spark用于解决 **Hadoop MapReduce 缺乏通用性与速度慢** 的问题，支持多种编程语言，支持比hadoop跟高级的api，执行速度更快，（内存计算），可独立运行支持批处理，流处理，sql查询，图计算，机器学习

核心组件：

spark core 提供rdd api，任务调度，内存管理等。

spark sql 处理结构化数据， dataframe

spark streaming 实时数据流处理（基于dstream 或 structured streaming）

mlbib machine learninglibrary机器学习裤（分类，聚类，回归）

RDD

不可变，可分区，容错机制，惰性求值

DataFrame

分布式计算、不可变 （不可修改，只能通过转换生成新的dataframe），更适合并行计算，支持多种数据源，与mllib streaming无缝接合，

Kafka 是一个**高吞吐、分布式、可扩展**的**消息队列系统**，用于**实时数据的发布、**订阅、存储与处理。Kafka 是大数据和实时系统的“高速公路”，用于连接各种系统，让数据**实时、稳定、高效**地流动。

| **比较项** | **Spark** | **MapReduce（Hadoop）** |
| --- | --- | --- |
| **计算模型** | 内存计算（RDD） | 磁盘计算（Map → Shuffle → Reduce） |
| **执行速度** | 极快（通常比 MapReduce 快 10-100 倍） | 慢（每一步都写磁盘） |
| **容错机制** | RDD 有 lineage，支持自动容错 | 基于任务重试 |
| **API 简洁性** | 高层抽象（如 DataFrame、Spark SQL、MLlib） | 低层次，需写 Map 和 Reduce 函数 |
| **迭代计算支持** | 支持，内存缓存 RDD | 不支持，需每次重新从磁盘读写 |
| **内存使用** | 内存为主，也可溢写磁盘 | 磁盘为主，几乎不使用内存缓存 |
| **容错策略** | 基于 RDD 血统重建丢失数据 | 任务失败后重新运行该任务 |
| **交互性** | 支持交互式查询（如 Spark Shell） | 不支持，批处理为主 |
| **使用语言** | 支持 Scala、Python、Java、R、SQL | 主要是 Java |
| **机器学习支持** | 强大（MLlib） | 弱，仅依赖 Mahout |
| **流处理能力** | 有（Spark Streaming / Structured Streaming） | 弱，需要额外组件如 Apache Storm |
| **生态系统** | 更现代，集成图计算（GraphX）、结构化数据 | 传统，缺乏结构化计算、高级 API |

Spark 是内存驱动的通用分布式计算框架，更适合实时、迭代、高速分析；而 MapReduce 是基于磁盘的传统批处理框架，适合一次性大规模处理但效率较低。

|  |
| --- |
| Zookeeper |

‘实验六

|  |  |
| --- | --- |
| Kafka 的元数据协调器（可选，3.x版本以后支持无Zookeeper） | |
| **步骤** | **含义** | |
| .filter("tag = 'CRON'") | 只处理 CRON 进程的日志 | |
| .withWatermark("timestamp", "1 minutes") | 设置水印，容忍 1 分钟延迟数据 | |
| .groupBy(window(...)) | 按 1 小时时间窗口分组 | |
| .count() | 统计每小时 CRON 日志数量 | |
| .sort(...) | 按时间先后排序窗口结果 | |
| **参数** | **含义** | |
| outputMode("complete") | 每次输出整个窗口统计表 | |
| format("console") | 输出到控制台 | |
| trigger(processingTime="3 seconds") | 每 3 秒处理一次流 | |
| .start() | 启动流处理流程 | |
| **withWaterMark作用** | **说明** | |
| ⏳ **允许处理迟到数据** | 指定多久以前的数据被认为是“太迟了”，丢弃 | |
| 📊 **减少状态保存压力** | 避免无限期保留状态，提升内存与性能表现 | |
| ⛔ **控制窗口聚合范围** | 保证基于时间窗口的聚合在延迟数据允许范围内处理 | |

withWatermark("event\_time", "delay") 让你可以处理一定程度延迟到达的数据，同时自动清理过期状态，保证性能。

streamingQuery 是 **PySpark Structured Streaming** 中用于 **控制和监控流式查询执行状态** 的对象，它是你在调用 .writeStream.start() 启动查询时返回的 StreamingQuery **实例**。

query 就是一个 **StreamingQuery** 对象

StreamingQuery 是你与 Spark Structured Streaming 查询交互的“控制句柄”，你可以用它控制、监控、诊断整个流处理的执行过程。

MLlib

| **类型** | **示例算法** |
| --- | --- |
| 分类 | Logistic Regression、Random Forest、GBT、Naive Bayes |
| 回归 | Linear Regression、Decision Tree Regression、GBT |
| 聚类 | KMeans、BisectingKMeans、GaussianMixture |
| 降维 | PCA、SVD |
| 推荐系统 | ALS（协同过滤） |
| 特征处理 | Tokenizer、HashingTF、StandardScaler、VectorAssembler 等 |
| 模型评估 | BinaryClassificationEvaluator、RegressionEvaluator、CrossValidator |

《实时大数据分析》课程答辩知识点详解文档（完整版）

## 一、Linux 系统及命令基础

Linux 是当前大数据平台的标准部署环境，其稳定性和可定制性强。

1. 常见命令分类：

（1）目录与文件管理：

\* `cd [目录]`：切换目录

\* `mkdir [目录名]`：创建目录

\* `rm -r [路径]`：删除文件或目录（加 -r 为递归）

\* `cp 源 目标`：复制文件/目录

\* `mv 源 目标`：移动或重命名

（2）压缩解压：

\* `tar -czf file.tar.gz dir/`：打包压缩

\* `tar -xzf file.tar.gz`：解压

（3）权限管理：

\* `chmod 755 file`：修改文件权限

\* `chown 用户:用户组 file`：修改文件所有者

（4）查找与查看内容：

\* `find / -name "文件名"`：查找

\* `grep '关键字' 文件`：按行匹配关键字

## 二、HDFS 与 Spark 基础

1. HDFS（Hadoop Distributed File System）：分布式文件系统。

\* 特点：适合存储海量、大文件，具备容错性（数据多副本存储）

\* 常用命令：

\* `hdfs dfs -mkdir /user/hadoop`：创建目录

\* `hdfs dfs -put 本地路径 HDFS路径`：上传文件

\* `hdfs dfs -get HDFS路径 本地路径`：下载文件

\* `hdfs dfs -ls`：查看目录

\* `hdfs dfs -rm -r 路径`：递归删除

2. SparkContext：创建 Spark 应用的入口，连接 Spark 集群并提交任务

```python

sc = SparkContext("local", "MyApp")

rdd = sc.textFile("file:///.../data.txt")

```

## 三、RDD 编程与核心算子

RDD 是弹性分布式数据集，支持容错与并行操作。

1. 转换算子（Transformation）：延迟执行

\* `map(f)`：每个元素作用函数 f

\* `filter(f)`：过滤符合条件的元素

\* `flatMap(f)`：返回迭代器展开所有子元素

\* `distinct()`：去重

\* `reduceByKey(f)`：用于 pair RDD，按 key 聚合

2. 行动算子（Action）：触发执行

\* `collect()`：收集所有数据到 driver

\* `count()`：计数

\* `take(n)`：取前 n 个

\* `saveAsTextFile("路径")`：输出到 HDFS/本地

## 四、Spark SQL 与 DataFrame 操作

DataFrame 是带 schema 的结构化数据表，适合 SQL 查询。

1. 创建 DataFrame：

\* 从 CSV 文件：

```python

df = spark.read.format("csv").option("header", "true").load("file:///.../data.csv")

```

\* 从 RDD：

```python

from pyspark.sql import Row

rdd = sc.textFile("...").map(lambda x: Row(...))

df = spark.createDataFrame(rdd)

```

2. 主要操作：

\* `select("col")`：选列

\* `filter("col > 10")` 或 `where()`：条件筛选

\* `groupBy("col").count()`：分组聚合

\* `withColumn("新列", 函数(col))`：新增列

\* `join(df2, on="key")`：连接

## 五、Spark Streaming 与 Flume 集成

Spark Streaming 提供基于时间窗口的实时计算能力。

1. 核心流程：

```python

ssc = StreamingContext(sc, 2) # 每2秒一个批次

lines = ssc.socketTextStream("localhost", 9999)

words = lines.flatMap(lambda line: line.split(" "))

```

2. Flume 是日志采集工具，可将数据实时发送到 Spark

\* 配置结构包含 Source（如 netcat）、Channel（内存）、Sink（Logger）

\* 示例：监听 33333 端口接收日志并通过 avro 发往 Spark

3. 使用 FlumeUtils 接收：

```python

stream = FlumeUtils.createStream(ssc, "localhost", 44444)

```

## 六、Structured Streaming 流式分析

Structured Streaming 是基于 DataFrame/Dataset 的流处理框架。

1. 数据输入：

```python

df = spark.readStream.format("socket").option("host", "localhost").option("port", 9988).load()

```

2. 时间处理与水印机制：

\* `withWatermark("timestamp", "1 minute")`：处理迟到数据

\* `window("timestamp", "1 hour")`：窗口聚合

3. 输出控制：

\* `outputMode`: append（追加）、complete（全输出）、update（只输出变化）

\* `.writeStream.format("console").start()`：控制台输出

## 七、机器学习 MLlib 模块

MLlib 是 Spark 的机器学习库，支持分类、回归、聚类等。

1. 特征工程：

\* `Vectors.dense([...])`：构建向量

\* `StringIndexer`：标签编码（字符串 → 数字）

\* `VectorAssembler`：组合多列为特征向量

\* `StandardScaler`：特征标准化（均值为0，方差为1）

2. 模型常用对象：

\* `LogisticRegression`：逻辑回归分类

\* `DecisionTreeClassifier`：决策树

\* `Pipeline`：整合步骤

## 八、PCA 主成分分析（降维）

PCA 用于高维数据降维，提取主要信息，消除冗余。

1. 示例：

```python

pca = PCA(k=3, inputCol="features", outputCol="pcaFeatures")

model = pca.fit(df)

result = model.transform(df)

```

2. 优点：降维可提升训练效率、防止过拟合

## 九、逻辑回归建模与预测

逻辑回归是处理二分类问题（如收入 >50K）的经典方法。

1. 模型初始化与训练：

```python

lr = LogisticRegression(maxIter=100, labelCol="indexedLabel", featuresCol="indexedFeatures")

model = lr.fit(train)

```

2. 模型参数：

\* `coefficients`：特征权重

\* `intercept`：偏置项

3. 模型评估：

```python

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction")

accuracy = evaluator.evaluate(predictions)

```

## 十、Pipeline 机器学习流水线

Pipeline 用于将多个数据处理步骤串联执行。

1. 示例：

```python

pipeline = Pipeline(stages=[indexer, pca, lr])

model = pipeline.fit(train)

pred = model.transform(test)

```

2. 作用：保持预处理 + 模型一致性，简化重复流程

## 十一、交叉验证与参数调优

使用 `CrossValidator` 搜索最优超参数组合，防止过拟合。

1. 构建参数网格：

```python

paramGrid = (ParamGridBuilder()

.addGrid(pca.k, [2,3,4])

.addGrid(lr.regParam, [0.01, 0.1])

.addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])

.build())

```

2. 构建验证器并训练：

```python

cv = CrossValidator(estimator=pipeline, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=3)

cvModel = cv.fit(train)

```

3. 提取最佳参数：

```python

best\_lr = cvModel.bestModel.stages[2]

print(best\_lr.getRegParam(), best\_lr.getElasticNetParam())

```

## 十二、答辩常见问题与建议回答

1. Q：为什么使用 PCA？

A：可以去除特征间的多重共线性，减少噪声，加速模型训练。

2. Q：Structured Streaming 与 Spark Streaming 区别？

A：Structured 基于 DataFrame API，支持水印/容错，语法更现代，性能更高。

3. Q：交叉验证的目的？

A：提升模型泛化能力，避免在特定训练集上的过拟合。

4. Q：什么时候用逻辑回归 vs 决策树？

A：逻辑回归适合线性问题，决策树适合非线性且可解释性要求高的问题。

5. Q：你实现了哪些优化手段？

A：使用 PCA 降维减少特征维度，使用 CrossValidator 自动调参，提升准确率。

——

建议：答辩中可展示实验运行截图、解释模型参数、比较结果变化，体现对工具和原理的理解。祝你答辩成功！