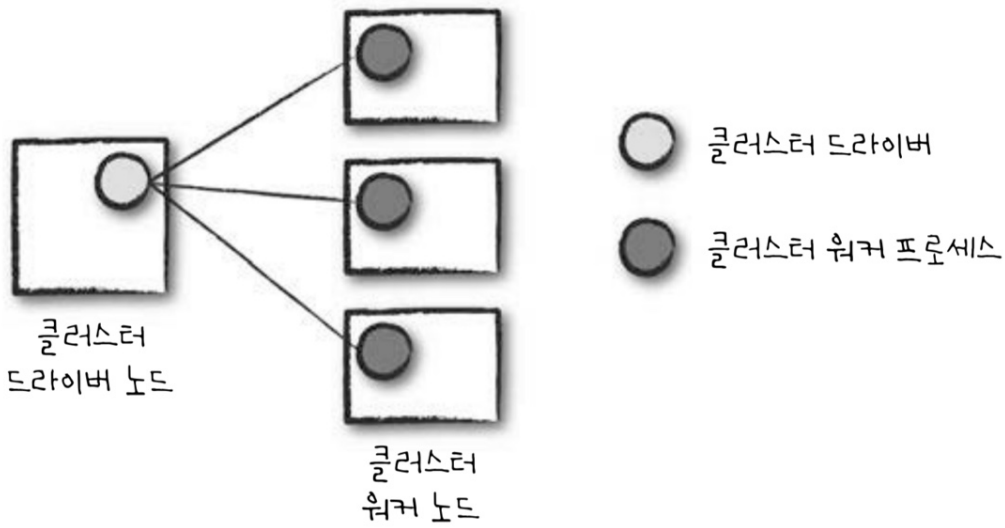
Week5 Part4 운영용 어플리케이션

생성일	@2022년 2월 15일 오전 12:15
태그	스파크 완벽가이드

Chapter 15 클러스터에서 스파크 실행하기

스파크 애플리케이션의 아키텍처

- 스파크 드라이버 : 물리적 머신의 프로세스이며 클러스터에서 실행 중인 애플리케이션의 상태를 유지한다.
- 스파크 익스큐터 : 스파크 드라이버가 할당한 태스크를 수행하는 프로세스
- 클러스터 매니저 : 스파크 애플리케이션을 실행할 클러스터 머신을 유지한다.



- 스파크 애플리케이션의 실행 방식으로는 클러스터 모드, 클라이언트 모드, 로컬 모드가 있다.

스파크 애플리케이션의 생애주기

스파크 내부

- 클라이언트 요청 : 스파크 애플리케이션을 제출하는 시점에 로컬 머신에서 코드가 실행되어 클러스터 드라이버 노드에 요청하고, 이 과정에서 스파크 드라이버 프로세스의 자원을 함께 요청한다.
- 시작 : 드라이버 프로세스가 클러스터에 배치되었으므로 사용자 코드를 실행한다.
- 실행 : 드라이버와 워커가 코드를 실행하고 데이터를 이동하는 과정에서 서로 통신하며 태스크를 할당한다.
- 완료 : 스파크 애플리케이션의 실행이 완료되면 드라이버 프로세스가 성공이나 실패 중 하나의 상태로 종료된다.

스파크 외부

- 스파크 애플리케이션은 하나 이상의 스파크 잡으로 구성된다. 애플리케이션의 스파크 잡은 대체로 차례대로 실행된다.
- SparkSession
  - 모든 스파크 애플리케이션은 가장 먼저 SparkSession을 생성한다.
  - SparkSession을 생성하면 스파크 코드를 실행할 수 있다.
  - SparkSession을 사용해 모든 수준의 저수준 API, 기존 컨텍스트 그리고 관련 설정 정보에 접근할 수 있다.
  - SparkContext를 이용해 RDD와 같은 스파크의 저수준 API를 사용할 수 있다.
- 논리적 명령
  - 사용자가 구성한 SQL, 저수준 RDD 처리, 머신러닝 알고리즘 등을 통한 각각의 잡이 트랜스포메이션이나 액션으로 호출되면 개별 스테이지와 태스크로 이루어진 스파크 잡이 실행된다.
- 스파크 잡
  - 액션 한나당 하나의 스파크 자이 생성되며 액션은 항상 결과를 반환한다.
- 스테이지
  - 스파크의 스테이지는 다수의 머신에서 동일한 연산을 수행하는 태스크의 그룹을 나타낸다.
  - 스파크는 가능한 한 많은 태스크를 동일한 스테이지로 묶으려 노력하고, 셔플 작업이 일어난 다음에는 반드시 새로운 스테이지를 시작한다.
- 태스크
  - 스파크의 스테이지는 태스크로 구성되고, 각 태스크는 단일 익스큐터에서 실행할 데이터의 블록과 다수의 트랜스포메이션의 조합이다.

세부 실행 과정

- 스파크는 map 연산 후 다른 map 연산이 이어진다면 함께 실행할 수 있도록 스테이지와 태스크를 자동으로 연결한다.
- 스파크는 모든 셔플을 작어발 때 데이터를 안정적인 저장소에 저장하므로 여러 잡에서 재사용할 수 있다.

파이프라이닝 기법

- 노드 간의 데이터 이동 없이 각 노드가 데이터를 직접 공급할 수 없는 연산만을 모아 태스크의 단일 스테이지로 만든다.
- 파이프라인으로 구성된 연산 작업은 단계별로 메모리나 디스크에 중간 결과를 기록하는 방식보다 후러씬 더 처리 속도가 빠르다.

셔플 결과 저장

- 항상 데이터 전송이 필요한 '소스' 태스크를 먼저 수행하고, 소스 태스크의 스테이지가 실행되는 동안 셔플 파일을 로컬 디스크에 기록한다.
- 잡이 실패하더라도 셔플 파일을 디스크에 저장했기 때문에 소스 태스크를 재실행할 필요 없이 실패한 리듀스 태스크부터 다시 시작할 수 있다.

Chapter 16 스파크 애플리케이션 개발하기

스파크 애플리케이션 테스트

- 자바, 스칼라, 파이썬 등으로 작성한 스파크 애플릭케이션을 테스트하려면 다음과 같은 원칙과 전략을 고려해야 한다.

전략적 원칙

- 입력 데이터에 대한 유연성
- 비즈니스 로직 변경에 대한 유연성
- 결과의 유연성과 원자성

테스트 코드 작성 시 고려사항

- SparkSession 관리하기
- 테스트 코드용 스파크 API 선정하기

단위 테스트 프레임워크에 연결하기

- 코드를 단위 테스트하려면 각 언어의 표준 프레임워크를 사용하고 테스트마다 SparkSession을 생성하고 제거하도록 설정하는 것이 좋다.

데이터소스 연결하기

- 가능하면 테스트 코드에서는 운영 환경의 데이터 소스에 접속하지 말아야 한다.

개발 프로세스

- 기존에 사용하던 개발 흐름과 유사!! (생략)

스파크 애플리케이션 환경 설정하기

- 애플리케이션 속성, 런타임 환경, 셔플 동작 방식, 스파크 UI, 압축과 직렬화, 메모리관리, 처리 방식, 네트워크 설정, 스케줄링, 동적 할당, 보안..

Chapter 17 스파크 배포 환경

스파크 애플리케이션 실행을 위한 클러스터 환경

- 설치형 클러스터
- 공개 클라우드

클러스터 매니저

스탠드얼론 모드

- 아파치 스파크 워크로드용으로 특별히 제작된 경량화 플랫폼

- 하나의 클러스터에서 다수의 스파크 애플리케이션을 실행할 수 있고 간단한 인터페이스를 제공하며 대형 스파크 워크로드로 확장할 수 있다.
- 기능의 제한이 있고 스파크 애플리케이션만 실행 가능하다.

하둡 YARN

- 하둡 YARN은 잡 스케줄링과 클러스터 자원 관리용 프레임워크.
- 스파크는 하둡과 거의 관련이 거의 없다! 하둡 YARN 클러스터 매니저를 지원하지만 하둡 자체가 꼭 필요한 것은 아님.
- spark-submit 명령의 —master 인수를 yarn으로 지정해 하둡 YARN 클러스터에서 스파크 잡을 실행할 수 있음.
- 클러스터 동장 방식을 제어할 수 있는 여러 가지 기능을 제공하고 다양한 실행 프레임워크를 진원한다.

메소스

- 아파치 메소스는 CPU, 메모리, 저장소, 그리고 다른 연산 자원을 머신에서 추상화하여 내고장성(fault-tolerant) 및 탄력적 분산 시스템(elastic distributed system)을 쉽게 구성하고 효과적으로 실행할 수 있다.
- 스파크에서 지원하는 클러스터 매니저 중 가장 무거워서 대규모의 메소스 배포 환경이 있는 경우에만 사용하는 것이 좋다.
- 메소스에서는 fine-grained와 coarse-grained 모드로 스파크를 실행할 수 있었으나 현재는 coarse-grained 스케줄링 모드만 지원한다.
  - coarse-grained : 스파크 익스큐터를 단일 메소스 태스크로 실행한다.

애플리케이션 스케줄링

- 스파크는 여러 연산 과정에서 필요한 자원을 스케줄링할 수 있다.
- 각 스파크 애플리케이션은 독립적인 익스큐터 프로세스를 실행한다.
  - 클러스터 매니저는 스파크 애플리케이션 전체에 대한 스케줄링 기능을 제공한다.
- 스파크 애플리케이션에서 여러 개의 잡을 다른 스레드가 제출한 경우 동시에 실행할 수 있다.
  - 네트워크를 통한 요청에 응답하는 애플리케이션에 적합하다.
- 스파크는 애플리케이션에서 자워을 스케줄링할 수 있도록 페어 스케줄러(pair scheduler) 기능을 제공한다.
- 스파크에서 하나의 클러스터에서 여러 스파크 애플리케이션을 실행하려면 워크로드에 따라 애플리케이션이 점유하는 자원을 동적으로 조정해야 한다. (동적 할당)
  - 동적 할당은 사용자 애플리케이션이 사용하지 않는 자원을 클러스터에 반환하고 필요할 때 다시 요청하는 방식

기타 고려사항

- 애플리케이션의 개수와 유형
- 스파크 버전 관리
- 디버깅 로그 기록 방식
- 메타스토어 사용
- 외부 셔플 서비스

Chapter 18 모니터링과 디버깅

스파크 잡의 어느 지점에서 오류가 발생했는지 파악하려면 스파크 잡을 모니터링해야 한다.

모니터링 범위

- 스파크 애플리케이션 잡
  - 클러스터에서 사용자 애플리케이션이 실행되는 상황을 파악하거나 디버깅하려면 가장 먼저 스파크 UI와 스파크 로그를 확인해야 한다.
- JVM
  - 스파크는 모든 익스큐터를 JVM에서 실행하므로 코드가 실행되는 과정을 이해하기 위해 JVM을 모니터링 해야 한다.
- OS와 머신
  - JVM은 OS에서 실행되므로 CPU, 네트워크 I/O 등의 자원에 대한 모니터링도 함께 해야 한다.
- 클러스터
  - 스파크 애플리케이션이 실행되는 클러스터도 모니터링 해야 한다. (YARN, 메소스, 스탠드얼론)
  - 인기있는 클러스터 모니터링 도구 : 강글리아(Ganglia), 프로메테우스(Prometheus)

모니터링 대상

💡 모니터링 범위는 크게 실행중인 사용자 애플리케이션의 **프로세스**(CPU, 메모리 사용률 등)와 프로세스 내부에서의 **쿼리 실행 과정**(잡과 태스크)로 나뉜다.

스파크 로그와 스파크 UI

- 스파크 애플리케이션의 로그나 스파크 자체의 로그에서 발견된 이상한 이벤트는 잡의 실패 지점이나 원인을 파악하는 데 도움이 된다.
- 스파크 UI는 실행 중인 애플리케이션과 스파크 워크로드에 대한 평가지표를 모니터링할 수 있다.

디버깅 및 스파크 응급 처치

- 교재 참고!!

Chapter 19 성능 튜닝

💡 **<요약>**
1. 파티셔닝과 효율적인 바이너리 포맷을 사용해 가능하면 작은 데이터를 읽는다.
2. 충분한 병렬성을 보장하고 파티셔닝을 사용해 데이터 치우침(skew) 현상을 방지한다.
3. 최적화된 코드를 제공하는 구조적 API를 최대한 활용한다.
→ 스파크 잡에서 적합한 연산을 사용하고 있는지 확인한다!!

간접적인 성능 향상 기법

- 설계 방안 : 스칼라 vs. 자바 vs. 파이썬 vs. R, DataFrame vs. SQL vs. Dataset vs. RDD
- RDD 객체 직렬화
- 클러스터 설정 : 클러스터/애플리케이션 규모 산정과 공유, 동적 할당
- 스케줄링 :
- 보관용 데이터: 파일 기반 데이터 저장소, 분할 가능한 파일 포맷과 압축, 테이블 파니셔닝, 버켓팅, 파일 수, 데이터 지역성, 통계 수집
- 셔플 설정
- 메모리 부족과 가비지 컬렉션

직접적인 성능 향상 기법

- 병렬화
- 향상된 필터링
- 파티션 재분배와 병합
- 사용자 정의 함수
- 임시 데이터 저장소 (캐싱)
- 조인
- 집계
- 브로드캐스트 변수