为 Slave 使用，不过发现在 NameNode 初始化的过程中以及 TaskTracker 执行过程中机器名配置好像有冲突（NameNode 和 TaskTracker 对于 Hosts 的配置有些冲突，究竟是把机器名对应 IP 放在配置前面还是把 Localhost 对应 IP 放在前面有点问题，不过可能也是我自己的问题吧，这个大家可以根据实施情况给我反馈）。最后反正决定一 台 Master，六台 Slave，后续复杂的应用开发和测试结果的比对会增加机器配置。

实施步骤

1. 在所有的机器上都建立相同的目录，也可以就建立相同的用户，以该用户的 home 路径来做 hadoop 的安装路径。例如我在所有的机器上都建立了/home/wenchu。
2. 下载 Hadoop，先解压到 Master 上。这里我是下载的 0.17.1 的版本。此时 Hadoop 的安装路径就是/home/wenchu/hadoop-0.17.1。
3. 解压后进入 conf 目录，主要需要修改以下文件：hadoop-env.sh，hadoop-site.xml、masters、slaves。

   Hadoop 的基础配置文件是 hadoop-default.xml，看 Hadoop 的代码可以知道，默认建立一个 Job 的时候会建立 Job 的 Config，Config 首先读入 hadoop-default.xml 的配置，然后再读入 hadoop-site.xml 的配置（这个文件初始的时候配置为空），hadoop-site.xml 中主要配置你需要覆盖的 hadoop-default.xml 的系

统级配置，以及你需要在你的 MapReduce 过程中使用的自定义配置（具体的一些使用例如 final 等参考文档）。

以下是一个简单的 hadoop-site.xml 的配置：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
   <name>fs.default.name</name>//你的 namenode 的配置，机器名加端口
   <value>hdfs://10.2.224.46:54310/</value>
</property>
<property>
   <name>mapred.job.tracker</name>//你的 JobTracker 的配置，机器名加端口
   <value>hdfs://10.2.224.46:54311/</value>
</property>
<property>
   <name>dfs.replication</name>//数据需要备份的数量，默认是三
   <value>1</value>
</property>
<property>
    <name>hadoop.tmp.dir</name>//Hadoop 的默认临时路径，这个最好配置，如果在新
增节点或者其他情况下莫名其妙的 DataNode 启动不了，就删除此文件中的 tmp 目录即可。不过如果删除了 NameNode 机器的此目录，那么就需要重新执行 NameNode 格式化的命令。
    <value>/home/wenchu/hadoop/tmp/</value>
</property>
<property>
   <name>mapred.child.java.opts</name>//java 虚拟机的一些参数可以参照配置
   <value>-Xmx512m</value>
</property>
<property>
  <name>dfs.block.size</name>//block 的大小，单位字节，后面会提到用处，必须是
512 的倍数，因为采用 crc 作文件完整性校验，默认配置 512 是 checksum 的最小单元。
  <value>5120000</value>
  <description>The default block size for new files.</description>
```

```
</property>
</configuration>
```

hadoop-env.sh 文件只需要修改一个参数：

```
# The java implementation to use.  Required.
 export JAVA_HOME=/usr/ali/jdk1.5.0_10
```

配置你的 Java 路径，记住一定要 1.5 版本以上，免得莫名其妙出现问题。

Masters 中配置 Masters 的 IP 或者机器名，如果是机器名那么需要在/etc/hosts 中有所设置。Slaves 中配置的是 Slaves 的 IP 或者机器名，同样如果是机器名需要在/etc/hosts 中有所设置。范例如下，我这里配置的都是 IP：

```
Masters:
 10.2.224.46

  Slaves:
 10.2.226.40
 10.2.226.39
 10.2.226.38
 10.2.226.37
 10.2.226.41
 10.2.224.36
```

4. 建立 Master 到每一台 Slave 的 SSH 受信证书。由于 Master 将会通过 SSH 启动所有 Slave 的 Hadoop，所以需要建立单向或者双向证书保证命令执行时不需要再输入密码。在 Master 和所有的 Slave 机器上执行：ssh-keygen -t rsa。执行此命令的时候，看到提示只需要回车。然后就会在/root/.ssh/下面产生 id_rsa.pub 的证书文件，通过 scp 将 Master 机器上的这个文件拷贝到 Slave 上（记得修改名称），例如：scp root@masterIP:/root/.ssh/id_rsa.pub /root/.ssh/46_rsa.pub，然后执行 cat /root/.ssh/46_rsa.pub >>/root/.ssh/authorized_keys，建立 authorized_keys 文 件即可，可以打开这个文件看看，也就是 rsa 的公钥作为 key，user@IP 作为 value。此时可以试验一下，从 master ssh 到 slave 已经不需要密码了。由 slave 反向建立也是同样。为什么要反向呢？其实如果一直都是 Master 启动和关闭的话那么没有必要建立反 向，只是如果想在 Slave 也可以关闭 Hadoop 就需要建立反向。
5. 将 Master 上的 Hadoop 通过 scp 拷贝到每一个 Slave 相同的目录下，根据每一个 Slave 的 Java_HOME 的不同修改其 hadoop-env.sh。
6. 修改 Master 上/etc/profile：
   新增以下内容：（具体的内容根据你的安装路径修改，这步只是为了方便使用）

```
export HADOOP_HOME=/home/wenchu/hadoop-0.17.1
 export PATH=$PATH:$HADOOP_HOME/bin
```

修改完毕后，执行 source /etc/profile 来使其生效。

7. 在 Master 上执行 Hadoop namenode -format，这是第一需要做的初始化，可以看作格式化吧，以后除了在上面我提到过删除了 Master 上的 hadoop.tmp.dir 目录，否则是不需要再次执行的。
8. 然后执行 Master 上的 start-all.sh，这个命令可以直接执行，因为在 6 中已经添加到了 path 路径，这个命令是启动 hdfs 和 mapreduce 两部分，当然你也可以分开单独启动 hdfs 和 mapreduce，分别是 bin 目录下的 start-dfs.sh 和 start-mapred.sh。
9. 检查 Master 的 logs 目录，看看 Namenode 日志以及 JobTracker 日志是否正常启动。
10. 检查 Slave 的 logs 目录看看 Datanode 日志以及 TaskTracker 日志是否正常。
11. 如果需要关闭，那么就直接执行 stop-all.sh 即可。

以上步骤就可以启动 Hadoop 的分布式环境，然后在 Master 的机器进入 Master 的安装目录，执行 hadoop jar hadoop-0.17.1-examples.jar wordcount 输入路径和输出路径，就可以看到字数统计的效果了。此处的输入路径和输出路径都指的是 HDFS 中的路径，因此你可以首先通过拷贝本地文件系统中的目录到 HDFS 中的方式来建立 HDFS 中的输入路径：

hadoop dfs -copyFromLocal /home/wenchu/test-in test-in。其中/home/wenchu/test-in 是本地路径，test-in 是将会建立在 HDFS 中的路径，执行完毕以后可以通过 hadoop dfs -ls 看到 test-in 目录已经存在，同时可以通过 hadoop dfs -ls test-in 查看里面的内容。输出路径要求是在 HDFS 中不存在的，当执行完那个 demo 以后，就可以通过 hadoop dfs -ls 输出路径看到其中的内容，具体文件的内容可以通过 hadoop dfs -cat 文件名称来查看。

经验总结和注意事项（这部分是我在使用过程中花了一些时间走的弯路）：

1. Master 和 Slave 上的几个 conf 配置文件不需要全部同步，如果确定都是通过 Master 去启动和关闭，那么 Slave 机器上的配置不需要去维护。但如果希望在任意一台机器都可以启动和关闭 Hadoop，那么就需要全部保持一致了。
2. Master 和 Slave 机器上的/etc/hosts 中 必须把集群中机器都配置上去，就算在各个配置文件中使用的是 IP。这个吃过不少苦头，原来以为如果配成 IP 就不需要去配置 Host，结果发现在执行 Reduce 的时候总是卡住，在拷贝的时候就无法继续下去，不断重试。另外如果集群中如果有两台机器的机器名如果重复也会出现问题。
3. 如果在新增了节点或者删除节点的时候出现了问题，首先就去删除 Slave 的 hadoop.tmp.dir，然后重新启动试试看，如果还是不行那就干脆把 Master 的 hadoop.tmp.dir 删除（意味着 dfs 上的数据也会丢失），如果删除了 Master 的 hadoop.tmp.dir，那么就需要重新 namenode -format。
4. Map 任务个数以及 Reduce 任务个数配置。前面分布式文件系统设计提到一个文件被放入到分布式文件系统中，会被分割成多个 block 放置到每一个的 DataNode 上，默认 dfs.block.size 应该是 64M，也就是说如果你放置到 HDFS 上的数据小于 64，那么将只有一个 Block，此时会被放置到某一个 DataNode 中，这个可以通过使用命令：hadoop dfsadmin -report 就可以看到各个节点存储的情况。也可以直接去某一个 DataNode 查看目录：hadoop.tmp.dir/dfs/data/current 就 可

以看到那些 block 了。Block 的数量将会直接影响到 Map 的个数。当然可以通过配置来设定 Map 和 Reduce 的任务个数。Map 的个数通常默认 和 HDFS 需要处理的 blocks 相同。也可以通过配置 Map 的数量或者配置 minimum split size 来设定，实际的个数为：max(min(block_size,data/#maps),min_split_size)。Reduce 可以通过这个公式计算：0.95*num_nodes*mapred.tasktracker.tasks.maximum。

总的来说出了问题或者启动的时候最好去看看日志，这样心里有底。

Hadoop 中的命令（Command）总结

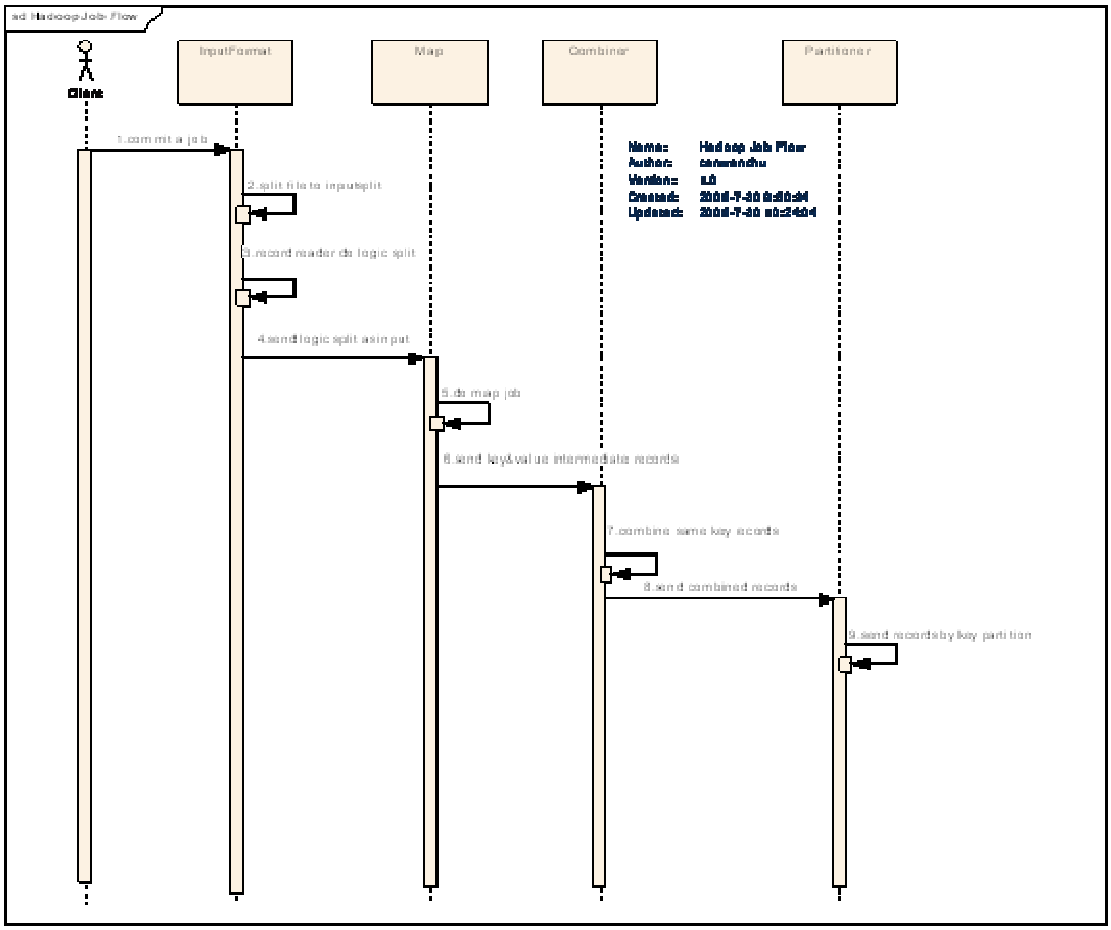
这部分内容其实可以通过命令的 Help 以及介绍了解，我主要侧重于介绍一下我用的比较多的几个命令。Hadoop dfs 这个命令后面加参数就是对于 HDFS 的操作，和 Linux 操作系统的命令很类似，例如：

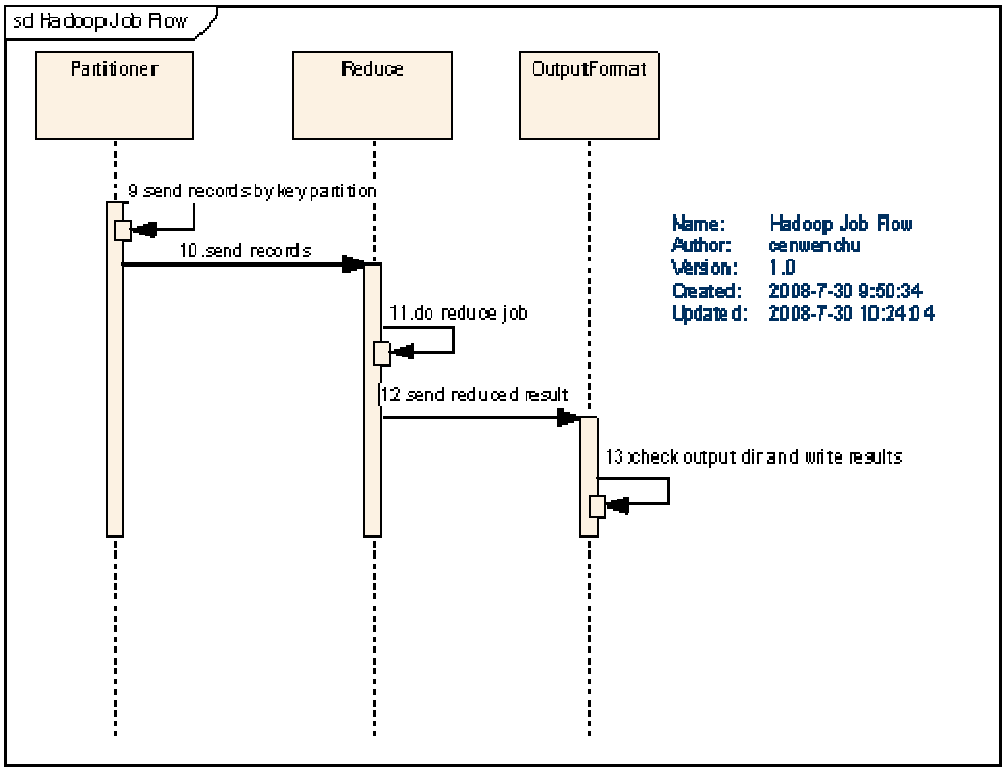
- Hadoop dfs –ls 就是查看/usr/root 目录下的内容，默认如果不填路径这就是当前用户路径；
- Hadoop dfs –rmr xxx 就是删除目录，还有很多命令看看就很容易上手；
- Hadoop dfsadmin –report 这个命令可以全局的查看 DataNode 的情况；
- Hadoop job 后面增加参数是对于当前运行的 Job 的操作，例如 list,kill 等；
- Hadoop balancer 就是前面提到的均衡磁盘负载的命令。

其他就不详细介绍了。

三、Hadoop 基本流程与应用开发

Hadoop 基本流程



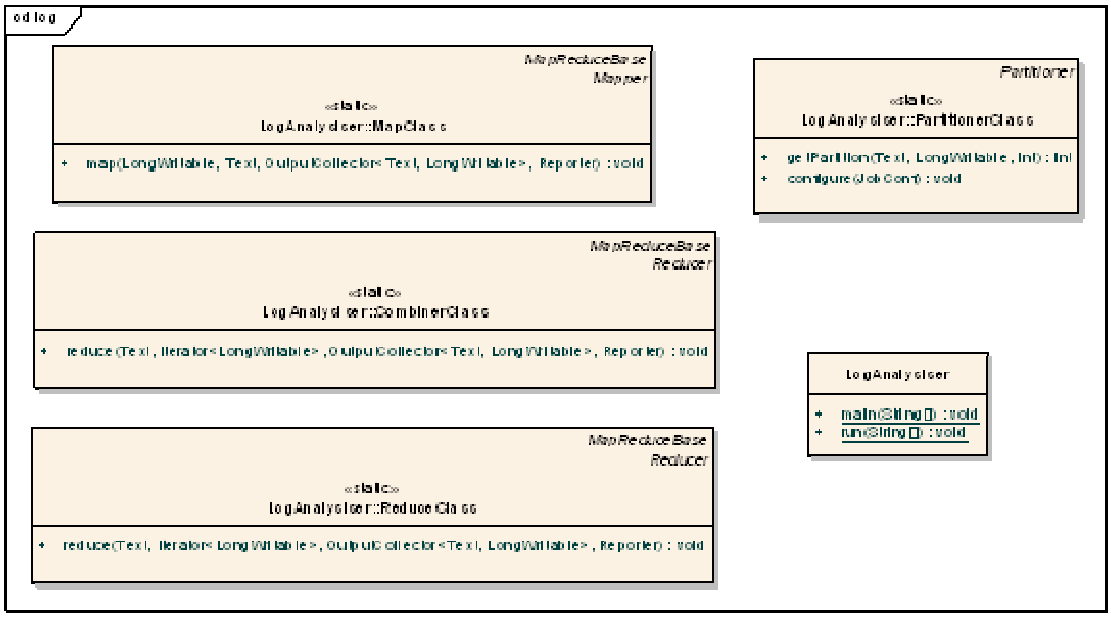


一个图片太大了，只好分割成为两部分。根据流程图来说一下具体一个任务执行的情况。

1. 在分布式环境中客户端创建任务并提交。
2. InputFormat 做 Map 前的预处理，主要负责以下工作：
    1. 验证输入的格式是否符合 JobConfig 的输入定义，这个在实现 Map 和构建 Conf 的时候就会知道，不定义可以是 Writable 的任意子类。
    2. 将 input 的文件切分为逻辑上的输入 InputSplit，其实这就是在上面提到的在分布式文件系统中 blocksize 是有大小限制的，因此大文件会被划分为多个 block。
    3. 通过 RecordReader 来再次处理 inputsplit 为一组 records，输出给 Map。（inputsplit 只是逻辑切分的第一步，但是如何根据文件中的信息来切分还需要 RecordReader 来实现，例如最简单的默认方式就是回车换行的切分）
3. RecordReader 处理后的结果作为 Map 的输入，Map 执行定义的 Map 逻辑，输出处理后的 key 和 value 对应到临时中间文件。
4. Combiner 可选择配置，主要作用是在每一个 Map 执行完分析以后，在本地优先作 Reduce 的工作，减少在 Reduce 过程中的数据传输量。
5. Partitioner 可选择配置，主要作用是在多个 Reduce 的情况下，指定 Map 的结果由某一个 Reduce 处理，每一个 Reduce 都会有单独的输出文件。（后面的代码实例中有介绍使用场景）
6. Reduce 执行具体的业务逻辑，并且将处理结果输出给 OutputFormat。
7. OutputFormat 的职责是，验证输出目录是否已经存在，同时验证输出结果类型是否如 Config 中配置，最后输出 Reduce 汇总后的结果。

业务场景和代码范例

业务场景描述：可设定输入和输出路径（操作系统的路径非 HDFS 路径），根据访问日志分析某一个应用访问某一个 API 的总次数和总流量，统计后分别输出到两个文件中。这里仅仅为了测试，没有去细分很多类，将所有的类都归并于一个类便于说明问题。



测试代码类图

LogAnalysiser 就是主类，主要负责创建、提交任务，并且输出部分信息。内部的几个子类用途可以参看流程中提到的角色职责。具体地看看几个类和方法的代码片断：

LogAnalysiser::MapClass

```
    public static class MapClass extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, LongWritable>
    {
        public void map(LongWritable key, Text value, OutputCollector
<Text,
                LongWritable> output, Reporter reporter)
                throws IOException
        {
            String line = value.toString();//没有配置 RecordReader，所
以默认采用 line
            的实现，key 就是行号，value 就是行内容
            if (line == null || line.equals(""))
                return;
            String[] words = line.split(",");
            if (words == null || words.length < 8)
                return;
            String appid = words[1];
            String apiName = words[2];
            LongWritable recbytes = new LongWritable(Long.parseLong(w
ords[7]));
            Text record = new Text();
            record.set(new StringBuffer("flow::").append(appid)
                            .append("::").append(apiName).toString());
            reporter.progress();

           //输出流量的统计结果，通过 flow::作为前缀来标示。output.col
lect(record, recbytes);
            record.clear();
            record.set(new StringBuffer("count::").append(appid).appe
nd("::")
            .append(apiName).toString());
            //输出次数的统计结果，通过 count::作为前缀来标示
            output.collect(record, new LongWritable(1));
        }
    }
```

LogAnalysiser:: PartitionerClass

```
    public static class PartitionerClass implements Partitioner<Text,
 LongWritable>
```

```
    {
        public int getPartition(Text key, LongWritable value, int num
Partitions)
        {
            if (numPartitions >= 2)//Reduce 个数，判断流量还是次数的统
计分配到不同的 Reduce
                if (key.toString().startsWith("flow::"))
                    return 0;
                else
                    return 1;
            else
                return 0;
        }
        public void configure(JobConf job){}
}
```

LogAnalysiser:: CombinerClass

参看 ReduceClass，通常两者可以使用一个，不过这里有些不同的处理就分成了两个。在 R
educeClass 中蓝色的行表示在 CombinerClass 中不存在。

LogAnalysiser:: ReduceClass

```
    public static class ReduceClass extends MapReduceBase
        implements Reducer<Text, LongWritable,Text, LongWritable>
    {
        public void reduce(Text key, Iterator<LongWritable> values,
                OutputCollector<Text, LongWritable> output, Reporter
reporter)throws IOException
        {
            Text newkey = new Text();
            newkey.set(key.toString().substring(key.toString().indexO
f("::")+2));
            LongWritable result = new LongWritable();
            long tmp = 0;
            int counter = 0;
            while(values.hasNext())//累加同一个 key 的统计结果
            {
                tmp = tmp + values.next().get();

                counter = counter +1;//担心处理太久，JobTracker 长时间
没有收到报告会认为 TaskTracker 已经失效，因此定时报告一下
                if (counter == 1000)
                {
                    counter = 0;
```

```
                    reporter.progress();
                }
            }
            result.set(tmp);
            output.collect(newkey, result);//输出最后的汇总结果
        }
    }
```

LogAnalysiser

```
        public static void main(String[] args)
        {
                try
                {
                        run(args);
                } catch (Exception e)
                {
                        e.printStackTrace();
                }
        }
        public static void run(String[] args) throws Exception
        {
                if (args == null || args.length <2)
                {
                        System.out.println("need inputpath and outputp
ath");
                        return;
                }
                String inputpath = args[0];
                String outputpath = args[1];
                String shortin = args[0];
                String shortout = args[1];
                if (shortin.indexOf(File.separator) >= 0)
                        shortin = shortin.substring(shortin.lastIndexO
f(File.separator));
                if (shortout.indexOf(File.separator) >= 0)
                        shortout = shortout.substring(shortout.lastInd
exOf(File.separator));
                SimpleDateFormat formater = new SimpleDateFormat("yyy
y.MM.dd");
                shortout = new StringBuffer(shortout).append("-")
                        .append(formater.format(new Date())).toString
();
```

```
                if (!shortin.startsWith("/"))
                        shortin = "/" + shortin;
                if (!shortout.startsWith("/"))
                        shortout = "/" + shortout;
                shortin = "/user/root" + shortin;
                shortout = "/user/root" + shortout;
                File inputdir = new File(inputpath);
                File outputdir = new File(outputpath);
                if (!inputdir.exists() || !inputdir.isDirectory())
                {
                        System.out.println("inputpath not exist or is
n’ t dir!");
                        return;
                }
                if (!outputdir.exists())
                {
                        new File(outputpath).mkdirs();
                }

                JobConf conf = new JobConf(new Configuration(),LogAna
lysiser.class);//构建 Config
                FileSystem fileSys = FileSystem.get(conf);
                fileSys.copyFromLocalFile(new Path(inputpath), new Pa
th(shortin));//将本地文件系统的文件拷贝到 HDFS 中

                conf.setJobName("analysisjob");
                conf.setOutputKeyClass(Text.class);//输出的 key 类型，
在 OutputFormat 会检查
                conf.setOutputValueClass(LongWritable.class); //输出
的 value 类型，在 OutputFormat 会检查
                conf.setMapperClass(MapClass.class);
                conf.setCombinerClass(CombinerClass.class);
                conf.setReducerClass(ReduceClass.class);
                conf.setPartitionerClass(PartitionerClass.class);
                conf.set("mapred.reduce.tasks", "2");//强制需要有两个
Reduce 来分别处理流量和次数的统计
                FileInputFormat.setInputPaths(conf, shortin);//hdfs 中
的输入路径
                FileOutputFormat.setOutputPath(conf, new Path(shortou
t));//hdfs 中输出路径

                Date startTime = new Date();
                System.out.println("Job started: " + startTime);
```

```
                JobClient.runJob(conf);
                Date end_time = new Date();
                System.out.println("Job ended: " + end_time);
                System.out.println("The job took " + (end_time.getTim
e() - startTime.getTime()) /1000 + " seconds.");
                //删除输入和输出的临时文件
                fileSys.copyToLocalFile(new Path(shortout),new Path(o
utputpath));
                fileSys.delete(new Path(shortin),true);
                fileSys.delete(new Path(shortout),true);
        }
```

以上的代码就完成了所有的逻辑性代码，然后还需要一个注册驱动类来注册业务 Class 为一个可标示的命令，让 hadoop jar 可以执行。

```
public class ExampleDriver {
  public static void main(String argv[]){
    ProgramDriver pgd = new ProgramDriver();
    try {
      pgd.addClass("analysislog", LogAnalysiser.class, "A map/reduce
program that analysis log .");
      pgd.driver(argv);
    }
    catch(Throwable e){
      e.printStackTrace();
    }
  }
}
```

将代码打成 jar，并且设置 jar 的 mainClass 为 ExampleDriver 这个类。在分布式环境启动以后执行如下语句：

```
hadoop jar analysiser.jar analysislog /home/wenchu/test-in /home/wenc
hu/test-out
```

在/home/wenchu/test-in 中是需要分析的日志文件，执行后就会看见整个执行过程，包括了 Map 和 Reduce 的进度。执行完毕会 在/home/wenchu/test-out 下看到输出的内容。有两个文件：part-00000 和 part-00001 分别记录了统计后的结果。 如果需要看执行的具体情况，可以看在输出目录下的_logs/history/xxxx_analysisjob，里面罗列了所有的 Map，Reduce 的创建情况以及执行情况。在运行期也可以通过浏览器来查看 Map,Reduce 的情况：http://MasterIP:50030 /jobtracker.jsp

Hadoop 集群测试

首先这里使用上面的范例作为测试，也没有做太多的优化配置，这个测试结果只是为了看看集群的效果，以及一些参数配置的影响。

文件复制数为 1，blocksize 5M

Slave 数	处理记录数(万条)	执行时间（秒）
2	95	38
2	950	337
4	95	24
4	950	178
6	95	21
6	950	114

Blocksize 5M

Slave 数	处理记录数(万条)	执行时间（秒）
2（文件复制数为 1）	950	337
2（文件复制数为 3）	950	339
6（文件复制数为 1）	950	114
6（文件复制数为 3）	950	117

文件复制数为 1

Slave 数	处理记录数(万条)	执行时间（秒）
6(blocksize 5M)	95	21
6(blocksize 77M)	95	26
4(blocksize 5M)	950	178
4(blocksize 50M)	950	54
6(blocksize 5M)	950	114
6(blocksize 50M)	950	44
6(blocksize 77M)	950	74

测试的数据结果很稳定，基本测几次同样条件下都是一样。通过测试结果可以看出以下几点：

1. 机器数对于性能还是有帮助的（等于没说^_^）。
2. 文件复制数的增加只对安全性有帮助，但是对于性能没有太多帮助。而且现在采取的是将操作系统文件拷贝到 HDFS 中，所以备份多了，准备的时间很长。
3. blocksize 对于性能影响很大，首先如果将 block 划分的太小，那么将会增加 job 的数量，同时也增加了协作的代价，降低了性能，但是配置的太大也会让 job 不能最大化并行处理。所以这个值的配置需要根据数据处理的量来考虑。

4. 最后就是除了这个表里面列出来的结果，应该去仔细看输出目录中的_logs/history 中的 xxx_analysisjob 这个文件，里面记录了全部的执行过程以及读写情况。这个可以更加清楚地了解哪里可能会更加耗时。

随想

"云计算"热的烫手，就和 SAAS、Web2 及 SNS 等一样，往往都是在搞概念，只有真正踏踏实实的大型互联网公司，才会投入人力物力去研究符合自 己的分布式计算。其实当你的数据量没有那么大的时候，这种分布式计算也就仅仅只是一个玩具而已，只有在真正解决问题的过程中，它深层次的问题才会被挖掘出 来。

这三篇文章（分布式计算开源框架 Hadoop 介绍，Hadoop 中的集群配置和使用技巧）仅仅是为了给对分布式计算有兴趣的朋友抛个砖，要想真的掘到金 子，那么就踏踏实实的去用、去想、去分析。或者自己也会更进一步地去研究框架中的实现机制，在解决自己问题的同时，也能够贡献一些什么。

前几日看到有人跪求成为架构师的方式，看了有些可悲，有些可笑，其实有多少架构师知道什么叫做架构？架构师的职责是什么？与其追求这么一个名号，还不如踏踏实实地做块石头沉到水底。要知道，积累和沉淀的过程就是一种成长。