《大数据综合实训实验指导书》

实验名称

基于 Flume + Kafka + Hive + Spark on YARN 的用户行为分析综合实训

一、实验目的

通过本综合实训，学生能够：

1. 理解大数据系统中多组件协同工作的整体流程
2. 掌握 Flume 采集日志数据的方法
3. 掌握 Kafka 在数据传输中的缓冲与解耦作用
4. 掌握 Hive 对用户行为数据的存储与管理
5. 掌握 Spark on YARN 的运行方式
6. 使用 Spark SQL 对用户行为进行统计分析
7. 理解 Spark Driver 与 Executor 在 YARN 上的运行位置

二、实验环境

组件	说明
Linux	master / slave1 / slave2
Hadoop	HDFS + YARN 已启动
Zookeeper	Kafka 依赖
Kafka	行为日志缓冲
Flume	日志采集
Hive	数据仓库
Spark	分布式计算（使用 Spark-Shell）

三、实验背景说明

在真实企业环境中，用户的浏览、收藏、加购等行为通常以日志形式产生。
这些数据需要经过 **采集 → 传输 → 存储 → 分析**，才能形成有价值的统计结果。

本实验模拟一个简化的电商用户行为分析场景，通过多个大数据组件的协同使用，完成一次完整的数据处理流程。

《大数据综合实训实验指导书（学生版）》_01.jpg

四、实验整体流程

```
用户行为日志文件
        ↓
     Flume
        ↓
     Kafka
        ↓
 Spark on YARN
        ↓
     Hive 表
        ↓
 Spark SQL 统计分析
```

五、实验步骤

5.1 启动环境

```
# 一、检查hadoop的环境
# 1.在master启动hdfs
start-dfs.sh
# 2.在slave1启动yarn
start-yarn.sh

# 二、启动zookeeper
# 在三个节点分别输入以下命令
zkServer.sh start

# 三、检查三个节点是否安装Kafka，并查看当前的主题有哪些
# 1.在三个节点分别启动以下命令
kafka-server-start.sh -daemon /opt/apps/kafka/config/server.properties
# 2.在三个节点分别检查kafka进程
jps
```

5.2 Kafka操作

5.2.1Kafka 创建 Topic

在 **slave1** 节点执行：

```
kafka-topics.sh \
--create \
--topic user_behavior \
--bootstrap-server master:9092 \
--partitions 3 \
--replication-factor 1
```

验证 Topic 是否创建成功：

```
kafka-topics.sh --list --bootstrap-server master:9092
```

5.2.2 启动 Kafka 消费者（用于验证）

在 **slave1 节点**上开启消费者，实时监听 `mylog` 主题：

```
# 在slave1运行Kafka自带的消费者脚本，监听mylog主题
kafka-console-consumer.sh \
--bootstrap-server master:9092,slave1:9092,slave2:9092 \
--topic user_behavior
```

⚠️ **不要关闭这个终端窗口**，后面验证数据是否成功发送全靠它来显示。

5.3 Flume 采集日志并发送到 Kafka

5.3.1 编写 Flume 配置文件

在master中创建配置文件：

```
cd /opt/apps/flume/conf
# 创建空文件
touch flume_kafka.conf
```

配置内容如下：

```
##########################################
# 1）定义 Flume Agent 的组件名称
##########################################
# a1：Agent 名称（固定写法）
# s1：Source（输入端）
# c1：Channel（通道）
# k1：Sink（输出端）
a1.sources = s1
a1.channels = c1
a1.sinks = k1

##########################################
```

```
# 2）配置 Source：spooldir（监听目录、读取新文件）
########################################
# Source 类型：spooldir
# 作用：监听一个目录，只要放入“新文件”，Flume 就会自动读取文件内容并发送
a1.sources.s1.type = spooldir
# 被监听的目录
# 注意：目录必须存在；建议开始实验前目录尽量为空
a1.sources.s1.spoolDir = /opt/data/flume/logs/2025-12-18
# 处理完成后的文件后缀
# 作用：文件读完后会被重命名为 *.done，避免重复读取
a1.sources.s1.fileSuffix = .done


########################################
# 3）配置 Channel：memory（内存通道，缓存数据）
########################################
# Channel 类型：memory
# 作用：Source 读到的数据先进入 Channel 缓冲，再由 Sink 取走发送
a1.channels.c1.type = memory
# Channel 最大可缓存事件数，防止数据突增时撑爆内存
a1.channels.c1.capacity = 1000
# 每次事务从 Channel 取/放的最大事件数
a1.channels.c1.transactionCapacity = 100


########################################
# 4）配置 Sink：KafkaSink（写入 Kafka）
########################################
# Sink 类型：KafkaSink
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
# Kafka 连接地址（新版写法，推荐）
# 作用：指定 Kafka broker 列表（一个也可，多台更稳）
a1.sinks.k1.kafka.bootstrap.servers = master:9092
# 写入 Kafka 的目标 Topic
a1.sinks.k1.kafka.topic = user_behavior
# 每批发送多少条 event 到 Kafka（提高吞吐）
a1.sinks.k1.batchSize = 20


########################################
# 5）绑定关系：Source -> Channel -> Sink
########################################
# Source 采集到的数据进入哪个 Channel
a1.sources.s1.channels = c1
# Sink 从哪个 Channel 取数据
a1.sinks.k1.channel = c1
```

5.3.2 启动 Flume Agent

（1）在 master 节点上创建监听目录：

```
mkdir -p /opt/data/flume/logs/2025-12-18
```

（2） 启动Flume

```
# 在master中启动flume
flume-ng agent \
-n a1 \
-c /opt/apps/flume/conf \
-f /opt/apps/flume/conf/flume_kafka.conf \
-Dflume.root.logger=INFO,console
```

注：保留该终端窗口，观察 Flume 日志输出情况，便于出错时排查问题。

当日志文件被读取后，文件名会自动加上 `.done` 后缀。

（3） 在 slave2 上准备源文件

在 **slave2 节点**上操作：

```
mkdir -p /opt/data/flume/logs

# 在 /opt/data/flume/logs 目录中创建 user_behavior.log 文件
touch /opt/data/flume/logs/user_behavior.log
```

示例内容：

```
u001,browse
u001,browse
u001,cart
u002,browse
u002,collect
u003,browse
u003,browse
u003,collect
u003,cart
```

（4） 将文件复制到 Flume 监听目录（master）

> 👉 **将日志文件复制到 Flume 的监听目录，触发 SpoolDir Source 自动检测并读取该文件，把内容发送到 Kafka。**

```
# 在slave2上执行：拷贝文件到master节点Flume监听的目录，模拟日志变化
scp /opt/data/flume/logs/user_behavior.log  master:/opt/data/flume/logs/2025-12-18
```

> 这一刻起，Flume 的 `SpoolDir Source` 会检测到 `master:/opt/data/flume/logs` 目录中新出现了一个 **完整文件**，并开始读取其中内容发送到 Kafka。

5.4 Hive 创建原始行为表（ODS 层）

```
# 新建一个master窗口
# 启动Hive的元数据服务
hive --service metastore &

# 启动hive客户端
hive
```

创建表：

```
-- 查看数据库
show databases;

-- 选择数据库
use default;

-- 创建表
CREATE TABLE ods_user_behavior (
  user_id STRING,
  action STRING
)
STORED AS PARQUET;

-- 查看表,是否有ods_user_behavior 表
show tables;
```

> 举例：PARQUET ['pɑ:keɪ] 列式存储文件格式
>
> CSV 格式（行式存储）
>
> > 1,张三,20,北京
> > 2,李四,21,上海
> > 3,王五,20,北京
>
> PARQUET：列式存储文件格式
>
> > id ：1 | 2 | 3
> > name：张三 | 李四 | 王五
> > age ：20 | 21 | 20
> > city：北京 | 上海 | 北京

5.5 Spark on YARN 读取 Kafka 数据并写入 Hive（核心）

（1）在 master 节点启动 Spark Shell：

```
# 用“低内存 + 低并发”方式启动 spark-shell
spark-shell \
```

```
--master yarn \
--deploy-mode client \
--driver-memory 512m \
--conf spark.driver.maxResultSize=256m \
--conf spark.executor.instances=1 \
--conf spark.executor.cores=1 \
--conf spark.executor.memory=512m \
--conf spark.sql.shuffle.partitions=2 \
--conf spark.sql.catalogImplementation=hive \
--conf spark.driver.extraClassPath=/opt/apps/hive/conf \
--conf spark.executor.extraClassPath=/opt/apps/hive/conf \
--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.1


# ========== 命令解释 ==============================================
spark-shell \                                                    # 启动 Spark 交互式 Shell
  --master yarn \                                                # 指定使用 YARN 作为资源管理器
  --deploy-mode client \                                         # client 模式：Driver 运行在当
前 master 节点
  --driver-memory 512m \                                         # 设置 Driver 可用内存为 512MB
  --conf spark.driver.maxResultSize=256m \                       # 限制 Driver 返回结果的最大大
小，防止内存溢出
  --conf spark.executor.instances=1 \                            # Executor 实例数量（教学环境设
为 1）
  --conf spark.executor.cores=1 \                                # 每个 Executor 使用 1 个 CPU
核心
  --conf spark.executor.memory=512m \                            # 每个 Executor 使用的内存大小
  --conf spark.sql.shuffle.partitions=2 \                        # Spark SQL shuffle 分区数，减
少资源消耗
  --conf spark.sql.catalogImplementation=hive \                  # 启用 Hive Catalog，允许访问
Hive 元数据
  --conf spark.driver.extraClassPath=/opt/apps/hive/conf \       # 为 Driver 加载 Hive 配置
（hive-site.xml）
  --conf spark.executor.extraClassPath=$/opt/apps/hive/conf \    # 为 Executor 加载 Hive 配置
  --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.1    # 引入 Spark-Kafka 连接器依赖
```

> 本实验通过 `--packages` 参数引入 Spark Kafka 连接器，Spark 会自动从 Maven 仓库下载所需依赖，并在 YARN 环境中将其分发到各 Executor 容器中，从而支持 Spark SQL 读取 Kafka 数据源。

---

(2) 使用 Spark 从 Kafka 订阅 user_behavior 主题数据：

```
// 导入隐式转换（DataFrame / Dataset 常用）
// 作用：支持 $"列名"、toDF 等写法（虽然本实验暂时不用，但属于规范写法）
import spark.implicits._

// 从 Kafka 订阅 user_behavior 主题，读取成 DataFrame
val kafkaDF = (
  spark.read
    .format("kafka")
    .option("kafka.bootstrap.servers", "master:9092")
    .option("subscribe", "user_behavior")
    .load()
    )
```

> **结果：** kafkaDF: org.apache.spark.sql.DataFrame = [key: binary, value: binary ... 5 more fields]
>
> **解释：** `kafkaDF` 是 Spark 从 Kafka 读取到的 DataFrame，
> 每一行对应一条 Kafka 消息，包含 key/value（默认 binary）以及 topic/partition/offset 等元数据。

(3) 查看 Kafka 原始数据的结构与内容

```
// 👉 查看 Kafka 读取后的表结构
// 查看表结构：Kafka 源会带 key/value/topic/partition/offset 等字段
kafkaDF.printSchema()

// 👉 查看前 10 条数据（此时 value 还是二进制）
// false 表示“不截断”，长字符串也完整显示。
kafkaDF.show(10, false)
```

(4) 解析 Kafka 消息内容，将 value 转换为字符串

```
// value 是消息正文（生产者发送的内容），Kafka 读出来默认是 binary
// CAST(value AS STRING)：把二进制转为可读字符串
// AS line：把这一列命名为 line，方便后续 split
val lines = kafkaDF.selectExpr("CAST(value AS STRING) AS line")

// 查看转换结果（此时 line 应该是类似：u001,browse）
lines.show(10, false)
```

(5) 拆分消息字段，构建结构化用户行为数据

```
// 假设消息格式：user_id,action
// 例如：u001,browse
// split(line, ',')[0]：取第 1 段作为 user_id
// split(line, ',')[1]：取第 2 段作为 action
val dataDF = lines.selectExpr(
  "split(line, ',')[0] AS user_id",
```

```
  "split(line, ',')[1] AS action"
)

// 查看拆分后的结构化结果
dataDF.show(10, false)

// 查看字段类型（应为 user_id/action 两列 string）
dataDF.printSchema()
```

(6) 创建 Hive ODS 层用户行为表（Parquet 存储）

```
// 切换到 default 数据库
spark.sql("USE default")

// 创建 ODS 层表：存原始行为数据（结构化后落地）
// STORED AS PARQUET：列式存储，查询更快、压缩更好
spark.sql("""
  CREATE TABLE IF NOT EXISTS ods_user_behavior (
    user_id STRING,
    action  STRING
  )
  STORED AS PARQUET
""")

// 验证表是否创建成功
spark.sql("SHOW TABLES").show(false)
```

(7) 将结构化数据写入 Hive 并进行基础行为分析

```
// coalesce(1)：把分区合并为 1 个分区（方便实验查看文件，但数据大时不建议）
// mode("append")：追加写入（不覆盖原数据）
// insertInto：写入已存在的 Hive 表（字段顺序要匹配表结构）
dataDF.coalesce(1).write.mode("append").insertInto("ods_user_behavior")

// 查看表前 10 行
spark.sql("SELECT * FROM ods_user_behavior LIMIT 10").show(false)

// 统计每种行为出现次数
spark.sql("""
  SELECT action, COUNT(*) AS cnt
  FROM ods_user_behavior
  GROUP BY action
""").show(false)
```

5.6 Spark SQL 用户行为统计分析

在 spark-shell 中执行：

```
// 5.6.1 基于 ODS 明细数据，统计每种用户行为的次数
// groupBy("action")：按用户行为分组
// count()：统计每种行为出现的次数
val summaryDF = dataDF.groupBy("action").count()

// 5.6.1 查看统计结果
summaryDF.show(false)
// 查看统计结果结构
summaryDF.printSchema()
```

5.7 保存统计结果表（DWS 层）

在 spark-shell 中执行：

```
// 保存统计结果表
summaryDF.write.mode("overwrite").saveAsTable("dws_user_behavior_summary")

// 查看表内容
spark.sql("SELECT * FROM dws_user_behavior_summary").show(false)
```

六、实验结果示例

用户ID	浏览次数	收藏次数	加购次数
u001	2	0	1
u002	1	1	0
u003	2	1	1

七、实验结论

> 本实验通过 Flume、Kafka、Hive 和 Spark on YARN 构建了完整的数据处理流程，实现了用户行为日志的采集、存储与分析。实验过程中未使用 PySpark，而是采用 Spark 原生的 Scala 接口完成计算，加深了对 Spark on YARN 运行机制的理解。

八、实验提交要求

1. Flume 配置文件
2. Kafka Topic 创建截图
3. Hive 建表 SQL
4. Spark on YARN 执行过程截图
5. 用户行为统计结果与分析说明