Slide 1:

Slide 2: .

Slide 3:

Slide 4:**목차**

1. RDD의 설계 배경
2. RDD의 기본 개념
3. RDD의 특성
4. RDD 간의 의존 관계
5. 단계의 분할
6. RDD의 완전한 운행 과정

Slide 1: **RDD概念、操作和特性**

Slide 2: **RDD概念、操作和特性**

Slide 3: **RDD概念、操作和特性**

Slide 4: 课程提纲(이번 시간은 아래 2개내용)

今天的课程安排如下：

首先我们会了解RDD的设计背景，

学习RDD的基本概念，

接着深入RDD的特性，

之后讨论RDD之间的依赖关系，然后学习阶段的划分，

最后了解RDD的完整运行过程。通过这六个部分，大家将全面掌握RDD的工作原理。

Slide 5:**RDD概念、操作和特性**

먼저 RDD의 설계 배경을 살펴보겠습니다. MapReduce 시대에 우리는 반복 시나리오 처리의 도전에 직면했습니다.

예를 들어 로지스틱 회귀, 시뮬레이티드 어닐링 알고리즘, 유전 알고리즘 등은 모두 여러 번의 반복 계산이 필요한 알고리즘들입니다.

Slide 6: **MapReduce의 한계**

전통적인 MapReduce는 반복 계산 처리 시 명백한 한계가 있었습니다:

- 매번 계산할 때마다 디스크에서 데이터를 읽어야 함
- 쿼리 계산 수행
- 결과를 다시 디스크에 쓰는 과정 필요
- 반복적인 디스크 I/O 작업으로 계산 효율성 저하

Slide 5: **RDD 概念、操作和特性**

让我们先来看看RDD的设计背景。

在MapReduce时代，我们面临着迭代场景处理的挑战(tiǎozhàn)。

比如逻辑斯蒂回归、模拟退火算法、遗传算法等，这些算法都需要进行多次迭代计算。

Slide 6: MapReduce的局限性

传统的MapReduce在处理迭代计算时存在明显的局限性。

- 每次计算都需要从磁盘读取数据，
- 进行查询计算，
- 然后将结果写回磁盘。
- 这种反复的磁盘I/O操作大大降低了计算效率。

Slide 7: **1.3 반복 계산의 문제점**

반복 계산이 필요할 때 MapReduce의 문제는 더욱 두드러졌습니다:

- 매번 반복할 때마다 작업 서브셋을 반복적으로 읽고 쓰기
- 막대한 디스크 IO 오버헤드 발생
- 직렬화/역직렬화 오버헤드로 성능에 심각한 영향

Slide 8:**1.4 직렬화와 역직렬화**

여기서 직렬화의 개념을 이해해야 합니다. Java 객체를 디스크에 저장할 때는 먼저 이진 또는 문자열 형식으로 변환해야 하는데, 이 과정이 바로 직렬화입니다. 이 변환 과정은 대량의 계산 자원을 소모합니다

Slide 9: **직렬화(Serialization)**

직렬화는 객체를 저장 및 전송 가능한 형식으로 변환하는 과정입니다.
보시다시피, 객체는 직렬화를 통해 저장 및 전송 가능한 형식으로 변환되며, 이 과정이 반복 계산에서 반복적으로 실행되어 성능 병목을 야기합니다..

Slide 10:**역직렬화(Deserialization)**

그 반대 과정을 역직렬화라고 하며, 저장 및 전송 형식에서 다시 객체를 생성하는 것입니다. MapReduce의 반복 계산에서는 이러한 직렬화와 역직렬화가 반복적으로 수행되어 효율성에 심각한 영향을 미쳤습니다.

Slide 7: MapReduce迭代计算的问题

当我们需要进行迭代计算时，MapReduce的问题就更加突出(**tūchū**)了。

- 每次迭代都要反复读写工作子集，
- 这带来了巨大的磁盘IO开销
- 序列化反序列化开销，严重影响了性能。

Slide 8: 序列化问题

这里我们需要理解序列化的概念。
当Java对象需要存储到磁盘时，必须先转化为二进制或字符串格式，这个过程就是序列化。
这个转换过程会消耗(**xiāohào)**大量的计算资源。

Slide 9:序列化过程详解

序列化是将对象转化为可保存和传输格式的过程。
我们可以看到，对象通过序列化转化为可保存和传输的格式，这个过程在迭代计算中会被反复执行，造成性能瓶颈(**píngjǐng)**。

Slide 10: 反序列化过程

相反的过程叫做反序列化，即从可保存和传输格式重新生成对象。
在MapReduce的迭代计算中，
这种序列化和反序列化会反复进行，
严重影响效率。

Slide 11: RDD的解決方案

이러한 문제들을 해결하기 위해 RDD가 등장했습니다. RDD는 추상적인 데이터 구조를 제공하여 전통적인 방법의 각종 단점을 효과적으로 피할 수 있습니다.

Slide 11: RDD的解决方案

为了解决这些问题，RDD应运而生。
RDD提供了抽象的数据结构，
可以有效避免传统方法的种种弊端
(**bìduān)**。

Slide 12: **2.2 RDD의 핵심 아이디어**

RDD의 핵심 아이디어는 **변환을 통해 새로운 RDD를 생성**하는 것입니다.
하나의 RDD는 다양한 변환 작업을 통해 다른 RDD를 생성할 수 있으며, 이는 처리 체인을 형성합니다.

Slide 12: RDD转换概念

RDD的核心思想是通过转换来生成新的RDD。
一个RDD可以通过各种转换操作生成另一个RDD，形成一个处理链**(liàn)**。

Slide 13:**2.3 RDD의 의존 관계와 DAG**

RDD들 간에는 의존 관계가 존재하며,
이러한 의존 관계는 **DAG(방향성 비순환 그래프)**를 구성합니다.

A부터 E까지의 각 RDD 노드들 간에는 명확한 의존 관계가 존재하며, 이러한 구조로 인해 Spark는 효율적인 작업 스케줄링과 최적화를 수행할 수 있습니다.

Slide 13: RDD依赖关系和DAG图

RDD之间存在依赖关系，
这些依赖关系构成了DAG（有向无环图）。

如图所示，从A到E的各个RDD节点之间存在清晰的依赖关系，
这种结构使得Spark可以进行高效的任务调度和优化。

Slide 14:**2.4 파이프라인화 최적화**
RDD의 중요한 특성 중 하나는 **파이프라인화(스트림라인화)**를 지원한다는 것입니다:

- 작업1에서 생성된 데이터는 메모리에서 직접 작업2로 전달
- 중간 결과를 디스크에 저장하는 오버헤드 회피
- 데이터는 메모리에서 직접 흘러가며 처리 효율성 크게 향상

Slide 15: 避免数据落地
파이프라인화를 통해 RDD는 데이터를 디스크나 다른 저장 장치에 저장할 필요를 피할 수 있습니다. 데이터는 메모리에서 직접 작업1에서 작업2로 흘러가며, 이는 처리 효율성을 크게 향상시킵니다.

Slide 16: RDD的分布式特性
변환 작업은 궤적만을 기록하고, 액션 작업이 계산을 트리거합니다.
예를 들어 .count() 작업은 레코드 수를 통계하며, 이때 실제 계산이 수행됩니다.

Slide 17: **2.5 RDD의 본질적 정의**
RDD는 본질적으로 읽기 전용 파티션 레코드 집합입니다.
이것이 RDD의 가장 핵심적인 정의이며, 이를 이해하는 것은 RDD를 파악하는 데 매우 중요합니다.

Slide 14: 管道化优化
RDD的一个重要特性是支持管道化（流水线化）。

操作1产生的数据可以直接在内存中传递给操作2，
避免了中间结果落地到磁盘的开销。

Slide 15:避免数据落地
通过管道化操作，RDD可以避免将数据存储在磁盘或其他存储设备中。数据直接从内存中由作业1流向作业2，这大大提高了处理效率。

Slide 16: RDD的分布式特性
RDD本质上是一个分布式对象的集合。
它由多个分布式对象组成，
可以在集群中并行处理

Slide 17: RDD的本质定义
RDD本质上是一个只读的分区记录集合。

这是RDD最核心的定义，
理解这一点对掌握RDD非常重要。

Slide 18: 从数据集到RDD

데이터셋 파일은 Spark 처리를 통해 읽기 전용 파티션 집합인 RDD를 생성합니다. 이 변환 과정은 Spark 데이터 처리의 시작점입니다.

Slide 18: 从数据集到RDD

数据集文件通过Spark处理后，会生成RDD这种只读的分区集合。

这个转换过程是Spark数据处理的起点。

Slide 19: RDD的分区结构

RDD는 파티션1, 파티션2 등과 같은 여러 파티션으로 구성됩니다.

이러한 분산 객체 집합의 설계로 인해 RDD는 여러 머신에서 병렬 처리가 가능합니다.

Slide 19: RDD的分区结构

RDD由多个分区组成，如分区1、分区2等等。

这种分布式对象集合的设计使得RDD可以在多台机器上并行处理。

Slide 20: RDD的并行处理

RDD의 데이터는 서로 다른 파티션에 분포되어 있으며, 각 파티션은 서로 다른 노드에서 분산 병렬 처리될 수 있어 효율적인 병렬 계산을 실현할 수 있습니다.

Slide 20: RDD的并行处理

RDD的数据分布在不同的分区中，

每个分区可以在不同的节点上进行分布式并行处理，

从而实现高效的并行计算。

Slide 21: 多个数据片段的集合

RDD를 여러 데이터 조각의 집합으로 이해할 수 있습니다. 각 파티션은 일부 데이터를 포함하고 있으며, 모든 파티션이 합쳐져서 완전한 데이터셋을 구성합니다.

Slide 21: 多个数据片段(**piànduàn**)的集合

可以将RDD理解为多个数据片段的集合。

每个分区包含一部分数据，

所有分区合在一起构成完整的数据集。

Slide 22: 高度受限的共享内存模型

RDD는 고도로 제한된 공유 메모리 모델을 채택했습니다. 이러한 설계는 어느 정도의 제한이 있지만 많은 장점을 가져다줍니다.

Slide 22: 高度受限的共享内存模型

RDD采用了高度受限的共享内存模型。

这种设计虽然有一定的限制，

但却带来了很多优势。

Slide 23: 只读特性

RDD의 고도로 제한된 공유 메모리 모델의 핵심은 읽기 전용 특성입니다. RDD가 메모리에 로드되면 더 이상 수정할 수 없으며, 이것이 RDD의 기본 원칙입니다.

Slide 23: 只读特性

RDD的高度受限共享内存模型的核心是只读特性。
一旦RDD被加载到内存中，
就不能再修改，
这是RDD的基本原则。

Slide 24: 数据修改的实现方式

데이터를 수정해야 할 때(예: 모든 학생에게 5점 추가), RDD는 원본 데이터를 직접 수정하지 않고 수정된 결과를 포함하는 완전히 새로운 RDD를 생성합니다.

Slide 24: 数据修改的实现方式

当我们需要对数据进行修改时（比如给所有同学加5分），RDD不会直接修改原数据，
而是生成一个全新的RDD来包含修改后的结果。

Slide 25: 转换生成新RDD

RDD는 변환을 통해 새로운 RDD를 생성하여 데이터 수정의 목적을 달성합니다. 원본 RDD는 변경되지 않고, 새로운 RDD가 수정된 데이터를 포함합니다.

Slide 25: 转换生成新RDD

RDD通过转换生成新的RDD来完成数据修改的目的。
原始RDD保持不变，
新的RDD包含修改后的数据。

Slide 26: RDD的API支持

Spark는 풍부한 RDD API를 제공하며, 프로그래머는 이러한 API를 호출하여 변환, 필터링, 집계 등을 포함한 RDD의 다양한 작업을 구현할 수 있습니다.

Slide 26: RDD的API支持

Spark提供了丰富的RDD API，程序员可以通过调用这些API实现对RDD的各种操作，包括转换、过滤、聚合等。

Slide 27: RDD的典型执行过程

RDD의 전형적인 실행 과정은 세 단계를 포함합니다:
먼저 RDD가 외부 데이터 소스를 읽어 생성되고, 그 다음 일련의 변환 작업을 거치며 매번 다른 RDD가 생성되고, 마지막으로 액션 작업을 통해 변환되어 외부 데이터 소스로 출력됩니다.

Slide 27: RDD的典型执行过程

RDD的典型执行过程包括三个步骤：
首先RDD读入外部数据源进行创建，
然后经过一系列的转换操作，每一次都会产生不同的RDD，
最后通过动作操作进行转换并输出到外部数据源

Slide 28: RDD的操作类型

RDD는 두 가지 주요 작업 유형을 지원합니다: 변환 타입 작업(Transformation)과 액션 타입 작업(Action). 이 두 작업은 서로 다른 특징과 용도를 가지고 있습니다.

Slide 29: 转换操作的特点

변환 타입 작업의 특징은 변환의 궤적만을 기록하고 즉시 계산이 발생하지 않는다는 것입니다. 이러한 설계를 지연 호출 메커니즘이라고 합니다.

Slide 30: 转换操作示例

성적 데이터에 5점을 추가하는 변환 작업을 실행할 때, RDD는 이 작업을 기록할 뿐 실제로 계산을 실행하지는 않습니다.

Slide 31: 动作操作触发计算

액션 타입 작업을 만날 때만 처음부터 끝까지의 계산이 수행됩니다. 액션 작업은 전체 계산 플로우의 실행을 트리거합니다.

Slide 32: RDD提供的转换操作

RDD는 map(매핑), filter(필터링), groupBy(그룹화), join(조인) 등의 기본 작업을 포함한 풍부한 변환 작업을 제공합니다.

Slide 33: 粗粒度操作特性

RDD가 지원하는 것은 거친 입도(coarse-grained) 작업으로, 한 번에 RDD 전체 집합에 대해서만 변환할 수 있습니다. 예를 들어 60명의 학생 모두에게 5점을 주는 것은 가능하지만, 그 중 특정 학생에게만 점수를 주는 것은 불가능합니다

Slide 28: RDD的操作类型

RDD支持两种主要的操作类型：
转换类型操作（Transformation）和动作类型操作（Action）。
这两种操作有着不同的特点和用途。

Slide 29: 转换操作的特点

转换类型操作的特点是只记录转换的轨迹(**guǐjì**)，不会立即发生计算。
这种设计叫做惰性调用机制。

Slide 30: 转换操作示例

当我们对成绩数据执行加5分的转换操作时，RDD只是记录下这个操作，并不会真正地执行计算。

Slide 31: 动作操作触发计算

只有当遇到动作类型操作时，才会进行从头到尾的计算。
动作操作会触发(**chùfā**)整个计算流程的执行。

Slide 32: RDD提供的转换操作

RDD提供了丰富的转换操作，包括map（映射）、filter（过滤）、groupBy（分组）、join（连接）等基本操作。

Slide 33: 粗粒度操作特性

RDD支持的是粗粒度(**cū lìdù**)操作，一次只能针对RDD全集进行转换。
比如给60个同学全部加5分，不能只给其中某个(**mǒu gè**)同学加分。

Slide 34:粗粒度的局限性

이러한 거친 입도의 특성은 어느 정도의 한계를 가져옵니다. 예를 들어 데이터베이스에서 95001번 학생에게만 5점을 추가하고 싶다면, RDD는 이런 세밀한 입도의 수정을 할 수 없습니다.

Slide 35: 粗粒度修改的特点

Spark는 거친 입도 수정을 지원하여, 한 번에 RDD 전체 집합에 대해서만 변환할 수 있습니다. 이는 RDD 설계의 중요한 특징입니다.

Slide 36:不支持细粒度修改

RDD는 세밀한 입도 수정을 지원하지 않으므로, 단일 레코드에 대한 수정이 필요한 데이터베이스 애플리케이션 시나리오에는 적합하지 않습니다.

Slide 37: 表达能力的疑问

여러분은 의문을 가질 수 있습니다: 이런 고도로 제한된 공유 메모리 모델이 표현 능력에 영향을 미치지 않을까요?

Slide 38: 表达能力很强大

답은 그렇지 않다는 것입니다.
고도로 제한된 공유 메모리 모델을 채택했지만, RDD의 능력은 매우 강력하여 표현 능력에 영향을 미치지 않습니다.

Slide 34:粗粒度的局限性

这种粗粒度的特性也带来一定的局限性(**júxiànxìng**)。
比如数据库中如果只想给95001号学生加5分，RDD无法做到这种细粒度(**xì lìdù**)的修改。

Slide 35: 粗粒度修改的特点

Spark支持粗粒度修改，一次只能针对RDD全集进行转换。这是RDD设计的一个重要特征。

Slide 36:不支持细粒度修改

RDD不支持细粒度修改，因此不适合需要对单条记录进行修改的数据库应用场景。

Slide 37: 表达能力的疑问

大家可能会疑问：
这种高度受限的共享内存模型会不会影响表达能力呢

Slide 38: 表达能力很强大

答案是不会。
虽然采用了高度受限的共享内存模型，
但RDD的能力非常强大，
不会影响表达能力。

Slide 39: 与其他系统的对比

비교표를 살펴보겠습니다. MapReduce는 작업1에서는 좋은 성능을 보이지만 작업2와 작업3에서는 제한이 있습니다. PreGel은 작업2에서 좋은 성능을 보이며, Spark는 세 가지 작업 모두에서 우수한 성능을 보입니다.

Slide 40: Spark的完整流程

Spark와 Scala 언어를 사용하여 API를 통해 RDD에 대해 작업을 수행하고, 최종적으로 계산 결과를 얻습니다. 이 플로우는 간결하면서도 효율적입니다.

Slide 41: 从HDFS生成RDD

일반적으로 우리는 HDFS에서 데이터를 읽어 RDD를 생성하며, 이는 빅데이터 처리의 전형적인 시작점입니다.

Slide 42: RDD的不可变性

강조해야 할 점은 RDD 자체는 수정을 허용하지 않는다는 것입니다. 모든 변환은 새로운 RDD를 생성하며, 이는 RDD 설계의 핵심 원칙입니다.

Slide 43: 完整的数据流

완전한 데이터 플로우는 변환을 통해 RDD를 생성하고, 그 다음 액션 작업을 통해 최종 결과를 생성하는 것입니다.

Slide 39: 与其他系统的对比

我们来看一个对比表。
MapReduce在操作1上表现良好，但在操作2和操作3上有限制。
PreGel在操作2上表现好，
而Spark在所有三种操作上都表现优秀。

Slide 40: Spark的完整流程

使用Spark和Scala语言，通过API对RDD进行操作，最终得到计算结果。
这个流程简洁而高效。

Slide 41: 从HDFS生成RDD

通常情况下，我们从HDFS读取数据来生成RDD，
这是大数据处理的典型起点

Slide 42: RDD的不可变性

需要强调的是，RDD本身不允许修改。
任何转换都会生成一个新的RDD，
这是RDD设计的核心原则。

Slide 43: 完整的数据流

完整的数据流程是：
通过转换生成RDD，
然后通过动作操作产生最终结果

Slide 44: **지연 실행 메커니즘**

RDD는 지연 실행 메커니즘을 채택합니다. 변환 연산은 단지 실행 계획만 기록할 뿐이며, 액션 연산이 발생해야 비로소 계산이 실제로 실행됩니다. 예를 들어, .count() 연산은 레코드 수를 세는 것으로, 이때 비로소 실제 계산이 수행됩니다.

Slide 45: Lineage血缘关系

RDD는 Lineage 혈연 관계를 가지고 있어 Input에서 Output까지의 완전한 변환 체인을 기록합니다. 친족을 찾거나 데이터를 복구해야 할 때 이 혈연 관계를 통해 추적할 수 있습니다.

Slide 46: DAG和流水线优化

RDD의 의존 관계는 DAG 방향성 비순환 그래프를 구성하며, 파이프라인 최적화를 지원하여 전체적인 처리 효율성을 향상시킵니다.

Slide 47: 流水线优化的好处

파이프라인 최적화를 통해 작업1의 출력이 직접 작업2의 입력으로 사용될 수 있으며, 중간 결과를 디스크에 쓸 필요가 없어 처리 속도가 크게 향상됩니다.

Slide 48: MapReduce的复杂性

이에 비해 MapReduce는 애플리케이션이 복잡할 때 매우 복잡한 Map 함수와 Reduce 함수 코드를 작성해야 하므로 개발 난이도가 높습니다.

Slide 44: 惰性调用机制

RDD采用惰性调用机制。转换操作只记录轨迹，动作操作才触发计算。

比如.count()操作会统计记录条数，这时才会进行实际计算。

Slide 45: Lineage血缘关系

RDD具有Lineage血缘关系，记录了从Input到Output的完整转换链路。

当需要寻亲或恢复数据时，可以通过这个血缘关系进行追溯。

Slide 46: DAG和流水线优化

RDD的依赖关系构成DAG有向无环图，支持流水线优化，提高了整体的处理效率。

Slide 47: 流水线优化的好处

流水线优化使得操作1的输出可以直接作为操作2的输入，中间结果不需要写入磁盘，大大提高了处理速度。

Slide 48: MapReduce的复杂性

相比之下，MapReduce在应用复杂时需要编写非常复杂的Map函数和Reduce函数代码，开发难度较大。

Slide 49: Spark的简洁性

반면 Spark는 풍부한 변환, 필터링, 정렬 등의 작업을 제공하여 복잡한 데이터 처리를 간결하고 명확하게 만듭니다.

Slide 50:- 高效的容错性

RDD는 자연스러운 장애 허용성을 가지고 있으며, 이는 중요한 장점 중 하나입니다.

Slide 51: - 现有容错机制的对比

전통적인 장애 허용 메커니즘에는 데이터 복제, 로그 기록, 트랜잭션 메커니즘 등이 있으며, 이러한 방법들은 각각 고유한 오버헤드를 가지고 있습니다.

Slide 52: - RDD的容错方案

RDD는 혈연 관계를 통해 장애 허용성을 구현합니다. 그림에 표시된 변환 체인에서, 특정 노드에 문제가 발생하면 혈연 관계를 통해 다시 계산하여 결과를 얻을 수 있습니다.

Slide 53: - RDD持久化

RDD는 지속화 작업을 지원하며, RDD.cache() 메서드를 통해 중간 결과를 메모리에 저장하여 중복 계산을 피할 수 있습니다.

Slide 54: - 持久化的优势

지속화는 불필요한 디스크 읽기/쓰기 오버헤드를 피하고, 메모리의 여러 RDD 간에 데이터를 전달하는 효율적인 작업을 구현합니다.

Slide 49: Spark的简洁性

而Spark提供了丰富的转换、过滤、排序等操作，使得复杂的数据处理变得简洁明了。

Slide 50:- 高效的容错性

RDD具有天然的容错性，这是它的一个重要优势。

Slide 51: - 现有容错机制的对比

传统的容错机制包括数据复制、记录日志、事务机制等，这些方法都有各自的开销。

Slide 52: - RDD的容错方案

RDD通过血缘关系实现容错。如图所示的转换链路，当某个节点出现问题时，可以通过血缘关系重新计算得到结果。

Slide 53: - RDD持久化

RDD支持持久化操作，通过RDD.cache()方法可以将中间结果保存在内存中，避免重复计算。

Slide 54: - 持久化的优势

持久化避免了不必要的读写磁盘开销，实现了数据在内存的多个RDD之间进行传递的高效操作。

Slide 55: - 避免序列化开销

메모리의 데이터 전달을 통해 RDD는 불필요한 직렬화 및 역직렬화 작업도 피할 수 있어 성능을 더욱 향상시킵니다.

Slide 55: - 避免序列化开销

通过内存中的数据传递，RDD还避免了不必要的序列化和反序列化操作，进一步提高了性能。

Slide 56:

한국어 : MapReduce의 주요 한계점으로 가장 적절한 것은?

분산 처리가 불가능하다

반복 계산 시 매번 디스크 I/O가 발생한다

병렬 처리를 지원하지 않는다

대용량 데이터 처리가 불가능하다

함수형 프로그래밍을 지원하지 않는다

正确答案 / 정답: 2)

Slide 57:

한국어 : RDD의 가장 핵심적인 특성은 무엇인가?

가변적(Mutable)이다

디스크 기반 저장이다

불변성(Immutable)이다

단일 노드에서만 동작한다

실시간 처리만 가능하다

正确答案 / 정답: 3)

Slide 58:

한국어 : RDD에서 데이터 수정이 필요할 때 어떤 방식으로 처리되는가?

기존 RDD를 직접 수정한다

새로운 RDD를 생성한다

데이터베이스에 업데이트한다

메모리를 초기화한다

디스크에 임시 저장한다

正确答案 / 정답: 2)

Slide 59:

한국어 : RDD의 두 가지 주요 연산 타입은 무엇인가?

Map과 Reduce

Read와 Write

Input과 Output

Transformation과 Action

Create와 Delete

正确答案 / 정답: 4)

Slide 60:

한국어 : RDD의 Transformation 연산의 특징으로 올바른 것은?

즉시 실행된다

결과를 바로 반환한다

연산 궤적만 기록한다 (Lazy Evaluation)

메모리를 많이 사용한다

디스크에 결과를 저장한다

正确答案 / 정답: 3)

Slide 61:

한국어 : RDD가 지원하는 연산의 입도(Granularity)는?

세밀한 입도 (Fine-grained)

거친 입도 (Coarse-grained)

중간 입도 (Medium-grained)

가변 입도 (Variable-grained)

혼합 입도 (Mixed-grained)

正确答案 / 정답: 2)

Slide 62:

한국어 : RDD의 Lineage(혈연관계)의 주요 목적은?

성능 향상

메모리 절약

장애 복구 (Fault Tolerance)

디스크 공간 절약

네트워크 속도 향상

正确答案 / 정답: 3)

Slide 63:

한국어 : RDD의 파이프라인 최적화의 장점으로 가장 적절한 것은?

CPU 사용량 감소

중간 결과의 디스크 저장 회피

네트워크 대역폭 증가

메모리 사용량 감소

코드 복잡도 감소

正确答案 / 정답: 2)

Slide 64:

한국어 : RDD.cache() 메서드의 주요 기능은?

RDD를 디스크에 저장

RDD를 압축

RDD를 메모리에 지속화

RDD를 삭제

RDD를 복제

正确答案 / 정답: 3)

Slide 65:

한국어 : RDD 기반 시스템이 MapReduce 대비 갖는 주요 장점이 아닌 것은?

반복 알고리즘에서의 성능 향상

인메모리 처리를 통한 속도 개선

단일 레코드 수정의 효율성

DAG 기반 최적화

풍부한 고수준 API 제공

正确答案 / 정답: 3)

Slide 66: 课程总结

오늘 우리는 RDD의 운행 원리를 종합적으로 학습했습니다. 설계 배경, 기본 개념, 특성, 의존 관계, 단계 분할, 운행 과정을 포함해서 말입니다. 여러분이 RDD의 작동 메커니즘을 깊이 이해하여 향후 Spark 학습을 위한 견고한 기초를 다지기를 바랍니다. 경청해 주셔서 감사합니다! 본질적으로 분산 객체의 집합입니다. 여러 분산 객체로 구성되어 클러스터에서 병렬 처리가 가능합니다.

Slide 66: 课程总结

今天我们全面学习了RDD的运行原理，包括它的设计背景、基本概念、特性、依赖关系、阶段划分和运行过程。希望大家能够深入理解RDD的工作机制，为后续的Spark学习打下坚实的基础。感谢大家的聆听！