

搜百度内部wiki

视频说明：<http://learn.baidu.com/courseInfo.html?courseId=3227&timestamp=v20131022>

hadoop规范及运维准入

背景

一、fs shell相关命令用法

ls part-\*

mv part-\*

对ls的结果的每一行du

setrep

distcp

hadoop-vfs及其他依赖环境

二、任务提交及参数

输入文件数量

map、reduce数量和数据量大小

运行时间和并发

map完成多少比例开始shuffle

cpu相关

内存相关

失败重试次数

允许任务失败的比例，

三、权限和资源

分组（group），队列（queue）及计算资源抢占

计算和存储账号

优先级

槽位和资源消耗

新任务上线准入

新任务上线接入

非线上数据

四、页面分析（以peta+abaci为例）

存储页面

计算页面

调度页面

五、任务异常及日志

任务失败基本处理

子进程返回信号

常见错误日志/信息

六、业务逻辑

rmr的异常

fork进程或起多线程任务

节点上的本地日志上传

多轮mapreduce优化

libhdfs中的hdfsFlush方法

七、不常见的坑

本地基础环境和salve差异的坑

某节点机器网卡出异常

八、资源控制临时手段

九、声明

## 背景

- 离线业务中hadoop任务较多，但是没有很好的准入规范
- 存储、计算资源的管理没有很好的规范
- 多次发生因为hadoop任务不合理而影响集群
- 任务失败原因不知
- 介于以上等原因，特制订此规范（初稿）作为参考。（主要针对peta和abaci版本集群）

## 一、fs shell相关命令用法

### ls part-\*

- 不合理命令举例：**hadoop fs -ls /user/rd/myinput/part-\***
- 分析：集群master会根据 **part-\*** 这个模式对所有 **/user/rd/myinput** 这个路径下的文件匹配一遍，因此，这个路径下文件很多的话，耗时非常长
- 建议：
  - 修改为 **ls /user/rd/myinput/ | grep "part-"**
  - 这样的话，匹配操作放在了本地上，而对master来说，一股脑儿把目录下的信息全给你了，省的它筛选一遍。实测发现，一个10几万文件的目录，该操作从十几分钟降低到几十秒

### mv part-\*

- 不合理命令举例：**hadoop fs -mv /user/rd/tmpoutput/part-\*/user/rd/output/**

- 分析：和上一个命令类似，也是需要每个文件匹配一遍
- 建议：
  - 如果目录下的所有文件都是 **part-\***，直接修改为：**mv /user/rd/tmpoutput/user/rd/output**（当然，事先 **rmr /user/rd/output**）
  - 如果目录下大部分都是 **part-\***，有少部分其他形式的文件，建议先mv出去然后效仿上一条
  - 如果目录下大部分的都不是 **part-\***，那么该操作一般不会耗费很长时间。

对ls的结果的每一行du

- 分析：当目录下文件梳理非常庞大的时候，假设20万，就会有20万次请求
- 建议：
  - 如果目录底下全是文件，那么 **ls** 的结果就包含大小，可以重定向到文件或者变量，后续做本地操作
  - 如果目录底下大部分是目录，那么可以对整个目录使用 **du** 命令，比如 **hadoop fs -du /user/rd/tmpoutput** 就会显示每一项的大小
  - 当然，最新版本的上了一个限制**hadoop fs shell**命令占用过多资源的功能，比如采用 **du、dus、ls、lsr** 等命令操作一个大目录的时候，可能会报超过inode限制的错误

setrep

- 不合理用法举例：得到一个目录下的所有文件列表，对一个目录下的所有文件**setrep**，并且不加 **-w** 参数
- 分析：当目录下文件梳理非常庞大的时候，请求非常多。并且，不加**-w**参数，命令之间没有等待，类似全部压后台跑，一瞬间集群有大量复制
- 建议：
  - 如果是对整个目录设置副本，可以用**setrep -R** 递归对整个目录下所有文件设置副本，只有一次请求。当然，**大目录禁止该操作。**
  - 建议一般情况下，加上**-w** 参数，表示阻塞执行，即等待副本复制完毕再返回。这样可以避免如下异常：
    - 很多时候我们会对某个频繁访问的文件设置 **15** 副本，然后马上启动一个**hadoop**任务，如果由于当时集群压力非常大，还没有复制到**15**个副本，就起了任务，会报读不到块的错误

distcp

- 不合理用法举例：不加限速起任务；在低版本 **hadoop-v2** 集群起 **distcp** 任务拷贝高版本 **peta** 集群的数据；**-update** 和 **-overwrite** 使用误区
- 分析：不加限速即影响单节点的网卡，又可能造成机房带宽资源被大量占用。**distcp** 不支持从 **hadoop-v2** 集群起任务拷贝 **peta** 集群的数据。**-update** 和 **-overwrite** 参数加上后，**会关闭预测执行，遇到慢节点后长尾可能性很大。**
- 建议：
  - 设置限速用法如下 **hadoop distcp -D distcp.map.speed.kb=81920 .....** **注意单位是KB，而我们一般用b来描述带宽的，注意换算。**参考：**http://wiki.babel.baidu.com/twiki/bin/view/Main/Distcp%E8%8C%83%E4%BE%8B**
  - 在**peta**集群起任务，设置合理的map槽位和所用计算分组
  - **distcp** 如果不加 **update** 和 **overwrite** 参数，**会强制创建目录（如果目录存在，会深入一层）。**建议删除 **distcp** 的目录路径之后起 **distcp**。如果中途跑失败的话再用 **update** 起（**此时注意目的路径要加上那个目录，因为update不会创建目录**）。**overwrite** 一般不建议。

hadoop-vfs及其他依赖环境

- 建议：
  - **vfs**的访问方式已经不推荐，原则上不再使用
  - 对于有特定的环境要求，如依赖特定的**python**环境，需要程序自带
  - 如果对操作系统如内核有特殊要求，不在本规范范围内

二、任务提交及参数

输入文件数量

- 不合理用法举例：任务的input目录中文件数量非常多，比如几十万以上
- 建议：
  - 超大的目录在起任务的时候会耗费大量时间。建议再生出这份数据的时候就用多个目录来存储。如果都是小文件，则在生成时，减少文件数量。
  - 如果总输入量确实非常庞大，跑任务时用多轮 **mapreduce**，并最后加一轮 **mapreduce** 来归并各步骤输出。
  - 更推荐的做法参考下面的 第六节 业务逻辑 -> 多轮**mapreduce**优化

map、reduce数量和数据量大小

- 建议：
  - 每个 **map** 的大小 **20M-2G** 比较合适，一般通过指定**spill**大小来限制。仅供参考，以业务实际为准
  - 每个 **reduce** 的大小 **1G-20G** 比较合适，一般通过**reduce**数目和合理的分环来限制。仅供参考，以业务实际为准
  - 禁止出现某个**part**非常大的情况，比如根据 **url**的长相 来分环，当某个域的url爆发时将造成灾难，打满磁盘，影响其他任务运行。
    - 一般通过合理的抽取标尺来解决。可以通过任务运行页面上的：“**Partition Data Info**”来查看分环是否不均，参考下面的 第四节 页面分析 -> 计算页面
    - 当然，业务逻辑如果支持**hash**分环（随机分环）就最好了。**hash分环时，建议reduce个数为质数。**

运行时间和并发

- 不合理用法举例：设置过大的并发；单个子任务的运行时间非常长
- 分析：过大的并发影响其他任务的运行；单个子任务的运行时间过长，一来增加失败重试的代价，长尾可能非常严重，二来影响高优任务的抢占
- 建议：
  - 并发应该按照预算所规定的设置，**严禁私自增加并发或在任务提交后擅自修改。**
  - 单个**map**的运行时间最好控制在半个小时内（以具体集群和任务特点为准），长达几个小时的任务需要优化
  - **reduce**任务长尾往往因为分环不均导致，需要优化分环逻辑，参考上一小节。

map完成多少比例开始shuffle

- 不合理用法举例：设置过小或者过大。涉及参数 **mapred.reduce.slowstart.completed.maps**
- 分析：这个参数是和“运行完的**map**数量百分比”做判断的，完成的百分比超过设定值时，开始**reduce**的**shuffle**。所以当**map**比**reduce**的执行时间多很多时，**map**运行完很早就开始了**reduce**，导致**reduce**的slot一直处于被占用状态。

- 建议：
  - 当map比reduce的执行时间多很多时，可以调整这个值（ **0.80**左右甚至以上）
  - 当然，如果任务的map数量非常大，不建议调很高的值，因为在刚达到这个设定值时，大量的map同时shuffle，给集群造成很大压力

cpu相关

- 不合理用法举例：任务使用的cpu较多，但是声明的cpu消耗量很大，或者反之很小。涉及参数 **abaci.job.map.cpu.percent**， **abaci.job.reduce.cpu.percent** 。参数默认值**30**（表示1个cpu的30%，即0.3个cpu）
- 分析：
  - 如果参数过小，那么实际使用的cpu会多很多，会占用更多资源，而调度器在任务刚启动时认为其cpu使用只有那么多，因此在高峰期会调度很多其他任务到这个节点，很容易造成慢节点。
  - 如果参数过大，那么可能导致在高峰期调度不上这个任务，因为有的碎片资源时，是有限满足那些需求申请比碎片资源小的任务的。
- 建议：根据子任务的实际消耗量加一点buffer来设置该值。（比如saver的merge3步骤，参数设置了60，表示占用0.6个cpu。）

内存相关

- 不合理用法举例：任务使用的内存较多，但是声明的内存消耗量很大，或者反之很小。涉及参数 **abaci.job.map.memory.mb**， **abaci.job.reduce.memory.mb** 。
  - (不设置的情况下，框架会在800的基础上多分配400，也就是调度时占用1200M。大部分任务用不掉，比如map使用1000M就够的，那么设置**abaci.job.map.memory.mb=600**)
  - 分析：
    - 如果参数过小，那么实际使用的内存会多很多，会占用更多资源，造成慢节点甚至oom。在matrix版本的集群中经常会被杀
    - 如果参数过大，那么可能导致在高峰期调度不上这个任务，因为有的碎片资源时，是有限满足那些需求申请比碎片资源小的任务的。
  - 建议：设置为稍大于单个任务的所用内存大小
- 
- 不合理用法举例：**stream.memory.limit**设置的过小或者过大
  - 分析：老版本的集群默认是**800M**。该参数是限制每个任务所能使用的内存大小，如果设置过大，比如20G，那么如果datanode的内存不够的话根本无法跑起任务。如果设置过小，任务运行时会被kill掉
  - 建议：
    - 设置一个尽可能小的又能满足内存消耗的值
    - 内存消耗特别大的任务禁止上线。
    - 如果实在无法通过优化逻辑解决，需要占用2G甚至更多的内存，必须设置该参数：**-D mapred.map.capacity.per.tasktracker=2** .表示**每个节点最多并行跑2个该job的map任务**（请根据内存情况适当增减该参数，默认是8）
    - 同样的，这个参数设置并行跑的reduce任务：**mapred.reduce.capacity.per.tasktracker**
  - 特别注意：**stream.memory.limit** 参数和前面的 **abaci.job.map/reduce.memory.mb** 参数的区别是：
    - 前者是控制内存上限，实际使用量超了就被杀。
    - 后者是控制子任务提交时声明的内存消耗量，**仅作用于任务提交阶段，他不管你实际使用了多少**。因此，你声明800M，但实际使用2000M也是可以的，但是风险如上所述。

失败重试次数

- 不合理用法举例：对一个非常重型的任务设置很大的值
- 建议：
  - 任务多次尝试导致资源浪费较多
  - 有时候太少的重试次数又导致任务轻易失败
  - 一般任务默认重试**3-4**次。特殊任务需要申请

允许任务失败的比例，

- 不合理用法举例：默认有**任何1个子任务达到重试次数了还没有跑成功就会fail整个job**，有些任务允许个别子任务失败但是没有设置，导致重跑浪费资源。涉及参数 **mapred.max.map.failures.percent**， **mapred.max.reduce.failures.percent**
- 建议：有时候，业务逻辑允许任务部分失败，只需要成功部分的结果，这个时候不用浪费资源再次运行任务。合理设置以上两个参数

三、权限和资源

分组（group），队列（queue）及计算资源抢占

- hadoop的资源调度方式有**分组**和**队列**两种方式
- 不同分组（队列）之间的资源抢占需要各应用方事先协商好：包括自己分组（队列）的资源多少，可抢占资源的比例，可借出的资源比例
- 以分组方式调度的集群，如果任务需要跑在多个分组。设置参数如：**mapred.job.groups=g1,g2,g3**
- 以队列方式调度的集群，任务只能指定一个分组，资源会按照配置自动抢占或借出。设置参数如：**mapred.job.queue.name=q1**

计算和存储账号

- hadoop的账号分为存储和计算，客户端在提交任务时通过指定 **ugi** 来说明使用什么账号；如果不跑计算，则只设置存储账号
- 计算账号在设置时需要**按照预算**设置能跑的队列（分组）
- 线下任务 **禁止** 跑在线上的队列（分组）。
- 禁止** 私自使用线上账号**ugi**来跑线下任务。
- 线上机器上面的 **hadoop-site.xml** 中包含**ugi**的设置，权限设置 **600**；
- hadoop-client的bin目录包含hadoop的可执行程序，权限设置 **700**；
- hdfs上面，形如 **/user/\*** 的线上目录的权限设置为 **755**，**root:root**。防止用户私自创建目录。模块上线需要联系**op**创建目录。

优先级

- 严格**按照预算**设置优先级，在资源不够用时，需要按照服务分级优先保证高优服务。
- 修改任务优先级需要走预算申请流程或者和其他任务协调

槽位和资源消耗

- 新的调度系统对槽位的说法和原来已经不同，不再严格规定每台机器8个map槽位和8个reduce槽位（在数量上，不再定死8个槽位；在性质上，只要有资源，既可以调度**map**，也可以调度**reduce**）
- 下面提供一种估算方法：
  - 如果以**cpu**来描述，那么 **0.3** 个核为一个槽位（目前新集群一般配置0.4核）
    - 举例**szjih-dbuild**集群的机器：一台机器提供8个核，**shuffle**占用掉2个核。因此有  $(8-2)/0.3=20$  个槽位。
    - 现在的好机器配置的cpu核更多，比如**nmg01-global**集群配置了12个核。
  - 如果以**内存**来描述，那么默认 **1800M**为提交一个任务的默认参数（任务启动时，集群给设置的默认内存占用），而按照 **nmg01-global** 和 **szjih-dbuild** 集群的实际情况，可以认为 **1000M** 为一个槽位，下面详述：
    - 举例**szjih-dbuild**集群的机器：一台机器大概提供28G内存，启动**shuffle**占用3200M左右。那么按1800来算的话，槽位数  $(28*1024-3200)/1800 \approx 14$  个
    - 但实际上，线上很多任务在运行**10%**之后，调度器会重新估算其消耗量，很多任务消耗量不大的话，会被调节到1400、1200、800甚至400M都有可能。因此针对内存的调度是比较精细的，槽位是实时变化的。
    - 原则上需要以这个集群队列所有任务所耗内存的平均数来估算槽位，比如大致抽样估算某个集群的分组所有任务的平均内存消耗为1600M，那么槽位数  $28G/1600M \approx 24$  个
    - 上面描述的这个过程以及每个任务的内存实时消耗量可以参考“第四节、页面分析->计算页面->metamaster页面”中的第三幅图，图中显示了当时这台机器上面每个任务的内存消耗情况。
    - 此处给出 **nmg01-global** 和 **szjih-dbuild** 集群的 **linkbase** 分组的**平均内存消耗值：1000M**。因此在评估这两个集群的 **linkbase** 分组资源使用时，按照**1000M**内存为一个槽位计算。
  - 基于以上两点，一个**map**或者**reduce**任务在描述槽位时，**需要明确说明其占用的cpu是多少个核，占用的内存是多少M**。
    - 估算槽位时，先根据提供的cpu消耗，用该值除以0.3得出单个任务所需槽位数 **X**，作为基于cpu的单任务所需槽位估算值(比如，如果你的任务非常耗cpu，需要占用0.6个cpu (此时起任务时注意设置参数，参见第二节提到的“cpu相关”)，那么单个子任务占两个槽位)。
    - 然后根据提供的内存消耗，用该值除以当前集群该分组任务的平均内存消耗值，得出单个任务所需槽位数 **Y**，作为基于内存的估算值(比如，如果你的任务非常耗内存，需要占用3000M内存 (此时起任务时注意设置参数，参见第二节提到的“内存相关”)，假设该分组任务平均占用1000M内存，那么单个子任务占用3个槽位)。。
    - 取 **X** 和 **Y** 之中的较大值，作为单个任务的槽位数。该数值乘以并发得出总槽位数。**map任务和reduce任务的估算方法同。这个数值用于告诉集群业务接口人，你的任务运行期间，会占用整个集群的多少资源。**
  - 任务的资源消耗情况不能单纯用槽位来说明。严格来讲是槽位对时间的积分。大致的说法是以多少**map**槽位运行多少时间，以多少**reduce**槽位运行多少时间。**槽位和运行时间两个数值结合才能让集群业务接口人知道，你的任务消耗多少资源，集群能否容下。**
  - 精确的槽位计算**可以根据这个视频27分钟的说明：<http://learn.baidu.com/play.html?courseId=3227&elementId=0c6766c2-cf80-4888-a87a-f2d61a044416&tamp=v20131022>
  - 另外，需要说明的是，目前资源调度除了**cpu**、内存，还有硬盘。硬盘方面：**禁止单节点占用超过100G**。
    - 这类不合理任务一般是因为把**hdfs**文件**get**到本地运算的，这完全可以用集群运算逻辑来处理。或者可能是大的词典，大的**conf**等，完全可以用相同标尺切分后进行归并运算。

## 新任务上线准入

- 集群上各应用方的接口人应该清楚自己分组上所跑任务的运行时间、资源消耗、集群高峰和空闲期等情况。
- 新任务上线需要有**计算资源的预算**
  - 预算时需要说明**map**槽位数和运行时间，**reduce**槽位和运行时间，运行的周期（每天，每周等）。**一共5个数值，缺一不可**
  - 槽位的计算严格按照上一小节的方法**，不能单纯用 **mapred.job.map.capacity** 和 **mapred.job.reduce.capacity** 这两个并发来替代槽位数。
  - 比如，默认配置中，每个任务有30个cpu单位，如果计算非常耗cpu，需要指定该参数为60，那么槽位消耗相当于多了一倍。
  - 再如，默认配置中，每个任务占用800M内存（根据集群的配置可能不同），如果非常吃内存，需要设置2048M内存，那么槽位的消耗也要按比例提高。
- 新任务上线需要有**存储资源的预算**
  - 在任务上线时，任务负责人提交 **spaceQuota** 和 **inodeQuota** 的预算，以及 存储数据的**hdfs**目录，**quota**报警接收人（建议是报警接收组）。
  - 由集群业务接口人评估当前集群容量之后，添加该目录的 **spaceQuota** 和 **inodeQuota** 及 报警接收人的申请 给**dpf**
- 新任务申请人要明确当前**client**机器上面的公用**hadoop**配置，模块中所有特殊的**hadoop**配置都需要放在模块自己的目录下，并经过集群业务接口人的审核，包括**动态生成的配置和写死在代码中的参数都要说明**。
- rd**需要分两步上线：1. 说明模块背景、功能、跑任务分组、数据存放的**hdfs**路径、**client**机器和路径、存储资源使用情况、计算资源使用情况（参考上一节《槽位和资源消耗》）。2. 将上述信息发邮件给op全组和op负责人进行申请，op负责人**通过之后**走**icafe**上线
- 由集群业务接口人根据以上信息并结合上述各节提到的**禁忌点**，评估新任务是否准入，以及起任务的时间和周期。
- 特别提醒：没有预算不得上线，除非能协调资源扩容或者让等量资源的老任务下线

## 新任务上线接入

- 任务命名
  - 除了**distcp**之外，其余任务名称都建议按照 “模块\_步骤名” 来命名
- hdfs**路径
  - 所有数据都放在 “**/user/**产品线/模块名” 下
  - 目录创建由集群业务接口人(**op**)负责创建
  - 同时需要设置该**hdfs**路径的**quota**

## 非线上数据

- 存储
  - 统一用一个存储用户如 **rd**，所有用户在 **/user/rd** 下创建自己的目录。
  - 集群业务接口人给 **/user/rd** 设置一个总**quota**，报警接收人为整个rd用户大组。rd目录如需细分，由各用户协商后自行向**dpf**提交申请
  - 超**quota**之后为便于追查几个大头，给出wiki帮助：<http://wiki.babel.baidu.com/twiki/bin/view/Ps/OP/HadoopQuotaExceededDeal>
- 计算
  - 统一用一个计算用户如**rd**，给一个统一的物理或者逻辑分组供计算。与线上任务隔离。
  - 内部任务抢占问题需要各用户协商 优先级、并发 和 起任务时间。

## 四、页面分析（以peta+abaci为例）

### 存储页面

- 【 namespace页面 】**
  - 线上集群一般都是配的8070端口为http地址的端口。机器名加端口访问namespace的页面，如 <http://szjih-dbuild-namenode.szjih01.baidu.com:8070/>（账号密码随便填）（可以在<http://dpfhelp.dmpop.baidu.com> 页面上点对应集群的HDFS页面快速找到）

下面给出每部分说明：

- Cluster Summary

733429683 files and directories, 581850591 blocks = 1315280274 total , data version: 203831.  
Heap Memory used 68.85 GB is 64% of Committed Heap Memory 107.06 GB. Max Heap Memory is 107.06 GB.  
Non Heap Memory used 32.31 MB is 66% of Committed Non Heap Memory 48.39 MB. Max Non Heap Memory is 130 MB
- 第一个数字是文件和目录的inode总和（我们知道目录也是占一个inode的）。
- 第二个数字是block的数量（hdfs实际上是以block来存储一个文件的，因此每个block对hdfs来说也是一个inode。）。
- 我们会看到前两个数字之后正好是第三个数字。这第三个数字才是真正的整个集群的inode数量。也直接和namespace和fms的内存使用量正相关，大致是线性关系。
- 从红框框出的第四个数字可以看到目前namespace的内存消耗百分比。
- 过多的小文件对master的压力很大，业务方在上传大批此类文件时要注意。比如：建库沙盒集群之前3亿多文件，两个64G内存的fms时刻处于oom边缘。
- 如果集群异常或者升级时，该处会提示 safemode，此时集群不可用。

- WARNING : There are about 268 missing blocks. Please check the log or run fsck.
- 如果集群丢块，会出现如上一行字。可以看到丢了268个块（就是上面提到的block，可不是说丢了268个文件）。
- 点击这行字，就跳到一个页面，列出一大堆数字。每个数字称为 objectid 。一个objectid对应一个文件，一个文件可能对应多个block。
- 可以通过 “hadoop fsck 路径 -files -blocks -locations ” 命令发现这些对应关系，比如下图：

```
/user/spider/dlb-linkbase/data/saver-inc/readyfor_filter/201406041100/201406031048/patch/part-0
0. blk_14355223980689960_176248096 len=1073741824 repl=3 [10.73.92.26:7001, 10.73.25.27:7001, 1
1. blk_14355223980713315_176269384 len=131127908 repl=3 [10.73.118.17:7001, 10.73.118.24:7001,
Object ID : /51-879621079 [ fms = c(/10.73.60.25:55310) ]

Status: HEALTHY
Total size:      1204869732 B
Total dirs:      0
Total files:      1
Total blocks (validated):      2 (avg. block size 602434866 B)
Minimally replicated blocks:  2 (100.0 %)
Over-replicated blocks:       2 (100.0 %)
Under-replicated blocks:      0 (0.0 %)
Mis-replicated blocks:        0 (0.0 %)
Default replication factor:    3
Average block replication:     3.0
Number of blocks:              2

Number of data-nodes:          2115
Number of racks:               111

The filesystem under path '/user/spider/dlb-linkbase/data/saver-inc/readyfor_filter/20140604110
```

- 如上图，这个文件超过1个G，因此被分成了两个block，0号blk是1个G，1号blk就不足1个G了，从这里可以看出文件和block的对应关系。每个block大小是由参数 dfs.block.size 控制。
- repl=3，说明三个副本，后面括号内的三个ip就是三个副本所在的三个节点。
- 要找丢失的文件可以参考：<http://wiki.babel.baidu.com/t> [bin/view/Ps/OP/findHadoopMissingBlock](http://wiki.babel.baidu.com/t/bin/view/Ps/OP/findHadoopMissingBlock)
- 下面这个图就是namespace和FMS的主备机了。真正在工作的是active那一列。第一项是namespace。
- NNMonitor字样的出现是说明开启了主备自动切换的检测，master不工作时，会自动切换为备机接管。注意，现在新的版本已经不再采用这个NNmonitor了。没显示这个字样也会自动切换

ZK Summary			
Node	Active	Status	Address
NS, NNMonitor	10.224.179.12:54310	10	10.224.180.12:54310
FMS [b], NNMonitor	10.224.179.33:55310	10	10.224.179.32:55310
FMS [a], NNMonitor	10.224.179.13:55310	10	10.224.180.53:55310
FMS [e], NNMonitor	10.224.13.30:55310	10	10.224.181.33:55310
FMS [d], NNMonitor	10.224.180.33:55310	10	10.224.13.40:55310
FMS [c], NNMonitor	10.224.179.53:55310	10	10.224.180.32:55310

- 下面这个图就是那5个fms的页面的链接了，随便点击一个链接进去，会看到一个新的页面，下一小节介绍细节



FMS Summary

fms address	corrupt file	metasave
<a href="#">szjjh-dbuild-fms01.szjjh01.baidu.com:8070</a>	<a href="#">result</a>	<a href="#">result</a>
<a href="#">szjjh-dbuild-fms00.szjjh01.baidu.com:8070</a>	<a href="#">result</a>	<a href="#">result</a>
<a href="#">szjjh-dbuild-fms08.szjjh01.baidu.com:8070</a>	<a href="#">result</a>	<a href="#">result</a>
<a href="#">szjjh-dbuild-fms05.szjjh01.baidu.com:8070</a>	<a href="#">result</a>	<a href="#">result</a>
<a href="#">szjjh-dbuild-fms02.szjjh01.baidu.com:8070</a>	<a href="#">result</a>	<a href="#">result</a>

•

◦

◦

【fms页面】

◦

下面这个图前面部分类似上面讲到的namespace的页面，只不过这是单个fms的信息。由这个fms负责部分目录结构和meta信息。同样的，如果有丢块，会出现上面ns同样的missing block字样，只不过丢块数一般比ns页面上的少，因为只显示这个fms管辖范围的丢块数量

◦

DFS Used%：这个百分比要多关注，超过90%，集群处于高危状态。这个数值和dpfhelp页面上那个根目录quota使用量经常是对不上的，前者往往要更小。这是因为对冷数据做了压缩（hdfs raid），这个压缩对master透明。实际上以这个fms页面上的百分比为准

◦

Live Nodes和Dead Nodes：顾名思义，就是当前活着的datanode数量和死掉的datanode数量。死节点数过多往往意味着集群异常并伴随丢块

◦

Decommission相关：主要是节点计划内的下线，主要由管理员操作

◦

Number of Under-Replicated Blocks：待复制块的数量。这个数据也比较重要，多次刷新页面会看到数字在变。这个数字表示还没有按照副本数要求复制完毕的block数量。因为不时的会有任务往集群上写数据，因此待复制块数量经常变动。集群空闲时应该是0或接近0。大量掉节点时该数字很大

129031800 files and directories, 119008660 blocks = 248040460 total , data version: 212773.  
Heap Memory used 97.46 GB is 90% of Committed Heap Memory 107.6 GB. Max Heap Memory is 107.6 GB.  
Non Heap Memory used 32.33 MB is 67% of Committed Non Heap Memory 48 MB. Max Non Heap Memory is 130 MB.

Configured Capacity	:	54.14 PB
DFS Used	:	45.72 PB
Non DFS Used	:	1022.38 TB
DFS Remaining	:	7.42 PB
DFS Used%	:	84.44 %
DFS Remaining%	:	13.71 %
<a href="#">Live Nodes</a>	:	5491
<a href="#">Dead Nodes</a>	:	486
<a href="#">Decommission Nodes</a>	:	3
<a href="#">Decommissioning Nodes</a>	:	15
Number of Under-Replicated Blocks	:	234456

•

◦

◦

下面这个图就是每个活节点的一些基本信息

◦

last Contact：是最新多久没汇报给master了，一般集群超时设置了10多分钟没汇报了才被认为是死节点

◦

Configured Capacity：是指单机上可以供hdfs使用的磁盘大小

◦

Non DFS Used：是指其中没被hdfs利用的磁盘大小，如果这个值很大，说明可能是mapred用户占用的临时计算资源占用过多（这部分数据在/home/disk\*/mapred下）或者是单机异常（如挂载的盘数量异常，hadoop配置的dfs.data.dir异常等）

•

•

Live Datanodes : 5491

Node	Last Contact	Admin State	Configured Capacity (TB)	Used (TB)	Non DFS Used (TB)	Remaining (TB)	Used (%)	U
szjjh-b13711	5	In Service	9.89	8.4	0.06	1.44	84.85	<div></div>
szjjh-b13867	5	In Service	9.07	7.71	0.04	1.32	85.01	<div></div>
szjjh-b13894	11	In Service	4.94	4.22	0.04	0.68	85.44	<div></div>

计算页面

•

【metamaster页面】

◦

线上集群一般都是配的8030端口为http地址的端口。机器名加端口访问metamaster的页面，如 <http://szjjh-dbuild-jobtracker.szjjh01.baidu.com:8030/>（账号密码随便填）（可以在 <http://dpfhelp.dmop.baidu.com> 页面点对应集群的abaci页面快速找到）

◦

下面给出各部分说明

◦

Job Queues

Queue Name	Scheduling Information
<a href="#">dew fresh</a>	CPU used 0.0%, MEMORY used 0.0%, DISK used 0.0%
<a href="#">default</a>	CPU used 6.3%, MEMORY used 5.7%, DISK used 0.1%
<a href="#">dew default</a>	CPU used 29.9%, MEMORY used 30.3%, DISK used 0.4%
<a href="#">mdoc</a>	CPU used 2.6%, MEMORY used 3.0%, DISK used 0.0%
<a href="#">linkbase-rd</a>	CPU used 95.4%, MEMORY used 98.1%, DISK used 1.1%
<a href="#">mis</a>	CPU used 3.4%, MEMORY used 6.0%, DISK used 0.0%

•

◦

◦

正如上面第三节讲到的，上图就是每个队列和其当前资源使用情况。资源分为cpu，内存和硬盘。正如第三节讲到的，ark版本的hadoop对于资源的调度，

http://wiki.babel.baidu.com/twiki/bin/view/Ps/OP/HadoopGuideAndAdmittance#一、fs shell相关命令用法\*

6/12

- 是精细化到cpu，内存和硬盘的。
- 有时候我们会看到百分比可能超过100%，这是因为集群开启了资源借用，在别的分组空闲时，可以抢占其资源。
  - 
  - 我们可以随便点其中一个队列的名称，进入一个新的页面，如下图

Jobs of Queue linkbase-rd

JobID	Priority	User	Queue	Job Name	Map % Complete	Map Total	Map Completed	Map Running	Reduce Complet
<a href="#">job_20140130203054_4413081</a>	VERY_HIGH	rd	linkbase-rd	ic_data-format	<div><div></div></div> 17.07%	41	0	7	<div><div></div></div> 0.00%
<a href="#">job_20140130203054_4412501</a>	HIGH	rd	linkbase-rd	use-spaminfo.GetWiseIndexList@WISE	<div><div></div></div> 100.00%	1446	1446	0	<div><div></div></div> 25.75%
<a href="#">job_20140130203054_4409738</a>	NORMAL	rd	linkbase-rd	dlb_splitter.1401944962	<div><div></div></div> 0.00%	100	0	0	<div><div></div></div> 0.00%
<a href="#">job_20140130203054_4408979</a>	NORMAL	rd	linkbase-rd	easy_hadoop_rd@szjjh-spi-cm5.szjjh01.baidu.com_1744_1	<div><div></div></div> 100.00%	202	202	0	<div><div></div></div> 0.00%
<a href="#">job_20140130203054_4398939</a>	NORMAL	rd	linkbase-rd	alias_eva_select_url	<div><div></div></div> 31.56%	83090	26215	9	<div><div></div></div> 3.05%
<a href="#">job_20140130203054_4388060</a>	VERY_HIGH	rd	linkbase-rd	ic_update-prophet-format	<div><div></div></div> 99.99%	356	337	24	<div><div></div></div> 0.00%

- 上图可以看到当前队列下每个任务的优先级，占用的map槽位、reduce槽位，提交的client机器等等，其中State显示SUBMIT的任务，一般是刚提交的。如果你的cache文件很大，提交会等待很久。如果一直提交不上来，如果集群没问题，一般是你任务的某些参数设错了（参考第二节）。
- 结合此图以及上个图中显示的当前队列的资源使用情况，你就可以判断为什么你的任务抢不到槽位，哪个任务占用了这个队列的很多槽位。
- 点击其中一个任务，就进入到了这个任务的详细信息展示页面，这个页面大家比较熟悉了，只讲几个可能平时不注意的点：
  - **Job Groups**：目前集群大部分都是queue调度的，此参数可能是一个你看不懂的名字，忽略就好
  - **Partition Data Info**：可以看到partition阶段是否分环不均，严禁分环非常不均的任务上线！
  - **Job Lifetime Chart**：可以看到运行到目前为止，每时每刻任务跑的并发。为什么任务跑的比平时慢，也许是抢不到槽位，从图上可以看出
  - **Failed/Killed Task Attempts**：集群任务单个attempt出错很正常，机器数量多，意外也多。只要点击去发现这个attempt任务的之后的attempt尝试成功了就行。
  - **Running**：有时候可能发现running的map数量比Job Map Capacity参数配置的数量少，一般是抢不到槽位。如果整个队列只有你一个任务在跑还抢不到足够槽位，一般是参数设错了（比如队列总共100台机器，cpu消耗设置2个，那就不可能打满8000的并发了）
  - 长尾：经常看到Running下面只有个别任务了，但是拖了好久没跑完，可以点击数字进入running task页面，再点某个attempt进入该attempt的详细信息页面。
    - 该attempt的页面上有当前任务所跑的宿主节点，点击这个机器名，进入这个机器的页面，你可以看到其下有多少任务在跑，每个任务消耗的cpu，内存，磁盘。慢节点往往是压力较大的机器。如下

ContainerId	JobId
slave_job_app-20140605125530-54_job_20140219185841_298865-map_20140605171404-8	job_20140219185841_298865-map
slave_job_app-20140605113610-1_job_20140219185841_298638-reduce_20140605160214-1334	job_20140219185841_298638-reduce
slave_job_app-20140605113610-1_job_20140219185841_298638-map_20140605160745-38	job_20140219185841_298638-map
slave_job_app-20140219193116-6_shuffle_20140604113424-0	shuffle
slave_job_app-20140605125530-54_job_20140219185841_298865-map_20140605170508-8	job_20140219185841_298865-map

- 再回到之前的attempt任务页面：
- 日志部分应该不用介绍了，点进去后，stdout是你程序中打出来的日志。syslog是集群自动打出来的日志
- **Counters** 这项可以看到这个子任务的读写量，慢节点往往由于这个子任务的读写量远大于其他的任务，也可以通过“Partition Data Info”看到分环不均。
- 长尾任务一般会看到这个子任务有多个attempt在跑（一般是两个，其中第二个是预测执行），长尾的常见处理方法是：kill掉其中最慢的那个。
  - 运气好的话，会调度到一台压力小的机器。如果还是原机器，多kill几次试试。
    - 如果你有节点机器权限的话，去那台机器上面把agent杀了，等一会儿再kill那个attempt，就肯定调度到其他机器了，注意之后要恢复刚才那个机器的agent。
  - 如果不是宿主节点机器慢，而是由于另外机器造成的，比如这台宿主机正在从其他节点机器拖数据过来，但是对方机器网卡打满了。
    - 那么你有权限的话，可以通过各种方法把对方机器压力降下来（比如kill掉mapred用户进程）。实在没办法，可以把datanode停了，那么任务就会从另外一台机器去拿hdfs数据（因为有三个副本嘛），还是注意等下要记得恢复datanode。
  - 以上操作仅限单机操作，不可大量停节点。新人操作需要监督。
  - 如果任务很重要，又自己没法处理，等了很久还是没跑完，可以找dpf同学帮忙看看那个机器为什么这么慢。
- 
- 再回到metamaster的主页面，拉到最下面，会发现如下的几个链接。
  -

- Job History

Resource Manager

Shuffle Services

5375 Alive, 564 Dead

- 
  - 
  - 下面逐个讲述
  - **【job history 页面】**
    - 点击上图的job history，会发现进入一个历史任务搜索页面，方便你查看已跑完（也包括跑挂了或者被kill的）任务的一些历史信息。
    - 最精确的查找方法是根据 **jobid** 查，然后点击下面搜出来的jobid，会跳转到一个新的页面。里面的信息非常全，下面详述：
      - **jobConf**：后面跟的链接点进去会看到你的任务所有的**参数设置**。
      - **status**：显示是任务是跑成功了还是失败了，还是被别人杀了
      - **Job Lifetime Chart**：一般有两个图（没有reduce阶段的话就一个图），显示的是整个任务生命周期内，并发数的时时情况。是否长尾清晰可见（仅剩一两个子任务拖很长的一段尾巴）
    - 再往下，信息更多，只讲几个重点
      - **For more details: Analyse This Job's Tasks**，点进去可以看到 **map,shuffle,reduce** 三个阶段的长尾任务，方便优化程序
      - **For more details: See This Job's Partition Info**，点进去看到 **partition** 信息，看是否有分环严重不均
      - **Total Map Slot Time**和**Total Reduce Slot Time**：这两个值就是**槽位消耗量**和**时间的积分**，就是**最精确的槽位资源的消耗量**。单位是槽位\*秒。比如“Total Map Slot Time”显示 **7200**，就相当于**1个槽位跑7200秒**，或者**120个槽位跑60秒** 等等
  - **shuffle service** 主要看下dead数不太多，就没问题了。一般一个节点会起一个shuffle
  - **resource manager** 页面很重要，点进去之后是整个集群的调度页面展示，详见下一小节。

调度页面

- State: Running

SchedulingThread: TIMED\_WAITING

Started: Thu May 29 15:07:53 CST 2014

Version: 1.3.10

Revision: 48395

Compiled: Mon May 5 15:45:07 CST 2014, build1@ml-scm-build72.ml.baidu.com

Configuration

Heap Memory used 11.42 GB is 57% of Committed Heap Memory 19.8 GB. Max Heap Memory is 19.8 GB.

Non Heap Memory used 34.76 MB is 68% of Committed Non Heap Memory 50.77 MB. Max Non Heap Memory is 130 MB.

Cluster Summary

Apps	Jobs	Live Agents	Dead Agents	Resource Nodes	CPU(free/total)	Memory(free/total)	Disk(free/total)
<a href="#">679</a>	899	<a href="#">5281</a>	<a href="#">192</a>	<a href="#">46</a>	265036/422971	83377854MB/138330577MB	104384987308MB/110171

- 如上图，**State: Running**是表示正常的。**resource manager** 异常或者在升级时，会提示**safemode**，此时调度不可用
  - **Apps**和**Jobs**：你提交一个任务会被集群当做一个应用（App），**appmaster**也作为一个常驻的App。而jobs包含三种：**reduce**任务，**map**任务和**appmaster**。因此，如果一个App只有只有map,那么就对应一个job。如果既有map又有reduce，那么这个App对应两个job。可以点击Apps下面的数字**679**进入查看
  - **Dead Agents**：挂掉的agent数量不要太大就ok
  - **CPU**，**Memory**，**Disk**：这三个数值就是**集群所有资源的总量和当前使用量**。要问集群是否空闲，采集一段时间这个三个值即可。
  - **Resource Nodes**:重点下面介绍**资源节点树**，点击下面的数字**46**会进入下图的页面
  - 
  - **【 Resource Node 页面 】**
  -



Resource Node List

Id	Name	Type	Parent Id	Host Num	Total Resource (CPU/Memory/Disk)	CPU% (/PhyNode)	Allows Lending% (/Total)	Allows Borrowing [Logical] or safemode [Physical]	Preempt Timeout	Local Free (CPU
1	levl_root	COMPOSITE	-	-	207254/54739984MB/56803265663MB	-	-	-	-	-/-/-
2	appmaster_phy	PHYSICAL	1	20 / 0	1600/559775MB/419430400MB	-	-	OFF	-	0/0MB
4	<a href="#">abaci.appmaster</a>	LOGICAL	2	-	1600/559775MB/419430400MB	100.0	false	50.0	0	110/1
3	ccdb_phy	PHYSICAL	1	43 / 2	3020/1087134MB/856886592MB	-	-	OFF	-	1978,1
5	<a href="#">ccdb1</a>	LOGICAL	3	-	2899/1043648MB/822419128MB	96.0	true	100.0	480	40/12
14	<a href="#">mis_dict</a>	LOGICAL	3	-	120/43485MB/34267463MB	4.0	true	0.0	480	0/0MB
6	build_phy	PHYSICAL	1	9 / 3	620/227511MB/179306496MB	-	-	OFF	-	400/1
10	<a href="#">build1</a>	LOGICAL	6	-	620/227511MB/179306496MB	100.0	true	100.0	480	0/0MB
7	dnews_phy	PHYSICAL	1	20 / 0	1200/505644MB/398458880MB	-	-	OFF	-	778/1
11	<a href="#">dnews</a>	LOGICAL	7	-	1200/505644MB/398458880MB	100.0	true	100.0	480	0/0MB
8	wdn_phy	PHYSICAL	1	60 / 2	5200/1516687MB/1195376640MB	-	-	OFF	-	3565,1
12	<a href="#">wdn</a>	LOGICAL	8	-	5200/1516687MB/1195376640MB	100.0	true	100.0	480	1600,1
9	wp_phy	PHYSICAL	1	14 / 2	840/353951MB/278921216MB	-	-	OFF	-	533/1
13	<a href="#">wp</a>	LOGICAL	9	-	840/353951MB/278921216MB	100.0	true	100.0	480	0/0MB
15	default_phy	PHYSICAL	1	2682 / 23	194657/50359709MB/53435603649MB	-	-	OFF	-	9809,1
16	<a href="#">bailing-default</a>	LOGICAL	15	-	52557/13597121MB/14427612985MB	27.0	true	100.0	480	7514,1
17	<a href="#">build</a>	LOGICAL	15	-	52557/13597121MB/14427612985MB	27.0	true	100.0	480	2034,1

- 篇幅有限，只显示上半部分
- 这个页面实际上是resource manager 那个机器上面的 resource-tree.conf 配置的一个页面展示，其实是json格式描述的一棵节点树
- 第一行id是1的那个根节点比较特殊，其实就是描述了所有的资源，一般不用关心
- 下面的每行分两种，PHYSICAL(物理分组)和LOGICAL(逻辑分组)。目前我们集群都是跑在逻辑分组上的，提交任务的时候指定的 mapred.job.queue.name 参数的值就是下面那些标注了LOGICAL的其中一个。
- 逻辑分组都是挂载在物理分组上的，一个物理分组可以对应多个逻辑分组，这些逻辑分组划分整个物理分组资源。资源分配见下面说明：
- 比较特殊的是appmaster，可以看到 Host num 是20，说明有20个appmaster，点击那个数字20可以看到集群的 appmaster 是哪些机器上面起的。
- 我们再看 id 3 和 id 5：
  - id 3是物理分组，id 5是逻辑分组；
  - id 5的Parent id是3，这个参数指定了逻辑分组所挂载的物理分组是哪个。
  - 因此，你如要看你提交的任务的分组 ccdb1 当前的资源使用情况，看id 5所在行。
  - 如果要看该分组死了多少agent，看其parent id即id 3所在行，发现43个活着，2个死了。
- 我们再看id 15，id 16，id 17。
  - 可以看到16和17是逻辑分组，挂载在15的物理分组上。
  - 再看 Cpu% 这一项，分别是0.27,0.27。实际上，图片的下半部分还有 id 18，id 19.....（篇幅有限未显示），都是属于id 15这个default这个物理分组的。
    - 而所有这些逻辑分组的所有 Cpu% 这一项之和是100%。这就是物理分组内部各逻辑队列的资源原始划分
  - 另外我们看 Allows Lending 和Allows Borrowing，这是表示这个逻辑分组是否可以把资源借出；以及可以向整个物理分组借用的资源百分比（100表示占用整个物理分组资源。也就是说
    - 其他逻辑分组空闲时可以独占整个物理分组）。可以看到16和17两个逻辑分组的最后一项 Borrowed 都不为0，这就是借到的资源
  - Preempt Timeout这个480表示：当本队列需要资源时，经过480秒之后，资源开始被拿回来。因此不用担心本逻辑分组要用资源时抢不回来。另外 resource-tree.conf 配置中还有一个参数 fairshare.preempt.timeout，超过这个时间还没归还资源，会kill掉对方子任务强制拿回。

五、任务异常及日志

任务失败基本处理

- 任务提交时会给出一个任务链接，可以在浏览器中进入到失败任务的详细页面。也可以通过jobid在history页面中搜索。distcp任务启动时仅给出jobid，也可以在history页面搜索到。
- 页面中会提示:导致作业失败的task。；或者有时候只显示一些Failed tasks，此时可以挑那个达到最大尝试次数还失败的任务看。
  - 点进去之后，随便看几个attempt的 Task Logs。一般结合 syslog 和 stderr 可以定位到集群还是程序自身问题。
  - 集群问题可以在 <http://dpfhelp.dmop.baidu.com/> 页面点击 问题反馈；程序自身问题请结合代码定位。下面会具体举一些例子

子进程返回信号

- 参考 <http://op.baidu.com/twiki/bin/view/Ps/OP/Hadoopwiki>
- 其他一般情况下，如果子进程返回码大于128，则可以用错误码减去128，表示子进程收到的信号（SIGNAL）；信号的意义可以通过 man 7 signal 来查看。当然，用户自己在程序中 exit XXX 就不用解释了。
- 子进程返回141
  - map或reduce程序超出平台内存限制或者超过CPU时间被limit杀掉，平台默认配置内存限制为800MB，CPU时间12小时（137-128=9，对应信号为SIGKILL）
- 子进程返回137
  - map或reduce异常退出，平台继续向管道推送数据，因管道异常出错(141-128=13，信号13代表着SIGPIPE错误，即管道错误)

- 根本原因还是程序异常退出导致。详细出错原因需rd自己结合Task Logs和程序源码定位
- 子进程返回1
  - 表示程序返回的就是1。请用户检查程序哪里引起的错误导致返回1

## 常见错误日志/信息

- 【超quota】
  - 举例:
    - org.apache.hadoop.fs.QuotaExceededException: The quota of /user/rd is exceeded: namespace quota=1800000000 file count= 35757505, diskspace quota=5037802324992000 diskspace= 5497558138880000
  - 说明:
    - 凡是显示形如上面这个报错的，就是超quota了：哪个目录超了（/user/rd），超了多少（后面两个数字相减），是spcae超还是inode超（space的数值大于quota，因此space超了），都在报错里面显示了
    - 小目录超了，赶紧删数据吧。如果要删目录下哪些是大头，参考：<http://wiki.babel.baidu.com/twiki/bin/view/Ps/OP/hadoopQuotaExceededDeal>
    - 大目录超了，如整个 /user/spider 超了，可能删除都删不了，那么得找集群业务接口人删数据
    - 如果删了之后过了迟迟没有生效（inode超了的话，删了之后很快生效），是因为dpf的quota采集脚本有延迟，紧急情况下可以找dpf同学用 setSpaceConsumed命令临时解决。
- 【ugi用错导致没权限】
  - 举例:
    - mkdir: org.apache.hadoop.security.AccessControlException: Permission denied: user= spider, access= WRITE, groups= default spider, inode="spider
  - 说明:
    - 凡是显示形如上面这个报错的，就是ugi用错导致没权限；你用的是哪个ugi(spider)，你尝试要用的权限（WRITE）。
    - 这个例子中，这个错误实际上来自于/user/spider 目录的权限是 root:root，755。而尝试用 spider 用户去创建 /user/spider/test 目录会报如上错误。
- 【目录前面//导致提示没权限】\* 举例:
  - hadoop fs -mkdir //tmp/xxx 提示报错
  - 说明:
    - 目录开头用//会出错，只能有一个/。倒是目录中间有//可以兼容
- 【提交的分组用错】
  - 举例: \* Error Launching job : java.io.IOException: queue wp is not allowed
  - 说明:
    - 凡是显示形如上面这个报错的，提交的分组用错导致该ugi没法提交到这个队列。当然也可能是ugi设错。请联系集群业务接口人确认ugi和队列之间是否有权限
    - 这个例子中，这个错误实际上来自于用spider用户向wp队列提交任务，但是metamaster的hadoop-user-info.properties配置中并没有开通让spider用户往wp队列提交任务的权限。权限方面申请参考上面第三节 权限和资源
    - 有时候因为公用的hadoop-site.xml，可能导致你提交到别人的分组，也报这个错，比如rd账号提交到了linkbase分组，也会报错
      - 如果是私用的hadoop-site.xml。对于nmg-global集群（queue调度），可以直接修改参数mapred.job.queue.name；对于szjh-dbuild集群（group调度），可以直接修改参数mapred.job.groups
      - 如果是公用的，不能随便修改，对于nmg-global集群（queue调度），提交任务的时候加上-D mapred.job.queue.name =webdata；对于szjh-dbuild集群（group调度），提交任务的时候加上-D mapred.job.groups =linkbase-rd
- 【文件写坏了或丢失】
  - 举例:
    - 错误日志里面显示某个路径的文件：file does not exist
    - 或者显示一个 ojectid 找不到
  - 处理:
    - 如果报出了具体路径，直接hadoop fs cat命令看下，如果报错了，说明文件写坏了，删了重来吧。也可以用hadoop fsck命令；如果cat和fsck没报错，有很小概率是集群抽风，曾经尝试cp到另一个地方，然后删到原来的，mv回来，居然好了。一般没报错可以按程序逻辑接下去跑。
    - 如果只报了objectid，没有报出路径，那么可以参考：<http://wiki.babel.baidu.com/twiki/bin/view/Ps/OP/findHadoopMissingBlock>。知道路径之后处理同上。
- 【并发太高】
  - 举例:
    - 报获取不到块的错误或者connection busy
  - 处理:
    - 一般是某个热点文件被频繁访问，副本数不够或者setrep之后没有sleep导致集群还来不及复制全部副本。建议任务setrep之后sleep几分钟或者加参数-w。
- 【NlineInput 会自动设置10个副本】
  - 举例:
    - 提交任务时，如果设置inputformat为Nlineinput。比如：-inputformat org.apache.hadoop.mapred.lib.NLineInputFormat。就算你之前把某个热点文件设置了15个副本，也会被强制修改为10个副本。可能导致读不到块的异常。
  - 处理:
    - Nlineinput在切分数据时会设置副本数，默认10个副本，可以在启动任务时设置这个参数修改mapred.line.input.format.replication=15
- 【/app超quota】
  - 举例:
    - 提交任务时或者任务运行中，提示因为/app超了而失败。但是任务的输入输出目录都不在/app下面
  - 处理
    - 因为/app是集群的管理目录，很多临时性的数据会存放在这里。比如distcp任务会在这个目录下写临时数据，如果/app超quota不可写了，任务当然挂了。得找dpf同学清理数据
- 【重跑可以成功的一些报错】
  - 凡下面列出的一些信息，只是少量出现，重试的attempt可以成功就可以忽略。一般是因为单机问题、网络问题等意外造成的。
    - Failed to connect to shuffle service
    - Failed to do local commitment to shuffle service
    - Too many fetch-failures
    - no space left

## 六、业务逻辑

## rmr的异常

- 分析：
  - 写程序的时候往往rmr一个文件，后面马上跟一个mv或者put操作。集群抽风时会报file exists。是因为虽然返回删除成功了，但是还没来得及真正注销掉这个inode。
- 处理：
  - rmr操作之后判断返回是否0，并且进一步用test命令判断是否目录或这件还存在，还存在的话sleep一段时间在进行下一步操作。建议将rmr操作进行封装。

## fork进程或起多线程任务

- 分析：
  - hadoop已经是多节点并发了，没有必要再用多线程或多进程。并且，目前hadoop还无法控制fork出来的进程所使用的内存，那么oom的风险很高
- 建议：
  - 原则上禁止fork进程或起多线程

## 节点上的本地日志上传

- 不合理用法举例：
  - 所有mapper打的日志统统上传到集群
- 分析：
  - 当一个任务的map数量非常大的时候，比如有些甚至达到百万级别的map，那么就会有百万个小文件上传。这样涉及百万次mkdir和put操作。另外，大量的带宽被消耗，影响自己任务和他人任务
- 建议：
  - 上传日志的模块加个判断，如果任务失败，才上传日志。

## 多轮mapreduce优化

- 分析：
  - 多轮mapreduce之间一般都是用hdfs作为之间的临时存储，那么对造成额外的读写消耗。
- 建议：
  - DAG改造，参考：<http://wiki.babel.baidu.com/twiki/bin/view/Com/Inf/DAGMR#DAG.pptx>

## libhdfs中的hdfsFlush方法

- 不合理用法举例：
  - 大并发的每个map频繁调用hdfsFlush
- 分析：
  - libhdfs提供hdfsFlush将写入的内容立马生效(flush)到集群。该操作对namenode的压力比较大，每次加写锁，磁盘I/O操作。大量并发操作时可能打垮集群。
- 建议：
  - 一般离线业务不需要那么高的时效性（写入的内容立马生效并能读到）。因此使用hdfsWrite写完之后调用close即可。

## 七、不常见的坑

### 本地基础环境和salve差异的坑

- 分析：
  - 追查问题发现任务报错：org.apache.hadoop.util.Shell\$ExitCodeException: tar: dlb\_sender.conf: Cannot hard link to `dlb\_sender.conf': No such file or directory tar
- 处理：
  - 提交client机器的tar版本被改了。回滚到和集群slave同个tar版本

### 某节点机器网卡出异常

- 分析：
  - 这是一个不多见的单节点异常导致整个集群异常的个例，因为某节点机器网卡入正常，因此master分配给client往这台机器写数据，并在写完第一个副本后返回client:成功（之后的复制由集群异步完成）。但由于网卡出异常，无法复制，也无法读取。
- 处理：
  - 先止损，干掉该机器的datanode进程。然后复制重要的block到其他节点的相应目录，然后重启那个节点就可被master感知到。参考：<http://wiki.babel.baidu.com/twiki/bin/view/Ps/OP/Publish1>

## 八、资源控制临时手段










- 近来有很多同学反馈任务抢占严重，高优任务跑不完，任务管理困难。这是容量管理没做好的恶果，如果实在短期没法梳理，下面介绍几个方法：
- 1. 优先级白名单机制，非白名单的任务统统设置为NORMAL或者更低优先级。你有hadoop-client和ugi就可以管理你自己组里的。
  - curl 下你的队列的页面（metamaster页面上点你们的队列进去），拿到当时跑的所有任务，根据jobname等信息，再比对白名单，拿到不该用HIGH以上优先级跑的任务
  - 然后使用命令：hadoop job -set-priority <job-id> <priority> 比如 **hadoop job -set-priority job\_20150507183057\_238261 NORMAL**
  - 参考demo: <http://wiki.baidu.com/pages/viewpage.action?pagelId=96398492>
- 2. 不允许提交任务的机器黑名单机制，非白名单的任务统统杀掉。你有hadoop-client和ugi就可以管理你自己组里的。
  - curl 下你的队列的页面（metamaster页面上点你们的队列进去），拿到当时跑的所有任务，根据client ip拿到机器所在的noah路径。根据你们的线上部署要不要杀
  - 用到的命令：
    - host \${IP}
    - get\_service\_by\_host -i \${HOSTNAME}
    - 针对上面获得的bns : get\_ancestor\_product\_by\_service \${BNS}
  - 然后使用命令杀：hadoop job -kill <job-id> 比如 **hadoop job -kill job\_20150507183057\_238261**

- 参考demo: <http://wiki.baidu.com/display/~yangsen01/kill+offline+job>
- 3. 高峰期高优任务保障，整个队列的cpu或memory使用接近100%时，VERY\_HIGH任务饥饿，而一些耗时的normal任务不释放槽位，此时suspend掉normal任务。你有hadoop-client和ugi就可以管理你自己组里的。
  - curl 下metamaster页面，或者hadoop queue -list 命令拿下你的队列的cpu和memory使用率，如果使用率不高，退出
  - 拿到当时跑的所有VERY\_HIGH任务，看看是否饥饿（有pending），有的话suspend掉部分NORMAL以下任务。
  - 用到的命令：
    - hadoop queue -info <job-queue-name> -showJobs | grep VERY\_HIGH 比如 hadoop queue -info linkbase -showJobs
    - 拿到上面的所有very\_high任务的jobid，for循环执行：hadoop job -info <job-id> | grep Pending
    - 然后把所有very\_high任务的map和reduce的pending个数拿到了，如果大于0，说明有very\_high任务得不到调度
    - 然后同理拿下NORMAL以下任务的map和reduce的running个数，如果大于0，说明鸡占鹊巢了
    - 此时，选择不重要的或者running数量最多的一个或几个NORMAL以下任务执行suspend：hadoop job -suspend <job-id> <hour> 比如 **hadoop job -suspend job\_20150507183057\_238261 4**
    - 然后，过几分钟（不建议马上就resume），尝试recover这个任务：**hadoop job -recover job\_20150507183057\_238261**
- 参考demo: <http://wiki.baidu.com/pages/viewpage.action?pageId=97551153>
- 注意，suspend和recover有一定概率失败，可以增加sleep的值，增加suspend和recover的重试次数
- 以上方法只是临时的，我们正在研究更合理，更通用的解决方案。基于框架上层的job、模块、业务层面的调度和保障系统，敬请期待。。。•

九、声明

- 本规范是初稿，仅供参考，如有错误，欢迎评论或email反馈，非常感谢。
- 很多疑问可以从下面这些地方获得答案：
  - <http://dpfhelp.dmpop.baidu.com/>
  - <http://op.baidu.com/twiki/bin/view/Ps/OP/Hadoopwiki>
  - <http://wiki.babel.baidu.com/twiki/bin/view/Com/Inf/HadoopDoc>
  - <http://wiki.babel.baidu.com/twiki/bin/view/Com/Inf/HadoopManual>
  - <http://wiki.babel.baidu.com/twiki/bin/view/Com/Inf/HadoopQA>

标签:  + 添加新标签, 查看所有标签

I	附件	变更	大小	日期	谁	评论
	<a href="#">a.png</a>	<a href="#">管理</a>	33.9 K	2014-06-05 - 11:31	<a href="#">UnknownUser</a>	
	<a href="#">ark-a.png</a>	<a href="#">管理</a>	21.8 K	2014-06-05 - 21:43	<a href="#">UnknownUser</a>	
	<a href="#">ark-b.png</a>	<a href="#">管理</a>	48.4 K	2014-06-05 - 21:54	<a href="#">UnknownUser</a>	
	<a href="#">ark-c.png</a>	<a href="#">管理</a>	80.6 K	2014-06-05 - 22:04	<a href="#">UnknownUser</a>	
	<a href="#">fms-a.png</a>	<a href="#">管理</a>	17.7 K	2014-06-05 - 15:20	<a href="#">UnknownUser</a>	
	<a href="#">fms-b.png</a>	<a href="#">管理</a>	20.1 K	2014-06-05 - 15:20	<a href="#">UnknownUser</a>	
	<a href="#">ms-a.png</a>	<a href="#">管理</a>	13.0 K	2014-06-05 - 15:57	<a href="#">UnknownUser</a>	
	<a href="#">ms-b.png</a>	<a href="#">管理</a>	53.0 K	2014-06-05 - 16:05	<a href="#">UnknownUser</a>	
	<a href="#">ms-c.png</a>	<a href="#">管理</a>	3.9 K	2014-06-05 - 16:26	<a href="#">UnknownUser</a>	
	<a href="#">ms-d.png</a>	<a href="#">管理</a>	24.9 K	2014-06-05 - 17:22	<a href="#">UnknownUser</a>	
	<a href="#">ns-2.png</a>	<a href="#">管理</a>	11.6 K	2014-05-30 - 11:58	<a href="#">UnknownUser</a>	
	<a href="#">ns-3.png</a>	<a href="#">管理</a>	10.6 K	2014-05-30 - 11:58	<a href="#">UnknownUser</a>	
	<a href="#">ns-a.png</a>	<a href="#">管理</a>	8.4 K	2014-06-05 - 11:36	<a href="#">UnknownUser</a>	
	<a href="#">ns-b.png</a>	<a href="#">管理</a>	2.1 K	2014-06-05 - 11:36	<a href="#">UnknownUser</a>	
	<a href="#">ns-c.png</a>	<a href="#">管理</a>	11.8 K	2014-06-05 - 11:36	<a href="#">UnknownUser</a>	
	<a href="#">ns-d.png</a>	<a href="#">管理</a>	10.5 K	2014-06-05 - 11:36	<a href="#">UnknownUser</a>	

主题版本： r42 - 2015-05-27 - yangsen01