单个节点的并行map,reduce个数

slots有点类似一个资源池，每个任务（map和reduce）执行时都必须获得一个slot才能继续，否则只能等待。当一个任务完成后，该任务就归还slot，这个过程有点类似释放资源到资源池中。显然，每一个获得资源的任务都可以立即执行，无需等待。另一方面，mapreduce的任务由tasktracker节点负责执行的，所以slots可进一步理解为tasktrackers能够并发执行多个任务。slots分为mapper slots和reducer slots，分别对应最大可并行执行的mapper和reducer数。

用户可以通过修改mapred-site.xml配置文件的**mapred.tasktracker.map.tasks.maxmum**来设置slots的值，默认为2.

每个tasktracker管理的任务数

**mapred.tasktracker.map.tasks.maximum**

**mapred.tasktracker.reduce.tasks.maximum**

另**mapred.jobtracker.reduce.tasks.maximum**控制一个job最大task个数，集群下最大并行数

一 \*\*\*\*\*\*应用程序编写规范\*\*\*\*\*\*\*\*\*

1.设置Combiner

对于一大批MapReduce程序，如果可以设置一个Combiner,那么对于提高作业性能是十分有帮助的。

Combiner可减少Map Task中间输出的结果，从而减少各个Reduce Task的远程拷贝数据量，

最终表现为Map Task和Reduce Task执行时间缩短。

2. 选择合理的Writable类型

在MapReduce模型中，Map Task和Reduce Task的输入和输出类型均为Writable。Hadoop本身已经提供了很多Writable实现，包括IntWritable、FloatWritable。为应用程序处理的数据选择合适的Writable类型可大大提升性能。比如处理整数类型数据时，直接采用IntWritable比先以Text类型读入在转换为整数类型要高效 。如果输出整数的大部分可用一个或两个字节保存，那么直接采用VIntWritable或者VLongWritable，

它们采用了变长整型的编码方式，可以大大减少输出数据量。

二 \*\*\*\*\*作业级别参数调优\*\*\*\*\*\*

1.规划合理的任务数目

2.增加输入文件的副本数

3.启动推测执行机制

4.设置失败容忍度

5.适当打开JVM重用功能

6.设置任务超时时间

7.合理使用DistributedCache

8.跳过坏记录

9.提高作业优先级

10.合理控制Reduce Task的启动时机

1.规划合理的任务数目

在Hadoop中，每个Map Task处理一个Input Split。Input Split的划分方式是由用户自定义的InputFormat决定的，

默认情况下，有以下三个参数决定。

**mapred.min.split.size ：Input Split的最小值 默认值1**

**mapred.max.split.szie: Input Split的最大值**

**dfs.block.size：HDFS 中一个block大小 默认值64MB**

golsize:它是用户期望的Input Split数目=totalSize/numSplits ,其中totalSize为文件的总大小；numSplits为用户

设定的Map Task个数，默认情况下是1.

splitSize = max{minSize,min{goalSize,blockSize}} 如果想让InputSize尺寸大于block尺寸，直接增大配置参数

mpared.min.split.size即可。

2.**增加输入文件的副本数**

如果一个作业并行执行的任务数目非常多，那么这些任务共同的输入文件可能成为瓶颈。为防止多个任务并行读取

一个文件内容造成瓶颈，用户可根据需要增加输入文件的副本数目。

**3.启动推测执行机制**

推测执行是Hadoop对“拖后腿”的任务的一种优化机制，当一个作业的某些任务运行速度明显慢于同作业的其他任务时，Hadoop会在另一个节点上为“慢任务”启动一个备份任务，这样两个任务同时处理一份数据，而Hadoop最终会将优先完成的那个任务的结果作为最终结果，并将另一个任务杀掉。

**4.设置失败容忍度**

Hadoop运行设置任务级别和作业级别的失败容忍度。作业级别的失败容忍度是指Hadoop允许每个作业有一定比例的任务运行失败，这部分任务对应的输入数据将被忽略；

任务级别的失败容忍度是指Hadoop允许任务失败后再在另外节点上尝试运行，如果一个任务经过若干次尝试运行后仍然运行失败，那么Hadoop才会最终认为该任务运行失败。

用户应该根据应用程序的特点设置合理的失败容忍度，以尽快让作业运行完成和避免没必要的资源浪费。

**5.适当打开JVM重用功能**

**如果task属于不同的job，那么JVM重用机制无效，不同job的task需要不同的JVM来运行。**为每个task启动一个新的JVM将耗时1秒左右，对于运行时间较长（比如1分钟以上）的job影响不大，但如果都是时间很短的task，那么频繁启停JVM会有开销。

如果我们想使用JVM重用技术来提高性能，那么可以将**mapred.job.reuse.jvm.num.tasks**设置成大于1的数。

**这表示属于同一job的顺序执行的task可以共享一个JVM，也就是说第二轮的map可以重用前一轮的JVM， 而不是第一轮结束后关闭JVM，第二轮再启动新的JVM。**

为了实现任务隔离，Hadoop将每个任务放到一个单独的JVM中执行，而对于执行时间较短的任务，JVM启动和关闭的时间将占用很大比例时间，为此，用户可以启用JVM重用功能，这样一个JVM可连续启动多个同类型的任务。

**6.设置任务超时时间**

如果一个任务在一定的时间内未汇报进度，则TaskTracker会主动将其杀死，从而在另一个节点上重新启动执行。用户 可根据实际需要配置任务超时时间。

**7.合理使用DistributedCache**

一般情况下，得到外部文件有两种方法：一种是外部文件与应用程序jar包一起放到客户端，当提交作业时由客户端上传到HDFS的一个目录下，然后通过Distributed Cache分发到各个节点上；另一种方法是事先将外部文件直接放到HDFS上，

从效率上讲，第二种方法更高效。第二种方法不仅节省了客户端上传文件的时间，还隐含着告诉DistributedCache :"请将文件下载到各个节点的pubic级别共享目录中”，这样，后续所有的作业可重用已经下载好的文件，不必重复下载"。

**8.跳过坏记录**

Hadoop为用户提供了跳过坏记录的功能，当一条或几条坏数据记录导致任务运行失败时，Hadoop可自动识别并跳过这些坏记录。

**9.提高作业优先级**

所有Hadoop作业调度器进行任务调度时均会考虑作业优先级这一因素。作业的优先级越高，它能够获取的资源（slot数目)也越多。

**Hadoop提供了5种作业优先级，分别为VERY\_HIGH、 HIGH、 NORMAL、 LOW、 VERY\_LOW。**

注：在生产环境中，管理员已经按照作业重要程度对作业进行了分级，不同重要程度的作业允许配置的优先级不同，用户可以擅自进行调整。

**10.合理控制Reduce Task的启动时机**

如果Reduce Task启动过早，则可能由于Reduce Task长时间占用Reduce slot资源造成"slot Hoarding"现象，从而降低资源利用率； 反之，如果Reduce Task启动过晚，则会导致Reduce Task获取资源延迟，增加了作业的运行时间

**mapred.reduce.slowstart.completed.maps** （map完成多少百分比时，开始shuffle）

当job指定了combiner的时候，我们都知道map介绍后会在map端根据combiner定义的函数将map结果进行合并。运行combiner函数的时机有可能会是merge完成之前，或者之后，这个时机可以由一个参数控制，即min.num.spill.for.combine（default 3），当job中设定了combiner，并且spill数最少有3个的时候，那么combiner函数就会在merge产生结果文件之前运行。

**三 任务级别参数调优**

1**.Map Task调优**

**map运行阶段分为:Read、Map、Collect、Spill、Merge五个阶段。**

map 任务执行会产生中间数据,但这些中间结果并没有直接IO到磁盘上,而是先存储在缓存(buffer)中,并在缓存中进行一些预排序来 优化整个map的性能, 存储map中间数据的缓存默认大小为100M，**由io.sort.mb 参数指定**。这个大小可以根据需要调整。当map任务产生了非常大的中间数据时可以适当调大该参数,使缓存能容纳更多的map中间数据，而不至于大频率的IO磁盘,当系统性能的瓶颈在磁盘IO的速度上,可以适当的调大此参数来减少频繁的IO带来的性能障碍。

由于 map任务运行时中间结果首先存储在缓存中,默认当缓存的使用量达到80%(或0.8)的时候就开始写入磁盘,这个过程叫做spill(也叫溢出),进行spill的缓存大小可以通过**io.sort.spill.percent** 参数调整，这个参数可以影响spill的频率。进而可以影响IO的频率。

当map任务计算成功完成之后，如果map任务有输出，则会产生多个spill。 接下来map必须将些spill进行合并,这个过程叫做merge,

merge过程是并行处理spill的,每次并行多少个spill是由参数**io.sort.factor**指定的默认为10个。但是当spill的数量非常大的时候，merge一次并行运行的spill仍然为10个,这样仍然会频繁的IO处理,因此适当的调大每次并行处理的spill数有利于减少merge数因此可以影响map的性能。

当map输出中间结果的时候也可以配置压缩。

**2. Reduce Task调优**

**reduce 运行阶段分为shuflle(copy) merge sort reduce write五个阶段。**

shuffle 阶段为reduce 全面拷贝map任务成功结束之后产生的中间结果,如果上面map任务采用了压缩的方式,那么reduce将map任务中间结果拷贝过来后首先进行解压缩,这一切是在reduce的缓存中做的,当然也会占用一部分cpu。为了优化reduce的执行时间,reduce也不是等到所有的map数据都拷贝过来的时候才开始运行reduce任务，而是当job执行完第一个map任务时开始运行的。

reduce在shuffle阶段 实际上是从不同的并且已经完成的map上去下载属于自己的数据,由于map任务数很多,所有这个copy过程是并行的,既同时有许多个reduce取拷贝map，这个并行的线程是通过 **mapred.reduce.parallel.copies** 参数指定，默认为5个,也就是说无论map的任务数是多少个，默认情况下一次只能有5个reduce的线程去拷贝map任务的执行结果。所以当map任务数很多的情况下可以适当的调整该参数，这样可以让reduce快速的获得运行数据来完成任务。

reduce线程在下载map数据的时候也可能因为各种各样的原因(网络原因、系统原因等），存储该map数据所在的datannode 发生了故障， 这种情况下reduce任务将得不到该datanode上的数据了,同时该 download thread 会尝试从别的datanode下载,可以通过**mapred.reduce.copy.backoff (默认为30秒)来调整下载线程的下载时间，如果网络不好的集群可以通过增加该参数的值来增加下载时间,**以免因为下载时间过长reduce将该线程判断为下载失败。

reduce 下载线程在map结果下载到本地时,由于是多线程并行下载，所以也需要对下载回来的数据进行merge,所以 map阶段设置的**io.sort.factor** 也同样会影响这个reduce的。

同map一样 该merge sort缓冲区大小也不是等到完全被占满的时候才写入磁盘而是默认当完成**0.66的时**候就开始写磁盘操作,该参数是通过 **mapred.job.shuffle.merge.percent** 指定的。另外一个影响参数是****mapred.inmem.merge.threshold(默认值1000)，拉过来的map端文件个数。****

当reduce 开始进行计算的时候通过 **mapred.job.reduce.input.buffer.percent** 来指定需要多少的内存百分比来作为reduce读，默认70%。

已经sort好的数据的buffer百分比,该值默认为0。Hadoop假设用户的reduce()函数需要所有的JVM内存，因此执行reduce()函数前要释放所有内存

。如果设置了该值，可将部分文件保存在内存中(不必写到磁盘上)。

三 管理员角度调优

管理员负责为用户作业提供一个高效的运行环境。管理员需要从全局出发，通过调整一些关键参数提高系统的吞吐率和性能。

总体上来看， 管理员需从

硬件选择、

操作系统参数调优、

JVM参数调优和Hadoop参数调优

等四个角度入手，为Hadoop用户提供一个高效的作业运行环境。

硬件选择

Hadoop自身架构的基本特点决定了其硬件配置的选项。Hadoop采用了Master/Slave架构，其中，master维护了全局元数据信息，重要性远远大于slave。在较低Hadoop版本中，master存在单点故障问题，因此，master的配置应远远好于各个slave。

操作系统参数调优

1.增大同时打开的文件描述符和网络连接上限

使用ulimit命令将允许同时打开的文件描述符数目上限增大至一个合适的值。同时调整内核参数net.core.somaxconn网络连接数目至一个足够大的值。

补充：net.core.somaxconn的作用

net.core.somaxconn是Linux中的一个kernel参数，表示socket监听（listen）的backlog上限。

什么是backlog呢？backlog就是socket的监听队列，当一个请求（request）尚未被处理或建立时，它会进入backlog。

而socket server可以一次性处理backlog中的所有请求，处理后的请求不再位于监听队列中。当server处理请求较慢，

以至于监听队列被填满后，新来的请求会被拒绝。在Hadoop 1.0中，参数 ipc.server.listen.queue.size控制了服务端socket的监听队列长度，

即backlog长度，默认值是128。而Linux的参数 net.core.somaxconn默认值同样为128。当服务端繁忙时，如NameNode或JobTracker，

128是远远不够的。这样就需要增大backlog，例如我们的3000台集群就将ipc.server.listen.queue.size设成了32768，

为了使得整个参数达到预期效果，同样需要将kernel参数 net.core.somaxconn设成一个大于等于32768的值。

2.关闭swap分区

避免使用swap分区，提供程序的执行效率。

除此之外，设置合理的预读取缓冲区的大小、文件系统选择与配置及I/O调度器选择等

JVM参数调优

由于Hadoop中的每个服务和任务均会运行在一个单独的JVM中，因此，JVM的一些重要参数也会影响Hadoop性能。

管理员可通过调整JVM FLAGS和JVM垃圾回收机制提高Hadoop性能。

Hadoop参数调优

1.合理规划资源

设置合理的槽位数目

在Hadoop中，计算资源是用槽位表示的。slot分为两种：Map Slot和Reduce Slot。

每种slot代表一定量的资源，且同种slot是同质的，也就是说，同种slot代表的资源量是相同的。管理员需要根据实际需要为TaskTracker配置一定数目的Map Slot和Reduce Slot数目，从而限制每个TaskTracker上并发执行的Map Task和Reduce Task的数目。

编写健康监测脚本

Hadoop允许管理员为每个TaskTracker配置一个节点健康状况监测脚本。TaskTracker中包含一个专门的线程周期性执行该脚本，并将脚本执行结果通过心跳机制汇报给JobTracker。一旦JobTracker发现某个TaskTracker的当前状况为“不健康”，则会将其加入黑名单，

从此不再为它分配任务。

2. 调整心跳配置

调整心跳的间隔 因根据自己集群的规模适度的调整心跳间隔

启用带外心跳 为了减少任务分配延迟，Hadoop引入了带外心跳 。带外心跳不同于常规心跳，它是任务运行结束或者任务运行失败时触发的，能够在出现空闲资源时第一时间通知JobTracker,以便它能够迅速为空闲资源分配新的任务。

除此之外，还包括磁盘块配置、设置合理的RPC Handler和HTTP线程数目、慎用黑名单机制、启用批量任务调度、选择合适的压缩算法、启用预读取机制等。

注：当一个集群的规模较小时，如果一定数量的节点被频繁的加入系统黑名单中，则会大大降低集群的吞吐率和计算能力。

四 小结

Hadoop 性能调优是一项工程浩大的工作，它不仅涉及Hadoop本身的性能调优，还涉及更底层的硬件、操作系统和Java虚拟机等系统的调优。

总体来说，提高作业运行效率需要Hadoop管理员和作业拥有者共同的努力，其中，管理员负责为用户提供一个高效的作业运行环境，而用户则根据自己作业的特点让它尽可能快地运行完成。