基于上网日志分析的Hadoop 练习手册

1. HDFS DFS命令行文件操作类API操作练习 4

1.1 练习目的： 4

1.2 练习案例说明： 4

1.3 练习步骤: 5

2. HDFS 创建快照、垃圾回收站操作练习 5

2.1 练习目的： 5

2.2 练习案例说明： 5

2.3 练习步骤: 5

3. HDFS Cache操作练习 6

3.1 练习目的： 6

3.2 练习案例说明： 6

3.3 练习步骤: 7

4. HDFS Dfsadmin基础操作练习 7

4.1 练习目的： 7

4.2 练习案例说明： 7

4.3 练习步骤: 8

5. HDFS Dfsadmin进阶操作练习 8

5.1 练习目的： 8

5.2 练习案例说明： 8

5.3 练习步骤: 8

5.3.1 HDFS 安全模式练习 8

5.3.2 HDFS 文件夹Quota 设置练习 9

5.3.3 HDFS NameNode 元数据相关操作练习 10

5.3.4 HDFS 负载均衡练习 11

5.3.5 HDFS distcp 12

5.3.6 HDFS OIV OEV练习 15

6. HDFS用户管理进阶操作练习 15

6.1 HDFS 用户管理 15

7. 上网日志分析MapReduce高级操作练习 18

7.1 练习目的： 18

7.2 练习案例说明： 18

7.3 练习步骤: 18

7.3.1 编写数据清洗脚本 18

7.3.2 编写数据统计脚本 18

7.3.3 数据统计脚本进阶－二次排序，序列化 19

7.3.4 数据统计脚本进阶－增加combiner 19

7.3.5 数据统计脚本进阶－增加partitioner 19

8. MapReduce 管理操作练习 19

8.1 练习目的： 19

8.2 练习案例说明： 20

8.3 练习步骤: 20

9. Hive 上网日志分析系统练习 21

9.1 练习目的： 21

9.2 练习案例说明： 21

9.3 练习步骤: 21

10. Hive 上网日志分析系统进阶练习－表格式 22

10.1 练习目的： 22

10.2 练习案例说明： 22

10.3 练习步骤: 24

11. Hive 上网日志分析系统进阶练习－自定义函数 25

11.1 练习目的： 25

11.2 练习案例说明： 25

11.3 练习步骤: 25

12. HBase 上网日志分析系统进阶练习－表结构设计 25

12.1 练习目的： 25

12.2 练习案例说明： 26

12.3 练习步骤: 26

13. HBase 上网日志分析系统进阶练习-bulkload, Coprocessor等 26

13.1 练习目的： 26

13.2 练习案例说明： 26

13.3 练习步骤: 26

# HDFS DFS命令行文件操作类API操作练习

## 1.1 练习目的：

通过本练习，学员可以掌握HDFS DFS命令行API基本使用。学会拷贝，删除，浏览文件等基础的操作。

## 1.2 练习案例说明：

数据位于hadoop home, 为上网日志分析的数据 log.data

## 1.3 练习步骤:

需要学员完成：

1. 将log.data 拷贝入HDFS的 /log 路径,借助 put ,mkdir命令
2. 通过<http://localhost:50070/> 浏览该文件，查看文件的block及其分布的机器
3. 使用命令查看 log.data的内容，并且尝试去掉crc校验和机制，借助cat命令
4. 将log.data 改名成log.tab，借助cp命令
5. 将log.tab 拷贝到本地, 借助copyToLocal命令

# 2. HDFS 创建快照、垃圾回收站操作练习

## 2.1 练习目的：

通过本练习，学员可以掌握HDFS 快照、垃圾回收站基本使用。学会制作快照，通过快照对数据进行恢复。

## 2.2 练习案例说明：

数据使用练习1的数据进行

## 2.3 练习步骤:

需要学员完成：

1. 管理员允许对log文件夹创建快照，借助：dfsadmin

[-allowSnapshot <snapshotDir>]

1. 对文件log.tab 创建快照，并起名字为当前系统时间

借助：hdfs dfs [-createSnapshot <snapshotDir> [<snapshotName>]]

1. 通过hdfs 命令查看快照数据,借助 hdfs dfs –ls
2. 模拟异常删除log.tab 借助hdfs dfs -rmr
3. 通过快照进行数据的恢复, 借助 hdfs dfs -cp 命令
4. 关闭快照 [-disallowSnapshot <snapshotDir>]
5. 打开垃圾回收站
6. 删除恢复回来的数据
7. 查看垃圾回收站的内容

# 3. HDFS Cache操作练习

## 3.1 练习目的：

通过本练习，学员可以掌握Cache基本使用。学会制作Cache

## 3.2 练习案例说明：

数据使用练习1的数据进行

命令说明

addPool/modifyPool

hdfs cacheadmin -addPool name [-owner owner] [-group group] [-mode mode] [-limit limit] [-maxTtl maxTtl]

-owner/group是该pool的属主/组，默认为当前用户

-mode是POSIX风格权限，默认为0755

-limit为该pool中可以缓存的最大字节数，默认没有限制

-maxTtl 最大的生存期，可以是120s, 30m, 4h, 2d等。

addDirective：添加指令

hdfs cacheadmin -addDirective -path path -pool pool-name [-force] [-replication replication] [-ttl time-to-live]

-path 添加的路径

-pool 加入的缓冲池名称

-force 不检查缓存池的资源限制

-replication 要使用的副本数，默认为1

-ttl 缓存指令可以保持多长时间。可以按照分钟，小时，天来指定，如30m，4h，2d。有效单位为[smhd]。“never”表示永不过期的指令。如果未指定该值，那么，缓存指令就不会过期。

## 3.3 练习步骤:

需要学员完成：

1. 创建pool，name为log，设置owner 为hadoop, -maxTtl为120秒
2. 对log文件夹创建cache，pool name为log 借助：hdfs cacheadmin –addDirective，设置要使用的副本数是2

# 4. HDFS Dfsadmin基础操作练习

## 4.1 练习目的：

通过本练习，学员可以掌握hdfs dfsadmin基本使用。

## 4.2 练习案例说明：

数据使用练习1的数据进行

## 4.3 练习步骤:

需要学员完成：

1. 使用report 命令，查看集群信息，并比较与50070界面的不同
2. 使用-printTopology，打印集群拓扑结构
3. 获取datanode info

[-getDatanodeInfo <datanode\_host:ipc\_port>]

1. 停止datanode 服务，

[-shutdownDatanode <datanode\_host:ipc\_port> [upgrade]]

# 5. HDFS Dfsadmin进阶操作练习

## 5.1 练习目的：

通过本练习，学员可以掌握hdfs dfsadmin进阶使用。

## 5.2 练习案例说明：

数据使用练习1的数据进行

## 5.3 练习步骤:

### 5.3.1 HDFS 安全模式练习

NameNode在启动的时候 首先进入安全模式，如果datanode丢失的block达到一定的比例（由hdfs-site.xml文件中dfs.safemode.threshold.pct决定，默认0.999f），则系统会一直处于安全模式状态即只读状态 ； 否则没有其他情况影响，一般情况下，系统会自动离开安全模式，一般用户修复系统时使用

dfs.safemode.threshold.pct 表示HDFS启动的时候，如果DataNode上报的 block个数0.999倍才可以离开安全模式，否则一直是这种只读状态。如果设为1则hdfs永远是处于SafeMode。

通常两种情况可以离开这处安全模式：

1、修改 dfs.safemode.threshold.pct为一个比较小的值，缺省值是0.999

2、Hadoop dfsadmin -safemode leave 命令强制离开

# hdfs dfsadmin -safemode get ## 返回安全模式是否开启的信息，返回 Safe mode is OFF/OPEN

# hdfs dfsadmin -safemode enter ## 进入安全模工

# hdfs dfsadmin -safemode leave ## 强制 NameNode 离开安全模式

# hdfs dfsadmin -safemode wait ## 等待，一直到安全模式结束

### 5.3.2 HDFS 文件夹Quota 设置练习

hadoop HDFS有以下两种Quota

* Name Quotas : 限制某个目录下的文件数量
* Space Quotas : 设置某个目录的空间大小

$hadoop fs -count -q /log

QUOTA REMAINING\_QUOTA SPACE\_QUOTA REMAINING\_SPACE\_QUOTA DIR\_COUNT FILE\_COUNT CONTENT\_SIZE FILE\_NAME

QUOTA : 命名空间的quota(限制文件数)

REMAINING\_QUOTA : 剩余的命名空间quota(剩余能创建的文件数目)

SPACE\_QUOTA : 物理空间的quota （限制磁盘空间占用大小）

REMAINING\_SPACE\_QUOTA : 剩余的物理空间

DIR\_COUNT : 目录数目

FILE\_COUNT : 文件数目

CONTENT\_SIZE: 目录逻辑空间大小

FILE\_NAME : 路径

设置Space Quota

hadoop dfsadmin -setSpaceQuota 1g /log

**比较fsck与**CONTENT\_SIZE 大小会发现不同

物理空间：文件实际占用HDFS的大小（包含replication）

逻辑空间：文件本身的大小，是逻辑上占用HDFS的大小。

物理空间 = 逻辑空间 \* Replication数量

hadoop fsck出来和hadoop fs -count 出来的CONTENT\_SIZE指的都是逻辑空间，而物理空间是逻辑空间\*备份数

如果replication是3 REMAINING\_SPACE\_QUOTA = SPACE\_QUOTA - CONTENT\_SIZE \* 3

清除Space Quota

hadoop dfsadmin -clrSpaceQuota /log

设置Name Quota

hadoop dfsadmin -setQuota 1000 /log

清除Name Quota:

hadoop dfsadmin -clrQuato /user/hadoop

### HDFS NameNode 元数据相关操作练习

* hdfs dfsadmin -metasave filename,

保存Namenode的主要数据结构到hadoop.log.dir属性指定的目录下的<filename>文件。对于下面的每一项，<filename>中都会一行内容与之对应

1. Namenode收到的Datanode的心跳信号

2. 等待被复制的块

3. 正在被复制的块

4. 等待被删除的块

* hdfs dfsadmin -saveNamespace

Save current namespace into storage directories and reset edits log.

Requires superuser permissions and safe mode

* hdfs dfsadmin –rollEdits

Rolls the edit log.Requires superuser permissions

* hdfs dfsadmin -fetchImage

Downloads the most recent fsimage from the Name Node and saves it in the specified local directory

hdfs oiv -i fsimage\_0000000000000022533 -o testfs

hdfs oev

### HDFS 负载均衡练习，建议一周做一次

尽量不要在NameNode上执行start-balancer.sh，而是找一台比较空闲的机器

负载均衡带宽设置

hdfs dfsadmin -setBalancerBandwidth <bandwidth in bytes per second>

千兆网的情况下，建议最多是带宽一半

负载均衡命令执行

start-balancer.sh [-threshold <threshold>]

-threshold 默认设置：10，参数取值范围：0-100，参数含义：判断集群是否平衡的目标参数，每一个 datanode 存储使用率和集群总存储使用率的差值都应该小于这个阀值 ，理论上，该参数设置的越小，整个集群就越平衡，但是在线上环境中，hadoop集群在进行balance时，还在并发的进行数据的写入和删除，所以有可能无法到达设定的平衡参数值。

dfs.balance.bandwidthPerSec 默认设置：1048576（1 M/S），参数含义：设置balance工具在运行中所能占用的带宽，设置的过大可能会造成mapred运行缓慢

### HDFS distcp

命令：hadoop distcp

DistCp（Distributed Copy）是用于大规模集群内部或者集群之间的高性能拷贝工具。 它使用Map/Reduce实现文件分发，错误处理和恢复，以及报告生成。 它把文件和目录的列表作为map任务的输入，每个任务会完成源列表中部分文件的拷贝。

DistCp是Apache Hadoop自带的工具，目前存在两个版本，DistCp1和DistCp2，FastCopy是Facebook Hadoop中自带的，相比于Distcp，它能明显加快同节点数据拷贝速度，尤其是Hadoop 2.0稳定版（第一个稳定版为2.2.0，该版本包含的特性可参考我的这篇文章：Hadoop 2.0稳定版本2.2.0新特性剖析）发布后，当需要在不同NameNode间（HDFS Federation）迁移数据时，FastCopy将发挥它的最大用武之地

FastCopy的主要流程如下：

查询源NS中文件的meta信息，获取源文件所有的block信息

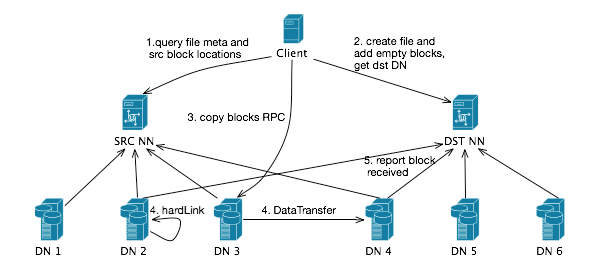
对于每个Block，获取其在原集群中的location信息

对于源文件中的每个block，在目标NS中的文件上添加空的block信息

对于所有的源Block，通过DN的copyBlock接口实现local copy

每个目标DN在完成block的copy之后向目标NS的NN中报告接收的Block

等待所有的block都copy完成后推出 基本结构如下：



2.1 File Meta复制

FastCopy首先获取源文件的meta信息(FileStatus)和blocks locations(LocatedBlocks)信息

检查源文件是否处于构建中，LocatedBlocks.isUnderConstruction()，如果是则跳过该文件

在目标NS中创建目标文件， 副本数，permission， blockSize等信息与源文件一致，为避免目标文件已存在，默认使用覆写模式创建

2.2 Block 复制

对于源文件的LocatedBlocks中所有的block信息，进行Block复制：

通过向目标NS中addBlock获取目标NS中block的DN 列表

对源block的DN列表和目标Block的DN列表进行排序对齐，使相同的DN在各自的列表中的位置相同

通过源Datanode的copyBlock()接口实现想目标DN的block数据复制

DataNode.copyBlock()的具体实现分为三种情况：

2.2.1 同一DN实例

这种情况存在于Federation中同一DN节点服务于两个NS的情况，事实上只需要为源Block的文件创建一个HardLink指向目标block文件

2.2.2 同一节点上不通DN实例

（云梯中暂时不存在这种单节点多DN实例的情况） 这种情况实际上文件位于同一台物理节点上，也可以通过HardLink完成，但由于两个DN实例维护不不同的VolumeMap，因此，需要源DN实例调用目标DN的copyBlockLocal（）接口实现，copyBlockLocal本质上也是使用HardLink来完成copy

2.2.3 不同DN节点

通常源DN和目标DN都是位于不同节点上的，需要通过网络传输block数据，这个传输过程与client向DN写入Block数据基本一致，因此可以直接使用DataTransfer来完成。

2.3 Lease更新

由于FastCopy在复制过程中需要对目标文件进行写入，但不是使用FileSystem的API（HDFS默认的lease机制是位于DFSClient中），因此FastCopy需要自己完成Lease的更新。对于每个copy的文件，FastCopy都需要启用一个LeaseChecker线程定期更新lease，保证数据写入的一致性。

2.4 复制状态监控

批量的文件copy是异步执行的，FastCopy内部通过一个fileStatusMap维护所有需要复制的文件，文件的状态中包括文件名，文件的block数以及已经完成复制的block， 以及一个blocksStatusMap维护每个需要复制的block的状态，block的状态包括block的副本总数，已经写入成功的副本数，以及写入失败的副本数。

3.2 实现逻辑

* FastCopy首先处理命令行参数，提取源文件和目标文件path，为每一个src：dst对构建FastCopyFileRequest
* 构建FastCopy实例处理FastCopyFileRequest
* FastCopy内部的调度处理通过ExecutorService维护一个线程池（线程池大小可以通过命令行-t 参数来控制，默认是5），每个线程是由实现了Future接口的FastFileCopy来对每个文件的copy进行处理。
* FastFileCopy内部实现上述设计文档描述的一个文件元信息copy的流程，并通过BlockCopyRpc异步调用DN的copyBlock接口实现block的复制
* FastCopy为每个文件的copy维护一个LeaseChecker，更新lease信息。
* 同时FastCopy通过内部维护fileStatusMap和blocksStatusMap来对copy过程状态进行管理

hdfs fastcopy

https://issues.apache.org/jira/browse/HDFS-2139%20HDFS-2139

### HDFS OIV OEV练习

* hdfs oiv

apply the offline fsimage viewer to an fsimage

* oev

apply the offline edits viewer to an edits file

# 6. HDFS用户管理进阶操作练习

## 6.1 HDFS 用户管理

Hadoop使用Linux用户管理，Hadoop中的用户就是Linux中的用户

Hadoop使用Linux分组管理，Hadoop中的分组就是Linux中的分组

操作用户组改变的命令

hdfs dfs

[-chgrp [-R] GROUP PATH...]

[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]

[-chown [-R] [OWNER][:[GROUP]] PATH...]

Linux权限说明

Linux操作系统中，会看到文件的操作权限，类似“drwxr-xr-x”

这串字符可以分成4段理解，结构为“d + 文件所有者操作权限 + 文件所有者所在组操作权限 + 其余人的操作权限”：

字母“d”，表示文件所在目录

字符串“rwx”，表示文件所有者对此文件的操作权限

字符串“r-x”，表示文件所有者所在组对些文件的操作权限

字符串“r-x”，表示除2、3两种外的任何人对此文件的操作权限

文件的读取、写入、执行权限：

执行：1、写入：2、 读取：4

数字：655

6：表示文件所有者的权限，4+2=6，即文件所有者对该文件有写入、读取。

5：表示文件所有者所在组的权限：4+1=5，即文件所有者所在组对文件有读、执行权限，没有写权限。

5：同上，其余人对该文件只有读、执行权限，没有写权限。

练习：

1. Linux创建账号HDFS
   1. useradd hdfs
   2. passwd hdfs
   3. 密码填入123456
   4. 可以使用userdel 删除账号
2. 修改/home/hadoop 权限为755 chmod –R 755 /home/hadoop
3. 切换到hdfs 账号下，尝试在hdfs中创建文件
4. 使用hadoop账号创建给hdfs用户使用的文件夹，hdfs dfs -mkdir /hdfs
5. 修改用户：hdfs dfs -chown hdfs /hdfs，并赋予700权限
6. 在使用hdfs账号像该文件夹放入文件，并尝试修改其他文件夹的权限

# 7. 上网日志分析MapReduce高级操作练习

## 7.1 练习目的：

通过本练习，学员可以掌握MapReduce基本编程与进阶使用。

## 7.2 练习案例说明：

数据使用练习1的数据进行，数据包含四个字段

手机号，url，访问的时间，产生的流量

## 7.3 练习步骤:

### 7.3.1 编写数据清洗脚本

要求：清洗出手机号长度超过13位，流量小于20字节的访问数据，并将清洗后的数据保存到/logtest文件夹中

### 7.3.2 编写数据统计脚本

要求

1. 统计URL PV
2. 统计每个手机号流量的汇总

### 7.3.3 数据统计脚本进阶－二次排序，序列化

要求：

1. 按照URL PV的降序进行结果输出
2. 增加logwritable 对象，按照流量比较对象的大小

### 7.3.4 数据统计脚本进阶－增加combiner

要求：

1. 在练习7.3.2 中url pv，增加combiner

### 7.3.5 数据统计脚本进阶－增加partitioner

要求：

1. 在练习7.3.2 中url pv，增加partitioner
   1. 设置reduce个数为4
   2. 将包含百度url的数据进入第一个reduce，sohu进入第二个，sina进入第三个，其余进入第四个

# 8. MapReduce 管理操作练习

## 8.1 练习目的：

通过本练习，学员可以掌握MapReduce管理编程进阶使用。

## 8.2 练习案例说明：

学习jobhistory ，mapred管理命令使用

mapred

Usage: mapred [--config confdir] COMMAND

where COMMAND is one of:

pipes run a Pipes job

job manipulate MapReduce jobs

queue get information regarding JobQueues

classpath prints the class path needed for running

mapreduce subcommands

historyserver run job history servers as a standalone daemon

distcp <srcurl> <desturl> copy file or directories recursively

archive -archiveName NAME -p <parent path> <src>\* <dest> create a hadoop archive

hsadmin job history server admin interface

## 8.3 练习步骤:

1. 启动JobhistoryServer，查看19888运行的mr日志

mr-jobhistory-daemon.sh start historyserver

http://localhost:19888

1. mapred queue 获取当前队列的信息
2. mapred job 杀掉，作业等练习

# 9. Hive 上网日志分析系统练习

## 9.1 练习目的：

通过本练习，学员可以掌握Hive高级管理使用。

## 9.2 练习案例说明：

Hive作为Hadoop上的的数据仓库工具，需要掌握基本的建表，使用等

上网日志分析中，使用Hive作为日志分析的基础

## 9.3 练习步骤:

1. 建立url pv 统计分析表，分区为日，分隔符为, 字段为 url,pv
2. 建立原始log表，分区为日，分隔符为， 字段为 mobile，url，starttime，traffic
3. 建立mobile 流量统计表，分区为日，分隔符为，字段为mobile，trafficsum
4. 将数据导入log
5. 计算url的pv，与手机号的流量
6. 编辑sql 执行脚本，log.sql
7. 使用hive ‘-f’ 加载该sql语句

# 10. Hive 上网日志分析系统进阶练习－表格式

## 10.1 练习目的：

通过本练习，学员可以掌握Hive表格式使用。

## 10.2 练习案例说明：

Hive作为Hadoop上的的数据仓库工具，需要掌握高级表结构使用，包括orcfile,parquet

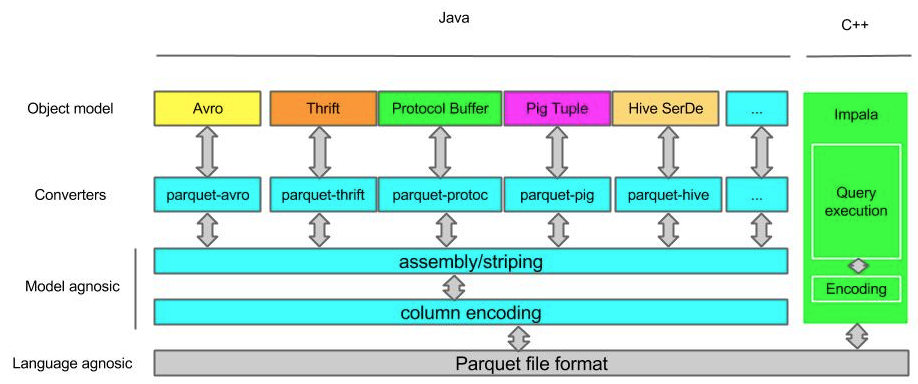
Parquet是语言无关的，而且不与任何一种数据处理框架绑定在一起，适配多种语言和组件，能够与Parquet配合的组件有：

查询引擎: Hive, Impala, Pig, Presto, Drill, Tajo, HAWQ, IBM Big SQL

计算框架: MapReduce, Spark, Cascading, Crunch, Scalding, Kite

数据模型: Avro, Thrift, Protocol Buffers, POJOs

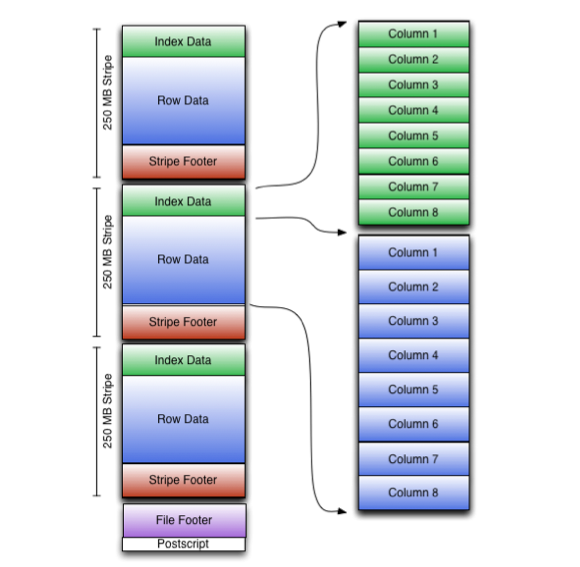
parquet 项目结构：



ORC File包含一组组的行数据，称为stripes，除此之外，ORC File的file footer还包含一些额外的辅助信息。在ORC File文件的最后，有一个被称为postscript的区，它主要是用来存储压缩参数及压缩页脚的大小。

在默认情况下，一个stripe的大小为250MB。大尺寸的stripes使得从HDFS读数据更高效。

在file footer里面包含了该ORC File文件中stripes的信息，每个stripe中有多少行，以及每列的数据类型。当然，它里面还包含了列级别的一些聚合的结果，比如：count, min, max, and sum。这里贴一下文档上的图



Stripe结构

每个Stripe都包含index data、row data以及stripe footer，Stripe footer包含流位置的目录，Row data在表扫描的时候会用到。

Index data包含每列的最大和最小值以及每列所在的行。行索引里面提供了偏移量，它可以跳到正确的压缩块位置。

通过行索引，可以在stripe中快速读取的过程中可以跳过很多行，尽管这个stripe的大小很大。在默认情况下，最大可以跳过10000行。

因为可以通过过滤预测跳过很多行，因而可以在表的 secondary keys 进行排序，从而可以大幅减少执行时间。比如你的表的主分区是交易日期，那么你可以对次分区（state、zip code以及last name）进行排序。

## 10.3 练习步骤:

1. 建立url pv\_orc表以orcfile格式

create external table orcfile (

url string,

pv bigint

) partitioned by (day date)

stored as orc

select \* from urlpv where day =’20160510’ order by pv limit 10

create external table orcfiletest (

url string,

pv bigint

)

1. 将urlpv数据导入该表，比较两个表的分析速度
2. 以parquet 建立urlpv\_par
3. 将urlpv数据导入该表，比较两个表的分析速度

# 11. Hive 上网日志分析系统进阶练习－自定义函数

## 11.1 练习目的：

通过本练习，学员可以掌握Hive自定义函数开发。

## 11.2 练习案例说明：

Hive作为Hadoop上的的数据仓库工具，需要掌握自定义函数UDF使用

## 11.3 练习步骤:

1. 建立udf函数，urlToDomain 将url转化成域名

add jar /home/hadoop/jar

create temporary function helloworld as 'hiveudf.UrlToDomain';

# 12. HBase 上网日志分析系统进阶练习－表结构设计

## 12.1 练习目的：

通过本练习，学员可以掌握HBase使用开发。

## 12.2 练习案例说明：

HBase作为Hadoop上的的NoSQL工具，需要掌握其开发与使用

## 12.3 练习步骤:

1. 建立表，url-keyword, rowkey为url,column 为keyword
2. 用户画像表建立
   1. rowkey 为用户的id
   2. 列族为tags

# 13. HBase 上网日志分析系统进阶练习-bulkload, Coprocessor等

## 13.1 练习目的：

通过本练习，学员可以掌握HBase高级开放使用开发。

## 13.2 练习案例说明：

HBase作为Hadoop上的的NoSQL工具，需要掌握其开发与使用

## 13.3 练习步骤:

1. 使用bulkload脚本导入数据
2. 创建触发器Coprocessor，用户数据插入数据时，如果插入的rowkey为test，输出hello world

alter 'kora','coprocessor' => '|org.apache.hbase.kora.coprocessor.RegionObserverExample|'

hdfs:///foo.jar|com.foo.FooRegionObserver

# 14. Flume NG 上网日志分析系统进阶练习导入HDFS数据

## 14.1 练习目的：

通过本练习，学员可以掌握Flume高级开放使用开发。

## 14.2 练习案例说明：

Flume作为Hadoop上的的数据采集工具，需要掌握其开发与使用

以下是flume 采集数据到hdfs的一个参考

agent1.sources = spooldirSource

agent1.channels = fileChannel

agent1.sinks = hdfsSink

agent1.sources.spooldirSource.type=spooldir

agent1.sources.spooldirSource.spoolDir=/data/lwq/new\_log

agent1.sources.spooldirSource.channels=fileChannel

agent1.sinks.hdfsSink.type=hdfs

agent1.sinks.hdfsSink.hdfs.path=hdfs://dev228:8020/raw/lwq/%y-%m-%d

agent1.sinks.hdfsSink.hdfs.filePrefix=lwq

agent1.sinks.sink1.hdfs.round = true

# Number of seconds to wait before rolling current file (0 = never roll based on time interval)

agent1.sinks.hdfsSink.hdfs.rollInterval = 3600

# File size to trigger roll, in bytes (0: never roll based on file size)

agent1.sinks.hdfsSink.hdfs.rollSize = 128000000

agent1.sinks.hdfsSink.hdfs.rollCount = 0

agent1.sinks.hdfsSink.hdfs.batchSize = 1000

#Rounded down to the highest multiple of this (in the unit configured using hdfs.roundUnit), less than current time.

agent1.sinks.hdfsSink.hdfs.roundValue = 1

agent1.sinks.hdfsSink.hdfs.roundUnit = minute

agent1.sinks.hdfsSink.hdfs.useLocalTimeStamp = true

agent1.sinks.hdfsSink.channel=fileChannel

agent1.sinks.hdfsSink.hdfs.fileType = DataStream

agent1.channels.fileChannel.type = file

agent1.channels.fileChannel.checkpointDir=/usr/share/apache-flume-1.5.0-bin/checkpoint

agent1.channels.fileChannel.dataDirs=/usr/share/apache-flume-1.5.0-bin/dataDir

启动命令

${FLUME\_HOME}/bin/flume-ng agent --conf ./conf/ -f conf/flume-site.xml -Dflume.root.logger=DEBUG,console -n agent1 > log.log 2>&1 &

## 14.3 练习步骤:

1. 采集log的本地数据进入hdfs

Flume NG 配置说明

# 15. Flume NG 上网日志分析系统进阶练习－Plugin 开发

## 15.1 练习目的：

通过练习掌握Flume NG插件开发方法

## 15.2 练习案例说明：

以下为自定义sink的例子

import com.google.common.base.Preconditions;

import com.google.common.base.Throwables;

import com.google.common.collect.Lists;

import org.apache.flume.\*;

import org.apache.flume.conf.Configurable;

import org.apache.flume.sink.AbstractSink;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.List;

public class MysqlSink extends AbstractSink implements Configurable {

private Logger LOG = LoggerFactory.getLogger(MysqlSink.class);

private String hostname;

private String port;

private String databaseName;

private String tableName;

private String user;

private String password;

private PreparedStatement preparedStatement;

private Connection conn;

private int batchSize;

public MysqlSink() {

LOG.info("MysqlSink start...");

}

@Override

public void configure(Context context) {

hostname = context.getString("hostname");

Preconditions.checkNotNull(hostname, "hostname must be set!!");

port = context.getString("port");

Preconditions.checkNotNull(port, "port must be set!!");

databaseName = context.getString("databaseName");

Preconditions.checkNotNull(databaseName, "databaseName must be set!!");

tableName = context.getString("tableName");

Preconditions.checkNotNull(tableName, "tableName must be set!!");

user = context.getString("user");

Preconditions.checkNotNull(user, "user must be set!!");

password = context.getString("password");

Preconditions.checkNotNull(password, "password must be set!!");

batchSize = context.getInteger("batchSize", 100);

Preconditions.checkNotNull(batchSize > 0, "batchSize must be a positive number!!");

}

@Override

public void start() {

super.start();

try {

//调用Class.forName()方法加载驱动程序

Class.forName("com.mysql.jdbc.Driver");

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

String url = "jdbc:mysql://" + hostname + ":" + port + "/" + databaseName;

//调用DriverManager对象的getConnection()方法，获得一个Connection对象

try {

conn = DriverManager.getConnection(url, user, password);

conn.setAutoCommit(false);

//创建一个Statement对象

preparedStatement = conn.prepareStatement("insert into " + tableName +

" (content) values (?)");

} catch (SQLException e) {

e.printStackTrace();

System.exit(1);

}

}

@Override

public void stop() {

super.stop();

if (preparedStatement != null) {

try {

preparedStatement.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if (conn != null) {

try {

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

@Override

public Status process() throws EventDeliveryException {

Status result = Status.READY;

Channel channel = getChannel();

Transaction transaction = channel.getTransaction();

Event event;

String content;

List<String> actions = Lists.newArrayList();

transaction.begin();

try {

for (int i = 0; i < batchSize; i++) {

event = channel.take();

if (event != null) {

content = new String(event.getBody());

actions.add(content);

} else {

result = Status.BACKOFF;

break;

}

}

if (actions.size() > 0) {

preparedStatement.clearBatch();

for (String temp : actions) {

preparedStatement.setString(1, temp);

preparedStatement.addBatch();

}

preparedStatement.executeBatch();

conn.commit();

}

transaction.commit();

} catch (Throwable e) {

try {

transaction.rollback();

} catch (Exception e2) {

LOG.error("Exception in rollback. Rollback might not have been" +

"successful.", e2);

}

LOG.error("Failed to commit transaction." +

"Transaction rolled back.", e);

Throwables.propagate(e);

} finally {

transaction.close();

}

return result;

}

}

## 15.3 练习步骤说明：

# 16. Sqoop 上网日志分析系统进阶练习－从mysql中导入数据到hdfs

## 16.1 练习目的：

通过Sqoop基本的使用，掌握其基本用法

## 16.2 练习案例说明：

将MySQL数据导出到HDFS

sqoop import -m 1 --connect jdbc:mysql://<HOST>:<PORT>/logs--username hdp\_usr --password test1 --table weblogs --target-dir /data/weblogs/import

1.Common arguments通用参数,主要是针对关系型数据库链接的一些参数

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 参数 | 说明 | 样例 |
| 1 | connect | 连接关系型数据库的URL | jdbc:**[MySQL](http://lib.csdn.net/base/14" \o "MySQL知识库" \t "_blank)**://localhost/sqoop\_datas |
| 2 | connection-manager | 连接管理类,一般不用 |  |
| 3 | driver | 连接驱动 |  |
| 4 | hadoop-home | hadoop目录 | /home/hadoop |
| 5 | help | 查看帮助信息 |  |
| 6 | password | 连接关系型数据库的密码 |  |
| 7 | username | 链接关系型数据库的用户名 |  |
| 8 | verbose | 查看更多的信息,其实是将日志级别调低 | 该参数后面不接值 |

**Importcontrol arguments:**

| **Argument** | **Description** |
| --- | --- |
| --append | Append data to an existing dataset in HDFS |
| --as-avrodatafile | Imports data to Avro Data Files |
| --as-sequencefile | Imports data to SequenceFiles |
| --as-textfile | Imports data as plain text (default) |
| --boundary-query <statement> | Boundary query to use for creating splits |
| --columns <col,col,col…> | Columns to import from table |
| --direct | Use direct import fast path |
| --direct-split-size <n> | Split the input stream every n bytes when importing in direct mode |
| --inline-lob-limit <n> | Set the maximum size for an inline LOB |
| -m,--num-mappers <n> | Use n map tasks to import in parallel |
| -e,--query <statement> | Import the results of statement. |
| --split-by <column-name> | Column of the table used to split work units |
| --table <table-name> | Table to read |
| --target-dir <dir> | HDFS destination dir |
| --warehouse-dir <dir> | HDFS parent for table destination |
| --where <where clause> | WHERE clause to use during import |
| -z,--compress | Enable compression |
| --compression-codec <c> | Use Hadoop codec (default gzip) |
| --null-string <null-string> | The string to be written for a null value for string columns |
| --null-non-string <null-string> | The string to be written for a null value for non-string columns |

**Incrementalimport arguments:**

| **Argument** | **Description** |
| --- | --- |
| --check-column (col) | Specifies the column to be examined when determining which rows to import. |
| --incremental (mode) | Specifies how Sqoop determines which rows are new. Legal values for mode include append and lastmodified. |
| --last-value (value) | Specifies the maximum value of the check column from the previous import. |

**Output lineformatting arguments:**

| **Argument** | **Description** |
| --- | --- |
| --enclosed-by <char> | Sets a required field enclosing character |
| --escaped-by <char> | Sets the escape character |
| --fields-terminated-by <char> | Sets the field separator character |
| --lines-terminated-by <char> | Sets the end-of-line character |
| --mysql-delimiters | Uses MySQL’s default delimiter set: fields: , lines: \n escaped-by: \ optionally-enclosed-by: ' |
| --optionally-enclosed-by <char> | Sets a field enclosing character |

**Hivearguments:**

| **Argument** | **Description** |
| --- | --- |
| --hive-home <dir> | Override $HIVE\_HOME |
| --hive-import | Import tables into Hive (Uses Hive’s default delimiters if none are set.) |
| --hive-overwrite | Overwrite existing data in the Hive table. |
| --create-hive-table | If set, then the job will fail if the target hive |
|  | table exits. By default this property is false. |
| --hive-table <table-name> | Sets the table name to use when importing to Hive. |
| --hive-drop-import-delims | Drops \n, \r, and \01 from string fields when importing to Hive. |
| --hive-delims-replacement | Replace \n, \r, and \01 from string fields with user defined string when importing to Hive. |
| --hive-partition-key | Name of a hive field to partition are sharded on |
| --hive-partition-value <v> | String-value that serves as partition key for this imported into hive in this job. |
| --map-column-hive <map> | Override default mapping from SQL type to Hive type for configured columns. |

**HBasearguments:**

| **Argument** | **Description** |
| --- | --- |
| --column-family <family> | Sets the target column family for the import |
| --hbase-create-table | If specified, create missing HBase tables |
| --hbase-row-key <col> | Specifies which input column to use as the row key |
| --hbase-table <table-name> | Specifies an HBase table to use as the target instead of HDFS |

**Codegeneration arguments:**

| **Argument** | **Description** |
| --- | --- |
| --bindir <dir> | Output directory for compiled objects |
| --class-name <name> | Sets the generated class name. This overrides --package-name. When combined with --jar-file, sets the input class. |
| --jar-file <file> | Disable code generation; use specified jar |
| --outdir <dir> | Output directory for generated code |
| --package-name <name> | Put auto-generated classes in this package |
| --map-column-java <m> | Override default mapping from SQL type to Java type for configured columns. |

## 16.3 练习案例说明：

# 17. Zookeper 上网日志分析系统进阶练习－开发一个进程状态保存的案例

## 17.1 练习目的：

通过课程，学习掌握如何进行Zookeeper的开发，掌握一般使用Zookeeper的方法

## 17.2 练习案例说明：

Zookeeper API Java 实例

public class ZookeeperTest {

private static final int SESSION\_TIMEOUT = 30000;

public static final Logger LOGGER = LoggerFactory.getLogger(ZookeeperTest.class);

private Watcher watcher = new Watcher() {

public void process(WatchedEvent event) {

LOGGER.info("process : " + event.getType());

}

};

private ZooKeeper zooKeeper;

/\*\*

\* 连接zookeeper

\* <br>------------------------------<br>

\* @throws IOException

\*/

@Before

public void connect() throws IOException {

zooKeeper = new ZooKeeper("localhost:2181,localhost:2182,localhost:2183", SESSION\_TIMEOUT, watcher);

}

/\*\*

\* 关闭连接

\* <br>------------------------------<br>

\*/

@After

public void close() {

try {

zooKeeper.close();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

/\*\*

\* 创建一个znode

\* 1.CreateMode 取值

\* PERSISTENT：持久化，这个目录节点存储的数据不会丢失

\* PERSISTENT\_SEQUENTIAL：顺序自动编号的目录节点，这种目录节点会根据当前已近存在的节点数自动加 1，然后返回给客户端已经成功创建的目录节点名；

\* EPHEMERAL：临时目录节点，一旦创建这个节点的客户端与服务器端口也就是 session过期超时，这种节点会被自动删除

\* EPHEMERAL\_SEQUENTIAL：临时自动编号节点

\* <br>------------------------------<br>

\*/

@Test

public void testCreate() {

String result = null;

try {

result = zooKeeper.create("/zk001", "zk001data".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

LOGGER.info("create result : {}", result);

}

/\*\*

\* 删除节点 忽略版本

\* <br>------------------------------<br>

\*/

@Test

public void testDelete() {

try {

zooKeeper.delete("/zk001", -1);

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

}

/\*\*

\* 获取数据

\* <br>------------------------------<br>

\*/

@Test

public void testGetData() {

String result = null;

try {

byte[] bytes = zooKeeper.getData("/zk001", null, null);

result = new String(bytes);

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

LOGGER.info("getdata result : {}", result);

}

/\*\*

\* 获取数据 设置watch

\* <br>------------------------------<br>

\*/

@Test

public void testGetDataWatch() {

String result = null;

try {

byte[] bytes = zooKeeper.getData("/zk001", new Watcher() {

public void process(WatchedEvent event) {

LOGGER.info("testGetDataWatch watch : {}", event.getType());

}

}, null);

result = new String(bytes);

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

LOGGER.info("getdata result : {}", result);

// 触发wacth NodeDataChanged

try {

zooKeeper.setData("/zk001", "testSetData".getBytes(), -1);

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

}

/\*\*

\* 判断节点是否存在

\* 设置是否监控这个目录节点，这里的 watcher 是在创建 ZooKeeper实例时指定的 watcher

\* <br>------------------------------<br>

\*/

@Test

public void testExists() {

Stat stat = null;

try {

stat = zooKeeper.exists("/zk001", false);

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

Assert.assertNotNull(stat);

LOGGER.info("exists result : {}", stat.getCzxid());

}

/\*\*

\* 设置对应znode下的数据 , -1表示匹配所有版本

\* <br>------------------------------<br>

\*/

@Test

public void testSetData() {

Stat stat = null;

try {

stat = zooKeeper.setData("/zk001", "testSetData".getBytes(), -1);

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

Assert.assertNotNull(stat);

LOGGER.info("exists result : {}", stat.getVersion());

}

/\*\*

\* 判断节点是否存在,

\* 设置是否监控这个目录节点，这里的 watcher 是在创建 ZooKeeper实例时指定的 watcher

\* <br>------------------------------<br>

\*/

@Test

public void testExistsWatch1() {

Stat stat = null;

try {

stat = zooKeeper.exists("/zk001", true);

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

Assert.assertNotNull(stat);

try {

zooKeeper.delete("/zk001", -1);

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 判断节点是否存在,

\* 设置监控这个目录节点的 Watcher

\* <br>------------------------------<br>

\*/

@Test

public void testExistsWatch2() {

Stat stat = null;

try {

stat = zooKeeper.exists("/zk002", new Watcher() {

@Override

public void process(WatchedEvent event) {

LOGGER.info("testExistsWatch2 watch : {}", event.getType());

}

});

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

Assert.assertNotNull(stat);

// 触发watch 中的process方法 NodeDataChanged

try {

zooKeeper.setData("/zk002", "testExistsWatch2".getBytes(), -1);

} catch (Exception e) {

e.printStackTrace();

}

// 不会触发watch 只会触发一次

try {

zooKeeper.delete("/zk002", -1);

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 获取指定节点下的子节点

\* <br>------------------------------<br>

\*/

@Test

public void testGetChild() {

try {

zooKeeper.create("/zk/001", "001".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

zooKeeper.create("/zk/002", "002".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

List<String> list = zooKeeper.getChildren("/zk", true);

for (String node : list) {

LOGGER.info("node {}", node);

}

} catch (Exception e) {

LOGGER.error(e.getMessage());

Assert.fail();

}

}

}

## 17.3 练习案例说明：

## 17.3 练习步骤：