尚硅谷大数据项目之在线教育-用户行为分析

版本：V1.0（cdh版）

# 第1章 前置知识

## 数据团队组织结构

数据平台负责人

数据平台架构师

大数据开发（ETL工程师、数据仓库工程师）

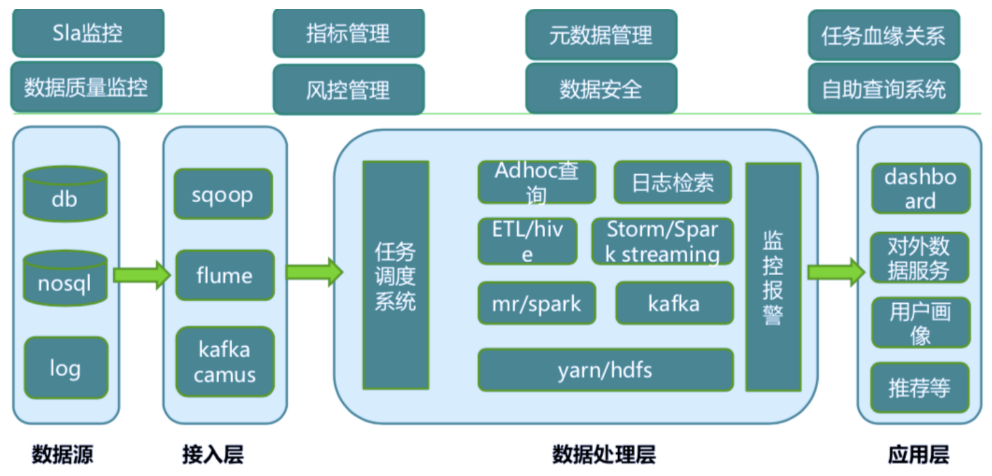
数据挖掘工程师

推荐系统

深度学习/AI工程师

BI工程师

## 一般企业大数架构图



## 从零开始搭建大数据集群

（1）思考：

如何确认集群规模？假设每台服务器8T硬盘

使用Apache/CDH/HDP版本？

服务器使用物理机还是云主机

大数据服务组件规划（出表）

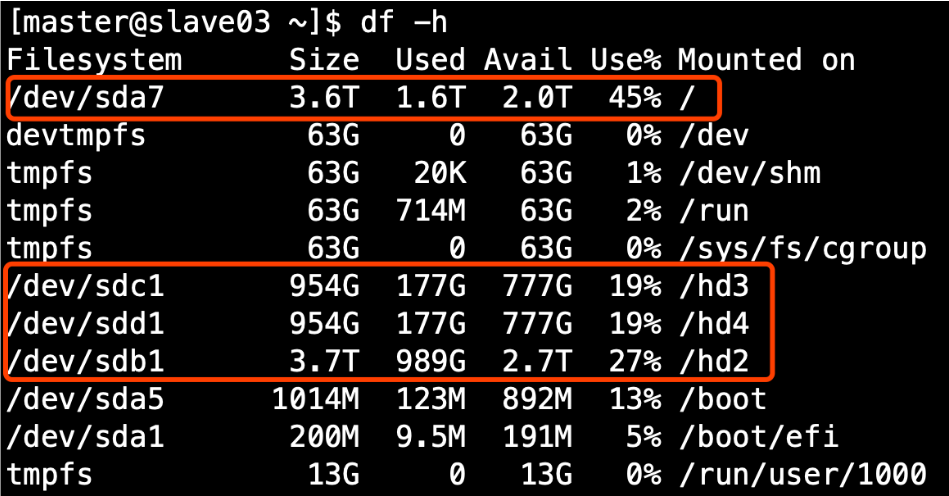
（2）安装过程中注意事项

a) 尽量使用离线方式安装

b)使用非root用户，配置免密码的sudo权限

c) 确认HDFS的存储目录，保证存储在空间最大硬盘上

hdfs存储参数：dfs.datanode.data.dir，多硬盘用逗号



d) 元数据备份（重点，如数据损坏，可能整个集群无法运行，至少要保证每日零点之后备份到其它服务器两个复本）

e) 基准测试

1） 测试HDFS写性能

[atguigu@hadoop102 mapreduce]$ hadoop jar hadoop-mapreduce-client-jobclient-2.7.2-tests.jar TestDFSIO -write -nrFiles 10 -fileSize 128MB

19/05/02 11:44:26 INFO fs.TestDFSIO: TestDFSIO.1.8

19/05/02 11:44:26 INFO fs.TestDFSIO: nrFiles = 10

19/05/02 11:44:26 INFO fs.TestDFSIO: nrBytes (MB) = 128.0

19/05/02 11:44:26 INFO fs.TestDFSIO: bufferSize = 1000000

19/05/02 11:44:26 INFO fs.TestDFSIO: baseDir = /benchmarks/TestDFSIO

19/05/02 11:44:28 INFO fs.TestDFSIO: creating control file: 134217728 bytes, 10 files

19/05/02 11:44:30 INFO fs.TestDFSIO: created control files for: 10 files

19/05/02 11:44:30 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.1.103:8032

19/05/02 11:44:31 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.1.103:8032

19/05/02 11:44:32 INFO mapred.FileInputFormat: Total input paths to process : 10

19/05/02 11:44:32 INFO mapreduce.JobSubmitter: number of splits:10

19/05/02 11:44:33 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1556766549220\_0003

19/05/02 11:44:34 INFO impl.YarnClientImpl: Submitted application application\_1556766549220\_0003

19/05/02 11:44:34 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application\_1556766549220\_0003/

19/05/02 11:44:34 INFO mapreduce.Job: Running job: job\_1556766549220\_0003

19/05/02 11:44:47 INFO mapreduce.Job: Job job\_1556766549220\_0003 running in uber mode : false

19/05/02 11:44:47 INFO mapreduce.Job: map 0% reduce 0%

19/05/02 11:45:05 INFO mapreduce.Job: map 13% reduce 0%

19/05/02 11:45:06 INFO mapreduce.Job: map 27% reduce 0%

19/05/02 11:45:08 INFO mapreduce.Job: map 43% reduce 0%

19/05/02 11:45:09 INFO mapreduce.Job: map 60% reduce 0%

19/05/02 11:45:10 INFO mapreduce.Job: map 73% reduce 0%

19/05/02 11:45:15 INFO mapreduce.Job: map 77% reduce 0%

19/05/02 11:45:18 INFO mapreduce.Job: map 87% reduce 0%

19/05/02 11:45:19 INFO mapreduce.Job: map 100% reduce 0%

19/05/02 11:45:21 INFO mapreduce.Job: map 100% reduce 100%

19/05/02 11:45:22 INFO mapreduce.Job: Job job\_1556766549220\_0003 completed successfully

19/05/02 11:45:22 INFO mapreduce.Job: Counters: 51

File System Counters

FILE: Number of bytes read=856

FILE: Number of bytes written=1304826

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=2350

HDFS: Number of bytes written=1342177359

HDFS: Number of read operations=43

HDFS: Number of large read operations=0

HDFS: Number of write operations=12

Job Counters

Killed map tasks=1

Launched map tasks=10

Launched reduce tasks=1

Data-local map tasks=8

Rack-local map tasks=2

Total time spent by all maps in occupied slots (ms)=263635

Total time spent by all reduces in occupied slots (ms)=9698

Total time spent by all map tasks (ms)=263635

Total time spent by all reduce tasks (ms)=9698

Total vcore-milliseconds taken by all map tasks=263635

Total vcore-milliseconds taken by all reduce tasks=9698

Total megabyte-milliseconds taken by all map tasks=269962240

Total megabyte-milliseconds taken by all reduce tasks=9930752

Map-Reduce Framework

Map input records=10

Map output records=50

Map output bytes=750

Map output materialized bytes=910

Input split bytes=1230

Combine input records=0

Combine output records=0

Reduce input groups=5

Reduce shuffle bytes=910

Reduce input records=50

Reduce output records=5

Spilled Records=100

Shuffled Maps =10

Failed Shuffles=0

Merged Map outputs=10

GC time elapsed (ms)=17343

CPU time spent (ms)=96930

Physical memory (bytes) snapshot=2821341184

Virtual memory (bytes) snapshot=23273218048

Total committed heap usage (bytes)=2075656192

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=1120

File Output Format Counters

Bytes Written=79

19/05/02 11:45:23 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write

19/05/02 11:45:23 INFO fs.TestDFSIO: Date & time: Thu May 02 11:45:23 CST 2019

19/05/02 11:45:23 INFO fs.TestDFSIO: Number of files: 10

19/05/02 11:45:23 INFO fs.TestDFSIO: Total MBytes processed: 1280.0

19/05/02 11:45:23 INFO fs.TestDFSIO: Throughput mb/sec: 10.69751115716984

19/05/02 11:45:23 INFO fs.TestDFSIO: Average IO rate mb/sec: 14.91699504852295

19/05/02 11:45:23 INFO fs.TestDFSIO: IO rate std deviation: 11.160882132355928

19/05/02 11:45:23 INFO fs.TestDFSIO: Test exec time sec: 52.315

2）测试HDFS读性能

[atguigu@hadoop102 mapreduce]$ hadoop jar hadoop-mapreduce-client-jobclient-2.7.2-tests.jar TestDFSIO -read -nrFiles 10 -fileSize 128MB

19/05/02 11:55:42 INFO fs.TestDFSIO: TestDFSIO.1.8

19/05/02 11:55:42 INFO fs.TestDFSIO: nrFiles = 10

19/05/02 11:55:42 INFO fs.TestDFSIO: nrBytes (MB) = 128.0

19/05/02 11:55:42 INFO fs.TestDFSIO: bufferSize = 1000000

19/05/02 11:55:42 INFO fs.TestDFSIO: baseDir = /benchmarks/TestDFSIO

19/05/02 11:55:45 INFO fs.TestDFSIO: creating control file: 134217728 bytes, 10 files

19/05/02 11:55:47 INFO fs.TestDFSIO: created control files for: 10 files

19/05/02 11:55:47 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.1.103:8032

19/05/02 11:55:48 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.1.103:8032

19/05/02 11:55:49 INFO mapred.FileInputFormat: Total input paths to process : 10

19/05/02 11:55:49 INFO mapreduce.JobSubmitter: number of splits:10

19/05/02 11:55:49 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1556766549220\_0004

19/05/02 11:55:50 INFO impl.YarnClientImpl: Submitted application application\_1556766549220\_0004

19/05/02 11:55:50 INFO mapreduce.Job: The url to track the job: http://hadoop103:8088/proxy/application\_1556766549220\_0004/

19/05/02 11:55:50 INFO mapreduce.Job: Running job: job\_1556766549220\_0004

19/05/02 11:56:04 INFO mapreduce.Job: Job job\_1556766549220\_0004 running in uber mode : false

19/05/02 11:56:04 INFO mapreduce.Job: map 0% reduce 0%

19/05/02 11:56:24 INFO mapreduce.Job: map 7% reduce 0%

19/05/02 11:56:27 INFO mapreduce.Job: map 23% reduce 0%

19/05/02 11:56:28 INFO mapreduce.Job: map 63% reduce 0%

19/05/02 11:56:29 INFO mapreduce.Job: map 73% reduce 0%

19/05/02 11:56:30 INFO mapreduce.Job: map 77% reduce 0%

19/05/02 11:56:31 INFO mapreduce.Job: map 87% reduce 0%

19/05/02 11:56:32 INFO mapreduce.Job: map 100% reduce 0%

19/05/02 11:56:35 INFO mapreduce.Job: map 100% reduce 100%

19/05/02 11:56:36 INFO mapreduce.Job: Job job\_1556766549220\_0004 completed successfully

19/05/02 11:56:36 INFO mapreduce.Job: Counters: 51

File System Counters

FILE: Number of bytes read=852

FILE: Number of bytes written=1304796

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=1342179630

HDFS: Number of bytes written=78

HDFS: Number of read operations=53

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Killed map tasks=1

Launched map tasks=10

Launched reduce tasks=1

Data-local map tasks=8

Rack-local map tasks=2

Total time spent by all maps in occupied slots (ms)=233690

Total time spent by all reduces in occupied slots (ms)=7215

Total time spent by all map tasks (ms)=233690

Total time spent by all reduce tasks (ms)=7215

Total vcore-milliseconds taken by all map tasks=233690

Total vcore-milliseconds taken by all reduce tasks=7215

Total megabyte-milliseconds taken by all map tasks=239298560

Total megabyte-milliseconds taken by all reduce tasks=7388160

Map-Reduce Framework

Map input records=10

Map output records=50

Map output bytes=746

Map output materialized bytes=906

Input split bytes=1230

Combine input records=0

Combine output records=0

Reduce input groups=5

Reduce shuffle bytes=906

Reduce input records=50

Reduce output records=5

Spilled Records=100

Shuffled Maps =10

Failed Shuffles=0

Merged Map outputs=10

GC time elapsed (ms)=6473

CPU time spent (ms)=57610

Physical memory (bytes) snapshot=2841436160

Virtual memory (bytes) snapshot=23226683392

Total committed heap usage (bytes)=2070413312

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=1120

File Output Format Counters

Bytes Written=78

19/05/02 11:56:36 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read

19/05/02 11:56:36 INFO fs.TestDFSIO: Date & time: Thu May 02 11:56:36 CST 2019

19/05/02 11:56:36 INFO fs.TestDFSIO: Number of files: 10

19/05/02 11:56:36 INFO fs.TestDFSIO: Total MBytes processed: 1280.0

19/05/02 11:56:36 INFO fs.TestDFSIO: Throughput mb/sec: 16.001000062503905

19/05/02 11:56:36 INFO fs.TestDFSIO: Average IO rate mb/sec: 17.202795028686523

19/05/02 11:56:36 INFO fs.TestDFSIO: IO rate std deviation: 4.881590515873911

19/05/02 11:56:36 INFO fs.TestDFSIO: Test exec time sec: 49.116

19/05/02 11:56:36 INFO fs.TestDFSIO:

3）删除测试生成数据

[atguigu@hadoop102 mapreduce]$ hadoop jar hadoop-mapreduce-client-jobclient-2.7.2-tests.jar TestDFSIO -clean

4）使用Sort程序评测MapReduce

（1）使用RandomWriter来产生随机数，每个节点运行10个map任务，每个map产生大约1G大小的二进制随机数

[atguigu@hadoop102 mapreduce]$ hadoop jar hadoop-mapreduce-examples-2.7.2.jar randomwriter random-data

（2）执行Sort程序

[atguigu@hadoop102 mapreduce]$ hadoop jar hadoop-mapreduce-examples-2.7.2.jar sort random-data sorted-data

（3）验证数据是否真正排好序了

[atguigu@hadoop102 mapreduce]$ hadoop jar hadoop-mapreduce-examples-2.7.2.jar testmapredsort -sortInput random-data -sortOutput sorted-data

f) HDFS参数调优hdfs-site.xml

1. dfs.namenode.handler.count=20 \* log2(Cluster Size)，NN启动后展开的线程数，比如集群规模为10台时，此参数设置为60

The Hadoop RPC server consists of a single RPC queue per port and multiple handler (worker) threads that dequeue and process requests. If the number of handlers is insufficient, then the RPC queue starts building up and eventually overflows. You may start seeing task failures and eventually job failures and unhappy users. It is recommended that the RPC handler count be set to 20 \* log2(Cluster Size) with an upper limit of 200.

1. dfs.namenode.service.handler.count=上面参数的一半

There is no precise calculation for the Service RPC handler count however the default value of 10 is too low for most production clusters. We have often seen this initialized to 50% of the dfs.namenode.handler.count in busy clusters and this value works well in practice.

1. dfs.namenode.edits.dir设置与dfs.namenode.name.dir尽量分开，达到最低写入延迟
2. dfs.namenode.accesstime.precision=0，HDFS文件的访问时间精确值。 默认值是1小时。 设置的值为0禁用HDFS的访问时间

The setting dfs.namenode.accesstime.precision controls how often the NameNode will update the last accessed time for each file. It is specified in milliseconds. If this value is too low, then the NameNode is forced to write an edit log transaction to update the file's last access time for each read request and its performance will suffer.  
The default value of this setting is a reasonable 3600000 milliseconds (1 hour). We recommend going one step further and setting it to zero so last access time updates are turned off. Add the following to your hdfs-site.xml.

# 第2章 项目需求及架构设计

## 2.1 项目需求分析

需求：

(1) 数据采集平台搭建

(2) 数据清洗、加载

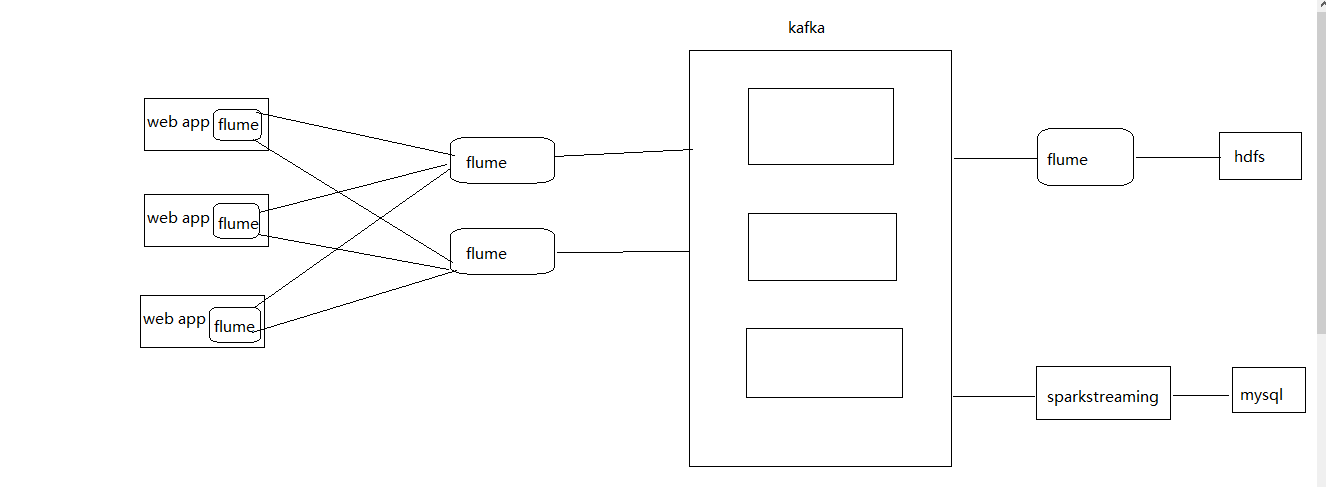
(3) 进行版本、渠道、访问频次、漏斗、7日留存等指标分析

## 2.2 项目框架

### 2.2.1 技术选型

* 数据采集传输：Flume，Sqoop，Kafka，Logstash，DataX
* 数据存储：HDFS，Mysql，HBase，Redis，MongoDB
* 数据清洗：Spark，Hive，Tez，Flink，Storm
* 指标统计：Hive，SparkSQL，Impala，Kylin，Druid，Clickhouse

### 2.2.2 系统数据流程设计



### 2.2.4 框架版本选型

|  |  |
| --- | --- |
| 产品 | 版本 |
| Hadoop | 2.7.2 |
| Flume | 1.7.0 |
| Hive | 1.2.1 |
| Sqoop | 1.4.6 |
| MySQL | 5.6.24 |
| Azkaban | 2.5.0 |
| Java | 1.8 |
| Zookeeper | 3.4.10 |

注意事项：框架选型尽量不要选择最新的框架，选择最新框架半年前左右的稳定版。

### 2.2.5 集群资源规划设计

|  |  |  |  |
| --- | --- | --- | --- |
|  | 服务器hadoop102 | 服务器hadoop103 | 服务器hadoop104 |
| HDFS | NameNode  DataNode | DataNode | DataNode  SecondaryNameNode |
| Yarn | NodeManager | Resourcemanager  NodeManager | NodeManager |
| Zookeeper | Zookeeper | Zookeeper | Zookeeper |
| Flume(采集日志) | Flume | Flume |  |
| Flume（消费Kafka） |  |  | Flume |
| Hive | Hive |  |  |
| MySQL | MySQL |  |  |
| Sqoop | Sqoop |  |  |
| Azkaban | Azkaban |  |  |

# 第3章 数据生成模块

## 3.1 埋点数据基本格式（以\t分隔）

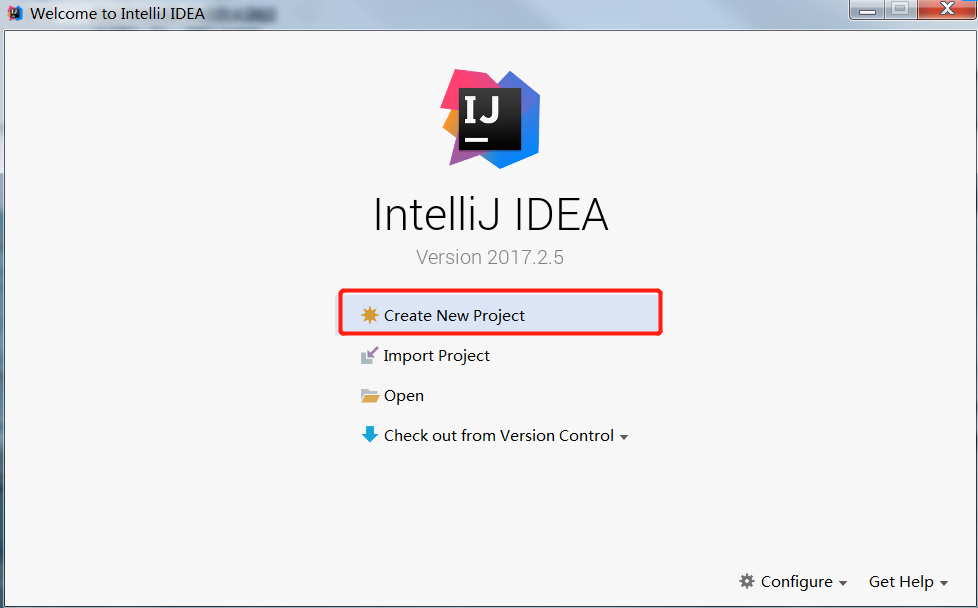
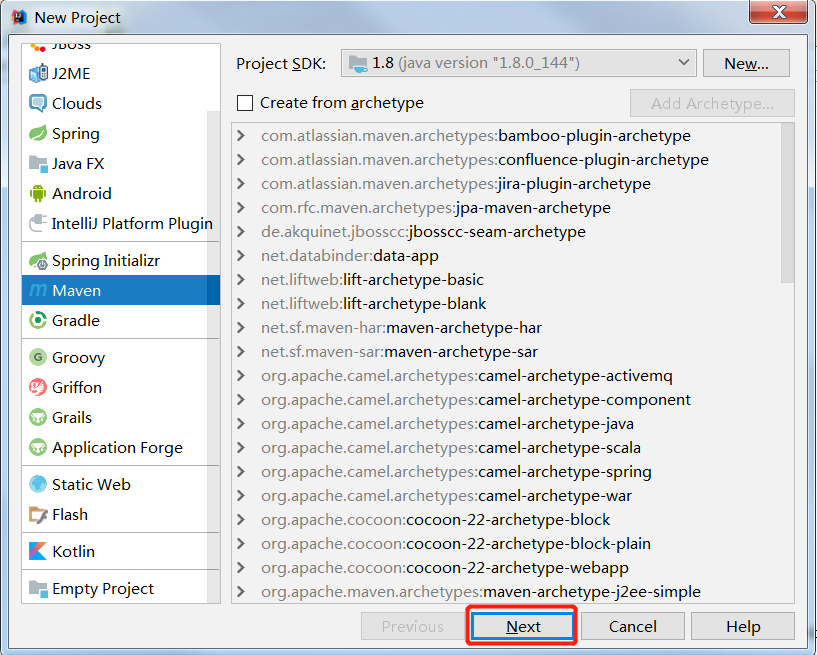
45660 45660 M 1 0 ios huawei wifi 59.48.116.0 18701445660 1 0 0 0 1.0 startHomework 1554134400

## 3.2 数据schema

uid STRING comment "用户唯一标识",  
username STRING comment "用户昵称",  
gender STRING comment "性别",  
level TINYINT comment "1代表小学，2代表初中，3代表高中",  
is\_vip TINYINT comment "0代表不是会员，1代表是会员",  
os STRING comment "操作系统:os,android等",  
channel STRING comment "下载渠道:auto,toutiao,huawei",  
net\_config STRING comment "当前网络类型",  
ip STRING comment "IP地址",  
phone STRING comment "手机号码",  
video\_id INT comment "视频id",  
video\_length INT comment "视频时长，单位秒",  
start\_video\_time BIGINT comment "开始看视频的时间缀，秒级",  
end\_video\_time BIGINT comment "退出视频时的时间缀，秒级",  
version STRING comment "版本",  
event\_key STRING comment "事件类型",  
event\_time BIGINT comment "事件发生时的时间缀，秒级"

### 3.4.1 创建Maven工程

1）创建bigdata

2）创建一个包名：com.atguigu.data\_monitor

3）在com.atguigu.data\_monitor包下创建一个Object: GeneratorUserBehaviorMonitorData。

4）在pom.xml文件中添加如下内容

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.atguigu</groupId>

<artifactId>bigdata</artifactId>

<version>1.0-SNAPSHOT</version>

<name>bigdata</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>2.3.2</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-assembly-plugin </artifactId>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass>com.atguigu.data\_monitor.GeneratorUserBehaviorMonitorData</mainClass>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

注意：com.atguigu.data\_monitor.GeneratorUserBehaviorMonitorData要和自己建的全类名一致。

### 3.4.2 生成测试数据代码

package com.atguigu.data\_monitor

import java.io.PrintWriter

import java.text.SimpleDateFormat

/\*\*

\* 生成用户行为模拟数据类

\*/

object GeneratorUserBehaviorMonitorData {

def main(args: Array[String]): Unit = {

if (args.length != 1) {

println("Usage:Please input date like 2019-04-02")

System.exit(1)

}

generatorMonitorData(args(0))

}

def generatorMonitorData(date: String): Unit = {

// 初始化手机号前6位，后5位自动化补齐

val initPhone = 187014

// 初始化时间缀，精确到秒

val sdf = new SimpleDateFormat("yyyy-MM-dd")

val eventTime = sdf.parse(date)

val initTimestamp = eventTime.getTime() / 1000

// 生成看视频但是没有看完的数据

writeMonitorData2File("watchVideo", false, 10000, 10000, initPhone, initTimestamp)

// 生成看视频且看完的数据

writeMonitorData2File("completeVideo", true, 20001, 8000, initPhone, initTimestamp)

// 生成看完视频且开始做作业的数据

writeMonitorData2File("startHomework", true, 30001, 7000, initPhone, initTimestamp)

// 生成看完视频且做完作业的数据

writeMonitorData2File("completeHomework", true, 40001, 6000, initPhone, initTimestamp)

// 生成进入订单页的数据

writeMonitorData2File("enterOrderPage", true, 50001, 4000, initPhone, initTimestamp)

// 生成进入订单页且完成订单的数据

writeMonitorData2File("completeOrder", true, 60000, 2000, initPhone, initTimestamp)

}

/\*\*

\* 通过实始化的时间缀和是否完成视频的条件生成开始视频，结束视频，事件发生的时间

\*

\* @param initTimestamp

\* @param isCompleteVideo

\* @return

\*/

def getVideoTimeAndEventTime(initTimestamp: Long, isCompleteVideo: Boolean) = {

// 定义开始视频时间为传入的initTimestamp

val startVideoTime = initTimestamp

// 因为视频时长统一定义为300秒，如果是未完成视频，则endVideoTime统一定义为initTimestamp + 100，如完成，则统一加300

val endVideoTime = if (isCompleteVideo) initTimestamp + 300 else initTimestamp + 100

// 事件发生时间eventTime也统一定义为initTimestamp即可

val eventTime = initTimestamp

(startVideoTime, endVideoTime, eventTime)

}

/\*\*

\* 根据传入的事件类型生成不同的模拟数据

\*

\* @param initUid

\* @param userAccount

\* @param initPhone

\* @param initTimestamp

\* @param isCompleteVideo

\* @param dataType

\*/

def writeMonitorData2File(dataType: String, isCompleteVideo: Boolean, initUid: Int, userAccount: Int, initPhone: Int, initTimestamp: Long): Unit = {

val writer: PrintWriter = new PrintWriter(s"./${dataType}\_${initTimestamp}.txt")

// 获取开始看视频时间，结束看视频时间和事件发生时间

val (startVideoTime, endVideoTime, eventTime) = getVideoTimeAndEventTime(initTimestamp, isCompleteVideo)

for (uid <- initUid until (initUid + userAccount)) {

// 拼接完整的11位手机号

val phone = initPhone + "" + uid

val event = dataType match {

case "watchVideo" => s"""|$uid\t$uid\tF\t2\t0\tSymbian\tauto\t4G\t27.129.32.0\t$phone\t1\t300\t$startVideoTime\t0\t1.0\tstartVideo\t$eventTime

|$uid\t$uid\tF\t2\t0\tSymbian\tauto\t4G\t27.129.32.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.0\tendVideo\t$eventTime\n""".stripMargin

case "completeVideo" => s"""|$uid\t$uid\tM\t1\t0\tios\tauto\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\tauto\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\tauto\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t0\t1.0\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\tauto\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.0\tendVideo\t$eventTime\n""".stripMargin

case "startHomework" => s"""|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t0\t1.0\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.0\tendVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.0\tstartHomework\t$eventTime\n""".stripMargin

case "completeHomework" => s"""|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.0\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t0\t1.0\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.0\tendVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.0\tstartHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\thuawei\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.0\tcompleteHomework\t$eventTime\n""".stripMargin

case "enterOrderPage" => s"""|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.1\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.1\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t0\t1.1\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t1.1\tendVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.1\tstartHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t1\t0\t0\t0\t1.1\tcompleteHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t59.48.116.0\t$phone\t0\t0\t0\t0\t1.1\tenterOrderPage\t$eventTime\n""".stripMargin

case "completeOrder" => s"""|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t0\t0\t0\t0\t2.0\tregisterAccount\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t0\t0\t0\t0\t2.0\tstartApp\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t1\t300\t$startVideoTime\t0\t2.0\tstartVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t1\t300\t$startVideoTime\t$endVideoTime\t2.0\tendVideo\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t1\t0\t0\t0\t2.0\tstartHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t1\t0\t0\t0\t2.0\tcompleteHomework\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t0\t0\t0\t0\t2.0\tenterOrderPage\t$eventTime

|$uid\t$uid\tM\t1\t0\tios\ttoutiao\twifi\t42.86.6.0\t$phone\t0\t0\t0\t0\t2.0\tcompleteOrder\t$eventTime\n|""".stripMargin

}

writer.write(event)

}

writer.close()

}

}

# 第4章 数据采集模块

## 4.1 CM及Hadoop安装(文档里修改了lzo安装)



**在安装CDH时候已经安装了hadoop集群**

**1.cdh安装的各个框架，其bin目录在/usr/bin, conf都在“/etc/框架名”内,**

**2.实际上，两个目录内的命令和配置文件都是链接到****/etc/alternatives下的命令和配置文件（比如hive命令，和hive-conf）,**

**3.进一步，发现/etc/alternatives里面的命令和配置文件，都是链接到其他具体路径，多数都是命令链接到/opt/cloudera/parcels/CDH-5.12.1-1.cdh5.12.1.p0.3/bin目录下的命令，**

**配置文件链接到/opt/cloudera/parcels/CDH-5.12.1-1.cdh5.12.1.p0.3/etc/框架名/，少数配置文件不是（比如hive-conf还是指向/etc/hive/conf.cloudera.hive）**

## 4.2 Zookeeper安装

详见：尚硅谷大数据技术之Zookeeper



|  |  |  |  |
| --- | --- | --- | --- |
|  | 服务器hadoop102 | 服务器hadoop103 | 服务器hadoop104 |
| Zookeeper | Zookeeper | Zookeeper | Zookeeper |

## 4.3 Flume安装

集群规划

|  |  |  |  |
| --- | --- | --- | --- |
|  | 服务器hadoop102 | 服务器hadoop103 | 服务器hadoop104 |
| Flume(采集日志) | Flume | Flume |  |

### 4.3.1 日志采集Flume安装

详见：尚硅谷大数据技术之Flume

文档内补充TLS/SSL验证错误问题和单个agent配置



### 4.3.2 日志采集Flume配置

0）第一层，hadoop103上的flume的配置文件file2flume.conf

# Name the components on this agent

a1.sources = r1

a1.channels = c1

a1.sinkgroups = g1

a1.sinks = k1 k2

# Describe/configure the source

a1.sources.r1.type = TAILDIR

a1.sources.r1.channels = c1

a1.sources.r1.positionFile = /opt/module/flume-2/checkpoint/behavior/taildir\_position.json

a1.sources.r1.filegroups = f1

a1.sources.r1.filegroups.f1 = /opt/module/flume-2/test-data/test.txt

a1.sources.r1.fileHeader = true

# Describe the channel

a1.channels.c1.type = file

a1.channels.c1.checkpointDir = /opt/module/flume-2/checkpoint/behavior

a1.channels.c1.dataDirs = /opt/module/flume-2/data/behavior/

a1.channels.c1.maxFileSize = 104857600

a1.channels.c1.capacity = 1000000

a1.channels.c1.keep-alive = 60

# round轮询

a1.sinkgroups.g1.processor.type = load\_balance

a1.sinkgroups.g1.processor.backoff = true

a1.sinkgroups.g1.processor.selector = round\_robin

a1.sinkgroups.g1.processor.selector.maxTimeOut=10000

# Describe the sink

a1.sinks.k1.type = avro

a1.sinks.k1.channel = c1

a1.sinks.k1.hostname = hadoop102

a1.sinks.k1.port = 2222

a1.sinks.k2.type = avro

a1.sinks.k2.channel = c1

a1.sinks.k2.hostname = hadoop104

a1.sinks.k2.port = 4444

# Bind the source and sink to the channel

a1.sources.r1.channels = c1

a1.sinkgroups.g1.sinks = k1 k2

a1.sinks.k1.channel = c1

a1.sinks.k2.channel = c1

第二层，hadoop102的配置文件flume2kafka\_1.conf

# Name the components on this agent

a2.sources = r1

a2.sinks = k1

a2.channels = c1

# Describe/configure the source

a2.sources.r1.type = avro

a2.sources.r1.bind = hadoop102

a2.sources.r1.port = 2222

# Describe the sink

a2.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink

a2.sinks.k1.topic = log-analysis

a2.sinks.k1.brokerList = hadoop102:9092,hadoop103:9092,hadoop104:9092

a2.sinks.k1.requiredAcks = 1

a2.sinks.k1.kafka.producer.type = sync

a2.sinks.k1.batchSize = 1

# Describe the channel

a2.channels.c1.type = memory

a2.channels.c1.capacity = 1000

a2.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel

a2.sources.r1.channels = c1

a2.sinks.k1.channel = c1

第二层，hadoop104的flume的配置文件flume2kafka\_2.conf

# Name the components on this agent

a3.sources = r1

a3.sinks = k1

a3.channels = c2

# Describe/configure the source

a3.sources.r1.type = avro

a3.sources.r1.bind = hadoop104

a3.sources.r1.port = 4444

# Describe the sink

a3.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink

a3.sinks.k1.topic = log-analysis

a3.sinks.k1.brokerList = hadoop102:9092,hadoop103:9092,hadoop104:9092

a3.sinks.k1.requiredAcks = 1

a3.sinks.k1.kafka.producer.type = sync

a3.sinks.k1.batchSize = 1

# Describe the channel

a3.channels.c2.type = memory

a3.channels.c2.capacity = 1000

a3.channels.c2.transactionCapacity = 100

# Bind the source and sink to the channel

a3.sources.r1.channels = c2

a3.sinks.k1.channel = c2

消费kafka数据的flume的配置

a4.sinks.k1.type=hdfs

a4.sinks.k1.hdfs.path=hdfs://hadoop102:9000/flume/%Y%m%d

a4.sinks.k1.hdfs.fileType=DataStream

a4.sinks.k1.hdfs.writeFormat=TEXT

# 每600秒滚动一个文件

a4.sinks.k1.hdfs.rollInterval=600

# 每128M滚动一个文件

a4.sinks.k1.hdfs.rollSize=134217728

a4.sinks.k1.hdfs.rollCount=0

# 每次拉取1000个event写入HDFS

a4.sinks.k1.hdfs.batchsize=1000

a4.sinks.k1.hdfs.threadsPoolSize=16

a4.sinks.k1.hdfs.filePrefix=flume.%Y%m%d%H%M

a4.sinks.k1.hdfs.idelTimeout=600

a4.sinks.k1.hdfs.round=true

a4.sinks.k1.hdfs.roundValue=10

a4.sinks.k1.hdfs.roundUnit= minute

# Describe the channel

a4.channels.c1.type = memory

a4.channels.c1.capacity = 1000

a4.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel

a4.sources.r1.channels = c1

a4.sinks.k1.channels = c1

1. Flume组件介绍
2. Source
3. Taildir Source相比Exec Source、Spooling Directory Source的优势（自我介绍时的亮点）

答：1.7版本之前，实现实时采集日志的Source只有Exec Source，但此Source可能会丢失数据（见官网描述）  
大家为了实现实时采集的效果，又保证数据安全，只能每隔半分钟产生一个并移动到Spooling Directory监控的目录中，此类做法会在web server中产生非常多的日志小文件，不利于管理  
1.7版本之后，出现了Taildir Source，即可以实时采集数据，又保证了数据安全，内部实现了类似断点续传的功能

1. batchSize大小如何设置？

答：event1K左右，500-1000合适（默认为100）

1. Channel
2. FileChannel和MemoryChannel区别(面试题)

答：

1、MemoryChannel传输数据速度更快，但因为数据保证在JVM的堆内存中，agent进程挂掉会导致数据丢失，适用于对数据质量要求不高的需求

2、FileChannel传输速度相对于Memory慢，但数据安全保障高，agent进程挂掉也可以从失败中恢复数据

1. FileChannel优化

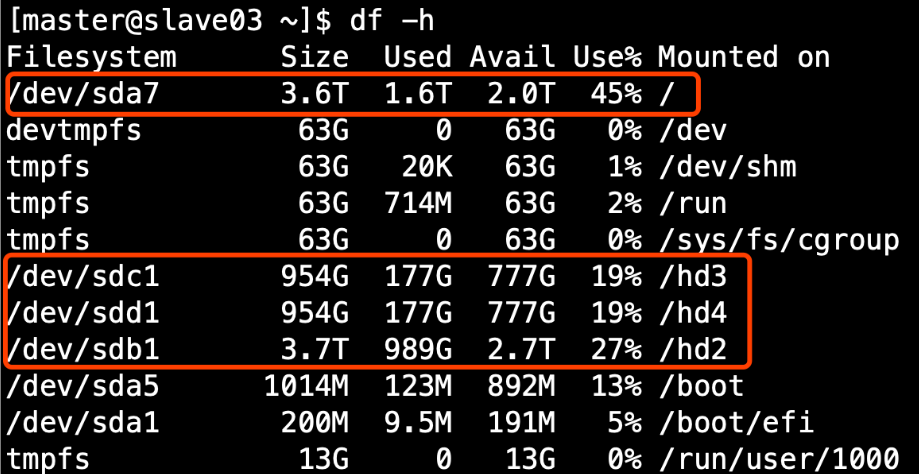
1、通过配置dataDirs指向多个路径，每个路径对应不同的硬盘，增大Flume吞吐量。

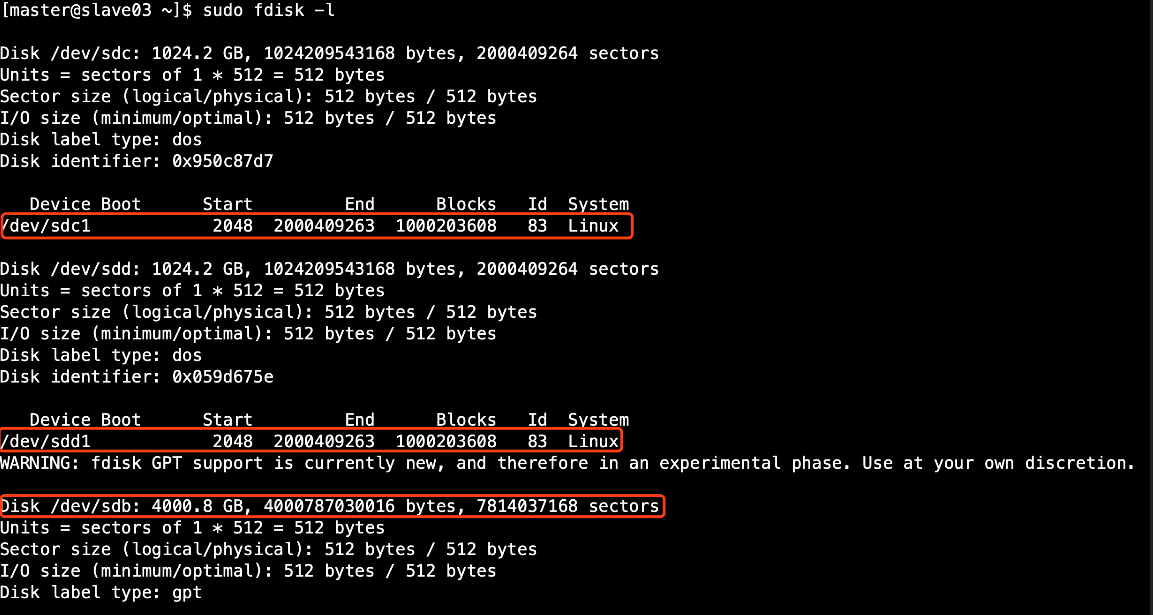
官方说明如下：

Comma separated list of directories for storing log files. Using multiple directories on separate disks can improve file channel peformance

2、checkpointDir和backupCheckpointDir也尽量配置在不同硬盘对应的目录中，保证checkpoint坏掉后，可以快速使用backupCheckpointDir恢复数据

如何查看目录对应哪块硬盘? df -h,fdisk -l命令





1. Sink：HDFS Sink
2. 官方默认的这三个参数配置写入HDFS后会产生小文件，hdfs.rollInterval、hdfs.rollSize、hdfs.rollCount
3. HDFS存入大量小文件，有什么影响？（面试题）

答：

元数据层面：每个小文件都有一份元数据，其中包括文件路径，文件名，所有者，所属组，权限，创建时间等，这些信息都保存在Namenode内存中。所以小文件过多，会占用Namenode服务器大量内存，影响Namenode性能和使用寿命

计算层面：默认情况下MR会对每个小文件启用一个Map任务计算，非常影响计算性能。同时也影响磁盘寻址时间。

1. 生产环境HDFS Sink建议配置

a1.sinks.k1.type=hdfs

a1.sinks.k1.hdfs.path=hdfs://atguigu:8020/guolong/%Y%m%d

a1.sinks.k1.hdfs.fileType=DataStream

a1.sinks.k1.hdfs.writeFormat=TEXT

# 每600秒滚动一个文件

a1.sinks.k1.hdfs.rollInterval=600

# 每128M滚动一个文件

a1.sinks.k1.hdfs.rollSize=134217728

a1.sinks.k1.hdfs.rollCount=0

# 每次拉取1000个event写入HDFS

a1.sinks.k1.hdfs.batchsize=1000

a1.sinks.k1.hdfs.threadsPoolSize=16

a1.sinks.k1.channel=c1

a1.sinks.k1.hdfs.filePrefix=guolong.%Y%m%d%H%M

a1.sinks.k1.hdfs.idelTimeout=600

a1.sinks.k1.hdfs.round=true

a1.sinks.k1.hdfs.roundValue=10

a1.sinks.k1.hdfs.roundUnit= minute

1. 基于以上hdfs.rollInterval=1800，hdfs.rollSize=134217728，hdfs.roundValue=10，hdfs.roundUnit= minute几个参数综合作用，效果如下：

1、tmp文件在达到128M时会滚动生成正式文件

2、tmp文件创建超10分钟时会滚动生成正式文件

举例：在2018-01-01 05:23的的时侯sink接收到数据，那会产生如下tmp文件：

/guolong/20180101/guolong.201801010520.tmp

即使文件内容没有达到128M，也会在05:33时滚动生成正式文件

1. 测试启动脚本

bin/flume-ng agent -n a1(agent的名称) -c conf -f conf/example(配置文件名称) **-Dflume.root.logger=DEBUG,console**

标红部分重点关注，可以通过DEBUG模式快速定位问题

1. Flume分层
   1. Flume为什么分两层

答：

如果只有一层，日志采集服务器非常多，此时会有很多个Flume agent，同时向HDFS写数据会产生多个client，对HDFS来说压力过大

只有一层时，部分业务配置只能在这层配置，如后续配置修改，则要修改的位置太多，不利于后期维护

* 1. 采集层
     1. 使用supervior方式保证agent挂掉后自动重启
     2. 因为要采集业务日志，所以需要部署在业务服务器上
     3. 根据服务器配置设置JVM heap，一般设置512M – 1G
  2. 渠聚层
     1. 使用load\_balance
     2. JVM heap一般设置为4G或更高
     3. 部署在单独的服务器上（4核8线程16G内存）
  3. JVM调优
     1. -Xmx与-Xms设置一样，减少内存抖动带来的性能影响

1. 基于以上双层架构配置，分析Flume如何保证数据至少处理一次
2. 采集层agent挂掉：因为使用了Taildir Source，所以可以断点续传
3. 汇聚层agent挂掉：因为使用load balance，实现了负载均衡和高可用
4. Flume的source向channel写入数据、sink从channel拉取数据本身带有事务机制

# 第5章 数据清洗、加载

## 5.1 需求

1. 假定现在已经将数据保存到HDFS的/user/hive/warehouse/ods.db/origin\_user\_behavior/${day}目录中，需要用SparkCore将数据清洗，清洗需求如下：
   1. 手机号脱敏：187xxxx2659
   2. 过滤重复行（重复条件，uid,event\_key,event\_time三者都相同即为重复）
   3. 最终数据保存到dwd.user\_behavior分区表，以dt（天）为分区条件，表的文件存储格式为ORC，数据总量为xxxx条
2. Hive 字段如下

uid STRING comment "用户唯一标识",  
username STRING comment "用户昵称",  
gender STRING comment "性别",  
level TINYINT comment "1代表小学，2代表初中，3代表高中",  
is\_vip TINYINT comment "0代表不是会员，1代表是会员",  
os STRING comment "操作系统:os,android等",  
channel STRING comment "下载渠道:auto,toutiao,huawei",  
net\_config STRING comment "当前网络类型",  
ip STRING comment "IP地址",  
phone STRING comment "手机号码",  
video\_id INT comment "视频id",  
video\_length INT comment "视频时长，单位秒",  
start\_video\_time BIGINT comment "开始看视频的时间缀，秒级",  
end\_video\_time BIGINT comment "退出视频时的时间缀，秒级",  
version STRING comment "版本",  
event\_key STRING comment "事件类型",  
event\_time BIGINT comment "事件发生时的时间缀，秒级"

1. 整体流程（使用调度系统调度）
   1. SparkCore清洗数据，写入到/user/hive/warehouse/tmp.db/user\_behavior\_${day}目录
   2. 建立tmp.user\_behavior\_${day}临时表，并加载上面清洗后的数据
   3. 使用hive引擎，并用开窗函数row\_number，将tmp.user\_behavior\_${day}表数据插入到dwd.user\_behavior表中
   4. 删除tmp.user\_behavior\_${day}临时表

## 5.2 数据清洗代码

inputPath:/user/hive/warehouse/ods.db/origin\_user\_behavior/${day}

outputPath:/user/hive/warehouse/tmp.db/user\_behavior\_${day}

package com.atguigu.user\_behavior

import org.apache.spark.rdd.RDD

import org.apache.spark.{SparkConf, SparkContext}

/\*\*

\* 用户行为数据清洗

\* 1、验证数据格式是否正确，切分后长度必须为17

\* 2、手机号脱敏，格式为123xxxx4567

\* 3、去掉username中带有的\n，否则导致写入HDFS时会换行

\*/

object UserBehaviorCleaner {

def main(args : Array[String]): Unit ={

if(args.length != 2){

println("Usage:please input inputPath and outputPath")

System.exit(1)

}

// 获取输入输出路径

val inputPath = args(0)

val outputPath = args(1)

val conf = new SparkConf().setAppName(getClass.getSimpleName).setMaster("local[2]")

val sc = new SparkContext(conf)

// 通过输入路径获取RDD

val eventRDD: RDD[String] = sc.textFile(inputPath)

// 清洗数据，在算子中不要写大量业务逻辑，应该将逻辑封装到方法中

eventRDD.filter(event => checkEventValid(event)) // 验证数据有效性

.map( event => maskPhone(event)) // 手机号脱敏

.map(event => repairUsername(event)) // 修复username中带有\n导致的换行

.coalesce(3)

.saveAsTextFile(outputPath)

sc.stop()

}

/\*\*

\* username为用户自定义的，里面有要能存在"\n"，导致写入到HDFS时换行

\* @param event

\*/

def repairUsername(event : String)={

val fields = event.split("\t")

// 取出用户昵称

val username = fields(1)

// 用户昵称不为空时替换"\n"

if(username != "" && !"Null".equals(username)){

fields(1) = username.replace("\n","")

}

fields.mkString("\t")

}

/\*\*

\* 脱敏手机号

\* @param event

\*/

def maskPhone(event : String): String ={

var maskPhone = new StringBuilder

val fields: Array[String] = event.split("\t")

// 取出手机号

val phone = fields(9)

// 手机号不为空时做掩码处理

if(phone != null && !"".equals(phone)){

maskPhone = maskPhone.append(phone.substring(0,3)).append("xxxx").append(phone.substring(7,11))

fields(9) = maskPhone.toString()

}

fields.mkString("\t")

}

/\*\*

\* 验证数据格式是否正确，只有切分后长度为17的才算正确

\* @param event

\*/

def checkEventValid(event : String) ={

val fields = event.split("\t")

fields.length == 17

}

}

## 部署

1. 面试题 ：Yarn cluster和Yarn client模式有什么别区
2. Spark on yarn完整提交命令

spark-submit --master yarn --deploy-mode cluster \

--num-executors 8 \

--executor-cores 4 \

--executor-memory 12G \

--class com.atguigu.user\_behavior.UserBehaviorCleaner UserBehaviorCleaner.jar \

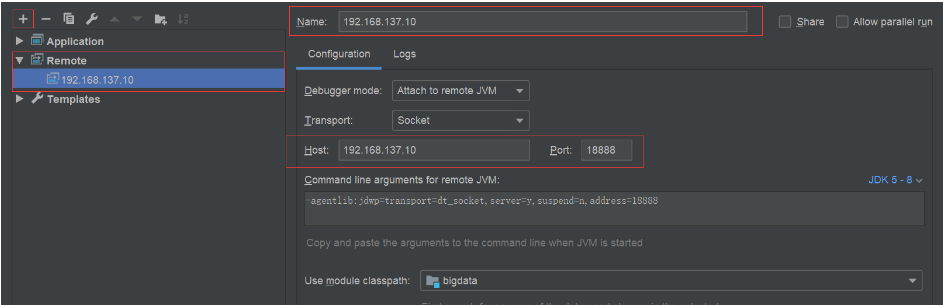
hdfs://atguigu:8020/user/hive/warehouse/ods.db/origin\_user\_behavior/${day} \

hdfs://atguigu:8020/user/hive/warehouse/tmp.db/user\_behavior\_${day}

## 5.4 远程调试

场景：以后工作中经常会遇到在本地执行没有问题，到了服务器跑的数据就是错误的

1. IDEA设置：Run --> Edit Configurations添加Remote



1. 在提交脚本中添加--driver-java-options参数

spark-submit --master local[2] \

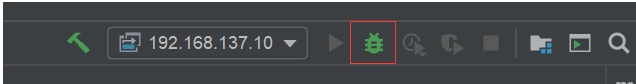
--driver-java-options "-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=18888" \

--class com.atguigu.user\_behavior.UserBehaviorCleaner UserBehaviorCleaner.jar \

hdfs://hadoop102:9000/user/hive/warehouse/ods.db/origin\_user\_behavior/${day} \

hdfs:// hadoop102:9000 /user/hive/warehouse/tmp.db/user\_behavior\_${day}

1. 在服务器提交任务，此时任务处理阻塞状态
2. 在idea中点击Remote服务对应的debug按钮



## 5.5加载、去重

1. 将清洗后的数据加载到临时表

create table if not exists tmp.user\_behavior\_${day}(

uid STRING comment "用户唯一标识",

username STRING comment "用户昵称",

gender STRING comment "性别",

level TINYINT comment "1代表小学，2代表初中，3代表高中",

is\_vip TINYINT comment "0代表不是会员，1代表是会员",

os STRING comment "操作系统:os,android等",

channel STRING comment "下载渠道:auto,toutiao,huawei",

net\_config STRING comment "当前网络类型",

ip STRING comment "IP地址",

phone STRING comment "手机号码",

video\_id INT comment "视频id",

video\_length INT comment "视频时长，单位秒",

start\_video\_time BIGINT comment "开始看视频的时间缀，秒级",

end\_video\_time BIGINT comment "退出视频时的时间缀，秒级",

version STRING comment "版本",

event\_key STRING comment "事件类型",

event\_time BIGINT comment "事件发生时的时间缀，秒级")

row format delimited fields terminated by "\t"

location "/user/hive/warehouse/tmp.db/user\_behavior\_${day}";

创建ORC格式的目标表

create external table if not exists dwd.user\_behavior(

uid STRING comment "用户唯一标识",

username STRING comment "用户昵称",

gender STRING comment "性别",

level TINYINT comment "1代表小学，2代表初中，3代表高中",

is\_vip TINYINT comment "0代表不是会员，1代表是会员",

os STRING comment "操作系统:os,android等",

channel STRING comment "下载渠道:auto,toutiao,huawei",

net\_config STRING comment "当前网络类型",

ip STRING comment "IP地址",

phone STRING comment "手机号码",

video\_id INT comment "视频id",

video\_length INT comment "视频时长，单位秒",

start\_video\_time BIGINT comment "开始看视频的时间缀，秒级",

end\_video\_time BIGINT comment "退出视频时的时间缀，秒级",

version STRING comment "版本",

event\_key STRING comment "事件类型",

event\_time BIGINT comment "事件发生时的时间缀，秒级") partitioned by(dt INT)

row format delimited fields terminated by "\t" stored as ORC

1. 将tmp.user\_behavior\_${day}的数据导入到ORC表中，使用开窗函数实现去重业务

insert overwrite table dwd.user\_behavior partition(dt=${day})

select

uid,

username,

gender,

level,

is\_vip,

os,

channel,

net\_config,

ip,

phone,

video\_id,

video\_length,

start\_video\_time,

end\_video\_time,

version,

event\_key,

event\_time

from (

select

uid,

username,

gender,

level,

is\_vip,

os,

channel,

net\_config,

ip,

phone,

video\_id,

video\_length,

start\_video\_time,

end\_video\_time,

version,

event\_key,

event\_time,

row\_number() OVER (PARTITION BY uid,event\_key,event\_time ORDER BY event\_time) u\_rank

from tmp.user\_behavior\_${day}

) temp where u\_rank = 1

# 第6章 指标实现

## 6.1 事件类型(event\_key)介绍

startApp 打开App

closeApp 关闭App

registerAccount 注册用户

startVideo 开始看视频

endVideo 结束看视频

startHomework 开始作业

completeHomework 完成作业

shareVideo 分享视频

enterOrderPage 进入订单详情页

completeOrder 支付完成订单，成为vip

说明：每类event\_key代表一种行为

## 6.2 课程学习反馈指标

1. 指标图示：
2. Mysql schema设计

CREATE TABLE app\_cource\_study\_report (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`watch\_video\_cnt` int(11) DEFAULT NULL,

`complete\_video\_cnt` int(11) DEFAULT NULL,

`dt` int(11) DEFAULT NULL,

`created\_at` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`updated\_at` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`),

UNIQUE KEY `app\_cource\_study\_report\_dt` (`dt`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

1. Hive schema设计

create table if not exists tmp.app\_cource\_study\_analysis\_${day}(

watch\_video\_count INT,

complete\_video\_count INT,

dt INT

) row format delimited fields terminated by "\t";

1. SQL实现

insert overwrite table tmp.app\_cource\_study\_analysis\_${day}

select sum(watch\_video\_count),sum(complete\_video\_count),dt from (

select count(distinct uid) as watch\_video\_count,0 as complete\_video\_count,dt from dwd.user\_behavior where dt = ${day} and event\_key = "startVideo" group by dt

union all

select 0 as watch\_video\_count,count(distinct uid) as complete\_video\_count,dt from dwd.user\_behavior where dt = ${day} and event\_key = "endVideo"

and (end\_video\_time - start\_video\_time) >= video\_length group by dt) tmp group by dt

1. 思考题：
   1. 插入时为什么要用overwrite而不是into
   2. **union**与**union** **all**的区别
2. Hive脚本实现整个流程

app\_course\_study\_analysis.sh

#! /bin/bash

day=$1

# 验证输入参数的合法性

if [ ${#day} -ne 8 ];then

echo "Please input date,eg:20190402"

exit 1

fi

# 创建临时表

hive -e "

create table if not exists tmp.app\_cource\_study\_analysis\_${day}(

watch\_video\_count INT,

complete\_video\_count INT,

dt INT

) row format delimited fields terminated by '\t';"

# 向临时表插入数据

hive -e "

insert overwrite table tmp.app\_cource\_study\_analysis\_${day}

select sum(watch\_video\_count),sum(complete\_video\_count),dt from (

select count(distinct uid) as watch\_video\_count,0 as complete\_video\_count,dt from dwd.user\_behavior where dt = ${day} and event\_key = 'startVideo' group by dt

union

select 0 as watch\_video\_count,count(distinct uid) as complete\_video\_count,dt from dwd.user\_behavior where dt = ${day} and event\_key = 'endVideo'

and (end\_video\_time - start\_video\_time) >= video\_length group by dt) tmp group by dt

"

1. SparkSQL实现整个流程

package com.atguigu.user\_behavior

import org.apache.spark.sql.SparkSession

object AppCourseStudyAnalysis {

def main(args: Array[String]): Unit = {

// 获取日期并验证

val day = args(0)

if("".equals(day) || day.length() != 8){

println("Usage:Please input date,eg:20190402")

System.exit(1)

}

// 获取SparkSession，并支持Hive操作

val spark: SparkSession = SparkSession.builder()

.appName(this.getClass.getSimpleName)

.config("spark.sql.warehouse.dir", "/user/hive/warehouse")

.enableHiveSupport()

.master("local[2]")

.getOrCreate()

import spark.sql

// 创建临时表

sql(s"""

|create table if not exists tmp.app\_cource\_study\_analysis\_${day}(

|watch\_video\_count INT,

|complete\_video\_count INT,dt INT)

|row format delimited fields terminated by '\t'

""".stripMargin)

// 将分析结果插入临时表

sql(

s"""

|insert overwrite table tmp.app\_cource\_study\_analysis\_${day}

|select sum(watch\_video\_count),sum(complete\_video\_count),dt from (

|select count(distinct uid) as watch\_video\_count,0 as complete\_video\_count,dt from dwd.user\_behavior where dt = ${day} and event\_key = 'startVideo' group by dt

|union

|select 0 as watch\_video\_count,count(distinct uid) as complete\_video\_count,dt from dwd.user\_behavior where dt = ${day} and event\_key = 'endVideo'

|and (end\_video\_time - start\_video\_time) >= video\_length group by dt) tmp group by dt

""".stripMargin)

spark.stop()

}

}

1. 思考：可以使用SparkSQL将分析结果直接写入到Mysql，而我们使用的是SparkSQL将结果写入到Hive后，再通过Sqoop导出到Mysql，两者哪个更好？
2. Sqoop导出

sqoop export --connect jdbc:mysql://192.168.137.10:3306/user\_behavior --username root --password 123456 --table app\_cource\_study\_report --columns watch\_video\_cnt,complete\_video\_cnt,dt --fields-terminated-by "\t" --export-dir "/user/hive/warehouse/tmp.db/app\_cource\_study\_analysis\_${day}" --input-null-string '\\N'

## 6.3 各系统版本访问统计

1. Hive schema

create table if not exists tmp.app\_version\_analysis\_${day}(

os STRING,

version STRING,

access\_count INT,

dt INT

) row format delimited fields terminated by "\t"

1. SQL实现

insert overwrite table tmp.app\_version\_analysis\_${day}

select os,version,count(1) as access\_count,dt from dwd.user\_behavior where dt = ${day} group by os,version,dt;

## 6.4 渠道新用户统计

1. Hive schema

create table if not exists tmp.app\_channel\_analysis\_${day}(

channel STRING,

new\_user\_cnt INT,

dt INT

) row format delimited fields terminated by "\t"

1. SQL实现

insert overwrite table tmp.app\_channel\_analysis\_${day}

select channel,count(distinct uid),dt from dwd.user\_behavior where dt = ${day} and event\_key = "registerAccount" group by channel,dt;

## 6.5 访问次数分布（我们的指标是1-2次(包含)，3-4次(包含)，大于4次）

1. Hive schema

create table if not exists tmp.app\_access\_cnt\_ranger\_analysis\_${day}(

le\_two INT,

le\_four INT,

gt\_four INT,

dt INT

) row format delimited fields terminated by "\t";

1. SQL实现

第一步：计算每个用户的访问次数并分组

drop table if exists tmp.user\_access\_cnt\_${day};

create table if not exists tmp.user\_access\_cnt\_${day} as select uid,count(1) as access\_cnt,dt from dwd.user\_behavior where dt = ${day} group by uid,dt;

第二步：根据访问次数来计算用户分步并插入最终表

insert overwrite table tmp.app\_access\_cnt\_ranger\_analysis\_${day}

select sum(le\_two) as le\_two,sum(le\_four) as le\_four,sum(gt\_four) as gt\_four,dt from

(select count(1) as le\_two,0 as le\_four,0 as gt\_four,dt from tmp.user\_access\_cnt\_${day} where access\_cnt <= 2 group by dt

union all

select 0 as le\_two,count(1) as le\_four,0 as gt\_four,dt from tmp.user\_access\_cnt\_${day} where access\_cnt <= 4 group by dt

union all

select 0 as le\_two,0 as le\_four,count(1) as gt\_four,dt from tmp.user\_access\_cnt\_${day} where access\_cnt > 4 group by dt) tmp

group by dt;

## 6.6 漏斗分析

1. 指标：打开app -> 开始看视频 - > 完成视频 -> 开始作业 -> 完成作业

说明：只有看了视频，才有可能完成视频，才能开始写作业，也可以不写，但是写作业不一定完成，所以每一步都会有数据流失

1. Hive schema

create table if not exists tmp.app\_study\_funnel\_analysis\_${day}(

start\_app\_cnt INT,

start\_video\_cnt INT,

complete\_video\_cnt INT,

start\_homework\_cnt INT,

complete\_homework INT,

dt INT

) row format delimited fields terminated by "\t";

1. SQL实现

insert overwrite table tmp.app\_study\_funnel\_analysis\_${day}

select count(distinct t1.uid) as start\_app\_cnt,count(distinct t2.uid) as start\_video\_cnt,count(distinct t3.uid) as complete\_video\_cnt,count(distinct t4.uid) as start\_homework,count(distinct t5.uid) as complete\_homework,t1.dt from

(select uid,dt from dwd.user\_behavior where dt = ${day} and event\_key = "startApp") t1

left join

(select uid from dwd.user\_behavior where dt = ${day} and event\_key = "startVideo") t2

on t1.uid = t2.uid

left join

(select uid from dwd.user\_behavior where dt = ${day} and event\_key = "endVideo" and (end\_video\_time - start\_video\_time) >= video\_length) t3

on t2.uid = t3.uid

left join

(select uid from dwd.user\_behavior where dt = ${day} and event\_key = "startHomework") t4

on t3.uid = t4.uid

left join

(select uid from dwd.user\_behavior where dt = ${day} and event\_key = "completeHomework") t5

on t4.uid = t5.uid group by t1.dt

思考：时序漏斗怎么做？全局时序2小时

## 6.7 7日留存分析

1. Hive schema

create table tmp.seven\_days\_retained\_analysis\_${day}(

register\_day INT,

zero\_interval\_retained\_rate DOUBLE,

one\_interval\_retained\_rate DOUBLE,

two\_interval\_retained\_rate DOUBLE,

three\_interval\_retained\_rate DOUBLE,

four\_interval\_retained\_rate DOUBLE,

five\_interval\_retained\_rate DOUBLE,

six\_interval\_retained\_rate DOUBLE,

dt INT

) row format delimited fields terminated by "\t";

1. SQL实现

// 获取近7日内全部用户的注册日期

select uid,dt as register\_day,event\_time from dwd.user\_behavior where dt between ${startDay} and ${endDay} and event\_key = "registerAccount"

// 获取近7日每天活跃的用户列表

select uid,dt as active\_day,max(event\_time) as event\_time from dwd.user\_behavior where dt between ${startDay} and ${endDay} group by uid,dt

// 两者整合，生成uid,register\_day,active\_day,interval(活跃时距离注册日期几天)

select t1.uid,t1.register\_day,t2.active\_day,datediff(from\_unixtime(t2.event\_time,"yyyy-MM-dd"),from\_unixtime(t1.event\_time,"yyyy-MM-dd")) as day\_interval from

(select uid,dt as register\_day,event\_time from dwd.user\_behavior where dt between ${startDay} and ${endDay} and event\_key = "registerAccount") t1

left join

(select uid,dt as active\_day,max(event\_time) as event\_time from dwd.user\_behavior where dt between ${startDay} and ${endDay} group by uid,dt) t2

on t1.uid = t2.uid

结果格式：

001 20190301 20190301 0

001 20190301 20190303 2

002 20190302 20190303 1

// 根据上面的表再生成留存用户数临时表

drop table if exists tmp.user\_retained\_${startDay}\_${endDay};create table if not exists tmp.user\_retained\_${startDay}\_${endDay} as

select register\_day,day\_interval,count(1) as retained from (

select t1.uid,t1.register\_day,t2.active\_day,datediff(from\_unixtime(t2.event\_time,"yyyy-MM-dd"),from\_unixtime(t1.event\_time,"yyyy-MM-dd")) as day\_interval from

(select uid,dt as register\_day,event\_time from dwd.user\_behavior where dt between ${startDay} and ${endDay} and event\_key = "registerAccount") t1

left join

(select uid,dt as active\_day,max(event\_time) as event\_time from dwd.user\_behavior where dt between ${startDay} and ${endDay} group by uid,dt) t2

on t1.uid = t2.uid) tmp group by register\_day,day\_interval

数据结果：

20190402 0 27000

20190402 1 19393

20190402 2 14681

20190402 3 9712

20190402 4 5089

20190402 5 1767

20190402 6 1775

// 计算近7日留存率

drop table if exists tmp.user\_retained\_rate\_${startDay}\_${endDay};create table if not exists tmp.user\_retained\_rate\_${startDay}\_${endDay} as

select register\_day,day\_interval,round(retained / register\_cnt,4) as retained\_rate,current\_dt from (

select t1.register\_day,t1.day\_interval,t1.retained,t2.register\_cnt,${endDay} as current\_dt from

(select register\_day,day\_interval,retained from tmp.user\_retained\_${startDay}\_${endDay}) t1

left join

(select dt,count(1) as register\_cnt from dwd.user\_behavior where dt between ${startDay} and ${endDay} and event\_key = "registerAccount" group by dt) t2

on t1.register\_day = t2.dt

group by t1.register\_day,t1.day\_interval ,t1.retained,t2.register\_cnt) tmp

数据结果

20190402 0 1.0 20190408

20190402 1 0.7183 20190408

20190402 2 0.5437 20190408

20190402 3 0.3597 20190408

20190402 4 0.1885 20190408

20190402 5 0.0654 20190408

20190402 6 0.0657 20190408

20190403 0 1.0 20190408

20190403 1 0.7183 20190408

20190403 2 0.5437 20190408

20190403 3 0.3597 20190408

20190403 4 0.1885 20190408

20190403 5 0.0654 20190408

// 到这里还没有结束，咱们再来个列转行

insert overwrite table tmp.seven\_days\_retained\_analysis\_${day}

select

register\_day,

max(case when day\_interval = 0 then retained\_rate else 0 end) as zero\_interval\_retained\_rate,

max(case when day\_interval = 1 then retained\_rate else 0 end) as one\_interval\_retained\_rate,

max(case when day\_interval = 2 then retained\_rate else 0 end) as two\_interval\_retained\_rate,

max(case when day\_interval = 3 then retained\_rate else 0 end) as three\_interval\_retained\_rate,

max(case when day\_interval = 4 then retained\_rate else 0 end) as four\_interval\_retained\_rate,

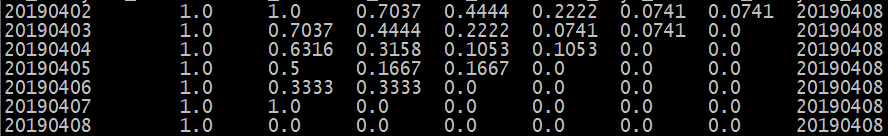
max(case when day\_interval = 5 then retained\_rate else 0 end) as five\_interval\_retained\_rate,

max(case when day\_interval = 6 then retained\_rate else 0 end) as six\_interval\_retained\_rate,

current\_dt

from tmp.user\_retained\_rate\_${startDay}\_${endDay} group by register\_day,current\_dt;

最终结果：



1. 思考：

上面的SQL完全可以写在一个大SQL中完成，为什么要分解成这么多步完成？

# 第7章 Sqoop深入

## 7.1 理解Hive中的Null底层存储的是什么

1. Mysql建立表，并执行count()操作

CREATE TABLE `student` (  
  `id` int(11) NOT NULL AUTO\_INCREMENT,  
  `username` varchar(16) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8

INSERT INTO student(username) VALUES("Jack")

INSERT INTO student(username) VALUES(NULL)

select count(username) from student

1. 将Mysql数据导入HDFS

sqoop import --connect jdbc:mysql://atguigu:3306/user\_behavior --username root --password 123456 --table student --fields-terminated-by '\t' --target-dir /user/hive/warehouse/tmp.db/student

1. 建表

create table if not exists tmp.student(id int,username string) row format delimited fields terminated by "\t";

1. Hive中执行count操作，发现与Mysql结果不一致

select count(username) from tmp.student

1. 删除tmp.student表，并重新通过sqoop导入

sqoop import --connect jdbc:mysql://atguigu:3306/user\_behavior --username root --password 123456 --table student --fields-terminated-by '\t' --target-dir /user/hive/warehouse/tmp.db/student --null-string '\\N'

1. 再次执行count操作，发现与mysql中结果一致，验证Hive中的Null在底层是以“\N”来存储，所以Sqoop导入，导出时，都会使用--null-string '\\N'进入数据转换来保证数据两端的一致性

## 7.2 如何Sqoop数据导出时数据一致性

1. 场景：如Sqoop在导出到Mysql时，使用4个map任务，过程中有2个任务失败，那此时Mysql中存储了另外两个map任务导入的数据，此时老板正好看到了这个报表数据。而开发工程师发现任务失败后，会调试问题并最终将全部数据正确的导入Mysql，那后面老板再次看报表数据，发现本次看到的数据与之前的不一致，这在生产环境是不允许的。
2. 面试题：
3. Sqoop脚本执行时，底层运行的是什么任务?
4. Sqoop导出数据到Mysql时，事务的基本单位是什么？
5. 如何解决上面场景说的问题？
6. 设置map数量为1个（不推荐，面试官想要的答案不只这个）
7. –staging-table方式

sqoop export --connect jdbc:mysql://192.168.137.10:3306/user\_behavior --username root --password 123456 --table app\_cource\_study\_report --columns watch\_video\_cnt,complete\_video\_cnt,dt --fields-terminated-by "\t" --export-dir "/user/hive/warehouse/tmp.db/app\_cource\_study\_analysis\_${day}" **--staging-table app\_cource\_study\_report\_tmp --clear-staging-table --input-null-string '\N'**

# 第8章 面试题及思考题总结

## 8.1 集群准备相关

1. 如何确认集群规模？假设每台服务器8T硬盘

答：按每条日志1K，每天1亿条，半年内不扩容服务器来算：100000000 / 1024 / 1024 = 约100G，保存半年约18T，保存3副本，共54T左右，再预留20%Buf，每台服务器8T硬盘预估，共需约9台服务器。如果要在数仓中再保存一份，服务器扩展1倍

1. 如何选择Apache/CDH/HDP版本？
2. Apache：运维麻烦，组件间兼容性需要自己调研。(一般大厂使用，技术实力雄厚，有专业的运维人员)
3. CDH：国内使用最多的版本，但CM不开源，但其实对中、小公司使用来说没有影响（建议使用）
4. HDP：开源，可以进行二次开发，但是没有CDH稳定,国内使用较少
5. 服务器使用物理机还是云主机
6. 机器成本考虑：
7. 物理机：以128G内存,20核物理CPU，40线程，8THDD和2TSSD硬盘，单台报价4W出头，需考虑托管服务器费用。一般物理机寿命5年左右
8. 云主机，以阿里云为例，差不多相同配置，每年5W
9. 运维成本考虑：
10. 物理机：需要有专业的运维人员
11. 云主机：很多运维工作都由阿里云已经完成，运维相对较轻松
12. 服务组件规划示例

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **服务名称** | **子服务** | **master.onion** | **slave01.onion** | **slave02.onion** | **slave03.onion** | **slave04.onion** |
| Zookeeper | zookeeper server | √ | √ | √ | √ | √ |
| zookeeper client | √ | √ | √ | √ | √ |
| HDFS | NameNode | √ |  | √ |  |  |
| ZKFailoverController | √ |  | √ |  |  |
| DataNodes | √ | √ | √ | √ | √ |
| JournalNodes | √ | √ | √ |  |  |
| Httpfs | √ |  |  |  |  |
| NFSGateways | √ | √ | √ | √ | √ |
| YARN | ResourceManager |  | √ |  |  |  |
| NodeManagers | √ | √ | √ | √ | √ |
| YARN Clients | √ | √ | √ | √ | √ |
| App Timeline Server |  | √ |  |  |  |
| MapReduce2 | History Server |  | √ |  |  |  |
| MapReduce2 Clients | √ | √ | √ | √ | √ |
| TEZ | Tez Clients | √ | √ | √ | √ | √ |
| Hive | Hive Metastore | √ |  |  |  |  |
| HiveServer2 | √ |  |  |  |  |
| WebHCat Server | √ |  |  |  |  |
| HCat Clients | √ | √ | √ | √ | √ |
| Hive Clients | √ | √ | √ | √ | √ |
| Pig | Pig Clients | √ | √ | √ | √ | √ |
| Ambari Metrics | Metrics Collector | √ |  |  |  |  |
| Grafana | √ |  |  |  |  |
| Metrics Monitors | √ | √ | √ | √ | √ |
| Spark | Spark History Server | √ |  |  |  |  |
| Livy Server | √ |  |  |  |  |
| Spark Thrift Server | √ |  |  |  |  |
| Spark Clients | √ | √ | √ | √ | √ |
| Slider | Slider Clients | √ | √ | √ | √ | √ |
| HUE | HUE | √ |  |  |  |  |
| mysql | mysql |  |  |  | √ | √ |
| **服务数总计** | | **27** | **18** | **17** | **14** | **14** |

## 8.2 数据采集

1. Flume为什么分两层
2. 如果只有一层，日志采集服务器非常多，此时会有很多个Flume agent，同时向HDFS写数据会产生多个client，对HDFS来说压力过大
3. 只有一层时，部分业务配置只能在这层配置，如后续配置修改，则要修改的位置太多，不利于后期维护
4. FileChannel和MemoryChannel区别
5. MemoryChannel传输数据速度更快，但因为数据保证在JVM的堆内存中，agent进程挂掉会导致数据丢失，适用于对数据质量要求不高的需求
6. FileChannel传输速度相对于Memory慢，但数据安全保障高，agent进程挂掉也可以从失败中恢复数据
7. HDFS存入大量小文件，有什么影响？
8. 元数据层面：每个小文件都有一份元数据，其中包括文件路径，文件名，所有者，所属组，权限，创建时间等，这些信息都保存在Namenode内存中。所以小文件过多，会占用Namenode服务器大量内存，影响Namenode性能和使用寿命
9. 计算层面：默认情况下MR会对每个小文件启用一个Map任务计算，非常影响计算性能。同时也影响磁盘寻址时间。

## 8.3 数据清洗、指标实现部分

1. yarn cluster和yarn client区别
2. yarn client会在提交job的服务器创建ApplicationMaster，job在运行过程中，AM会与Executor和ResourceManager进行通信，如多个任务在同一服务器使用yarn client模式提交，会造成此服务器网卡被”打满”，导致其它服务提交失败。一般情史下yarn client适于用代码测试，因为日志全部返回给Driver端，方便调试
3. yarn cluster会把AM分配到不同的服务器上，实现负载均衡的效果，生产环境一定要使用此模式
4. Textfile、Parquet、ORC格式如何选择
5. TextFile是最基本的行式存储，占用空间大，唯一的好处是方便阅读
6. Parquet和ORC都是列式存储格式，占用空间小，查询效率高

企业中使用ORC替代TextFile后，原空间为200G，替代后为20G，节省90%空间，SQL查询性能提升3-5倍

1. union与union all的区别
2. union会将联合的结果集去重，效率较union all差
3. union all不会对结果集去重，所以效率高

## 8.4 Sqoop数据导出

1. Sqoop脚本执行时，底层执行的是什么任务？

底层执行是的MapReduce中的Mapper任务

1. Sqoop导出数据到Mysql时，事务的基本单位是什么？

保证事务的基本单位是Mapper任务

1. 如何保证Sqoop导出的一致性？
2. 设置map数量为1个（不推荐，面试官想要的答案不只这个）
3. 使用—staging-table方式

原理是Sqoop首先将数据导入到Mysql中一个与目标表结构完全相同的临时表，在全部导入到临时表成功后，以1个事务将临时表的数据导入到目标表

1. Hive中的Null在底层存储的是什么内容？

在底层存储的是 \N ，所以在Sqoop导入、导出时一般会指定 --null-string '\\N'

## 8.5 离线项目真实场景分析

1. hadoop集群情况说明：
2. 20台 128G内存，40线程CPU，4T硬盘 \* 4
3. 每台服务器分给Yarn管理的资源为 36线程，108G内存（因为CPU数量尽量与内存量是一个整数比，此处为1：3）
4. 所以Yarn能够管理的总资源为2160G内存和720线程CPU
5. 数据情况：用户行为数据每日新增200G，约2亿条数据，每条数据120个字段
6. 使用SparkCore清洗时，使用On Yarn模式，如何设置job资源
7. 首先确认--executor-cores，官方建议2-5个，超过5个时不会有性能的明显提升，但是却占用了更多的cpu资源，此处我们设置为4
8. 再确认--executor-memory，因为上面说了比例为1：3，所以此处设置为12G
9. 最后确认--num-executors，而此项又需要根据提交任务所在队列的最大资源和多少应用程序并行执行，比如Yarn中只有4个队列，各占25%，每个队列需要并发执行4个任务，所以任务可以获取45线程CPU和135G内存，而上面2点我们已经确认了每个executor需要的cpu和内存，直接相除，向下取整即可，所以executor数量在此场景下设置为10最合适
10. 使用上面的资源提升任务时，比如需求是清洗200G数据，大概耗时5-10分钟
11. Flume集群：

第一层：采集层--与web server(10台)部署在一起，所以flume agent就是10个

第二层：汇聚--3台单独的服务器，32G内存，16线程CPU，4T \* 4

6）Kafka集群

3台32G 16线程CPU 4T \* 4

7）DELL品牌服务器，CPU为至强E5系列