Ⅳ. 클러스터 컴퓨팅을 위한 시스템 구축

4.1 실험 환경 구성

Windows Server 2022 Datacenter OS에서, Hyper-V를 사용하여 7개의 Virtual Machine을 구축하여 실험을 진행하였다(그림 부록 1). 여기서 Virtual Machine의 OS는 Ubuntu Server 22.04 LTS이며, 각 노드는 dual-core vCPU, 16GB memory, 80GB storag로 구성하였다(그림 부록 2). 또한, 각 Virtual Machine은 1Gbps의 네트워크로 연결되어 있으며, 사용된 소프트웨어 환경에 관한 정보는 아래와 같다.

<표 4.1> 소프트웨어 환경

Software Framework	
Hadoop	3.3.5 Version
Spark	3.4.0 Version
Zookeeper	3.8.0 Version
Programming Language	
Java	Java 1.8.0_362 Version
Scala	2.12.17 Version
Python	3.10.6 Version
Operating System	
Host	Windows Server 2022 Datacenter
Node	Ubuntu Server 22.04 LTS

클러스터 시스템(Clustered System)은 2개의 네임노드(Namenode), 5개의 데이터노드(Datanode)로 구성되어 있고, 각 노드들의 Java Virtual Machine Process는 아래와 같다.

<표 4.2> Java Virtual Machine Process

Node/ Java Virtual Machine Process	namenode 1	namenode 2	datanode 3	datanode 4 ~ 7
NameNode	YES	YES	NO	NO
ResourceManager	YES	YES	NO	NO
JournalNode	YES	YES	YES	NO
DFSZKFailover Controller	YES	YES	NO	NO
HistoryServer	YES	NO	NO	NO
DataNode	NO	NO	YES	YES
NodeManager	NO	NO	YES	YES

여기서 Java Virtual Machine Process들의 역할은 아래와 같다.

- NameNode: HDFS의 핵심 구성 요소로, 파일 시스템 네임스페이스 관리, 파일 및 디렉터리에 대한 메타데이터 저장, 파일 블록과 데이터노드 간의 매핑 유지 관리를 담당한다.

- ResourceManager: YARN의 구성 요소로, 애플리케이션에 리소스를 할당하고, 애플리케이션 수명 주기를 관리하며, 전역 클러스터 리소스 사용을 유지 관리하는 역할을 한다.

- JournalNode: 활성 네임노드의 편집 로그를 저장하고 대기 네임노드와 동기화하여 활성 네임노드가 다운되는 경우 원활한 장애 조치를 보장하는 역할을 한다.

- DFSZKFailoverController(HDFS Zookeeper Failover Controller): 활성 및 대기 네임노드를 조정하고 필요할 때 자동으로 장애 조치를 수행하는 Hadoop 고가용성 기능의 구성 요소이다.

- HistoryServer: YARN의 구성 요소로, 완료된 응용 프로그램의 기록을 유지 관리하여 사용자가 실행 후 응용 프로그램을 분석하고 디버그(debug)할 수 있도록 한다.

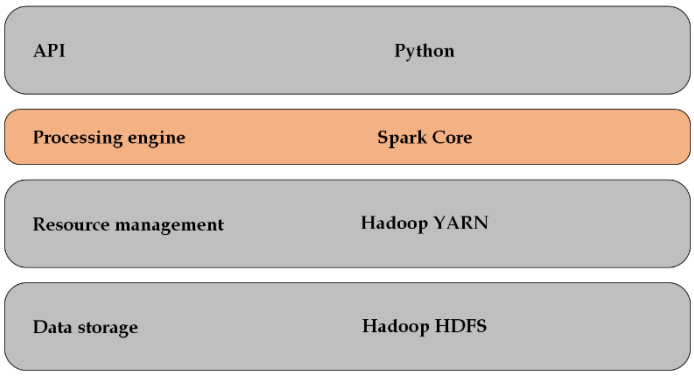
- DataNode: 실제 데이터 블록을 저장하고 관리하는 HDFS의 워커 노드이다. 클라이언트의 읽기 및 쓰기 요청을 처리하고 네임노드의 지시에 따라 블록 생성, 삭제 및 복제를 수행한다.

- NodeManager: YARN의 구성 요소로, 클러스터의 각 워커노드에서 실행된다. 노드에서 리소스 및 컨테이너(격리된 실행 환경)를 관리하고, 컨테이너 리소스 사용량을 모니터링하고, 리소스 매니저의 상태를 다시 보고하는 일을 담당한다.

HDFS 블록 용량은 128M, HDFS 파일 블록 복제 개수는 3개, JVM Heap Memory는 500MB로 구성하였다.

4.2 클러스터 시스템 구축

본 연구를 위해 구축한 클러스터 시스템은 API, Processing engine, Resource management, Data storage로 역할을 나눌 수 있으며, 이는 아래와 같은 상위 수준 스택 구조로 구성되어 있다.[39]



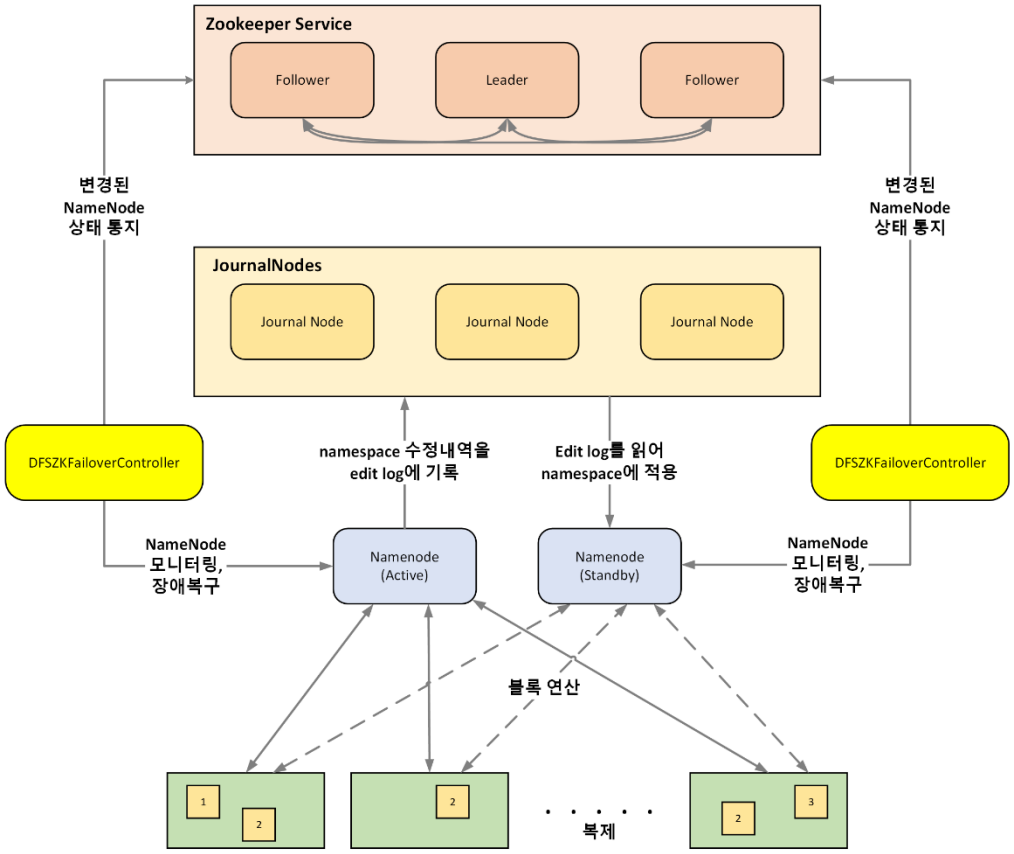
<그림 4.1> 클러스터 시스템 상위 수준 스택 구조

클러스터 시스템의 데이터 저장소(Data storage)는 HDFS를 사용하였고, 리소스는 YARN 리소스 매니저(Resource management)를 사용하여 관리하였다. 빈발 패턴 마이닝을 위한 프로세싱 엔진(Processing engine)은 Spark Core를 활용하여 수행하였으며 애플리케이션에 관한 코드는 Python 언어와 PySpark 라이브러리를 활용하여 작성하였다.

[39] 구축한 클러스터 시스템의 컴포넌트들은 상술한 Apache Hadoop, Apache Spark, Apache Zookeeper와 같은 오픈소스 프레임워크를 사용하였으며, 이를 설명하는 그림 자료는 인용 자료들을 참고하여 재구성하였다.

4.2.1 Hadoop 고가용성 구성

클러스터 시스템에서 데이터 저장소와 리소스 매니저는 Hadoop HDFS, Hadoop YARN을 사용하였으며, 데이터 저장소의 역할을 하는 HDFS의 아키텍처는 아래와 같이 구성되어있다.

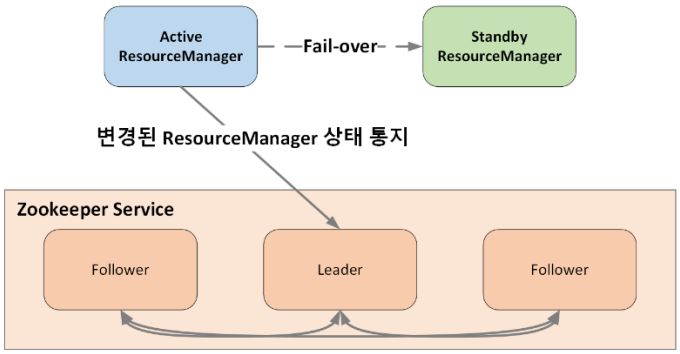


<그림 4.2> HDFS 고가용성 아키텍처[40]

해당 시스템은 2개의 네임노드와 5개의 데이터노드 총 7개의 서버로 구성되어 있으며, HDFS 및 YARN에 대한 관리를 위한 네임노드는

[40] [74], [114], [115]을 참고하여 재구성.

고가용성(High Avaliability)을 위해 이중화(Duplication)된, 두 대의 서버로 구성되어 있다. 두 대의 서버는 활성(Active)노드와 대기(Standby)노드로 역할이 나뉘어 있으며, 두 서버 모두 데이터노드에 있는 데이터 블록(Data Block)에 대한 정보와 하트비트(Heartbeat)를 모두 받아서 메타데이터(Metadata)를 유지한다.



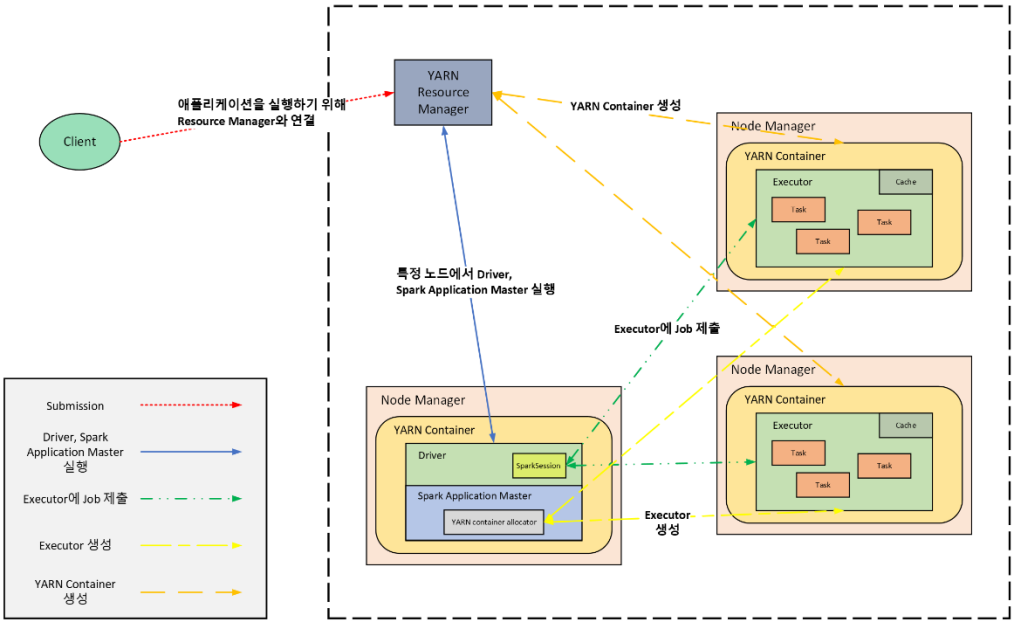
<그림 4.3> YARN 리소스 매니저 고가용성 구조[110].

또한, YARN 리소스 매니저는 위 그림과 같은 방식으로 Zookeeper를 사용하여 쓰기 요청을 유실하지 않고 서비스를 진행하는 고가용성 상태를 유지한다.

4.2.2 Spark on YARN 클러스터 구성

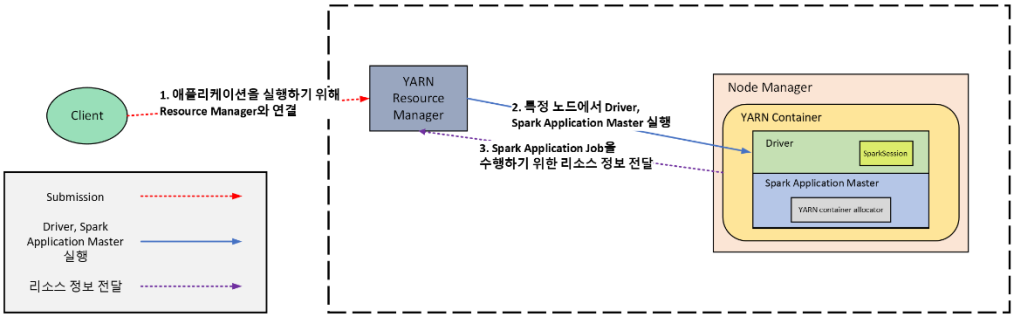
Spark 애플리케이션은 YARN 리소스 매니저로 관리되며, 클러스터 모드에서 클라이언트의 요청을 처리하는 과정에 대한 상위 수준 아키텍처는 <그림 4.4>와 같다. Spark 애플리케이션은 '1. Spark 애플리케이션에 관한 코드 제출, 2. Spark 애플리케이션 마스터 및 드라이버 실행, 3. 워커노드에 YARN 컨테이너 할당, 4. 스파크 애플리케이션 마스터가 워커노드에 익스큐터 생성, 5. Spark 애플리케이션에 관한 잡(Job)을 수행하기 위한 태스크로 구성된 작업을 각 익스큐터로 전달, 6. 드라이버가 클라이언트에게 Spark

애플리케이션의 최종 결과 전달'과 같은 절차를 수행하여 클라이언트에게 최종 결과를 전달하게된다. 이에 대한 자세한 내용은 아래와 같다.



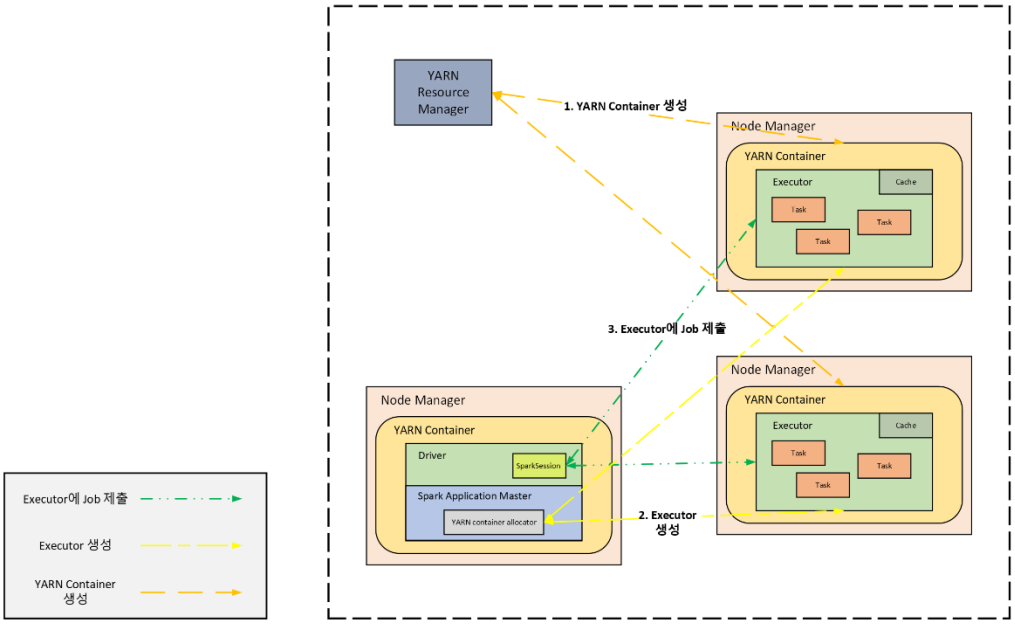
<그림 4.4> Spark on YARN 클러스터 아키텍처[85].

우선 아래의 그림과 같이 클라이언트가 리소스 매니저에게 Spark 애플리케이션을 실행하기 위한 리소스를 요청하면, 리소스 매니저는 모니터링하고 있는 워커노드의 자원 상태와 Spark 애플리케이션에 필요한 리소스를 파악하여 특정 워커노드에 컨테이너를 할당하고 클라이언트에게 알려준다. 이후 클라이언트가 애플리케이션에 대한 코드를 리소스 매니저에게 제출하면, 리소스 매니저는 해당 컨테이너에 Spark 애플리케이션 마스터 및 드라이버를 실행시킨다. 이후, 해당 컨테이너는 실행된 드라이버가 애플리케이션에 관한 잡을 수행하기 위한 리소스 정보를 리소스 매니저에게 전달한다.



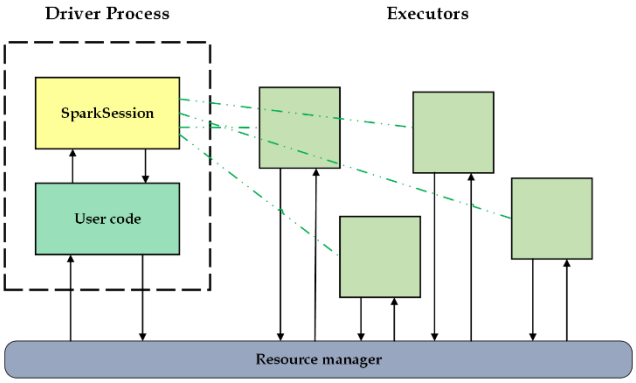
<그림 4.5> 애플리케이션 마스터 및 드라이버 실행 과정

Spark 애플리케이션을 수행하기 위한 리소스 정보를 기반으로 리소스 매니저는 잡을 수행할 수 있는 워커노드들에 컨테이너를 할당한다. 이후, YARN Allocator는 리소스 매니저가 할당한 컨테이너에 익스큐터를 생성한다. 마지막으로 드라이버에서 애플리케이션에 관한 잡을 수행하기 위해, 각 익스큐터에게 태스크로 구성된 작업을 제출한다.



<그림 4.6> 익스큐터 생성 및 작업 제출 과정

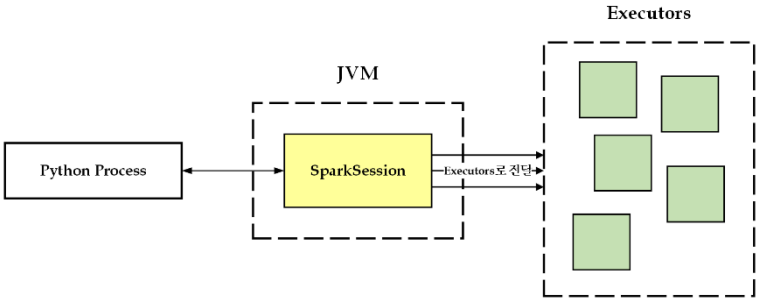
위 그림에서 드라이버가 각 익스큐터에게 태스크로 구성된 작업을 제출하고 익스큐터가 드라이버가 할당한 작업을 수행하는 과정을 다르게 표현하면, 아래의 그림과 같이 나타낼 수 있다.



<그림 4.7> 작업 제출 구조[77].

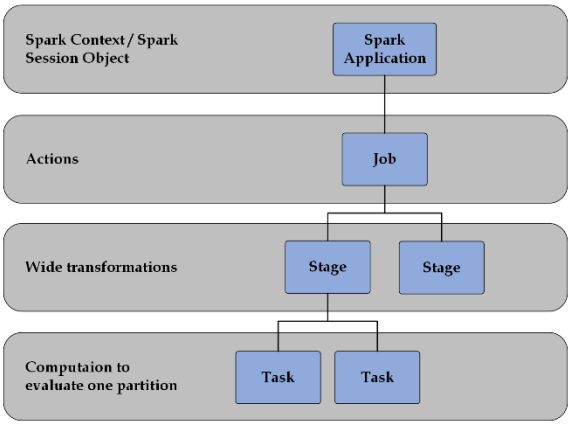
여기서 드라이버가 할당한 작업을 수행하는 익스큐터는, 할당된 작업에 관한 코드를 실행하고, 다시 드라이버 노드에 보고하는 두 가지 역할을 수행한다.

드라이버는 Spark의 Language APIs를 통해 Java, Python, R과 같은 다양한 프로그래밍 언어를 실행할 수 있으며, 본 연구에서는 Spark 코드를 실행하기 위해 Python 프로그래밍 언어를 사용하였다. 또한 SparkSession 객체를 진입점으로 사용하였으며, 이를 그림으로 나타내면 아래와 같다.



<그림 4.8> SparkSession과 Python API 간의 관계[41]

이처럼 Python 코드 기반의 애플리케이션은 익스큐터의 JVM에서 실행할 수 있는 코드로 변환되어 전달되게 되는데, 이때 익스큐터들의 태스크로 구성된 작업은 아래의 그림과 같이 계획(schedule)된다. 그리고 Spark 애플리케이션을 수행하는 작업이 여러 개의 액션(Action)[42]으로 구성되어 있다면, 드라이버 프로세스는 액션을 수행하기 위한 잡을 계획하게 된다.



<그림 4.9> 애플리케이션 트리 구조[90].

[41] [77], [116]을 참고하여 재구성.

[42] 여기서 액션(action)은 데이터를 반환하거나 저장하는 Spark의 연산을 의미한다.

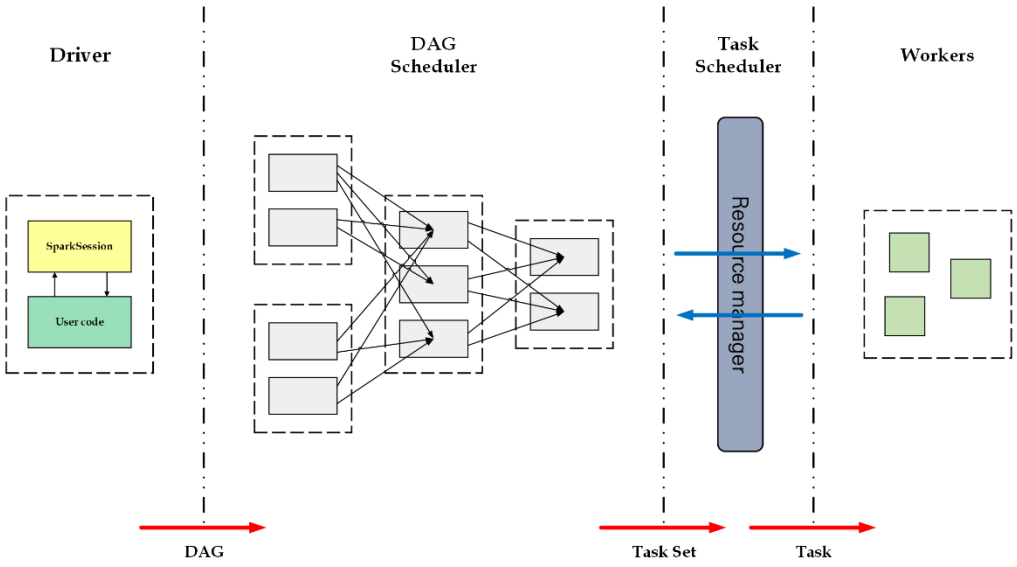
여기서, 잡은 액션을 수행하기 위한 계획에 관한 정보를 의미하며, 액션을 수행하기 위해서 각자의 잡이 구성된다.

드라이버 프로세스에는 잡을 여러 개의 익스큐터가 분담하여 작업할 수 있도록 계획을 수립하는, Spark Scheduler가 있다. Spark Scheduler는 DAG Scheduler, Task Scheduler라는 두 가지 주요 컴포넌트로 구성되어 있으며, 아래와 같은 역할을 수행한다.

- DAG Scheduler: Spark 애플리케이션을 수행하기 위해 드라이버 프로세스는 잡을 태스크에 관한 방향성 비순환 그래프(Directed Acyclic Graph, DAG)로 생성하게 되는데, DAG Scheduler는 이 DAG를 통해 도출된 결과를 바탕으로 계획 수립하는 역할을 한다.
- Task Scheduler: DAG Scheduler를 통해 수립된 계획을 적절한 익스큐터에 할당하는 역할을 한다.

액션마다 드라이버 프로세스의 Spark Scheduler는 아래의 그림과 같이 DAG를 만들고, 이후 익스큐터를 실행하기 위한 리소스에 대해 리소스 매니저[43]와 통신하며, Task Scheduler를 통해 익스큐터들이 수행할 각자의 태스크로 구성된 작업을 전달한다.

[43] 'Hadoop YARN, Mesos, 쿠버네티스'와 같은 클러스터 매니저.



<그림 4.10> Job Scheduling 과정[85].

DAG Scheduler로 도출된 결과는, 익스큐터들이 잡을 적절히 나누어 수행할 수 있게 나누어지며, 이에 대한 작업 단위를 스테이지(Stage)[44]라고 한다. 즉, 스테이지는 잡을 여러 개의 익스큐터가 분담하여 작업할 수 있도록 나눈 것을 의미한다. 그리고 스테이지는 태스크(Task)라는 가장 작은 작업 단위로 실행된다(그림 4.9). 이때 태스크는 데이터의 한 부분(partition)을 처리하며, 모든 태스크가 완료되면 스테이지가 완료된다.

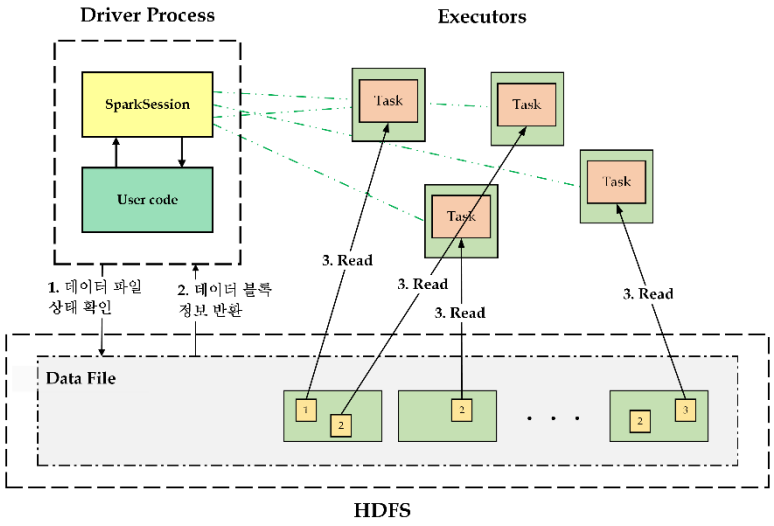
4.2.3 HDFS 읽기-쓰기 프로세스

이처럼 Spark 애플리케이션은 태스크라는 작업 단위들로 구성되어 있으며, HDFS에 데이터를 읽기-쓰기(read-write)하는 프로세스는 다음과 같다.

[44] 데이터가 셔플(shuffle)이 필요하지 않은 연산들, 즉 다른 익스큐터나 드라이버와의 통신 없이 하나의 익스큐터에서 독립적으로 계산 가능한 태스크들의 집합.

(1) HDFS 읽기 프로세스

아래의 그림과 같이 잡을 수행하는 스테이지 중, HDFS에 데이터를 읽어 들이는 과정이 있다면, 드라이버는 HDFS에 데이터 파일(File)에 정보를 얻기 위해 HDFS와 통신을 하게 된다. 여기서 드라이버가 HDFS에 파일에 대한 정보를 요청하면, HDFS는 해당 파일의 데이터블록 정보를 드라이버에 전달한다.



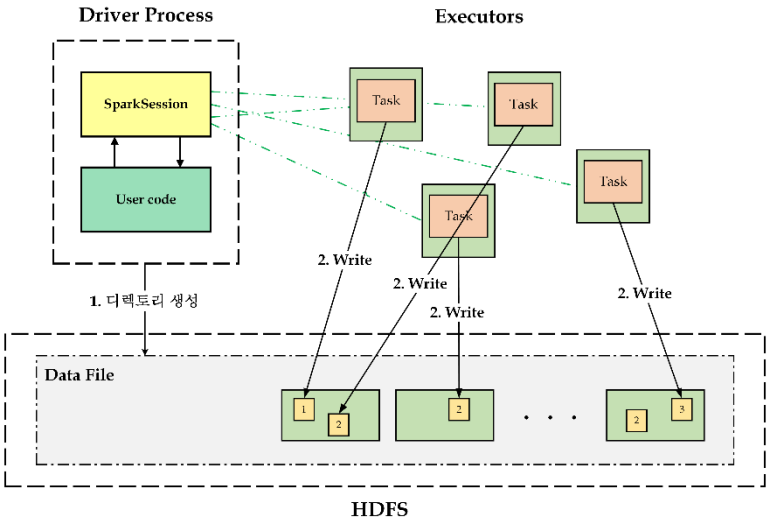
<그림 4.11> HDFS 데이터블록 읽기[45]

이러한 정보를 기반으로 드라이버는 잡을 수행하기 위한 리소스를 계산하고, 익스큐터를 할당하게 된다. 이후, 익스큐터는 태스크 작업을 수행하면서 HDFS에 데이터를 읽어 들이는 단계가 되면, HDFS와 통신을 하며 할당된 태스크를 수행하기 위한 데이터블록에 데이터를 읽어 들이게 된다.

[45] [73], [77], [79], [110], [117], [118]을 참고하여 재구성. 여기서, HDFS의 내부 구조에 대한 표현은, '여러 노드가 Data File에 포함되어있다.'가 아닌, 'Data File이 여러 노드의 데이터 블록으로 분산되어 저장되어 있다.'를 의미한다.

(2) HDFS 쓰기 프로세스

아래의 그림과 같이 잡을 수행하는 스테이지 중, HDFS에 데이터를 기록하는 과정이 있다면, 드라이버는 HDFS와 통신을 하여, HDFS에 데이터 파일(File)을 저장하기 위한 디렉터리를 생성한다.



<그림 4.12> HDFS 데이터블록 쓰기[46]

이후 드라이버는 잡에 관한 상태 정보를 기반으로, 데이터 쓰기 작업을 익스큐터로 전달한다. 이후, 익스큐터는 태스크 작업을 수행하면서 HDFS에 데이터를 기록하는 단계가 되면, HDFS와 통신을 하며 해당 디렉터리에 데이터를 저장한다.

[46] [73], [77], [79], [110], [117], [118]을 참고하여 재구성. 여기서, HDFS의 내부 구조에 대한 표현은, '여러 노드가 Data File에 포함되어있다.'가 아닌, 'Data File이 여러 노드의 데이터 블록으로 분산되어 저장되어 있다.'를 의미한다.

Computing Environment

<그림 부록 1> Host 서버 정보

이름 ↓	가상 프로세서	할당된 메모리	메모리 압력	메모리 요구량	하트비트	상태
dn1	2	16 GB	23 %	3.68 GB	확인	실행 중
dn2	2	16 GB	19 %	3.04 GB	확인	실행 중
dn3	2	16 GB	18 %	2.88 GB	확인	실행 중
dn4	2	16 GB	18 %	2.88 GB	확인	실행 중
dn5	2	16 GB	17 %	2.72 GB	확인	실행 중
nn1	2	16 GB	30 %	4.8 GB	확인	실행 중
nn2	2	16 GB	25 %	4 GB	확인	실행 중

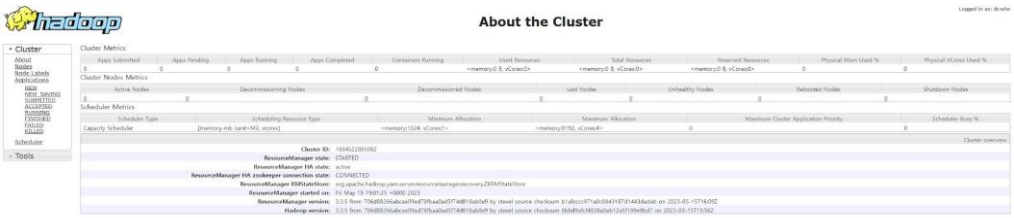
<그림 부록 2> Virtual Machine 정보

이름 ↑↓	상태 ↑↓	리소스 그룹 ↑↓
DataNode-1	연결됨	Hadoop-Ecosystem
DataNode-2	연결됨	Hadoop-Ecosystem
DataNode-3	연결됨	Hadoop-Ecosystem
NameNode-2	연결됨	Hadoop-Ecosystem
Namenode-1	연결됨	Hadoop-Ecosystem

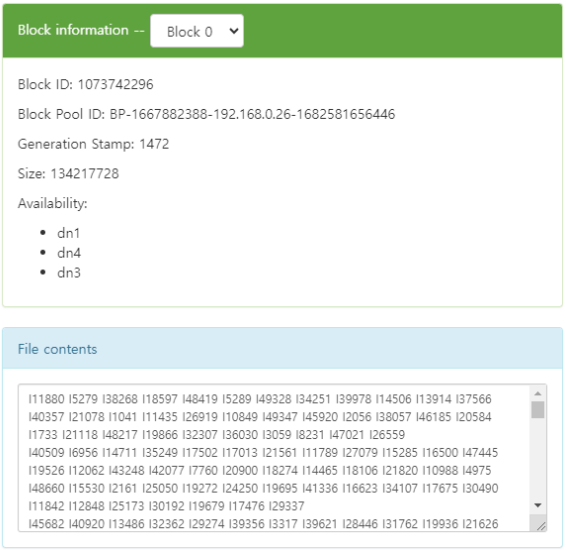
<그림 부록 3> 퍼블릭 클라우드 서비스 연결 예시

```
Starting namenodes on [nn1 nn2]
Starting datanodes
Starting journal nodes [nn1 nn2 dn1]
Starting ZK Failover Controllers on NN hosts [nn1 nn2]
Starting resourcemanagers on [ nn1 nn2]
Starting nodemanagers
```

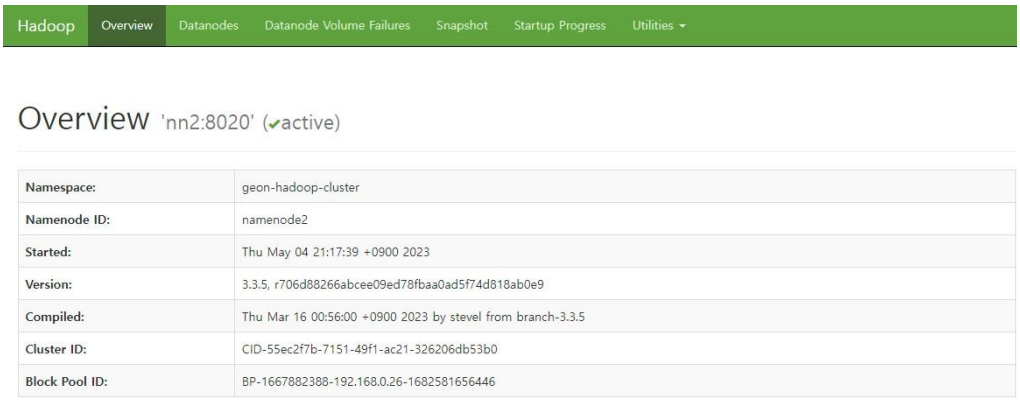
<그림 부록 4> 클러스터 시스템 실행 예시



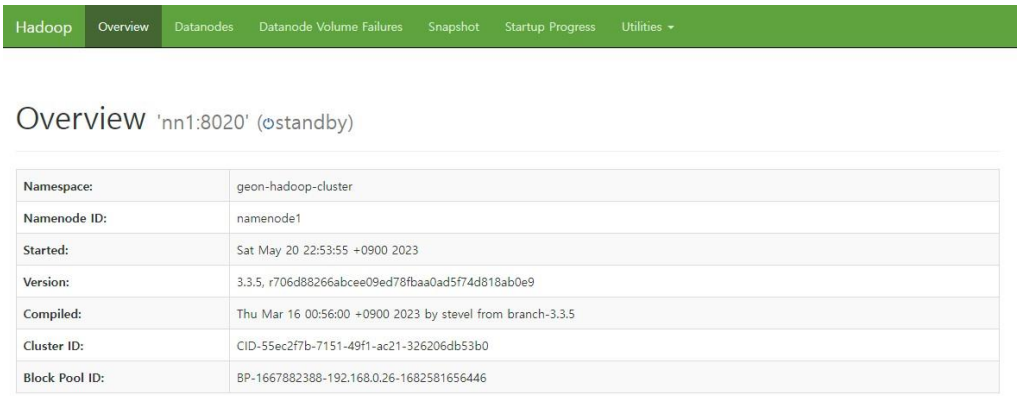
<그림 부록 5> Hadoop 노드 매니저



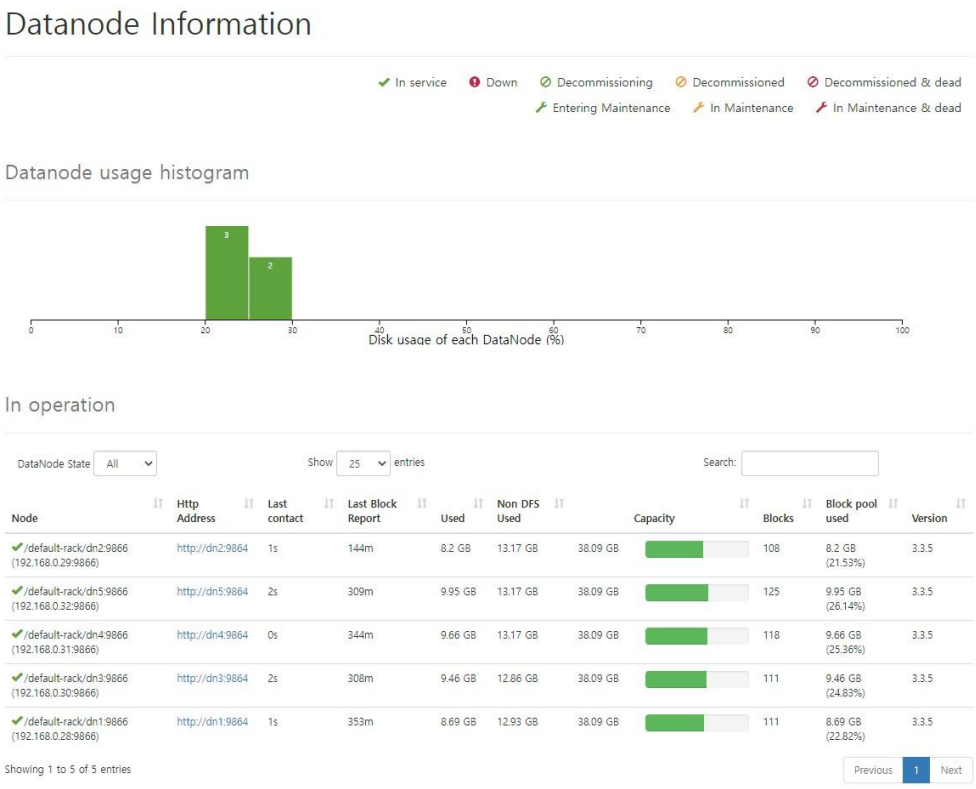
<그림 부록 6> HDFS 저장 데이터 예시



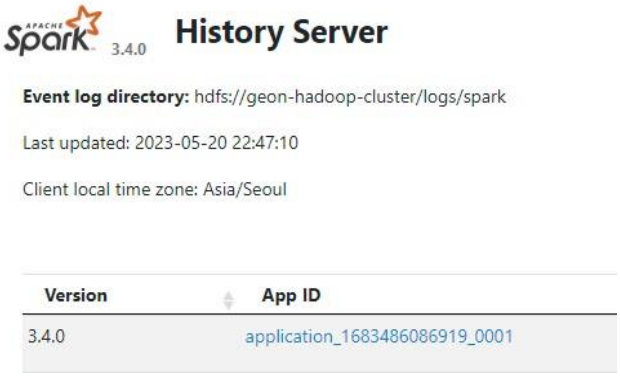
<그림 부록 7> Hadoop 활성 네임노드



<그림 부록 8> Hadoop 대기 네임노드



<그림 부록 9> Hadoop 데이터노드



<그림 부록 10> Spark History 서버