논문 21-46-02-25 The Journal of Korean Institute of Communications and Information Sciences '21-02 Vol.46 No.02
https://doi.org/10.7840/kics.2021.46.2.401

# 스마트시티를 위한 실시간 데이터 처리 아키텍처

이 모 세•, 강 민 수*, 김 홍 준*, 김 재 헌°

# Real-Time Data Processing Architecture for a Smart Cities

Mo-se Lee•, Min-su Kang*, Hong-joon Kim*, Jae-hun Kim°

요 약

스마트시티는 교통, 환경, 시설 등의 다양한 도시 문제를 사물인터넷 기반의 정보통신기술을 활용하여 해결함으로써 시민들의 편리하고 쾌적한 생활을 누릴 수 있도록 한다. IoT 기반의 스마트시티 서비스는 스마트시티에서 발생하는 방대한 양의 데이터를 수집, 저장, 처리하고 기계학습에 의한 예측 및 추론을 하는 인공지능 플랫폼 기술을 기반으로 한다. 본 논문에서는 스마트시티에 적용하기 위한 실시간 빅데이터 분석 플랫폼 개발 기술들에 대해 간략히 살펴보고 실시간 빅데이터 처리를 위한 아키텍처를 제안하고자 한다. 스마트시티를 위한 빅데이터 처리 아키텍처에 대한 모의 검증을 진행하기 위해 텐서플로우 기반 실시간 데이터 분석 모듈과 데이터 시각화 모듈을 추가하여 전체적인 데이터 플로우를 구성하였다. 본 논문에서 제안한 플랫폼을 사용하면 스마트시티에서 발생하는 방대한 데이터들에 대한 실시간 데이터 처리가 가능하며 스마트시티 관리자가 원하는 기계학습 모델을 적용하여 대시보드를 통해 예측 및 추론에 대한 결과를 확인할 수 있다. 이로써 제안한 실시간 빅데이터 분석 플랫폼이 도시 문제 해결에 기여할 수 있을 것으로 사료된다.

**키워드 :** 스마트시티, 빅데이터 플랫폼, 빅데이터 아키텍처, 빅데이터 처리, 실시간 데이터
**Key Words :** Smart City, Big data platform, Big data architecture, Big data processing, Real-time data

ABSTRACT

Smart City uses IoT-based ICT to solve various urban problems such as traffic, environment, and facilities, so that citizens can enjoy a convenient and comfortable life. The IoT-based smart city service is based on artificial intelligence platform technology that collects, stores and processes vast amounts of data generated in a smart city, and makes predictions and inferences by machine learning. In this paper, we briefly review technologies for real-time big data analysis platform development for application to a smart city and propose an architecture for real-time big data processing. In order to verify the big data processing architecture for a smart city, a TensorFlow-based real-time data analysis module and a data visualization module were added to construct the overall data flow. The platform proposed in this paper enables real-time data processing for vast amounts of data generated in a smart city, and the smart city manager can apply the desired machine learning model and check the results of prediction and inference through the dashboard. It is estimated that the proposed real-time big data analysis platform will contribute to solving urban problems.

※ 본 연구는 스마트시티 혁신성장동력프로젝트(10-2020-0167753) 과제의 연구비 지원에 의해 수행되었습니다.
• First Author : VReducation Co., Ltd, lms7140@vreducation.kr, 책임연구원, 정회원
° Corresponding Author : VReducation Co., Ltd, huns@vreducation.kr, 대표이사, 정회원
* VReducation Co., Ltd, kmsjks79@vreducation.kr, 정회원; Namutech Co., Ltd, hongjoon.kim@namutech.co.kr, 정회원
논문번호 : 202011-273-0-SE, Received October 28, 2020; Revised January 12, 2021; Accepted January 12, 2021

## Ⅰ. 서 론

4차 산업혁명의 새로운 성장 동력으로 떠오르고 있는 스마트시티는 도시에서 발생하는 교통, 환경, 시설 등의 다양한 도시 문제를 사물인터넷 기반의 정보통신기술을 활용하여 해결함으로써 시민들의 편리하고 쾌적한 생활을 누릴 수 있도록 한다.

스마트시티의 기반이 되는 IoT(Internet of Things) 기술은 모든 사물의 센서 데이터를 연결하고 실시간으로 데이터를 공유할 수 있게 한다. 실시간으로 물의 생산량과 소비량을 체크하는 수자원 관리 시스템인 스마트워터그리드(Smart Water Grid), 전기 관리를 위한 지능형 전력망인 스마트그리드(Smart Grid), 주차관리시스템, 실시간 교통정보 관리 시스템 등이 스마트시티를 구성하는 주요 서비스로써 도시 거주자들의 삶의 질을 향상시켜 줄 것으로 전망된다[1].

위와 같은 IoT 기반의 스마트시티 서비스는 스마트시티에서 발생하는 방대한 양의 데이터를 수집, 저장, 처리하고 기계학습(Machine Learning)에 의한 예측 및 추론을 하는 인공지능 플랫폼 기술을 기반으로 한다. 기존의 인공지능 플랫폼 기술이 빅데이터를 이용하여 도시의 문제를 파악하는 수준이었다면, 스마트시티의 인공지능 플랫폼 기술은 도시의 문제를 지능적이고 능동적으로 해결하기 위한 자동화 서비스를 제공해야 한다[2].

스마트시티에 적용하기 위한 방대한 양의 데이터를 분석하는 인공지능 기술들이 개발되고 있으며 인공지능의 딥러닝 기술을 기반으로 구현되는 빅데이터 플랫폼을 제안하는 연구도 진행되었다[3].

하지만 기존에는 다양한 데이터의 융합이 요구되는 사업에서 단위 연구만 진행되었다는 점에서 데이터 간 공유 및 융합에 대한 한계점이 나타났다. 스마트시티 서비스는 도시에서 발생하는 데이터의 복합적인 융합을 요구하기 때문에 인공지능 플랫폼 기술을 복합적인 데이터에 대한 선제적인 문제 해결이 가능한 플랫폼으로 발전시키고자 한다[4].

위와 같은 빅데이터 기반 인공지능 플랫폼을 위해서는 효과적인 데이터 처리 기술이 필요하다. 현재 실시간 빅데이터 처리를 위해 Kafka, Spark, Storm과 같은 기술들이 많이 나와 있으며, 이 기술들은 Netflix, Uber와 같은 빅데이터를 다루는 서비스에도 사용되고 있다.

스마트시티의 다양한 구성 요소와 그로부터 발생하는 방대한 양의 데이터를 이용하여 복합적인 서비스를 제공하기 위해선 여러 요소에서 발생하는 데이터를 한 곳으로 수집하여 대용량 데이터를 저장 및 처리할 수 있는 아키텍처가 필요하다. 따라서 본 논문에서는 스마트시티에 적용하기 위한 실시간 빅데이터 분석 플랫폼 개발 기술들에 대해 간략히 살펴보고 실시간 빅데이터 처리를 위한 아키텍처를 제안하고자 한다.

서론에 이어 2장에서는 실시간 빅데이터 분석 플랫폼에 관한 이론적 배경을 기술하고, 3장에서는 본 논문에서 제안하는 빅데이터 처리 아키텍처 및 적용사례를 설명한다. 마지막 4장에서는 결론 및 향후과제를 제시한다.

## Ⅱ. 이론적 배경

이번 장에서는 본 논문에서 제안하는 스마트시티를 위한 실시간 빅데이터 아키텍처를 설계할 때 데이터의 수집·저장·처리에 사용되는 오픈소스 소프트웨어들의 기본적인 개념과 각각이 갖는 특장점 등 이론적 배경에 대하여 다룬다.

### 2.1 Kafka

아파치 카프카(Apache Kafka)는 아파치 소프트웨어 재단이 개발한 오픈 소스 분산 메시지 처리 시스템이다. 카프카의 데이터 기본 단위는 메시지(Message)로 데이터베이스의 행(Row)이나 레코드(Record)에 비유된다. 또한 카프카의 메시지는 토픽(Topic)으로 분류되는데, 이는 데이터베이스 테이블이나 파일 시스템의 폴더와 동일한 개념을 가진다. 메시지를 메모리에 저장하는 것이 아니라 파일 시스템에 저장하기 때문에 메시지가 많이 축적되어도 시스템 성능에 영향을 미치지 않는다[5].

카프카는 대용량 실시간 처리에 특화되어 확장 가능한 발행-구독(Publish-subscribe) 모델을 기반으로 정의되며 크게 프로듀서(Producer)와 컨슈머(Consumer), 브로커(Broker)로 구성된다. 아래 <그림

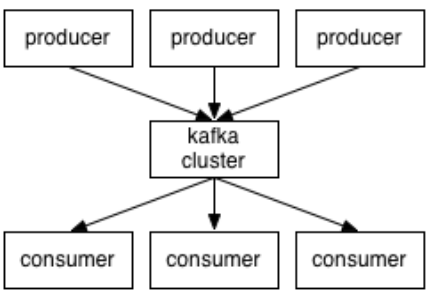


그림 1. 카프카 발행-구독 모델 구조
Fig. 1. Structure of Kafka Publish-Subscribe Model

1>은 카프카의 구성을 보여준다.

프로듀서가 특정 토픽의 메시지를 생성한 뒤 브로커에 전달하면, 브로커는 메시지를 토픽별로 분류하여 쌓아놓는다. 그리고 해당 토픽을 구독하는 컨슈머들이 메시지를 가져가서 처리하게 된다[6,7].

카프카의 브로커들은 클러스터로 구성되어 동작하여 확장성과 고가용성을 보장한다. 브로커가 1개일 경우에도 클러스터로써 동작한다. 클러스터 내의 브로커에 대한 분산 처리는 아파치 주키퍼(Apache ZooKeeper)가 담당한다.

클러스터는 JVM(Java Virtual Machine)을 기반으로 동작하기 때문에 운영체제(OS)가 설치된 리소스에 영향을 받는다[8].

카프카의 토픽은 파티션(Partition)이라는 단위로 구성되어 메시지의 처리는 토픽이 아닌 파티션별로 관리가 된다. <그림 2>는 1개의 토픽이 3개의 파티션에 분산되어 있는 것을 보여주며 추가되는 메시지는 각 파티션의 끝에 수록된다. 각 파티션은 각기 다른 서버에 분산될 수 있으며 하나의 토픽이 여러 서버에 걸쳐 수평적으로 확장되기 때문에 단일 서버로 처리할 때보다 우수한 성능을 낼 수 있다. 아래 그림과 같이 파티션은 0부터 1씩 증가하면서 오프셋(Offset) 값을 메시지에 부여하는데 이 값은 개별 파티션 내에서 메시지 식별 ID로 사용되며, 토픽 내에서 메시지를 식별할 때에는 파티션 값과 오프셋 값을 함께 사용해야 한다.

카프카의 파티션은 오직 1개의 컨슈머의 접근만을 허용하며, 컨슈머의 개수가 파티션의 개수보다 작으면 하나의 컨슈머가 여러 개의 파티션을 소유하게 되고, 반대로 컨슈머의 개수가 파티션의 수보다 많으면 여분의 컨슈머는 메시지를 처리하지 않게 되므로 조정이 필요하다.

스트림(Stream)은 파티션의 개수와 관계없이 1개의 토픽으로 간주되며 데이터를 쓰는 프로듀서와 데이터를 읽는 컨슈머로 이동되는 연속적인 데이터를 말한다. 실시간으로 메시지를 처리하는 프레임워크인 카프카 스트림즈(Kafka Streams) 같은 스트림 방법과 반대로 오프라인으로 대량의 데이터를 처리하는 프레임워크인 하둡(Hadoop)과 대비된다[9].

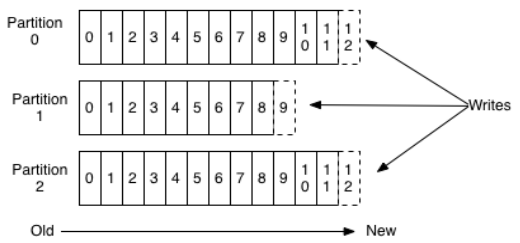


그림 2. 카프카 토픽 구조
Fig. 2. Anatomy of a Kafka-Topic

### 2.2 HDFS(Hadoop Distributed File System)

하둡은 정형데이터 및 사진영상 등의 비정형 데이터를 효과적으로 처리하는 오픈소스 빅데이터 솔루션으로, 포춘 500대 기업 모두가 하둡을 활용하고 있을 정도로 업계에서는 '빅데이터가 곧 하둡' 이라고 표현한다.

하둡 관련 오픈소스 솔루션들은 해마다 발전하여 '하둡 에코시스템'이라 불리는 하둡과 연동된 하둡 생태계를 구성하였다. 하둡은 간단한 프로그래밍 모델을 사용하여 여러 대의 컴퓨터 클러스터에서 대규모 데이터 세트를 분산 처리 할 수 있게 해주는 프레임워크이며, 단일 서버에서 수천대의 머신으로 확장 할 수 있도록 설계되었다. 일반적으로 하둡의 분산 저장 파일 시스템(HDFS)과 맵리듀스(MapReduce) 프레임워크를 의미하였으나, <그림 3>과 같이 여러 데이터저장, 실행엔진, 프로그래밍 및 데이터처리 같은 하둡 생태계 전반을 포함하는 의미로 확장 발전되었다[10].

하둡 생태계를 구성하는 여러 소프트웨어들 중 기본적으로 하둡이 사용하는 대용량 분산 저장소인 HDFS는 빅 데이터의 안정적인 저장 및 검색을 위해 설계 된 Hadoop의 분산 파일 시스템이다[11].

HDFS는 높은 장애 결함 허용성(내결함성)을 제공하고, 저비용의 범용 하드웨어 기반으로도 효율적인 분산 파일 시스템 구성 가능하며, 고가용성(High-availability)을 위하여 대용량의 데이터를 미리 정해진 크기의 단위로 구성노드들에게 분산 저장하여 기본적으로 3개의 데이터 노드에 중복으로 데이터를 저장하기 때문에 대용량의 데이터를 안정적으로 관리할 수 있다[12].

HDFS의 기본적인 구조는 하나의 네임노드(Name

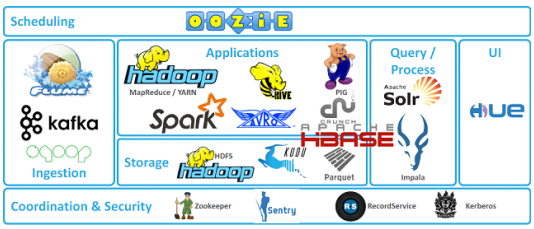


그림 3. 하둡 에코 시스템 구성도
Fig. 3. Block Diagram of Hadoop-eco System

Node)와 하나 이상의 데이터노드(Data Node)로 구성되며 이는 각각 마스터 노드(Master Node), 슬레이브 노드(Slave Node)라고 불리기도 한다.

네임 노드는 HDFS의 디렉토리와 파일 정보를 유지하기 때문에 자체에 실제 데이터를 저장하기보다는 HDFS를 구성하는 파일, 디렉터리 등의 메타데이터를 유지함으로써 전반적인 파일 시스템을 관리하는 역할을 한다. 하지만 네임노드에게만 의존하여 HDFS를 운용하게 될 경우 네임노드에 문제가 생겼을 때 전체 HDFS를 사용하지 못하게 될 수 있다.

이를 방지하기 위하여 네임 노드의 데이터를 자체 저장장치에 파일로 백업하고 있는 체크포인트 노드와 세컨더리 네임 노드, 또는 네임노드 메모리가 갖고 있는 메타데이터 정보를 독립적인 메모리에 백업하는 백업 노드(Backup Node) 등을 네임노드와 함께 운용함으로써 문제 발생 시 유연한 대처를 할 수 있는 구성을 제공하여 HDFS의 안정성을 높일 수 있다[11].

HDFS를 구성하는 또 다른 노드인 데이터노드는 분산 처리할 정보가 실제 저장되는 노드로써, <그림 4>와 같이 여러 데이터 노드가 하나의 Rack으로 묶여 있다. 여기서 Rack은 데이터 센터에서 서버 등의 다양한 장비들을 효율적으로 장착하기위한 프레임을 의미하는 용어이지만, HDFS에선 데이터의 복사본을 배치할 때 고려하는 요소로써 사용된다. HDFS에선 데이터를 저장할 때 파일 시스템 클라이언트로부터의 요청 및 디렉토리 생성/삭제/조회 등의 작업이 데이터 노드에서 실행되게 되는데, 기본적으로 데이터의 블록 단위 분할처리 이후 각 블록마다 두 개의 복사본을 생성해 원본 블록을 포함하여 총 세 개의 동일한 블록을 유지하도록 한다. 데이터 노드는 데이터를 저장할 때 첫 번째 블록이 저장되지 않은 Rack을 찾아 그 Rack에 포함된 데이터노드 두 개를 조회하여 두 번째와 세 번째 복제된 블록을 저장하는 방식을 사용한다. 이를 통해 특정 Rack 전체가 문제가 생기더라도 다른 Rack에서 해당 데이터를 구성하는 블록을 찾을 수 있기 때문에 HDFS의 데이터 신뢰성을 높이는 핵심적인 기능이다[11].

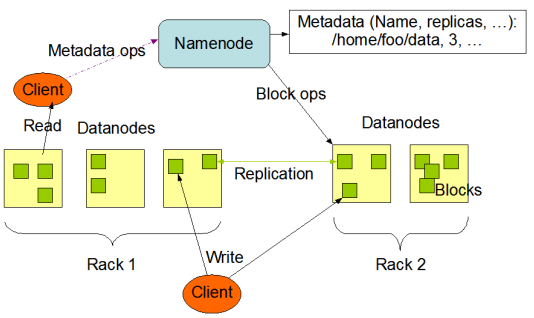


그림 4. HDFS 작업 흐름 구조
Fig. 4. Task Flow of HDFS

이렇게 HDFS의 데이터 신뢰성을 유지하는 기능은 데이터 노드뿐 만 아니라 데이터 노드를 관리하는 네임노드에서도 제공한다. 네임 노드는 주기적으로 클러스터의 각 데이터노드에서 하트비트(Heartbeat) 및 블록에 대한 보고서를 수신한다. 하트비트를 수신하면 데이터 노드가 제대로 작동하고 있음을 나타내며, 만약 네임노드가 특정 데이터노드로부터 하트비트를 수신 받지 못했다면, 네임노드는 해당 데이터노드를 사용불가 상태로 등록하고, 해당 데이터노드에 저장되어 있던 복제 본을 다른 데이터노드에 복제하여 블록의 복제본 개수를 기존 개수와 동일하도록 관리하는 기능을 제공한다[13].

### 2.3 Spark

스파크(Spark)는 빅 데이터 처리를 위한 인 메모리 기반의 대용량 데이터 고속처리 엔진을 보유하고 있는 확장성이 뛰어난 분산데이터 처리 시스템이다. 스파크는 기존의 디스크 입출력 방식이었던 빅 데이터 엔진들, 즉 예를 들어 맵 리듀스, 하이브(Hive)와 같은 엔진들과 비교하였을 때 데이터를 메모리에 저장하고 처리함으로써 데이터 처리 속도가 높다. 메모리는 일반적으로 디스크를 사용한 처리속도와 비교하였을 때 100배의 차이를 나타내는데 스파크는 이를 이용해 성능측면에서 맵 리듀스 보다 인 메모리 방식의 데이터 처리에서 100배 이상의 성능 향상을 보여주었고, 메모리로 처리할 수 없는 대용량의 데이터에 대해서도 디스크 처리 성능에서 10배 이상 빠른 속도를 발휘하였다. 성능뿐만 아니라 범용의 분산 클러스터 컴퓨팅 프레임워크 형태를 최근 모든 분야에 각광받는 머신러닝 알고리즘을 수용할 수 있는 모델로 개발되어 있다. 또한 Java, Python, Scala, R 등의 언어로 개발할 수 있어 생산성과 효율성이 높다[14].

스파크는 HDFS의 데이터뿐만 아니라 Hive, HBase, PostgreSQL, Mysql, MariaDB 등의 데이터베이스와의 호환이 가능하도록 설계되어 있다.

스파크는 위에서 나열한 데이터소스로부터 데이터를 가져와서 처리할 때 RDD(Resilient Distributed Data)라는 임의의 방향성 비 순환 그래프(Directed Acyclic Graph. DAG)형식의 데이터 단위 처리 구조를 사용하며 이를 스테이지(Stage)라고 정의한다. 스파크가 실행될 때 스테이지는 다수의 태스크로 분할

되고, 각 태스크는 맵 리듀스의 태스크와 같이 분산된 RDD 파티션에서 병렬로 실행되기 때문에 클러스터 구성하는 모든 Slave 노드의 메모리를 효율적으로 사용할 수 있으며, 이러한 이유로 스파크는 기존 하둡에서 사용 되어왔던 맵 리듀스 적용 시 발생하는 디스크 병목 현상이 제거되어 분석에 소요되는 시간을 획기적으로 줄일 수 있다[15].

RDD(Resilient Distributed Dataset)는 스파크에서 가장 중요한 개념으로써, 하둡 혹은 데이터베이스와 같은 클러스터 다수의 머신에 분할되어 저장된 데이터에 대한 읽기 전용 컬렉션이다. 일반적인 스파크 프로그램은 하나 또는 그 이상의 RDD를 입력받고 일련의 변환 작업을 거쳐 목표 RDD 집합으로 변형되는 과정을 거치게 된다. RDD는 직역하였을 때 탄력적인 분산 데이터 셋이라고 명명 할 수 있는데, 유실된 파티션이 존재할 때 스파크가 RDD의 처리 과정을 다시 계산하여 자동으로 복구가 가능하기 때문이다.

스파크에서 모든 작업은 새로운 RDD를 생성하거나 존재하는 RDD를 변형하거나, 결과 계산을 위해 RDD를 연산하는 것이다[14]. 앞서 소개한 Spark가 제공하는 라이브러리 중 실시간 데이터 처리에 초점을 둔 Spark Streaming은 다양한 소스데이터로부터의 실시간 처리를 지원하며 스트림 데이터를 마이크로 배치 형태로 쪼개서 처리하는 방식을 사용한다.

Spark Streaming 시스템 작업의 수행은 DStream을 입력데이터로 받아 처리된다. DStream이란 아래 <그림 5>와 같이 입력 받은 스트림데이터를 일정 시간 간격으로 나눈 데이터이며, 이는 연속적인 Spark RDD로 이루어진 자료 구조라고 설명할 수 있다.

Spark Streaming에서는 분산 자원 관리 프레임워크로 MESOS나 YARN을 선택하여 사용할 수 있다. 해당 프레임워크의 주된 목적은 잡이 필요로 하는 리소스를 확보하여 안정적인 서비스를 제공할 수 있도록 관리하는 것이다. 잡은 위와 같은 리소스 관리 프레임워크에게 CPU 코어 수와 사용할 메모리 공간의 크기를 요청 할 수 있으며, 리소스 관리 프레임워크는 잡이 요청한 자원에 대한 정보를 받아 해당 태스크에 대해 요청 자원 사용률을 보장해준다[16,17].

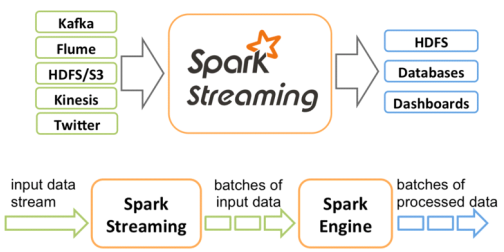


그림 5. 스파크 스트리밍의 유연성 및 데이터 처리 과정
Fig. 5. Flexibility of Spark Streaming and Data Handling Process

## Ⅲ. 제 안

3장에서는 본 논문에서 제안하고자 하는 빅데이터 처리 아키텍처의 구성 요소와 서론에서 언급되었던 각 IoT 장비에서 발생하는 개별 단위 데이터 간의 유연한 융합을 지원하기 위해 설계된 아키텍처의 코어 모듈에 대해 설명하고, 각 모듈들의 기능과 실제 서비스 운용을 위한 검증 시나리오를 구성하여 데이터 플로우(Data Flow)에 대해 상세히 설명한다.

### 3.1 스마트시티 빅데이터 처리 아키텍처 구성요소

스마트시티 빅데이터 처리 아키텍처는 도시의 IoT 센서에서 실시간으로 발생하는 방대한 데이터들을 효율적으로 처리하고 관리하고자 설계되었다.

이론적 배경에서 언급되었듯이 빅데이터 처리 아키텍처를 구성하고 있는 핵심 소프트웨어는 카프카, Spark, HDFS 세 가지이다. 이외에도 <그림 6>에서 볼 수 있듯이 카프카와 HDFS 클러스터들의 구성 정보를 관리하고 분산 동기화 및 그룹 서비스를 제공하는 ZooKeeper[18], 대용량 저장 장치의 노드들을 관리하는 노드매니저(Node Managers)에 리소스에 대한 접근 권한을 부여해 자원 제약, 스케줄링 전략, 작업 우선순위와 같은 규칙을 수립하도록 조정자 역할을 하는 YARN 또한 빅데이터 처리 아키텍처의 필수 구성요소이다[19].

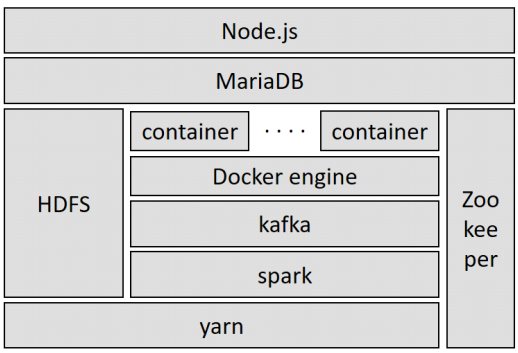


그림 6. 빅데이터 처리 아키텍처 소프트웨어 구성도
Fig. 6. Block Diagram of Bigdata Processing Architecture S/W

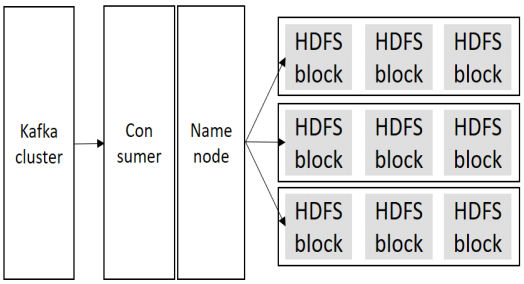


그림 7. 카프카 컨슈머를 통한 실시간 HDFS 저장 모듈
Fig. 7. Real-time HDFS Store Module via Kafka Consumers

또한 검증 시나리오 적용 시 데이터 플로우를 위한 웹 백엔드(Web-backend), 관계형 데이터베이스인 mariaDB, 데이터 분석 모듈을 포함한 도커 컨테이너 등을 구성하였다. 이러한 구성요소를 사용하여 설계 및 구축한 아키텍처 내 코어 모듈은 다음과 같은 세부 구조를 갖는다.

#### 3.1.1 실시간 데이터 저장 모듈

위의 <그림 7>은 스마트시티 빅데이터 처리 아키텍처를 구성하는 코어 모듈 중 첫 번째는 실시간 데이터 저장 모듈이다.

데이터 저장 모듈은 카프카 인터페이스를 통해 실시간으로 들어오는 데이터의 Id 값을 참조 HDFS에 해당 데이터와 관련된 디렉토리가 없더라도 데이터를 받는 즉시 생성되도록 데이터 처리 모듈과의 연계성을 고려하여 설계하였다.

HDFS의 데이터노드에 저장되는 데이터는 한 번에 3개의 노드로 분산되어 저장하지 않고 요청마다 2개의 데이터 노드에 복제되어 분산 저장되도록 설정하였다.

또한 HDFS 작업의 우선순위 및 리소스 관리 툴인 YARN 설정에서 Scheduler 속성의 Capacity Scheduler를 이용하여 트리 형태로 큐를 선언하고 각 큐별로 이용할 수 있는 자원의 용량을 정하여 작업마다 할당하는 방식으로 리소스를 분산하여 병렬로 여러 작업이 진행될 수 있도록 설정하였다[19].

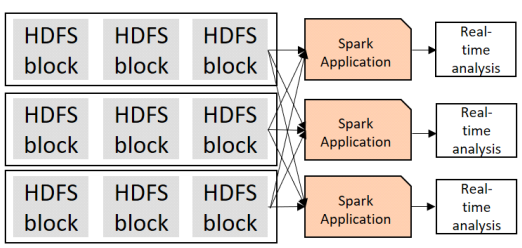


그림 8. 스파크 스트리밍을 통한 HDFS 데이터 처리 모듈
Fig. 8. HDFS Data Processing Module with Spark Streaming

#### 3.1.2 실시간 데이터 처리 모듈

위의 <그림 8>은 스마트시티 빅데이터 처리 아키텍처를 구성하는 코어 모듈 중 두 번째는 실시간 데이터 처리 모듈이다.

데이터 저장 모듈을 통해 HDFS의 디렉토리에 분산 저장되어 있는 데이터에 대해, 데이터 처리 모듈은 각 데이터 노드 블록들의 스트림 데이터를 일정 단위로 나누는 배치 처리를 통해 RDD의 연속적인 객체로 정의하는 Dstream(Discreatized Stream)으로 데이터를 가져온다. 건 별로 들어오는 데이터를 모아 처리하는 Receiver를 통해 Spark 메모리에 저장한다.

스파크 어플리케이션은 각각 1종 이상의 데이터를 분석 모델에 적용할 수 있는 형태로 전처리되어, 학습된 모델을 서빙하는 분석 모듈의 도커 컨테이너로 전달한다. 이때 데이터 전처리는 일반적으로 넘파이 배열(Numpy Array) 혹은 리스트 형태로 처리된다. 넘파이란 Python 개발 언어가 제공하는 강력한 행렬연산 라이브러리이다[20].

스파크 어플리케이션을 통해 전처리를 진행하는 순서는 다음과 같다. 예를 들어 JSON 형태의 원본 데이터가 실시간으로 들어올 때 스파크 스트리밍 어플리케이션은 일반적으로 JSON을 파싱하여, {"key": "value"} 쌍 중에서 value 값을 추출한다. 이때 추출된 값은 데이터 타입을 확인 후 같은 데이터 타입으로 통일시킨다. 이렇게 수치화된 넘파이 배열 형태의 Dstream 데이터를 Spark Streaming에서 제공하는 foreachRDD 연산을 통해 RDD 단위로 전처리된 데이터를 분석 모듈에 일정 시간 간격으로 입력하는 작업을 수행한다. 모델의 경우 다양한 종류의 데이터가 입력될 수 없기 때문에 스파크 어플리케이션과 실시간 분석 모듈은 1:1 구조를 갖도록 구성하였다[16].

### 3.2 적용 시나리오

스마트시티를 위한 빅데이터 처리 아키텍처에 대한 모의 검증을 진행하기 위해 딥러닝 프레임워크인 텐서플로우 기반 실시간 데이터 분석 모델 모듈과 데이터 시각화 모듈을 추가하여 전체적인 데이터 흐름도를 위의 <그림 9>와 같이 구성하고 시나리오를 수립하였다.

스마트시티의 IoT 센서를 통해 실시간 데이터가 들어온다는 가정을 위해 자체 보유 중인 미세먼지 센서

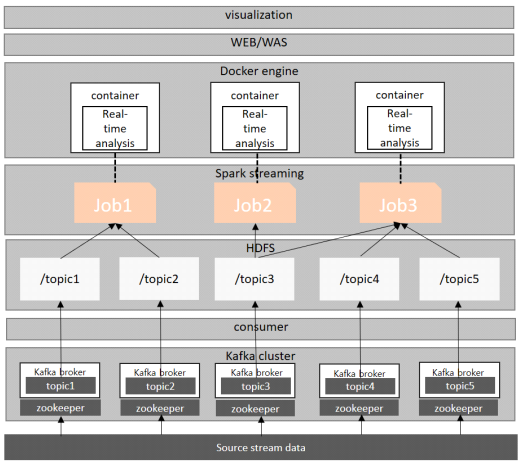


그림 9. 시나리오 적용을 위한 데이터 흐름도
Fig. 9. Data Flow Diagram for Applying Scenario

를 이용하였다. 1초 주기로 데이터를 확인할 수 있으며, 도시에서 생성될 수 있는 시계열 데이터이기 때문에 모의 검증을 위해 적합하다. 데이터의 명세와 속성은 아래 <그림 10>에서 확인할 수 있다.

앞에서 설명한 시계열 공기질 데이터에 대한 예측 결과를 도출하기 위해 LSTM(Long Short-term Memory) 모델을 사용하였다. LSTM 모델은 인공신경망의 한 종류인 RNN(Recurrent Neural Network)의 데이터 장기 의존성 문제를 해결하여 RNN에 비해 예측정확도 부분에서 뛰어나다[21].

모의 검증 프로세스는 데이터 저장 및 데이터 처리 모듈을 거쳐 실시간으로 예측 결과를 제공하는 모델 서빙 모듈에 데이터를 보내면 모델에 의한 예측 결과


```
{
  "O3":41.45000076293945,
  "created":"2020-07-16 16:17:25",
  "H2S":70.19999694824219,
  "VOC":587.4299926757812,
  "logKey":"1594883845719",
  "CO":930.0,
  "deviceId":"0358777071041256",
  "connected":true,
  "NO2":29.299999237060547,
  "RAWPM25":15.0,
  "Temp":26.520000457763672,
  "Humidity":52.79999923706055,
  "SO2":102.19999694824219,
  "AIPM25":12.109999656677246
}
```


그림 10. 아키텍처 모의 검증 시 사용된 실시간 미세먼지 데이터
Fig. 10. Real-time Air-quality Data used in Simulated Architecture Verification

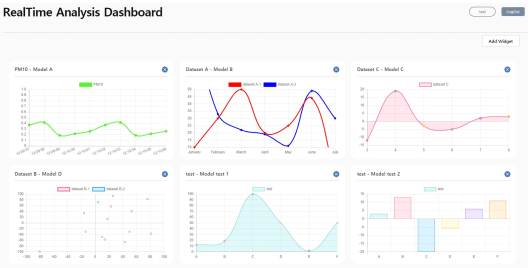


그림 11. 스마트시티를 위한 실시간 분석 대시보드
Fig. 11. Real-time Analysis Dashboard for a Smart City

값이 도출되고, 최종적으로 아래 <그림 11>과 같이 웹서버를 통해 결과 데이터에 대해 모니터링을 하게 된다.

여기에서 데이터 저장 모듈의 HDFS에 저장되는 블록의 크기는 16MB로 설정하였다. HDFS의 기본 블록 값은 64MB이지만 수집하는 데이터의 크기에 따라 디스크 자원에 영향을 줄 수 있기 때문에, 오버헤드는 늘어나지만 디스크 자원을 최적화하기 위해 블록의 크기를 줄였다[22].

여기에서 모델 서빙 모듈인 TFServing은 텐서플로우를 사용하여 학습된 모델에 한해서 실시간 모델 서빙을 제공할 수 있는데, 단일 CPU 코어 기준으로 초당 100,000건의 추론요청을 처리할 수 있는 성능을 보유하고 있다. 모델 서빙은 컨테이너 방식을 사용하기 때문에, 분석 프로세스를 스파크 작업이 실행되고 있는 호스트 운영체제와 격리된 상태로 동작하기 때문에 가볍고 빠르다[23].

LSTM 모델을 서빙하고 있는 TFServing 컨테이너로 REST API의 POST 방식으로 데이터를 전달한다[24].

앞서 설명한 핵심 역할을 하는 코어 모듈부터 이와 연결된 부가적인 모듈들을 통해 보여주고자 하는 아키텍처 내의 데이터 플로우은 다음과 같다.

먼저, 카프카 클러스터를 통해 수집된 공기질 데이터가 실시간으로 HDFS에 적재된다. 둘째, HDFS의 내부 디렉토리에 쌓인 데이터에 대해 Spark Streaming이 즉시 읽고 처리하는 작업을 수행한다. 셋째, 전처리된 데이터는 Tensor Flow 기반의 LSTM 모델로 입력된다. 넷째, 모델을 통해 반환된 예측 결과 데이터를 웹서버를 통해 모니터링 할 수 있게 된다.

예측 결과 데이터를 반환 받기까지 소요되는 시간은 요청하는 데이터의 크기 및 모델 신경망의 복잡도에 따라 상이하지만 일반적으로 10ms 이내에 반환 받을 수 있을 것으로 기대된다.

## Ⅳ. 결 론

본 논문에서는 스마트시티에 적용하기 위한 실시간 빅데이터 분석 플랫폼에 대한 아키텍처를 제안하였다. 제안한 플랫폼을 사용하면 스마트시티에서 발생하는 방대한 데이터들에 대한 실시간 데이터 처리가 가능하며 스마트시티 관리자가 원하는 기계학습 모델을 적용하여 대시보드를 통해 예측 및 추론에 대한 결과를 확인할 수 있다. 이로써 제안한 실시간 빅데이터 분석 플랫폼이 도시 문제 해결에 기여할 수 있을 것으로 사료된다.

하지만 본 논문은 다음과 같은 한계점들을 갖는다. 먼저, 스마트시티에서 발생하는 데이터를 다루지 못했다는 점이다. 실제 스마트시티에서는 다양한 IoT 센서 기반으로 방대한 데이터가 발생하는데 제안한 아키텍처에 적용하지 못하였다. 따라서 향후 연구에서는 더욱 방대한 데이터를 적용시키는 연구를 진행할 필요가 있다.

둘째, 실시간 데이터 처리를 위한 아키텍처에 대한 세부 튜닝에 대한 비교분석을 하지 않았다는 점이다. 예를 들어 본 논문에서 제안한 아키텍처 중 카프카의 경우 토픽의 개수, 클러스터의 브로커 개수, 브로커마다 파티션 개수 등을 고려하여 구성을 해야 성능을 고려할 수 있는데 성능 튜닝에 대한 면밀한 분석이 필요하다. 따라서 다음 연구에서는 아키텍처를 구성하는 프레임워크들의 성능 튜닝에 따른 아키텍처별 성능 비교 연구가 필요하다.

셋째, 실시간 빅데이터 분석 플랫폼을 구성하는 아키텍처이므로 예측 및 추론을 위한 기계학습 모델에 대한 예측 및 추론 성능에 대해 고려하지 않았다는 점이다. 본 논문에서는 모델을 이용한 예측 및 추론으로 넘어가기 전인 실시간 데이터 처리에 중점을 두었기 때문에 다음 연구에서는 데이터 전처리부터 다양한 기계학습 모델에 대한 입력 및 출력되는 모델 서빙(Serving) 성능까지 고려하여 실시간 빅데이터 분석 플랫폼으로서 기능을 갖출 수 있는 성능 비교 연구가 필요하다.

마지막으로 실제 운영되고 있는 어플리케이션 서비스들의 실시간 빅데이터 분석 아키텍처들과의 비교 검증을 하지 않았기 때문에 제안하는 아키텍처에 대한 일반성이 충분히 검증되지 못하였다. 따라서 실제 서비스 되고 있는 어플리케이션들의 빅데이터 아키텍처들과의 기능적인 차이점을 비교하는 것이 향후 연구과제가 될 것이다.

## References


[1] Y. K. Park and S. M. Rue, "Analysis on smart city service technology with IoT," *Korea Inst. Inf. Technol. Mag.*, vol. 13, no. 2, pp. 31-37, Dec. 2015.

[2] I. H. Lee, D. H. Park, Y. S. Son, Y. H. Lee, and T. J. Park, "Technology trends of IoT-based smart city application," *Commun. KIISE*, vol. 36, no. 7, pp. 61-68, Jul. 2018.

[3] H. J. Kim, "Deep learning city: A big data analytics framework for smart cities," *Inf. Policy*, vol. 24, no. 4, pp. 79-92, Dec. 2017.

[4] B. J. Jeon and H. W. Kim, "An exploratory study on the sharing and application of public open big data," *Inf. Policy*, vol. 24, no. 3, pp. 27-41, Sep. 2017.

[5] S. W. Kang, Y. S. Yoon, and H. W. Lee, "Big data analysis platform technology based on lambda architecture using spark, kafka and cassandra," in *Proc. Symp. KICS*, pp. 313-314, Daegu, Korea, Nov. 2017.

[6] J. H. Moon, Y. Yun, and Y. T. Shin, "Design of hierarchical distributed SDN controller based on apache kafka," in *Proc. KISS Conf.*, pp. 1295-1297, Jeju Island, Korea, Jun. 2019.

[7] http://kafka.apache.org/24/documentation.html

[8] D. S. Kim, S. W. Son, M. S. Gil, and Y. S. Moon, "Apache kafka benchmark test on HDD and SSD environment," in *Proc. KISS Conf.*, pp. 1691-1693, Jeju Island, Korea, Jun. 2017.

[9] N. Neha, S. Gwen, and P. Todd, *Kafka the definitive guide*, jpub, 2018.

[10] http://hadoop.apache.org

[11] D. C. Kang, J. Y. Choi, and J. S. Kim, "Study on performance and stable execution environment of distributed message processing system based on Hadoop," in *Proc. KISS Conf.*, pp. 1475-1477, Busan, Korea, Jun. 2014.

[12] D. C. Kang, J. Y. Choi, and J. S. Kim, "Study on performance and stable execution environment of distributed message processing system based on Hadoop," in *Proc. KISS*

*Conf.*, pp. 55-57, Jeju Island, Korea, Dec. 2018.
[13] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
[14] K. Y. Ji and Y. M. Kwon, "Performance comparison of python and scala APIs in spark distributed cluster computing system," *J. Multimedia Inf. Syst.*, vol. 23, no. 2, pp. 241-246, Feb. 2020.
[15] K. J. Park, "A design on informal big data topic extraction system based on spark framework," *KIPS Trans. Software and Data Eng.*, vol. 5, no. 11, pp. 521-526, 2016.
[16] E. S. Lee, J. H. Kim, and S. S. Hong, "Selective resource managing technique for real-time processing in spark streaming system," in *Proc. KISS Conf.*, pp. 64-66, Jeju Island, Korea, Jun. 2017.
[17] https://spark.apache.org/docs/latest/streaming-programming-guide.html
[18] S. H. Kim, M. K. Sung, and Y. D. Jung, "An implementation of two-phase locking protocol on distributed cloud systems using ZooKeeper," in *Proc. KISS Conf.*, pp. 7-9, Jeju Island, Korea, Jun. 2012.
[19] https://classic.yarnpkg.ccom/en/docs
[20] https://numpy.org
[21] H. S. Lee, K. H. Kim, H. M. Jung, H. S. Lee, H. S. kim, and J. H. Park, "A study on wind power forecasting using LSTM method," *The Trans. KIEE*, vol. 69, no. 8, pp. 1157-1164, Aug. 2020.
[22] S. R. Alapati, *Expert hadoop administration*, 1st Ed., Addison-Wesley Professional, 2016.
[23] J. T. Park, H. G. Kim, and I. Y. Moon, "Design and implementation of web compiler for learning of artificial intelligence," *J. Korea Navig. Inst.*, vol. 21, no. 6, pp. 674-679, 2017.
[24] Y. M. Bae, S. J. Jung, and W. Y. Soh, "Comparative analysis of the virtual machine and containers methods through the web server configuration," *J. ICCE*, vol. 18, no. 11, pp. 2670-2677, Nov. 2014.

**이 모 세** (Mo-se Lee)

2014년 2월 : 국민대학교 기계시스템공학 졸업
2019년 2월 : 국민대학교 비즈니스IT전문대학원 공학석사
2020년 : 이노커스 연구원
2021년~현재 : 브이알에듀 책임연구원
<관심분야> Data Engineering, Deep Learning
[ORCID:0000-0003-4919-1867]

**강 민 수** (Min-su Kang)

2020년 2월 : 한서대학교 항공소프트웨어공학과 졸업
2020년 2월 : 이노커스 연구원
2021년~현재 : 브이알에듀 선임연구원
<관심분야> Deep learning, Data engineering, Linux, System Architecture
[ORCID:0000-0002-7855-1342]

**김 홍 준** (Hong-joon Kim)

1996년 7월 : 연세대학교 기계공학과 졸업
1998년 8월 : 연세대학교 기계공학과 석사
2004년 2월 : 아주대학교 경영대학원 MBA
<관심분야> 기계공학, 전파공학, 마케팅, 5G, 스마트시티, 클라우드, 인공지능, 빅데이터, 블록체인
[ORCID:0000-0003-2701-6953]

**김 재 헌** (Jae-hun Kim)

2009년 3월 : 와세다대학교 경제학과 졸업
2017년 2월 : 고려사이버대학교 실용외국어학과 졸업
2019년 2월 : 고려대학교 교려대학원 중일지역비교문화 석사 수료
<관심분야> 마케팅, 클라우드, 인공지능, 빅데이터
[ORCID:0000-0002-2218-0976]