Slide 1:

Slide 2:

Slide 3:

Slide 1:

Slide 2:

Slide 3: 오늘 배울내용은 4번 **RDD之间的依赖关系**

Slide 4: RDD 간의 의존관계 - 개념 소개

RDD 운행원리의 핵심은 RDD 간의 의존관계를 이해하는 것입니다.

Spark에서 하나의 애플리케이션은 여러 작업으로 분해되고, 각 작업은 다시 서로 다른 단계로 나뉘며, 각 단계는 태스크의 집합을 포함합니다.

이러한 계층 구조의 분할은 RDD 간의 의존관계를 바탕으로 결정됩니다.

Slide 4:RDD 간의 의존관계 - 개념 소개

RDD运行原理的核心在于理解RDD之间的依赖关系。

在Spark中，一个应用程序会被分解成多个作业，每个作业又会被划分成不同的阶段，而每个阶段包含一个任务的集合。

这种层次结构的划分是基于RDD之间的依赖关系来确定的。

Slide 5: DAG와 의존관계

RDD 간의 의존관계는 DAG(유향무환그래프)를 형성합니다.

그림에서 볼 수 있듯이, 데이터는 Input에서 시작하여 A, B, C, D 등 RDD의 생성과 변환 연산을 거쳐, 최종적으로 E, F의 변환과 액션 연산을 통해 Output으로 출력됩니다.

이 과정에서 각 RDD는 부모 RDD에 의존하여 유향의 의존 체인을 형성합니다.

Slide 5:DAG와 의존관계

RDD之间的依赖关系形成了一个DAG（有向无环图）。

从图中可以看到，数据从Input开始，经过A、B、C、D等RDD的创建和转换操作，最终通过E、F的转换和动作操作输出到Output。

这个过程中，每个RDD都依赖于它的父RDD，形成了一个有向的依赖链(liàn)。

Slide 6: 의존관계의 두 가지 유형

RDD 간의 의존관계는 두 가지 유형으로 나뉩니다: 좁은 의존과 넓은 의존입니다.

좁은 의존은 단계를 나누지 않지만, 넓은 의존은 계산을 여러 단계로 나눕니다.

이러한 구분은 Spark의 실행 효율성과 태스크 스케줄링에 중요한 의미를 갖습니다.

Slide 6: 의존관계의 두 가지 유형

RDD之间的依赖关系分为两种类型：窄依赖和宽依赖。

窄依赖不会划分阶段，而宽依赖会将计算划分成多个阶段。

这种划分对于Spark的执行效率和任务调度具有重要意义。

Slide 7: 넓은 의존과 Shuffle 연산

넓은 의존은 Shuffle 연산과 밀접한 관련이 있습니다.

Shuffle 연산은 셔플링 연산이라고도 하며, 네트워크상에서 대규모 데이터 재분배가 필요합니다.

넓은 의존이 나타날 때 Shuffle 연산이 필요하게 되고, 이는 많은 네트워크 오버헤드와 성능 영향을 가져옵니다.

Slide 7: 넓은 의존과 Shuffle 연산

宽依赖与Shuffle操作密切相关。

Shuffle操作也称为洗牌(**xǐ pái)**操作，它需要在网络中进行大规模的数据重新分发。

当出现宽依赖时，就需要进行Shuffle操作，这会带来大量的网络开销和性能影响。

Slide 8: Shuffle 연산의 구체적 예시

구체적인 Shuffle 연산 예시를 살펴보겠습니다.

여기서는 전형적인 MapReduce 패턴을 보여줍니다:

파일이 분할된 후 Map1, Map2, Map3에서 처리되고, 각 Map은 ("a",1), ("b",1), ("c",1)과 같은 키-값 쌍을 생성합니다.

그 다음 대량의 네트워크 데이터 분배를 통해 동일한 키가 해당하는 Reduce 노드로 전송되어 집계되고, 최종적으로 ("a",3), ("b",3), ("c",3)과 같은 결과를 출력합니다.

Slide 8: Shuffle 연산의 구체적 예시

让我们看一个具体的Shuffle操作例子。

这里展示了一个典型的MapReduce模式：

文件被分割后由Map1、Map2、Map3处理，每个Map产生键值对如("a",1)、("b",1)、("c",1)。

然后通过大量的网络数据分发，相同的键被发送到对应的Reduce节点进行聚合，最终输出("a",3)、("b",3)、("c",3)这样的结果。

Slide 9: 좁은 의존과 넓은 의존의 시각적 비교

이 그림은 좁은 의존과 넓은 의존의 차이를 명확히 보여줍니다.

좁은 의존에서는 map, filter, union 연산과 같이 부모 RDD의 각 파티션이 자식 RDD의 하나의 파티션에만 대응되어 데이터 흐름이 간단하고 직접적입니다.

반면 넓은 의존에서는 join, groupByKey 연산과 같이 부모 RDD의 파티션이 자식 RDD의 여러 파티션과 연결을 맺어야 하므로 복잡한 데이터 재분배가 필요합니다.

Slide 10: 좁은 의존의 특징

좁은 의존의 특징은 부모 RDD의 파티션과 자식 RDD의 파티션 간에 일대일 대응 관계가 존재한다는 것입니다.

이는 부모-자식 관계와 같아서 각 부모 RDD의 파티션이 하나의 자식 RDD 파티션에만 영향을 줍니다.

이러한 관계는 계산을 로컬에서 수행할 수 있게 하여 네트워크 통신이 필요 없으므로 매우 효율적입니다.

Slide 9:좁은 의존과 넓은 의존의 시각적 비교

这张图清楚地展示了窄依赖和宽依赖的区别。

在窄依赖中，如map、filter、union操作，父RDD的每个分区只对应子RDD的一个分区，数据流动简单直接。

而在宽依赖中，如join、groupByKey操作，父RDD的分区需要与子RDD的多个分区建立联系，需要进行复杂的数据重新分布。

Slide 10:좁은 의존의 특징

窄依赖的特点是父RDD的分区与子RDD的分区之间存在一对一的对应关系。

这就像父子关系一样，每个父RDD的分区只影响一个子RDD的分区。

这种关系使得计算可以在本地进行，不需要网络通信，因此效率很高。

Slide 11: 좁은 의존의 구체적 예시

여기서는 좁은 의존의 구체적인 예시를 보여줍니다.

RDD1의 파티션1이 RDD2의 파티션4에 대응되고, 파티션2가 파티션5에, 파티션3이 파티션6에 대응됩니다.

이러한 일대일 대응 관계는 데이터 처리의 지역성을 보장하여, 각 태스크는 로컬 데이터만 처리하면 되고 다른 노드의 데이터를 기다릴 필요가 없어 병렬적으로 효율적인 실행이 가능합니다.

Slide 11:窄依赖的具体例子

这里展示了窄依赖的具体例子。

RDD1的分区1对应RDD2的分区4，分区2对应分区5，分区3对应分区6。

这种一一对应的关系确保了数据处理的局部性，每个任务只需要处理本地的数据，不需要等待其他节点的数据，可以并行高效地执行。

Slide 12: 좁은 의존의 개념도

개념적으로 보면, 좁은 의존은 여러 부모 RDD가 하나의 자식 RDD에 데이터를 제공하는 것과 같습니다.

여러 부모 RDD(01, 02, 03)가 있지만, 각 부모 RDD의 파티션은 자식 RDD의 대응 파티션과만 관계를 가지므로, 이러한 관계는 데이터 처리의 단순성과 효율성을 유지합니다.

Slide 12:窄依赖的概念图

从概念上看，窄依赖就像多个父RDD向一个子RDD提供数据。

虽然有多个父RDD（01、02、03），但每个父RDD的分区只与子RDD的对应分区有关系，这种关系保持了数据处理的简单性和高效性。

Slide 13: join 연산의 넓은 의존

여기서는 join 연산이 어떻게 넓은 의존을 생성하는지 중점적으로 보여줍니다.

RDD6과 RDD7이 join 연산을 수행하여 RDD8을 생성할 때, RDD6의 파티션 15, 16, 17과 RDD7의 파티션 18, 19, 20이 모두 RDD8의 파티션 21, 22, 23의 계산에 참여해야 합니다.

이러한 다대다 관계가 넓은 의존의 전형적인 특징입니다.

Slide 13: join 연산의 넓은 의존

这里重点展示了join操作如何产生宽依赖。

RDD6和RDD7进行join操作生成RDD8时，RDD6的分区15、16、17和RDD7的分区18、19、20都需要参与到RDD8的分区21、22、23的计算中。

这种多对多的关系就是宽依赖的典型特征。

Slide 14: 좁은 의존의 다중 부모 RDD

좁은 의존의 개념을 다시 한번 강조하면:

여러 부모 RDD가 있더라도 각 부모 RDD의 파티션이 하나의 자식 RDD 파티션에만 대응된다면 이는 여전히 좁은 의존입니다.

그림에서는 두 개의 부모 RDD가 하나의 자식 RDD에 데이터를 제공하는 상황을 보여주며, 이는 union 연산에서 매우 일반적입니다.

Slide 14: 좁은 의존의 다중 부모 RDD

再次强调窄依赖的概念：

即使有多个父RDD，但只要每个父RDD的分区只对应一个子RDD的分区，这仍然是窄依赖。

图中显示了两个父RDD向一个子RDD提供数据的情况，这在union操作中很常见。

Slide 15: 넓은 의존의 일대다 관계

좁은 의존에 비해 넓은 의존은 일대다 관계를 보여줍니다:

하나의 부모 RDD가 여러 자식 RDD에 데이터를 제공해야 합니다.

이러한 관계는 필연적으로 Shuffle 연산이 필요하게 되는데, 데이터가 특정 규칙(예: 키 값)에 따라 서로 다른 파티션으로 재분배되어야 하기 때문입니다.

Slide 15: 넓은 의존의 일대다 관계

相对于窄依赖，宽依赖展现的是一对多的关系：

一个父 RDD 需要为多个子 RDD 提供数据。

这种关系必然需要进行 Shuffle 操作，因为数据需要根据某种规则（如键值）重新分布到不同的分区中

Slide 16: Shuffle 연산의 특징

Shuffle 연산의 핵심 특징은 네트워크상에서 대규모로 데이터를 주고받아야 하고, 서로 다른 노드 간에 데이터를 상호 전송해야 한다는 것입니다.

이것이 우리가 "셔플링 연산"이라고 부르는 이유입니다 - 카드를 섞는 것처럼 데이터를 다시 혼합하고 분배해야 하기 때문입니다.

넓은 의존은 필연적으로 Shuffle 연산을 동반하며, 이는 Spark 성능에 영향을 주는 핵심 요소이기도 합니다.

Slide 16: Shuffle 연산의 특징

Shuffle操作的核心特征包括：需要在网络中大规模地来回传输数据，以及不同节点之间需要互相传输数据。

这就是为什么我们称之为"洗牌操作"——就像洗牌一样，数据需要重新混合和分布。

宽依赖必然伴随着Shuffle操作，这也是影响Spark性能的关键因素。

Slide 17: RDD의 의존 관계 - 넓은 의존 관계 (Wide Dependencies)

개념: 하나의 상위 RDD 파티션이 여러 하위 RDD 파티션에 영향을 주는 관계입니다.

특징:셔플(Shuffle) 발생: 데이터가 서로 다른 노드들 사이에서 대규모로 전송됩니다.

네트워크 비용 큼: "노드 간 데이터 교환"이 필요하여 성능에 중요한 영향을 미칩니다.

예시: RDD9의 파티션 24, 25, 26이 RDD12의 파티션 27, 28을 생성하기 위해 데이터를 섞어서(shuffling) 보냅니다.

Slide 17:RDD的依赖关系 - 宽依赖 (Wide Dependencies)

概念：一个父RDD分区可以对应多个子RDD分区。

特点：会引起Shuffle（洗牌）操作：需要在网络中大规模地传输数据。

网络开销大：需要"不同节点之间互相传输数据"，对性能影响关键。

示例：RDD9 的分区24、25、26需要经过Shuffle过程，将其数据重新分布以创建 RDD12 的分区27和28。

Slide 18: 의존관계의 종합 정리

마지막으로 RDD 의존관계의 완전한 그림을 종합적으로 살펴보겠습니다.

왼쪽은 좁은 의존의 예시로 map, filter, union 등의 연산을 보여주며, 데이터 흐름이 간단하고 효율적입니다.

오른쪽은 넓은 의존의 예시로 join, groupByKey 등의 연산을 보여주며, 복잡한 데이터 재분배가 필요합니다.

이 두 가지 의존관계를 이해하는 것은 Spark 애플리케이션의 성능을 최적화하는 데 매우 중요합니다.

Slide 18: 의존관계의 종합 정리

最后，让我们综合看一下RDD依赖关系的完整图景。

左侧展示的是窄依赖的例子，包括map、filter、union等操作，数据流动简单高效。

右侧展示的是宽依赖的例子，包括join、groupByKey等操作，需要进行复杂的数据重新分布。

理解这两种依赖关系对于优化Spark应用程序的性能至关重要。

연습 문제

Slide 19: 문제 1

질문: 다음 중 좁은 의존에 해당하는 연산은?

- A. groupByKey()
- B. reduceByKey()
- C. map()
- D. join()

정답: C

해설: map() 연산은 각 파티션이 독립적으로 처리되며, 부모 RDD의 각 파티션이 자식 RDD의 하나의 파티션에만 대응되므로 좁은 의존입니다.

Slide 20: 문제 2

질문: 넓은 의존은 어떤 연산을 발생시키나요?

- A. 로컬 계산
- B. 메모리 캐싱
- C. Shuffle 연산
- D. 데이터 압축

정답: C

해설: 넓은 의존은 부모 RDD의 데이터를 여러 자식 RDD 파티션으로 재분배해야 하므로 Shuffle 연산이 필요합니다.

Slide 21:문제 3

질문: RDD 의존관계에서 DAG는 무엇을 의미하나요?

- A. 데이터 집계 그래프
- B. 유향무환그래프
- C. 분산 알고리즘 그래프
- D. 동적 할당 그래프

정답: B

해설: DAG는 Directed Acyclic Graph(유향무환그래프)의 약자로, RDD 간의 의존관계를 나타내는 그래프 구조입니다.

Slide 22:문제 4

질문: 다음 중 Shuffle 연산에 대한 설명으로 올바른 것은?

- A. Shuffle 연산은 로컬 노드 내에서만 수행됩니다
- B. Shuffle 연산은 성능에 영향을 주지 않습니다
- C. Shuffle 연산은 대량의 네트워크 데이터 전송이 필요합니다
- D. Shuffle 연산은 좁은 의존에서만 발생합니다

정답: C

해설: Shuffle 연산은 서로 다른 노드 간에 데이터를 재분배하기 위해 네트워크를 통

한 대량의 데이터 전송이 필요하며, 이는 성능에 큰 영향을 줍니다.

Slide 23:**문제 5**

질문: union 연산은 다음 중 어떤 의존 유형에 속하나요?

- A. 넓은 의존
- B. 좁은 의존
- C. 혼합 의존
- D. 무의존

정답: B

해설: union 연산은 여러 RDD를 단순히 합치는 연산으로, 각 파티션이 독립적으로 처리되므로 좁은 의존에 해당합니다.

Slide 24:문제 6: 在窄依赖中，父RDD的一个分区对应子RDD的几个分区？

0个

B. 1个

C. 多个

D. 不确定

정답: B

在Apache Spark中，窄依赖（Narrow Dependency）的定义是：父RDD的每个分区最多被子RDD的一个分区所使用。

Slide 25:문제 7

질문: 다음 중 넓은 의존을 생성하는 연산은?

- A. filter()
- B. map()
- C. groupByKey()
- D. mapPartitions()

정답: C

해설: groupByKey() 연산은 동일한 키를 가진 데이터를 하나의 파티션으로 모아야 하므로 데이터 재분배가 필요하여 넓은 의존을 생성합니다.

Slide 26:문제 8

질문: Spark에서 단계(Stage) 분할은 무엇을 기준으로 하나요?

- A. 데이터 크기
- B. 노드 수량
- C. 넓은 의존의 위치
- D. 메모리 사용량

정답: C

해설: Spark는 넓은 의존이 발생하는 지점에서 단계를 나눕니다. 넓은 의존은 Shuffle을 필요로 하므로 단계 경계가 됩니다.

Slide 27:문제 **9**

질문: 다음 중 filter 연산에 대한 설명으로 올바른 것은?

- A. filter 연산은 Shuffle을 발생시킵니다
- B. filter 연산은 넓은 의존에 속합니다
- C. filter 연산은 좁은 의존에 속합니다
- D. filter 연산은 파티션 수를 변경하지 않습니다

정답: C

해설: filter 연산은 각 파티션에서 독립적으로 조건에 맞는 데이터만 선별하므로 좁은 의존에 해당합니다. (D도 맞지만 C가 더 핵심적인 특성입니다)

Slide 28:**문제 10:** 在宽依赖中，为什么需要进行Shuffle操作？

A. 为了提高计算速度

B. 为了节省内存

C. 为了将相关数据重新分布到同一分区

D. 为了减少网络传输

한국어: 넓은 의존에서 왜 Shuffle 연산이 필요한가요?

A. 계산 속도를 높이기 위해 B. 메모리를 절약하기 위해 C. 관련된 데이터를 같은 파티션으로 재분배하기 위해 D. 네트워크 전송을 줄이기 위해

정답: C 해설: 넓은 의존에서는 연산을 수행하기 위해 관련된 데이터(예: 같은 키를 가진 데이터)가 동일한 파티션에 있어야 하므로 Shuffle을 통한 데이터 재분배가 필요합니다.

강의 마무리

열심히 들어주셔서 감사합니다!

오늘은 Spark RDD 운행원리 중 의존관계 개념에 대해 깊이 있게 학습했습니다.

좁은 의존과 넓은 의존의 차이점을 이해하고, Shuffle 연산의 작동 메커니즘을 배웠으며, 이러한 개념들이 Spark의 성능과 태스크 스케줄링에 어떤 영향을 주는지 알아보았습니다.

여러분이 이 지식을 실제 Spark 개발에 적용할 수 있기를 바랍니다.

질문이 있으시면 언제든지 연락 주시기 바랍니다.

Slide 29: 강의 마무리

感谢大家认真听讲！

今天我们深入学习了Spark RDD运行原理中的依赖关系概念。

我们了解了窄依赖和宽依赖的区别，学习了Shuffle操作的工作机制，以及这些概念如何影响Spark的性能和任务调度。

希望大家能够将这些知识应用到实际的Spark开发中。

如果有任何问题，请随时联系我。