2.3.1 기본 개념과 아키텍처 설계 - RDD 소개

Slide 1:

Slide 2:

Slide 3:

Slide 4:

Slide 5:

Slide 6:**RDD란 무엇인가?**
RDD는 Spark의 핵심

먼저 2.3.1절을 학습하겠습니다: 기본 개념과 아키텍처 설계.
Spark에서 가장 핵심적인 개념은 바로 RDD입니다.
RDD는 Spark의 기초이며, RDD를 이해하면 Spark의 핵심을 이해하게 됩니다.

Slide 7:- RDD는 가장 핵심적인 데이터 추상화

RDD는 Spark에서 가장 핵심적인 데이터 추상화입니다.
데이터 추상화란 무엇일까요?
간단히 말해서, RDD는 분산 데이터를 처리하는 통일된 방식을 제공합니다.
데이터가 어디서 오든 상관없이 RDD로 표현하고 조작할 수 있습니다

Slide 1:

Slide 2:

Slide 3:

Slide 4:

Slide 5:

Slide 6: 2.3.1 基本概念和架构设计 - RDD介绍

我们首先来学习基本概念和架构设计。

Spark中最核心的概念就是RDD。

RDD是Spark的基础，理解了RDD，就理解了Spark的精髓(**jīngsuǐ**)。

Slide 7: RDD是最核心的数据抽象

RDD是Spark最核心的数据抽象。

什么是数据抽象呢？
简单来说，RDD为我们提供了一种统一的方式来处理分布式数据，
无论数据来自哪里，
都可以用RDD来表示和操作.
(여기서 추상화의 의미를 이렇게 설명하자. 고양이→동물, 토기 → 동물)

Slide 8: 파일에서 RDD로

모든 데이터 파일이 Spark를 통해 RDD로 변환될 수 있음을 볼 수 있습니다.
이것이 Spark가 데이터를 처리하는 첫 번째 단계입니다:
다양한 형식의 데이터를 RDD로 통일하여 추상화하는 것입니다.

Slide 8: 从文件到RDD

我们可以看到，任何数据文件都可以通过Spark转换成RDD。
这就是Spark处理数据的第一步：
将各种格式的数据统一抽象为RDD

Slide 9: 데이터베이스 vs Spark의 데이터 처리 방식

여기서 전통적인 데이터베이스와 Spark의 데이터 처리 방식을 비교해보겠습니다:

- **데이터베이스**: 데이터는 레코드 단위이며, 레코드에 대해 연산을 수행
- **Spark**: 데이터는 RDD에 대한 연속적인 연산

이는 근본적인 사고 전환으로, 레코드 지향에서 데이터셋 지향으로의 변화입니다.

Slide 9: 数据库vs Spark的数据处理方式

这里我们对比一下传统数据库和Spark的数据处理方式：

- 数据库：数据是一条条记录，我们对记录进行操作
- Spark：数据是对RDD进行一次又一次的操作

这是一个根本性的思维转变，从面向记录到面向数据集。

Slide 10: RDD의 전체 명칭

RDD는 **Resilient Distributed Dataset**의 줄임말로, 한국어로는 "탄력적 분산 데이터셋"을 의미합니다. 이 이름은 매우 중요하며, 각 단어마다 깊은 의미가 있습니다:

- **Resilient**: 탄력적, 내결함성을 가짐
- **Distributed**: 분산적, 데이터가 여러 노드에 분산
- **Dataset**: 데이터셋, 데이터의 집합

Slide 10: RDD的全称

RDD是Resilient Distributed Dataset的简写，中文意思是"弹性分布式数据集"。
这个名字很重要，每个词都有深刻含义：

- Resilient：弹性的，具有容错能力
- Distributed：分布式的，数据分布在多个节点
- Dataset：数据集，是数据的集合

Slide 11:Spark 기본 개념 상세 설명

Spark의 기본 개념들을 자세히 알아보겠습니다:

Application (애플리케이션)
사용자가 작성한 Spark 애플리케이션 프로그램

Executor (실행기)
작업 노드(WorkerNode)에서 실행되는 프로세스로, Task 실행을 담당

Task (태스크)
Executor에서 실행되는 작업 단위

RDD
Resilient Distributed Dataset(탄력적 분산 데이터셋)의 줄임말로, 분산 메모리의 추상적 개념이며, 고도로 제한된 공유 메모리 모델을 제공

DAG (방향성 비순환 그래프)
Directed Acyclic Graph(방향성 비순환 그래프)의 줄임말로, RDD 간의 의존 관계를 반영

Job (작업)
하나의 Job은 여러 RDD와 해당 RDD에 대한 다양한 연산을 포함

Stage (단계)
Job의 기본 스케줄링 단위로, 하나의 Job은 여러 Task 그룹으로 나뉘며, 각 Task 그룹을 Stage 또는 TaskSet이라고 하며, 상호 간에 Shuffle 의존 관계가 없는 관련 태스크들의 집합을 나타냄

Slide 11: Spark基本概念详解

(슬라이드 보고 설명)

让我们详细了解Spark的基本概念：

Application（应用）：用户编写的Spark应用程序

Executor：是运行在工作节点（WorkerNode）的一个进程，负责运行Task

Task（任务）：运行在Executor上的工作单元

RDD：是Resilient Distributed Dataset（弹性分布式数据集）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型

DAG：是Directed Acyclic Graph（有向无环图）的简称，反映RDD之间的依赖关系

Job（作业）：一个Job包含多个RDD及作用于相应RDD上的各种操作

Stage（阶段）：是Job的基本调度单位，一个Job会分为多组Task，每组Task被称为Stage，或者也被称为TaskSet，代表了一组关联的、相互之间没有Shuffle依赖关系的任务组成的任务集

Slide 12:**기본 개념의 계층 관계**

Spark의 각 개념 간의 계층 관계를 살펴보면, 모든 개념이 핵심을 중심으로 둘러싸여 완전한 Spark 생태 아키텍처를 형성하고 있음을 볼 수 있습니다.

Slide 13:**RDD의 특성**

RDD는 탄력적 분산 데이터셋으로, 고도로 제한된 공유 메모리 모델을 제공합니다.

여기서 "고도로 제한된"이란 무슨 의미일까요?
RDD는 읽기 전용(read-only)이라는 뜻입니다. 한 번 생성되면 수정할 수 없으며, 이러한 설계는 데이터의 일관성과 내결함성을 보장하기 위함입니다.

Slide 14:**RDD와 DAG의 관계**

메모리에서 여러 RDD를 생성할 때, 그들 사이에는 방향성 비순환 그래프(DAG)가 형성됩니다.
DAG는 RDD 간의 의존 관계를 반영하며, 이는 Spark가 태스크 스케줄링과 최적화를 수행하는 중요한 근거입니다.

Slide 12: 基本概念树状图**(shùzhuàngtú)**

这张图很好地展示了Spark各个概念之间的层次关系。我们可以看到，所有概念都围绕着(**wéiràozhe**)核心，
形成了完整的Spark生态架构

Slide 13: RDD的特性

RDD是弹性分布式数据集，它提供了高度受限的共享内存模型。

这里"高度受限"是什么意思呢？

就是RDD是只读的，一旦创建就不能修改，这样设计是为了保证数据的一致性和容错性。

Slide 14: RDD与DAG的关系

当我们在内存中生成多个RDD时，
它们之间会形成有向无环图（DAG）。

DAG反映了RDD之间的依赖关系，这是Spark进行任务调度和优化的重要依据(**yījù**)。

Slide 15:**Executor 상세 설명**

Executor는 작업 노드에서 실행되는 프로세스입니다.
각 작업 노드는 하나 또는 여러 개의 Executor 프로세스를 실행할 수 있으며, 구체적인 계산 태스크를 실행하는 역할을 담당합니다.

Slide 16:**Executor 태스크 실행**

각 Executor는 여러 태스크를 실행할 수 있습니다.
이러한 태스크들은 병렬로 실행되어 처리 효율을 크게 향상시킵니다.
작업 노드는 Executor 프로세스를 실행하여 할당받은 다양한 태스크를 수행합니다.

Slide 17:**프로세스, 스레드, 태스크의 관계**

이 중요한 계층 관계를 이해해야 합니다:

1. 작업 노드가 Executor 프로세스를 실행
2. Executor 프로세스는 여러 스레드를 포함
3. 각 스레드가 하나의 태스크를 실행

이러한 설계로 효율적인 병렬 컴퓨팅을 구현합니다.

Slide 15: Executor详解

Executor是运行在工作节点上的一个进程。

每个工作节点可以运行一个或多个Executor进程，负责执行具体的计算任务。

Slide 16: Executor运行任务

每个Executor可以运行多个任务。
这些任务是并行执行的，大大提高了处理效率。

工作节点通过运行Executor进程来执行分配给它的各种任务

Slide 17: 进程、线程、任务的关系

我们需要理解这个重要的层次关系：

1. 工作节点运行Executor进程
2. Executor进程包含多个线程
3. 每个线程执行一个任务

这种设计实现了高效的并行计算。

Slide 18:애플리케이션 프로그램

Application은 사용자가 작성한 Spark 애플리케이션 프로그램입니다.

예를 들어 단어 빈도 통계 코드 파일을 작성하고 컴파일하면 Spark 애플리케이션 프로그램이 되어 클러스터에 제출하여 실행할 수 있습니다.

Slide 18: Application应用程序

Application是用户编写的Spark应用程序。

比如我们写一个词频统计的代码文件，编译(**biānyì**)后就成为一个Spark应用程序，可以提交到集群运行。

Slide 19:Task 태스크 상세 설명

Task는 Executor에서 실행되는 작업 단위입니다. 하나의 Task는 Executor 프로세스의 한 스레드에서 실행됩니다. 여러 Task가 동일한 Executor에서 병렬 실행될 수 있으며, 각 Task는 독립적인 스레드에서 실행됩니다.

Slide 19: Task任务详解

Task是运行在Executor上的工作单元。

一个Task运行在Executor进程的一个线程上。

多个Task可以在同一个Executor中并行执行，每个Task都在独立的线程中运行。

Slide 20:Job 작업 구조

하나의 Spark 애플리케이션 프로그램은 여러 작업(Job)을 포함할 수 있습니다. 각 작업은 다시 여러 단계(Stage)로 나뉘고, 각 단계는 여러 태스크(Task)를 포함합니다. 이것이 Spark의 기본 실행 계층 구조입니다.

Slide 20: Job作业结构

一个Spark应用程序可以包含多个作业（Job）。

每个作业又分为多个阶段（Stage），每个阶段包含多个任务（Task）。

这是Spark的基本执行层次结构。

Slide 21:Stage 단계 상세 설명

Stage는 작업의 기본 스케줄링 단위입니다. 각 Stage는 병렬 실행이 가능한 여러 Task를 포함합니다. 이러한 Task들은 서로 다른 작업 노드에서 병렬 실행되어 전체적인 처리 효율을 향상시킵니다.

Slide 21: Stage阶段详解

Stage是作业的基本调度单位。

每个Stage包含多个可以并行执行的Task。

这些Task在不同的工作节点上并行运行，提高了整体的处理效率.

Slide 22: Spark**Spark 아키텍처 전체 개요**

이 그림은 Spark의 완전한 아키텍처를 보여줍니다:
태스크 제어 노드(Driver Program): 애플리케이션의 "지휘소"로 SparkContext를 포함
클러스터 자원 관리자(Cluster Manager): 클러스터 자원을 관리
작업 노드(Worker Node): Executor와 Task를 실행하며 Cache 캐시를 포함
저장 시스템: HDFS, HBase 등이 데이터 저장을 제공

Slide 22: Spark架构总览

这张图展示了Spark的完整架构：

任务控制节点（Driver Program）：
应用程序的"指挥所"，包含SparkContext

集群资源管理器（Cluster Manager）：
管理集群资源

工作节点（Worker Node）：
运行Executor和Task，并包含Cache缓存

存储系统：HDFS、HBase 等提供数据存储

Slide 23:**클러스터 자원 관리자 (Cluster Manager)**

클러스터 자원을 관리하며, 다음을 포함합니다:

- CPU 자원,
- 메모리 자원,
- 대역폭 자원

Spark는 여러 종류의 자원 관리자를 지원합니다:

- 자체 내장 Standalone 모드
- YARN
- Mesos

Slide 23: 集群资源管理器类型

Cluster Manager集群资源管理器负责管理：

- CPU资源
- 内存资源
- 带宽资源

Spark支持多种资源管理器：

- 自带的Standalone模式
- YARN
- Mesos

Slide 24: 작업 노드 (Worker Node)

작업 노드(Worker Node)의 내부 구조는 다음을 포함합니다:

- Executor: 실행기, 태스크 실행을 담당
- Cache: 캐시, 중간 결과를 저장
- Task: 구체적인 계산 태스크
- Task (Receiver): 스트림 처리의 수신기 태스크

Slide 24: Worker Node详细结构

工作节点（Worker Node）
的内部结构包括：

- Executor：执行器，负责运行任务
- Cache：缓存，存储中间结果
- Task：具体的计算任务
- Task (Receiver)：
  流处理中的接收器任务

Slide 25 Executor 내부 구조

Executor 프로세스 내부는 여러 스레드를 포함하며, 각 스레드가 하나의 태스크를 실행합니다. 이러한 설계로 단일 Executor에서 여러 태스크를 병렬 실행할 수 있어 자원 활용 효율을 향상시킵니다.

Slide 25: Executor内部结构

Executor进程内部包含：
多个线程，每个线程执行一个任务
这种设计允许在单个Executor中并行执行多个任务，提高资源利用效率

Slide 26:완전한 계층 구조

Spark의 완전한 계층 구조는 다음과 같습니다:

1. **애플리케이션(Application)**은 여러 작업(Job)을 포함
2. **작업(Job)**은 여러 단계(Stage)를 포함
3. **단계(Stage)**는 여러 태스크(Task)를 포함
4. **태스크 제어 노드(Driver)**가 전체 과정을 조정

Slide 26:完整的层次结构

这张图完整展示了Spark的层次结构：

1. 应用（Application）包含多个作业（Job）
2. 作业（Job）包含多个阶段（Stage）
3. 阶段（Stage）包含多个任务（Task）
4. 任务控制节点（Driver）负责协调整个过程

Slide 27: 코드 주도의 계층 구조

애플리케이션 프로그램의 계층 구조는 내부의 관련 코드에 따라 결정됩니다.
코드의 복잡성과 연산 유형이 얼마나 많은 Job, Stage, Task가 생성될지를 결정합니다.
이는 매우 중요한 개념으로, 코드 구조가 실행 구조에 직접적인 영향을 미칩니다.

Slide 28:**자원 신청 흐름**

1. 태스크 제어 노드(Driver)가 클러스터 자원 관리자(Cluster Manager)에 자원을 신청
2. 클러스터 자원 관리자가 작업 노드의 Executor를 시작
3. Driver가 태스크를 Executor로 전송하여 실행

Slide 29:**결과 반환 흐름**

태스크 실행이 완료된 후, 결과는 Executor에서 태스크 제어 노드(Driver)로 반환됩니다.
이렇게 하나의 완전한 계산 순환이 완성됩니다.

Slide 30:**데이터 기록 흐름**

계산 완료 후, 결과 데이터는 HDFS나 기타 데이터베이스에 기록되어 영구 저장될 수 있습니다.
이것이 Spark가 데이터를 처리하는 완전한 흐름입니다: **읽기 → 계산 → 저장**

Slide 27:代码驱动的层次结构

一个应用程序的层次结构取决于里面的相关代码。
代码的复杂性和操作类型决定了会产生多少个Job、Stage和Task。
这是一个很重要的概念，代码结构直接影响执行结构。

Slide 28: Spark的资源申请流程：

1. 任务控制节点（Driver）向集群资源管理器（Cluster Manager）申请资源
2. 集群资源管理器启动工作节点上的Executor
3. Driver将任务发送给Executor执行

Slide 29: 结果返回流程

任务执行完成后，结果会从Executor返回给任务控制节点（Driver）。

这样就完成了一个完整的计算循环。

Slide 30: 数据写入流程

计算完成后，结果数据可以写入到HDFS或其他数据库中进行持久化存储。

这是Spark处理数据的完整流程：读取 → 计算 → 存储。

Slide 31:Job과 Stage의 관계

하나의 작업(Job)이 어떻게 여러 단계(Stage)로 분해되는지, 그리고 각 단계가 다시 병렬 실행 가능한 여러 태스크(Task)를 포함하는지를 명확하게 이해하는 것이 중요합니다.

Slide 31:Job和Stage的关系

这张图清晰地展示了一个作业（Job）如何分解为多个阶段（Stage），每个阶段又包含多个可以并行执行的任务（Task）。

연습 문제

문제 1: 개념 이해 및 논리적 추론

RDD가 "고도로 제한된 공유 메모리 모델"을 제공한다고 했을 때, 이 설계가 분산 시스템에서 가지는 가장 중요한 장점은 무엇인가?

A) 메모리 사용량을 줄여 비용을 절약할 수 있다

B) 데이터 일관성과 내결함성을 보장하여 분산 환경에서의 복잡성을 줄인다

C) 네트워크 통신 속도를 향상시킨다

D) CPU 연산 속도를 최적화한다

정답: B

해설: RDD의 읽기 전용(immutable) 특성은 분산 환경에서 데이터 일관성 문제와 동시성 제어 문제를 근본적으로 해결하며, 장애 발생 시 lineage를 통한 복구가 가능하게 합니다.

문제 2: 아키텍처 이해 및 분석

하나의 Spark 애플리케이션에서 Job의 개수가 결정되는 가장 직접적인 요인은 무엇인가?

A) 클러스터의 노드 개수

B) 사용 가능한 메모리 크기

C) 코드에서 Action 연산이 호출되는 횟수

D) RDD의 파티션 개수

정답: C

해설: Spark에서 Job은 Action 연산(collect, save, count 등)이 호출될 때마다 생성됩니다. Transformation은 lazy evaluation이므로 Job을 생성하지 않습니다.

문제 3: 실행 흐름 분석

다음 중 Spark의 실행 과정에서 올바른 순서는?

A) Driver가 Task를 Executor로 전송 → Cluster Manager가 자원 할당 → Executor 시작

B) Cluster Manager가 자원 할당 → Executor 시작 → Driver가 Task를 Executor로 전송

C) Executor 시작 → Driver가 Task 전송 → Cluster Manager가 자원 할당

D) Driver가 DAG 생성 → Task 전송 → 자원 할당

정답: B

해설: 올바른 순서는 1) Driver가 Cluster Manager에 자원 신청, 2) Cluster Manager가 자원 할당 및 Executor 시작, 3) Driver가 Task를 시작된 Executor들로 전송입니다.

문제 4: 성능 및 설계 분석

Executor가 프로세스 내에서 여러 스레드로 태스크를 실행하는 설계의 주된 이유는?

A) 메모리 사용량을 줄이기 위해

B) JVM 시작 비용을 줄이고 데이터 공유 효율을 높이기 위해

C) 네트워크 대역폭을 절약하기 위해

D) 디스크 I/O 속도를 향상시키기 위해

정답: B

해설: 하나의 Executor 프로세스에서 여러 스레드로 태스크를 실행하면 JVM 시작 오버헤드를 줄이고, 스레드 간 메모리 공유로 효율성을 높일 수 있습니다.

문제 5: 복합적 사고

대용량 데이터 처리 시 Stage가 여러 개로 나뉘는 주된 이유와 그로 인한 이점은?

A) 메모리 부족을 방지하고, 각 Stage를 독립적으로 최적화할 수 있다

B) CPU 사용률을 높이고, 병렬 처리 효율을 극대화할 수 있다

C) Shuffle 연산의 경계에서 나뉘며, 장애 복구 시 전체가 아닌 해당 Stage만 재실행하면 된다

D) 네트워크 통신을 줄이고, 로컬 데이터 접근을 최대화할 수 있다

정답: C

해설: Stage는 Shuffle 의존성이 있는 지점에서 나뉩니다. 이를 통해 장애 발생 시 전체 작업을 재시작할 필요 없이 해당 Stage만 재실행하여 효율성을 높입니다.

문제 6: 시스템 설계 논리

Spark에서 DAG(방향성 비순환 그래프)를 사용하는 가장 핵심적인 이유는?

A) 메모리 사용량을 최적화하기 위해

B) RDD 간의 의존 관계를 표현하고 실행 계획을 최적화하기 위해

C) 네트워크 통신 비용을 줄이기 위해

D) 사용자 인터페이스를 단순화하기 위해

정답: B

해설: DAG는 RDD 간의 의존 관계를 명확히 표현하여 Spark가 실행 계획을 최적화하고, pipeline 최적화, stage 분할 등을 수행할 수 있게 합니다.

문제 7: 자원 관리 분석

Spark가 Standalone, YARN, Mesos 등 다양한 Cluster Manager를 지원하는 설계의 핵심 의도는?

A) 사용자가 선호하는 인터페이스를 제공하기 위해

B) 서로 다른 환경과 요구사항에 맞는 자원 관리 정책을 선택할 수 있게 하기 위해

C) 시스템의 복잡도를 높여 전문성을 요구하기 위해

D) 각 클러스터 매니저의 장점만을 결합하기 위해

정답: B

해설: 서로 다른 조직과 환경은 각기 다른 자원 관리 정책과 기존 인프라를 가지고 있으므로, 다양한 Cluster Manager 지원으로 유연성을 제공합니다.

문제 8: 데이터 처리 패러다임 이해

전통적인 데이터베이스의 "레코드 단위 처리"와 Spark의 "RDD 연산 체인" 방식의 근본적 차이점과 그 장점은?

A) RDD 방식은 메모리 사용량이 적어 비용 효율적이다

B) RDD 방식은 함수형 프로그래밍 패러다임으로 복잡한 데이터 변환을 간결하게 표현할 수 있다

C) RDD 방식은 SQL과 완전히 호환되어 기존 시스템과 통합이 쉽다

D) RDD 방식은 실시간 처리에만 특화되어 있다

정답: B

해설: RDD의 함수형 프로그래밍 패러다임은 map, filter, reduce 등의 고차 함수를 통해 복잡한 데이터 변환 로직을 간결하고 재사용 가능하게 표현할 수 있습니다.

문제 9: 종합적 시스템 분석
대규모 머신러닝 파이프라인에서 Spark의 캐시(Cache) 기능이 특히 중요한 이유는?
A) 디스크 공간을 절약할 수 있기 때문
B) 여러 번 재사용되는 중간 결과를 메모리에 보관하여 반복 연산의 성능을 크게 향상시키기 때문
C) 네트워크 보안을 강화할 수 있기 때문
D) 동시 사용자 수를 늘릴 수 있기 때문
정답: B
해설: 머신러닝에서는 동일한 데이터셋에 대해 여러 알고리즘을 적용하거나 반복 학습을 수행합니다. 캐시를 통해 중간 결과를 메모리에 보관하면 반복적인 디스크 I/O를 피할 수 있습니다.

문제 10: 고급 설계 이해
한 회사에서 배치 처리와 실시간 스트림 처리를 모두 수행해야 한다면, Spark 아키텍처를 선택하는 가장 합리적인 이유는?
A) Spark는 가장 빠른 처리 속도를 보장하기 때문
B) 통일된 API와 아키텍처로 두 가지 워크로드를 처리하여 운영 복잡성을 줄일 수 있기 때문
C) Spark는 다른 프레임워크보다 저렴하기 때문
D) Spark만이 대용량 데이터를 처리할 수 있기 때문
정답: B
해설: Spark 생태계는 Spark Core, Spark Streaming, Spark SQL 등을 통해 배치와 스트림 처리를 통일된 프레임워크에서 제공하므로, 별도의 시스템을 구축하고 운영하는 복잡성을 크게 줄일 수 있습니다.

Slide 42: **요약**

오늘 우리는 Spark의 운행 아키텍처를 심층적으로 학습했습니다:

- RDD라는 핵심 개념
- Spark의 기본 구성 요소와 그들 간의 관계
- 완전한 실행 흐름과 계층 구조

이러한 개념들을 이해하는 것은 Spark의 작동 원리를 파악하는 데 매우 중요합니다. 질문이 있으시면 언제든지 교류하고 토론하실 수 있습니다.
감사합니다!

Slide 42: 课程结束

谢谢大家的聆听！今天我们深入学习了Spark的运行架构，包括：
RDD这个核心概念
Spark的基本组件和它们之间的关系
完整的执行流程和层次结构
理解这些概念对于掌握Spark的工作原理至关重要。如果有任何问题，欢迎随时交流讨论。谢谢大家！