공학석사학위논문

하둡 에코시스템 기술을 적용한 교통 스트리밍 데이터의 저장 및 처리기술 설계 및 구현

A Design and Implementation of Storage and Processing Method for Traffic Streaming Data Using Hadoop Ecosystem

충 북 대 학 교 대 학 원

빅데이터협동과정 빅데이터 전공

김 진 혁

2017 년 2 월

공학석사학위논문

하둡 에코시스템 기술을 적용한 교통 스트리밍 데이터의 저장 및 처리기술 설계 및 구현

A Design and Implementation of Storage and Processing Method for Traffic Streaming Data Using Hadoop Ecosystem

지도교수 조 완 섭

빅데이터협동과정 빅데이터 전공

김 진 혁

이 논문을 공학석사학위 논문으로 제출함.

2017 년 2 월

본 논문을 김진혁의 공학석사학위 논문으로 인정함.

심 사 위 원 장　　최 상 현 ㊞

심 사 위 원　　조 완 섭 ㊞

심 사 위 원　　이 경 희 ㊞

충 북 대 학 교 대 학 원

2017 년 2 월

차 례

그림 차례

표 차례

A Design and Implementation of Storage and Processing Method for Traffic Streaming Data Using Hadoop Ecosystem*

Jin Hyuk Kim

Department of Big Data

Graduate School, Chungbuk National University

Cheongju, Korea

Supervised by Professor Cho, Wan-Sup

Abstract

Traffic big data analysis system in Cheongju City in Korea collects big data from various transportation devices such as BIS(Bus Information System), ATMS(Advanced Traffic Management System), and transportation card transaction system each day in a batch fashion (4 million log data per day). We propose a streaming big data processing system for realtime collection, storage, and processing of the transportation data. We adopt Kafka, Hadoop/Spark packages on a cloud system for realtime collection, storage, and processing, respectively. The cloud system consists of 14 nodes currently. For performance evaluation, we divided the system into 3 components: collection capability, storage capability, and processing capability. For collection capability, the size of the message set is the major factor for the performance. The storage an processing capability mostly depend on the number of nodes in the cloud, and the system seems to be suitable for the current traffic load in Cheongju city.

Key words: Streaming Data Processing, Hadoop Ecosystem, Spark Streaming

---

* A thesis for the degree of Master in February 2017.

Ⅰ. 서 론

1.1 연구 배경

본 연구는 교통 스트리밍 데이터의 수집·저장 시스템과 실시간 처리 시스템을 구현하기 위한 시스템 설계와 구현 방법에 대해 논한다. 스트리밍 데이터는 다양한 디바이스로부터 수집되는데 그 기반은 유비쿼터스 컴퓨팅(Ubiquitous Computing) 환경에서 시작된다. 유비쿼터스 컴퓨팅의 아버지인 Mark D. Weiser는 여러 논문들을 통해 기술이 인간의 삶에 녹아들어 인간이 기술의 존재를 인지하지 못하도록 하는 것이 유비쿼터스 컴퓨팅 환경이라고 주장하였다(Mark D. Weiser, 1991; Mark D. Weiser, 1994; Mark D. Weiser, 1996). 유비쿼터스 컴퓨팅 환경은 최근 사물인터넷(Internet of Things) 기술을 통해 자리 잡고 있으며, 이런 환경을 IoT 환경이라 부른다. IoT 환경은 다양한 IoT를 구성하는 디바이스들이 사용자의 인지 또는 간섭 없이 서로 연결과 통신을 통해 사용자에게 서비스를 제공하는 환경을 말한다(김영준, 2013).

IoT 환경의 확산은 여러 분야의 산업에서 다양한 형태의 서비스로 구축되고 있으며 IoT 환경의 규모적 변화에 의해 새로운 서비스가 형성되기도 한다(신승혁, 2015). 이런 환경은 교통 환경에서도 적용되고 있으며 교통량, 통행시간, 통행속도, 기종점 통행량 등 실시간으로 수집되는 데이터를 통해 실시간 교통정보 제공, 실시간 신호 제어를 포함한 교통 운영에 활용 가능할 것이며, 이미 국내에 IoT를 도시 교통에 적용하는 사례도 생겨나고 있다(서원호, 2015).

IoT 디바이스는 실시간으로 끊임없이 데이터를 생성하는데 이를 스트리밍 데

이터(Streaming Data)라 한다. 이런 스트리밍 데이터는 빅데이터의 3가지 특징을 모두 포함한다. 즉, 거대한 데이터의 양(Volume), 빠르게 생성되며(Velocity), 그리고 데이터의 다양성(Variety)이 특징이다. 빅데이터를 저장 및 처리하기 위한 대표적인 플랫폼으로 하둡(Hadoop)이 있다(J. Dean, 2004). 하둡은 대용량 데이터를 저장하기 위한 HDFS(Hadoop Distributed File System)를 지원하며 대용량 데이터를 처리하기 위해 MapReduce Framework를 지원한다. 하둡은 대용량 데이터를 저장하고 일괄(Batch) 처리하는 용도에는 적합하지만 데이터를 빠르게 처리하는데 한계가 있다(복경수, 2016). 또한 하둡에서 스트리밍 데이터 수집이 어렵기 때문에 이를 위한 시스템도 별도로 필요하다.

스트리밍 데이터는 수 만 개 이상의 데이터 소스에서 연속적으로 생성되며 일정 시간마다 반복적으로 데이터 레코드를 전송한다. 스트리밍 데이터는 시간 정보가 포함된 시계열 데이터이며, 한 번 지나간 데이터는 복원이 불가능한 특징이 있다. 또한 빠른 주기로 값이 유동적으로 변하는 특징으로 기존 일괄처리 방식을 이용한 기법으로는 스트리밍 데이터의 생성 속도를 따라갈 수 없다(정성민, 2016). 따라서 실시간으로 스트리밍 데이터를 수집・저장・처리할 수 있는 시스템을 요구한다.

스트리밍 데이터를 수집하기 위해 분산 메시지 처리 시스템인 아파치 카프카(Apache Kafka)를 고려할 수 있다. 카프카는 로그 처리 시스템에서 필요 없는 기능을 제거하고 성능 위주로 설계함으로써 실제로 링크 속도에 육박하는 성능을 보여준다(Jay Kreps, 2011). 로그 역시 스트리밍 데이터로 빠르게 생성되고 짧은 시간에 대용량 데이터를 만들어낸다. 스트리밍 데이터 처리는 스파크 스트리밍(Spark Streaming), 스톰(Storm), 플룸(Flume) 등을 고려할 수 있다. CEP(Complex

Event Processing) 엔진인 Esper도 스트리밍 데이터 처리가 가능하지만 분산 처리를 지원하지 않아 분산 환경에 적용하기 어려운 단점이 있다. 따라서 본 연구는 CEP 엔진을 고려하지 않고 분산 환경을 기반으로 하는 하둡 에코시스템(Hadoop Ecosystem) 플랫폼 기반으로 시스템을 설계하고 구현방법에 대해 논한다.

1.2 연구 목적

본 연구의 목적은 교통 스트리밍 데이터의 수집·저장을 위한 시스템과 교통 스트리밍 데이터의 실시간 처리를 위한 시스템의 설계 및 구현을 목적으로 한다. 시스템 설계 및 구현은 하둡 에코시스템의 API를 기반으로 자바언어를 활용하여 객체지향 방법으로 문제를 해결한다.

논문에서는 제안된 방법을 실제로 구축하여 검증하기 위해 청주시 교통빅데이터 분석시스템(조완섭, 2015)에서 스트리밍 빅데이터의 실시간 수집·저장 시스템과 처리 시스템을 설계하고 구현하였다. 청주시 교통 빅데이터 분석시스템은 청주시 전역에서 발생하는 BIS, ATMS, 교통카드 정보를 일단위로 수집하고 분석하는 배치 형태의 시스템 (일단위로 데이터 수집)이며, 향후 예측 기능을 보완하기 위해서 실시간 시스템으로의 확장이 필요하다.

본 연구에서 구축한 시스템은 교통 빅데이터 수집을 위해 카프카를 사용하며, 배치형태의 저장과 처리를 위해 Hadoop를 활용한다. 또한 실시간 처리를 위해 스파크 스트리밍 기술을 활용한다. 또한 제안된 시스템의 성능을 평가하기 위해 카프카를 이용한 데이터 수집능력, 표준화된 밴치마킹 방식인

TPCx-HS를 활용한 저장능력, 스트리밍 데이터 처리능력 세가지로 구분하여 성능을 평가한다. 수집능력의 경우 메시지셋의 크기를 환경에 적절하게 결정하는 것이 성능에 큰 영향을 미치며, 저장능력의 경우에는 스캐일아웃 방식이므로 노드수의 증가에 따라 개선될 수 있으며, 처리능력은 초당 200건 정도 처리할 수 있어 청주시 교통데이터를 처리하는 데는 충분하며, 필요한 경우 노드수 증가를 통해 필요한 만큼의 성능을 확보할 수 있을 것이다.

1.3 논문의 구성

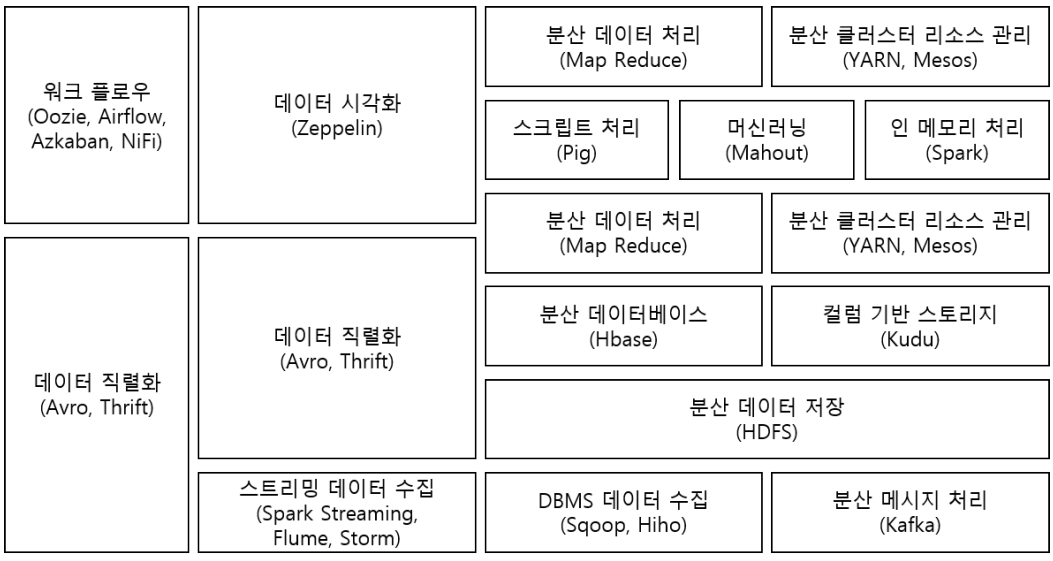
본 논문은 5개의 장으로 구성된다. 제1장은 연구에 대한 배경과 목적에 대하여 기술하고, 제2장은 이론적 접근 방법(배경)과 스트리밍 데이터 처리 관련 연구 및 사례에 대해 기술한다. 제3장은 본 연구에서 제안하는 시스템의 설계와 구현에 대해 설명한다. 제4장은 기존 시스템과의 비교와 평가에 대해 서술하며 기술하고, 마지막 제5장은 본 연구의 결론으로 연구에 따른 공헌과 한계점 및 향후 연구과제에 대해 논한다.

Ⅱ. 관련연구

본 장에서는 스트리밍 데이터 수집·저장·처리를 위한 하둡 에코시스템 기반 빅데이터 플랫폼과 본 연구와 관련된 연구들을 설명한다.

2.1 하둡 에코시스템

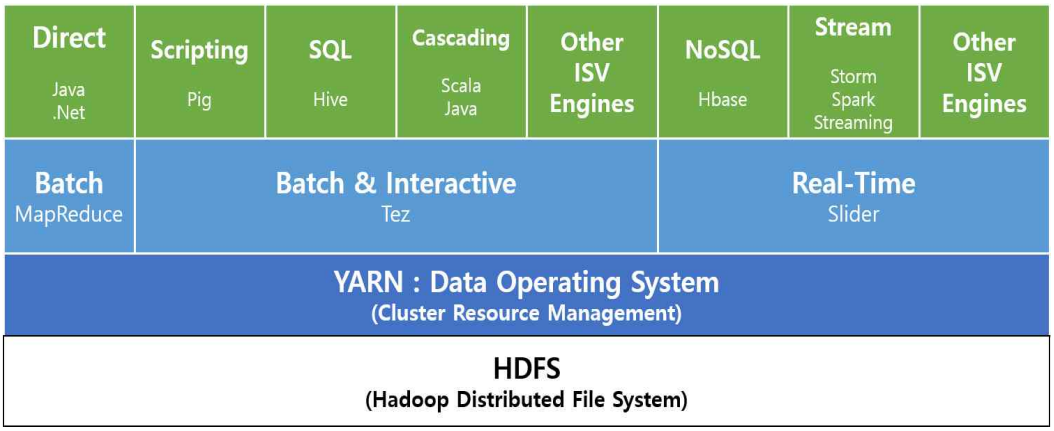
하둡은 대량의 자료를 처리할 수 있는 클러스터 컴퓨터 환경에서 동작하는 분산 응용 프로그램을 지원하는 프레임워크이다. 빅데이터 기술의 표준으로 자리 잡은 하둡을 기반으로 주변 시스템들이 개발되었는데 이를 하둡 에코시스템이라 부른다(그림 1).



[그림 1] 하둡 에코시스템 구성도 (정재화, 2016)

2.1.1 아파치 하둡(Apache Hadoop)

하둡은 대용량 데이터를 분산 처리할 수 있는 오픈소스 프레임워크로 구글이 논문으로 발표한 GFS(Google File System)와 맵리듀스(MapReduce) 프레임워크를 2005년 더그 커팅에 의해 구현된 결과물이다(정재화, 2016). 하둡은 HDFS(Hadoop Distributed File System)에 데이터를 저장하고 분산처리 시스템인 맵리듀스 프레임워크를 통해 데이터를 일괄(batch) 처리한다. 하둡 1.0은 HDFS와 MapReduce 프레임워크 기반의 일괄 처리에 최적화 된 빅데이터 플랫폼이었으나, Hadoop 2.0에서 Yarn의 등장과 함께 interactive, streaming, graph 처리 등과 같이 다양한 데이터 처리 시스템을 Hadoop이 제공하며 시스템 간 자원을 공유할 수 있다(김직수, 2016)(그림 2).



[그림 2] Hadoop 2.0의 시스템 구성도

HDFS는 마스터/슬레이브 구조를 갖는다. 게다가 클러스터의 각 노드에는 데이터노드가 하나씩 존재하고, 이 데이터노드는 실행될 때마다 노드에 추가되는 저장소(Storage)를 관리한다. HDFS는 네임스페이스를 공개하여 유저데이터

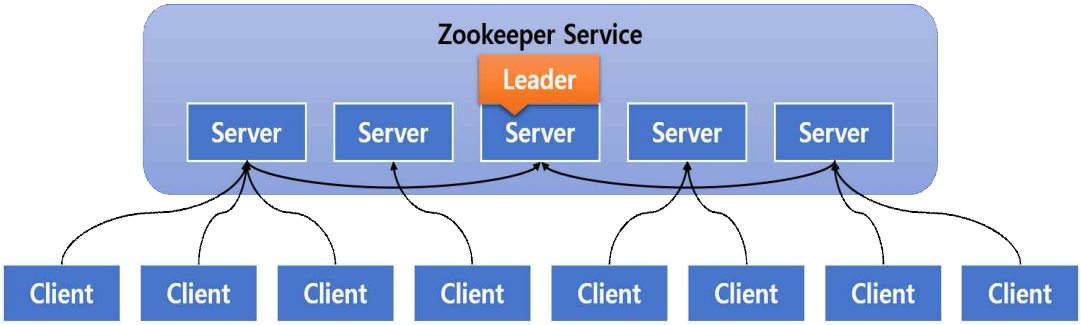
가 파일에 저장되는 것을 허락한다. 내부적으로 하나의 파일은 하나 이상의 블록으로 나뉘어 있고, 이 블록들은 데이터노드들에 저장되어 있다. 네임노드는 파일과 디렉터리의 읽기, 닫기, 이름 바꾸기 등, 파일 시스템 네임스페이스의 여러 기능을 수행한다. 또한, 데이터노드와 블록들의 맵핑(Mapping)을 결정한다. 데이터 노드는 파일시스템의 클라이언트가 요구하는 읽기, 쓰기 기능들을 담당한다. 또한 데이터노드는 네임노드에서의 생성, 삭제, 복제 등과 같은 기능도 수행한다.

2.1.2 아파치 주키퍼(Apache Zookeeper)

주키퍼는 분산 코디네이션 서비스 시스템으로 분산된 시스템간의 정보 공유, 클러스터에 있는 서버들의 상태 체크, 분산된 서버들 간의 동기화 이슈 처리 등의 역할을 담당한다. 주키퍼는 분산 시스템 내에서 중요한 상태 정보나 설정 정보를 유지하기 때문에, 코디네이션 서비스의 장애는 전체 시스템의 장애를 유발하게 된다. 따라서 이중화를 통해 고가용성(High Availability)을 제공하는 것이 필요하다. 주키퍼는 오직 분산 시스템을 코디네이션 하는 용도로 설계되었기 때문에 데이터 접근 속도가 빨라야 하며 자체적으로 장애에 대한 대응성을 가져야 한다. 그래서 주키퍼는 자체적으로 클러스터링 기능을 제공하고 장애에도 데이터 유실 없이 서비스 복구(fail-over)가 가능하다.

주키퍼는 리더가 되는 서버가 하나 존재하며 모든 서버에 중심이 된다. 주키퍼 서비스의 각 서버는 서로에 대한 정보를 공유해야 한다. 각 서버는 하드디스크와 같은 영구 저장소(Persistence Storage) 내에 있는 서버 상태 이미지,

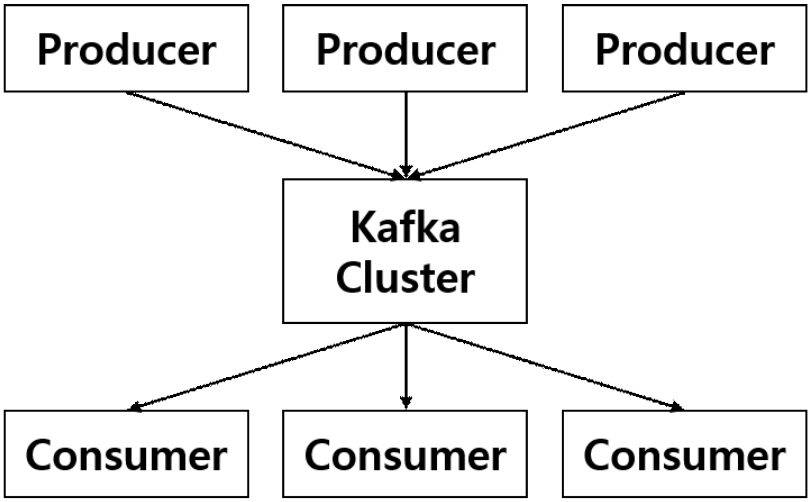
트랜잭션 로그, 스냅샷 정보 등을 시스템 메모리 내에 유지한다. 클라이언트는 하나의 주키퍼 서버에 TCP 방식으로 연결된다. 연결된 하나의 세션은 요청, 응답, 이벤트, 그리고 하트 비트(heart beat) 등의 정보를 보낸다. 만약 TCP 연결이 종료되면 클라이언트는 다른 서버에 연결요청을 수행한다(그림 3).



[그림 3] 주키퍼 서비스 개념도
(이미지 출처: Zookeeper 3.3 Documentation)

2.1.3 아파치 카프카(Apache Kafka)

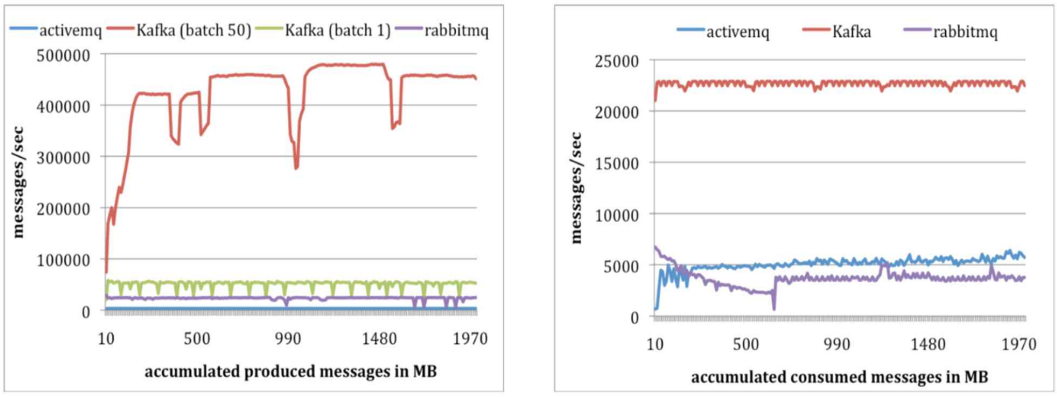
카프카는 LinkedIn에서 개발된 분산 메시징 시스템으로써 2011년 오픈소스로 공개되었다. 카프카는 발행-구독(publish-subscribe) 모델을 기반으로 동작하며 프로듀서(Producer), 컨슈머(consumer), 브로커(broker)로 구성된다(그림 4).



[그림 4] 카프카 구성 요소
(이미지 출처: Apache Kafka 0.8.1 Documentation)

카프카의 브로커는 토픽(topic)을 기준으로 메시지를 관리한다. 프로듀서는 특정 토픽의 메시지를 생성한 뒤 해당 메시지를 브로커에 전달한다. 브로커가 전달받은 메시지를 토픽별로 분류하여 쌓아두면, 해당 토픽을 구독하는 컨슈머들이 메시지를 가져가서 처리한다.

카프카의 토픽은 파티션(partition)이라는 단위로 쪼개어져 클러스터의 각 서버들에 분산되어 저장되고, 고가용성을 위해 복제 설정을 할 경우 파티션 단위로 각 서버에 복제되며, 장애 발생 시 파티션 단위로 복구(fail over)가 수행된다. 카프카는 대용량의 실시간 로그 처리에 특화된 아키텍처 설계를 통해 기존 메시징 시스템보다 우수한 TPS(Throughput Per Second)를 보여준다(Jay Kreps, 2011).



[그림 5] 카프카 생산자/소비자의 TPS (Jay Kreps, 2011)

붉은색 그래프는 메시지를 한 번에 50개씩 전송한 결과이고, 연두색 그래프는 한 번에 하나씩 전송한 결과이다. 마이크로 배치처리 한 경우 더 우수한 성능을 보여준 것을 알 수 있다. 또한 경쟁 시스템인 RabbitMQ, ActiveMQ 보다 나은 성능을 보여주는 것을 알 수 있다(그림 5).

카프카는 확장성과 고가용성을 지원하기 위해 클러스터로 구성되어 동작하게끔 설계되었다. 클러스터 내의 브로커에 대한 분산처리는 주키퍼가 담당한다.

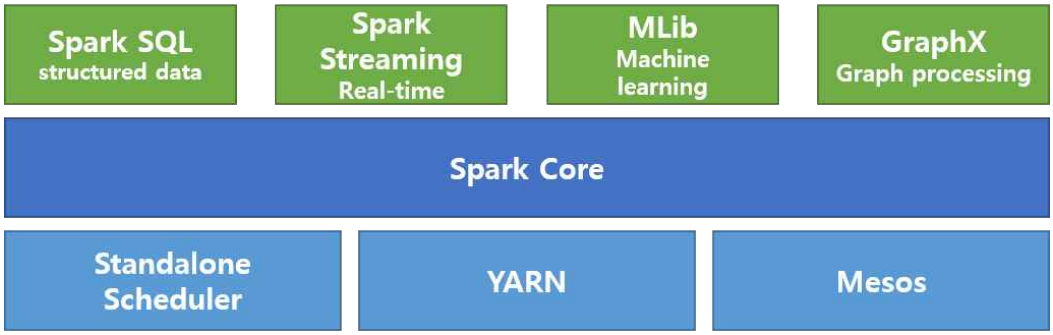
2.2 스트리밍 데이터 프로세싱 플랫폼

스트리밍 데이터 프로세싱 플랫폼은 오로라(Aurora), S4, 스파크 스트리밍(Spark Streaming), 스톰(Storm), 플린크(Flink), 삼자(Samza), 그리고 아마존 키네시스(Kinesis) 등 종류가 다양하다. 각 플랫폼의 디자인, 연산 모델, 그리고 시스템의 구조는 서로 상이하며 연산하고자 하는 알고리즘에 따라 다른 성능을 보여준다.

본 절은 스트리밍 데이터 프로세싱 플랫폼으로 오픈 소스 소프트웨어 중 가장 많이 사용되며 오픈소스 프로젝트 기여자(contributor)가 활발히 활동하고 있는 3가지 플랫폼인 아파치 스파크, 아파치 스톰, 아파치 플린크에 대해 소개한다.

2.2.1 아파치 스파크(Apache Spark)

스파크는 오픈소스 클러스터 컴퓨팅 프레임워크로 2009년 University of California at Berkeley의 AMPLab에서 처음 개발되었고 2010년 오픈소스화 되어 아파치 프로젝트에 올라갔다. 스파크는 대용량의 데이터를 고속처리를 위해 설계되었고 RDD(Resilient Distributed Dataset) 형식으로 메인 메모리에서 데이터를 처리 한다. 스파크는 클러스터 매니저(Standalone Scheduler, YARN, Mesos), 코어 엔진, 그리고 워크로드 컴포넌트(Spark SQL, Spark Streaming, MLib, GraphX)로 구성된다(그림 6).



[그림 6] 스파크 컴포넌트 구성도
(이미지 출처: Apache Spark Streaming 2.0.2 Documentation)

스파크는 모든 작업을 새로운 RDD로 만들거나, 이미 존재하는 RDD를 변형해서 작업을 수행한다. 스파크는 내부적으로 RDD에 있는 데이터를 분산 저장하며 클러스터 위에서 수행되는 연산을 병렬 처리한다. 스파크는 배치 애플리케이션, 반복 알고리즘, 대화형 쿼리, 스트리밍과 같은 다양한 컴퓨팅 방법을 지원한다.

스파크 스트리밍(Spark Streaming)은 코어 스파크 API의 확장으로 스트리밍 데이터를 처리하기 위한 API를 제공한다. 또한 높은 처리 성능(high throughput), 내고장성(fault-tolerant)을 제공하며 스파크가 기존에 갖고 있던 확장성(scale-out)의 유연함을 그대로 갖는다. 끊임없이 들어오는 스트리밍 데이터를 스파크 스트리밍이 배치 간격(batch interval) 마다 데이터를 나누고, 나누어진 배치 데이터를 스파크 코어(Spark Core) 엔진이 처리해서 배치 간격 마다 결과를 반환 한다(그림 7).

[그림 7] 스파크 스트리밍 데이터 흐름
(이미지 출처: Apache Spark Streaming 2.0.2 Documentation)

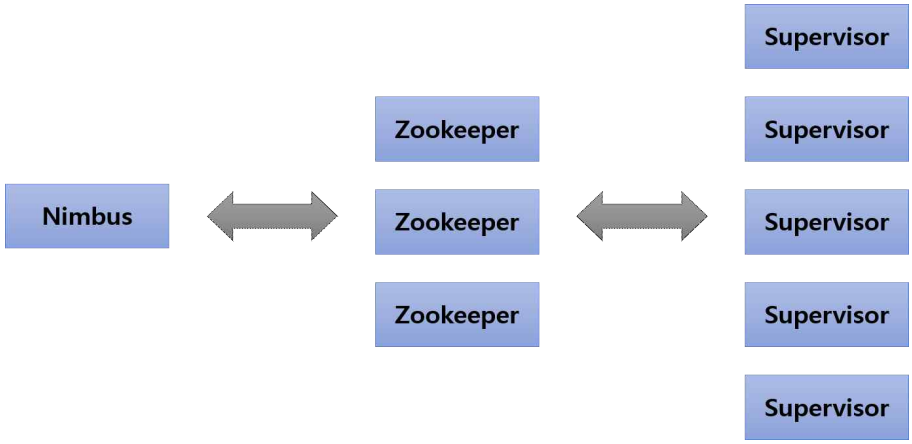
스파크 스트리밍은 다양한 플랫폼으로부터 데이터를 받아 처리할 수 있다. 대표적인 플랫폼으로 하둡 파일 시스템(HDFS), 카프카, 플룸(Flume)을 들 수 있다. 마이크로 배치 크기로 데이터를 받아 데이터를 처리 후 하둡 파일 시스템, 데이터베이스 또는 시각화 Dashboard에 결과를 반환할 수 있다(그림 8).

[그림 8] 스파크 스트리밍 시스템 연계 흐름
(이미지 출처: Apache Spark Streaming 2.0.2 Documentation)

2.2.2 아파치 스톰(Apache Storm)

아파치 스톰은 오픈소스 소프트웨어로 분산 실시간 연산 시스템이다. 스톰의 아키텍처는 하둡과 매우 유사한데 하둡은 맵리듀스 프레임워크를 이용해서 데이터를 처리하는데 반해 스톰은 토폴로지 작업을 수행한다. 맵리듀스와 토폴로지의 차이는, 맵리듀스는 정해진 데이터 셋을 배치 처리하지만 토폴로지는 실시간으로 지속적으로 메시지를 처리한다는 점이다. 스톰에도 하둡의 잡트레커(Job Tracker) 역할을 하는 님버스(Nimbus) 데몬과 하둡의 테스크 트레커(Task Tracker) 역할을 하는 슈퍼바이저(Supervisor) 데몬이 있다.

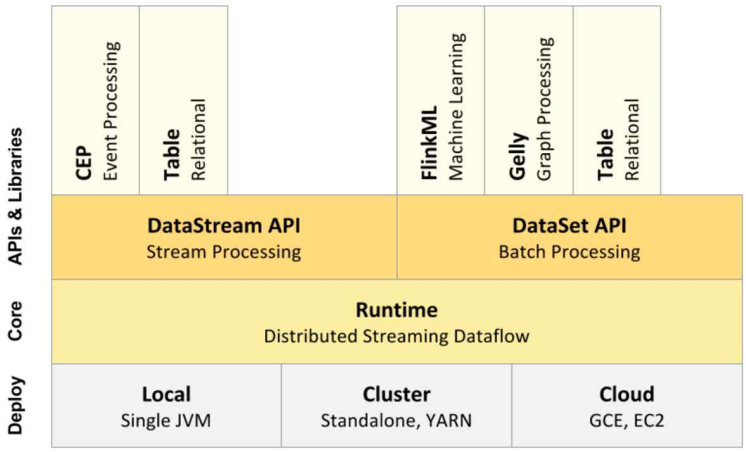
님버스 데몬은 슈퍼바이저 노드에 작업을 할당하고 장애 발생시 조치를 취한다. 슈퍼바이저 노드는 할당된 토폴로지의 일부를 처리할 작업 프로세스(Worker Process)의 구동을 담당한다. 님버스와 슈퍼바이저 노드는 주키퍼를 사용해 장애 상황에 대처한다(그림 9).



[그림 9] 스톰의 클러스터 시스템 구성도

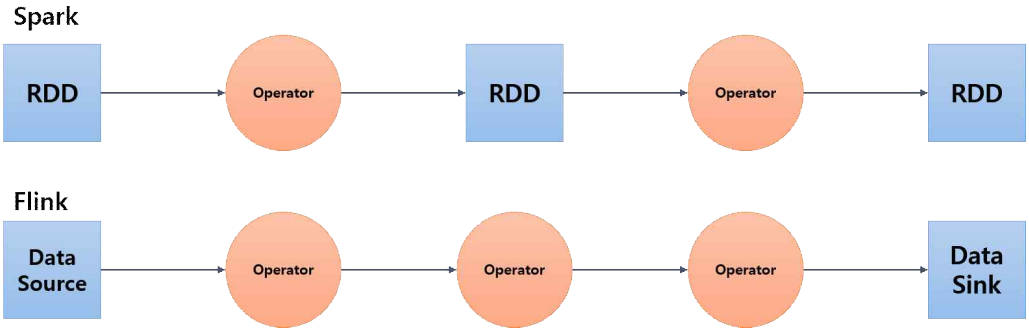
2.2.3 아파치 플린크(Apache Flink)

아파치 플린크는 분산 스트리밍과 배치 데이터 처리 엔진으로 오픈소스 플랫폼이다. 플린크의 컴포넌트는 클러스터 매니저, 플린크 코어, 그리고 워크로드 컴포넌트(API & Libraries)로 이루어져 있으며(그림 10), 스파크와 유사하다.



[그림 10] 플린크 컴포넌트 구성도
(이미지 출처: Apache Flink 1.1.3 Documentation)

스파크는 데이터를 표현할 때 RDD를 사용하는 반면 플린크는 DataSet을 사용한다. 스파크는 RDD 변환하고 변환이 끝나면 또 변환하는 일련의 과정(Data Centric)을 거치는 반면 플린크는 연산자(operator)들 사이로 데이터를 흘려보낸다는 차이점이 있다(Operator Centric)(그림 11).



[그림 11] 스파크와 플린크의 데이터 처리 과정
(이미지 출처: NFLabs http://www.nflabs.com/)

스파크와 플린크는 성격이 유사하기에 서로 비교가 되고 있다. 표 1은 스파크와 플린크의 특징들을 나열한 것이다. 가장 큰 차이점은 처리 방식으로 볼 수 있다. 스파크는 마이크로-배치 처리를 하는 반면 플린크는 이벤트가 발생했을때 처리를 하는 방식으로 이루어진다. 스트리밍 데이터 프로세서의 신뢰도에서 두 플랫폼 Exactly Once를 보이는 점이 인상적이다. Exactly Once는 데이터의 중복과 유실로부터 보장해준다는 의미이다. 두 플랫폼 모두 스트리밍 데이터 처리에서 높은 신뢰도를 보인다는 점에서 동일하지만 오픈소스 contributor와 연구 사례 등에서 차이를 보인다. 이 점은 플린크보다 스파크가 월등하다.

[표 1] 스파크와 플린크의 주요 특징 비교
(출처: Javier Lopez and Mihail Vieru, Big Data Engineer, Zalando Tech)

Feature	Apache Spark	Apache Flink
Processing mode	micro-batch	event at a time
temporal processing support	processing time	event time ingestion time processing time
batch processing	yes	yes
latency	seconds	sub-seconds
back pressure handling	through manual configuration	implicit, through system architecture
state storage	distributed dataset	in memory distributed key/value store
state access	full state scan for each micro-batch	value lookup by key
state check pointing	yes	yes
high availability mode	yes	yes
event processing guarantee	exactly once	exactly once
Apache kafka connector	yes	yes
Amazon S3 connector	yes	yes
language support	Java, Scala, Python	Java, Scala, Python
deployment	standalone, YARN, Mesos	standalone, YARN

2.3 선행 연구 및 사례

2.3.1 선행 연구

스트리밍 데이터를 처리하거나, 기본 방법보다 빠르게 처리하기 위한 기법, 또는 기계학습을 활용하여 스트리밍 데이터를 분석하기 위한 응용 연구들이 활발히 진행 중에 있다. 스트리밍 데이터는 주로 사물인터넷 환경에서 발생하는 경우가 많다. 이런 사물인터넷 환경에서 스트리밍 데이터를 수집하기 위한 연구도 활발히 이루어지고 있다. 강윤희(2014)는 스트리밍 데이터 처리를 위해 프레임워크를 개발하였는데 아파치 플룸(Apache Flume)을 사용하여 센서 디바이스로부터 스트리밍 데이터를 수집하였고 수집된 데이터를 HDFS 저장을 위한 수집시스템을 구현하였다. 하둡이 주목받았던 초창기에 박수현(2012)은 하둡 기반의 시스템에서 스트리밍 데이터처리를 위해 하둡에서 스트리밍 데이터를 분할하기 위한 기법을 연구하기도 했다.

스트리밍 데이터 처리 시스템의 처리방법을 개선함으로써 성능향상을 꽤한 사례도 있다. 복경수(2016)는 스트리밍 데이터 처리에 사용되는 슬라이딩 윈도우 기법에서 타임 윈도우가 겹쳐서 처리되는 영역의 데이터를 재사용하는 기법을 점진적 처리 방법을 적용하여 기존 스파크에서 처리하는 기법보다 약 55% 뛰어난 성능 향상을 보인바 있으며, 최도진(2015)은 이동객체를 위한 연속 질의 기법으로 효율적인 처리 및 복구를 위해 스파크 스트리밍의 RDD를 기반으로 하는 연속질의 처리 기법을 제안하였다.

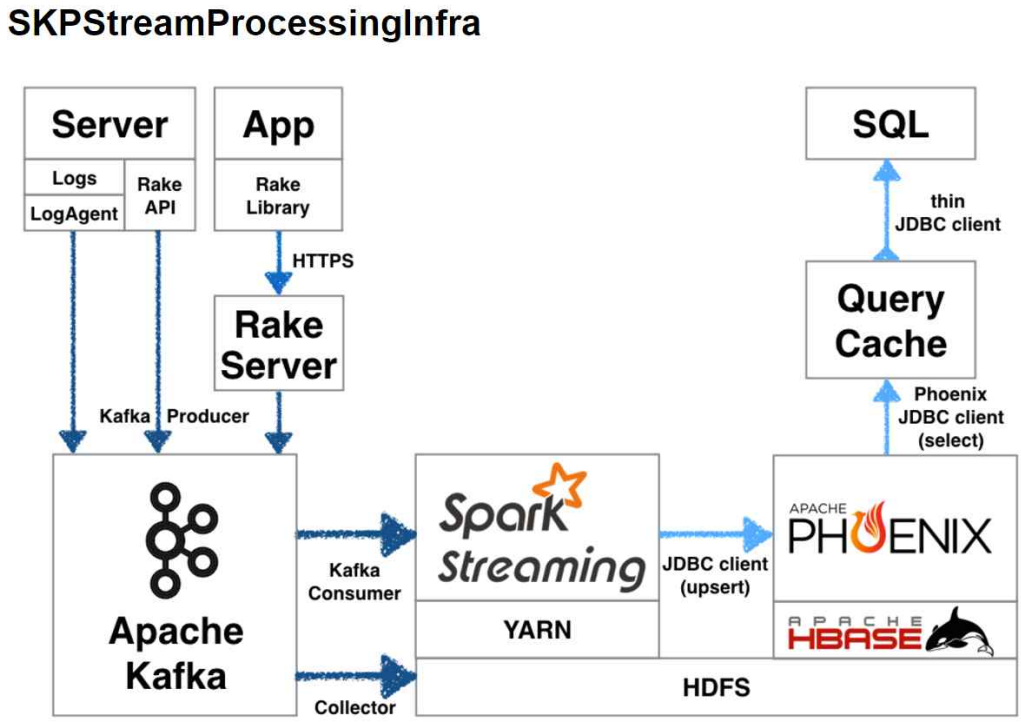
연속질의 처리 속도를 높이기 위해 분산 이동객체에 대한 그리드 색인기법을 스파크 스트리밍에 맞도록 변형한 것이다.

스트리밍 데이터를 활용하는 응용 연구 사례들도 있다. 기계학습을 활용하여 스트리밍 데이터의 실시간 분석 사례도 있다. 박정희(2016)는 스트리밍 데이터를 수집하고 이를 실시간 분석하기 위해 기계학습 알고리즘을 적용하였고, 권순현(2015)은 사물인터넷 환경에서 발생하는 대용량 스트리밍 센서 데이터의 실시간・병렬처리를 통해 시맨틱(Semantic) 데이터로 변환하는 기법을 연구하였다. 정보보안 분야에서도 활발히 연구되고 있다. 강지우(2016)은 스파크 스트리밍을 활용하여 ARP(Address Resolution Protocol) Spoofing 공격 감지와 대응 방법에 대한 연구를 하였다. 실시간 패킷 분석을 하여 공격 감지를 하고 대응책을 찾겠다는 전략이다.

2.3.2 관련 사례

우리나라 최대 규모의 포털 중 하나인 네이버(http://www.naver.com/)는 대용량 스트리밍 데이터 분석이라는 주제로 네이버 TV연예 서비스에 스트리밍 처리 기술을 적용한 사례가 있다. 웹에서 콘텐츠는 빠르게 생성되고 빠르게 소비된다. 이런 상황에서 흐름을 놓치지 않으려면 실시간 데이터를 분석해 서비스 운영에 적극 반영해야 한다. 네이버는 카프카와 스톰을 사용하여 대용량 스트리밍 데이터 처리를 하였다. 스트리밍 데이터 실시간 분석 플랫폼은 네이버 TV 연예 서비스에서 사용자가 어떤 콘텐츠에 관심이 있는지 감지하기 위한 용도로 활용되었다.

SK 플래닛의 Data Infrastructure 팀에서 스파크 스트리밍을 활용한 데이터 유실 없는 스트리밍 처리 인프라를 구축했던 사례가 있다.

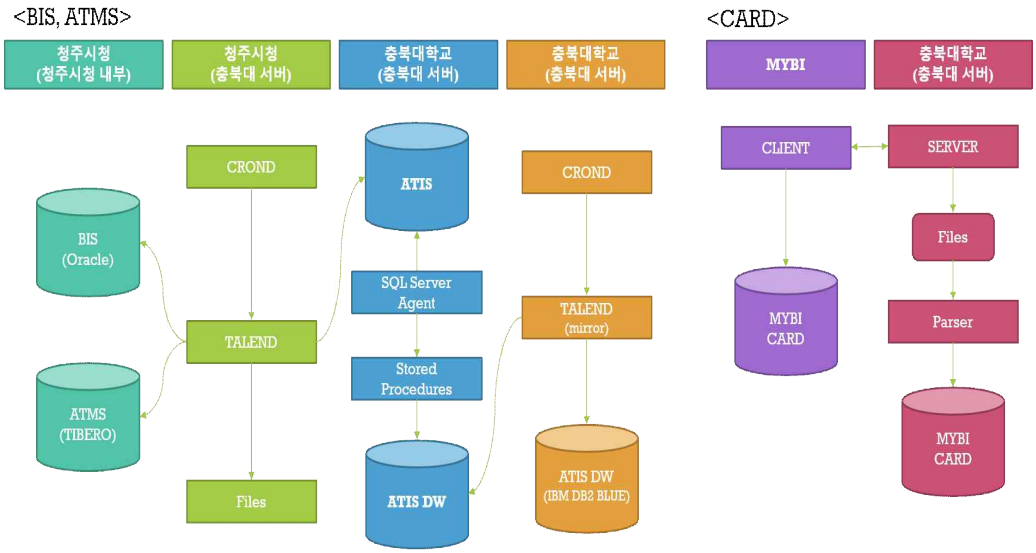


[그림 12] SK플래닛 스트림 처리 인프라 아키텍처
(이미지 출처: SK플래닛 README, 엄태욱 Data Infrastructure팀 프로그래머)

스트리밍 처리 인프라는 로그데이터를 처리하기 위한 목적으로 개발되었으며 대용량 데이터 수집·처리와 데이터 질의 기능을 구현하였다. 클라이언트 애플리케이션(안드로이드, iOS, 그리고 자바스크립트)에서 발생하는 대용량 로그처리를 위해 카프카와 스파크 스트리밍을 조합하여 데이터를 처리한다. 스트리밍 데이터 처리 플랫폼의 데이터 신뢰성을 고려하여 스파크 스트리밍을 적용한 사례이다.

2.4 기존 시스템의 한계점

청주시 교통 빅데이터 분석 시스템(조완섭, 2015)에서 교통 빅데이터 수집을 위한 기존 시스템은 RDBMS 기반의 시스템이다. 여기서 교통 데이터는 ETL(Extract Transform Load) 도구인 TALEND를 활용하여 하루 1회 BIS(Bus Information System), ATMS(Advanced Traffic Management System) 데이터를 ATIS DB에 수집한다. ATIS DB에 저장된 원시 데이터를 DBMS의 Stored Procedure를 통해 ATIS DW로 변환한다. ATIS DW의 데이터는 하루 1회 TALEND를 통해 IBM DB2 BLUE로 데이터를 이전한다. 이는 메인 메모리 기반 고속 데이터 서비스를 위함이다. 카드 데이터는 1회 TCP 전문통신을 통해 카드사로부터 전달받는다. 지금까지의 ETL 과정에 대한 설명은 그림 13과 같다.



[그림 13] 교통 빅데이터 수집을 위한 기존 RDBMS 기반 시스템 구조

기존의 교통 데이터 수집 시스템은 배치 처리를 기반으로 구성되어 있다. 따라서 이 시스템을 활용할 경우 실시간으로 끊임없이 발생하는 스트리밍 데이터를 수집하지 못하며, DW는 배치 처리를 기반으로 이루어지는 부분이라 실시간 데이터 처리도 불가능하다. 또한 DBMS는 확장성이 제한적으로 분산 처리 환경 구성이 어렵다.

따라서 본 연구는 스트리밍 데이터 수집·저장 시스템과 스트리밍 데이터 처리 시스템을 통해 실시간 스트리밍 데이터 수집·저장·처리를 지원하고 분산 환경에 자유로운 빅데이터 플랫폼을 적용함으로써 확장에 유연한 시스템(scale-out)을 제안한다.

Ⅲ. 시스템 설계 및 구현

본 장에서는 교통 스트리밍 데이터 수집·저장 시스템과 실시간 처리 시스템의 설계와 구현방법을 기술한다. 3.1절에서는 시스템 전체 아키텍처, 3.2절은 수집·저장 시스템 그리고 3.3절은 실시간 처리 시스템에 대해 논한다.

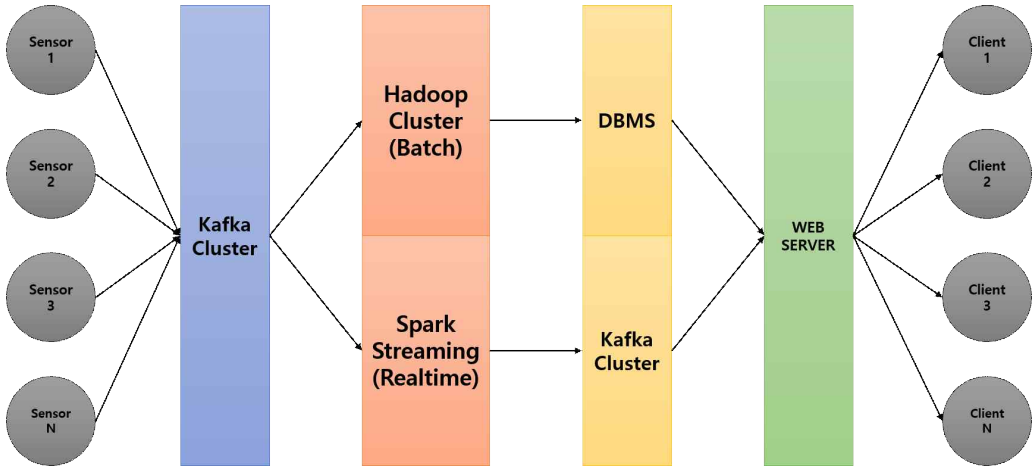
본 연구에서 스트리밍 데이터 처리에 스파크를 사용하였다. 이유는 스트리밍 데이터 처리 신뢰도 때문이다. 아래는 스트리밍 데이터 처리 신뢰도의 세 가지 종류에 대한 설명이다.

1. *At-most-once: 데이터의 유실이 있을 수 있다.*
2. *At-least-once: 데이터 유실은 없으나 재전송으로 인해 중복이 발생할 수 있다.*
3. *Exactly-once: 데이터가 오직 한번만 처리되며 유실도 중복도 없다.*

위 세 가지 중 세 번째 Exactly-once는 모든 상황에 대해 완벽히 보장하긴 어렵지만 스트리밍 데이터 처리에서 가장 이상적인 방식이다. 바로 스파크가 Exactly-once를 완벽히 보장하기 때문에 교통 스트리밍 데이터 처리에 가장 적합하다. 즉, 스파크는 스트리밍 데이터를 처리하는 부분에서 데이터의 중복과 유실을 방지할 수 있다. 이런 이유가 스트리밍 데이터 처리에 스파크를 선택하게 된 이유이다. 참고로 스톰은 At-least-once 방식이고 플린크는 Exactly-once 방식이다. 플린크의 사례가 스파크에 비해 상대적으로 적어 본 연구에서는 스트리밍 데이터 처리에서 Exactly-once의 보장과 동시에 다양한 연구 사례가 있는 스파크를 스트리밍 데이터 프로세서로 선택하였다.

3.1 시스템 아키텍처(System Architecture)

시스템의 구성은 크게 2가지로 구분된다 : 교통 스트리밍 데이터의 수집·저장 시스템, 교통 스트리밍 데이터의 처리 시스템. 그림 14는 시스템의 논리적 구성도를 보여준다. 스트리밍 데이터의 수집은 카프카를 통해 수집되며 카프카에서 수집된 데이터를 하둡 클러스터로 배치 저장되며 하둡의 맵리듀스 프레임워크를 활용해 HDFS에 저장된 스트리밍 데이터를 배치 처리한다. 배치 처리된 데이터는 DBMS에 저장하고 웹 서버의 웹 애플리케이션을 통해 클라이언트에게 제공된다.

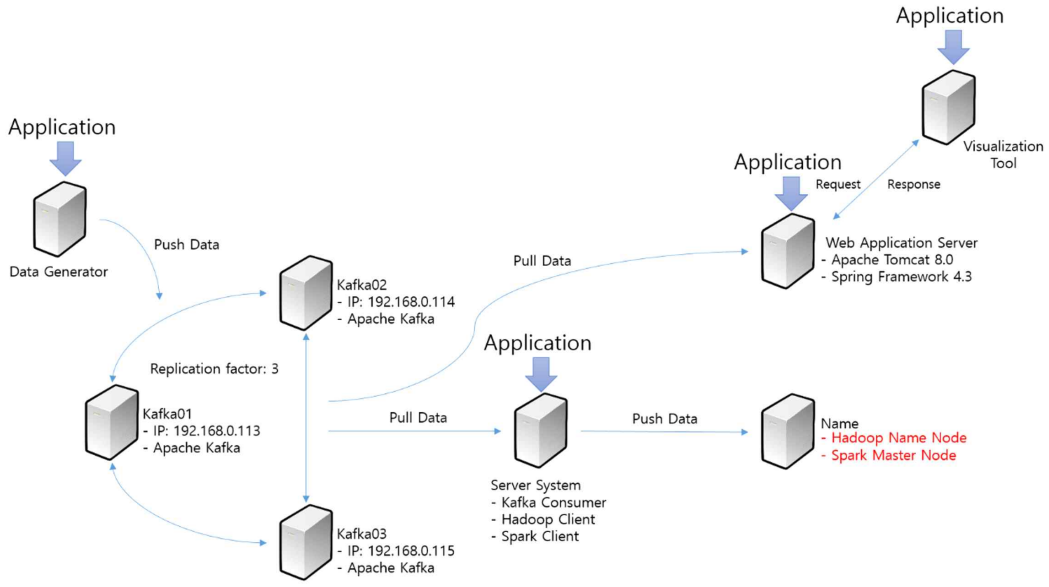


[그림 14] 시스템의 논리적 구성도

스트리밍 데이터의 실시간 처리는 스파크 스트리밍을 이용하여 해결한다. 스파크 스트리밍 API를 활용하면 카프카와 스파크에서 데이터를 마이크로-배치(micro-batch) 처리할 수 있는 방법을 제공한다. 스파크 스트리밍으로 마이크로-배치로 카프카로부터 데이터를 가져와 처리하고 결과를 다시 카프카에

저장한다. 카프카에 저장된 데이터는 웹 애플리케이션을 통해 클라이언트로 서비스 된다.

하둡 클러스터는 1대의 네임노드와 14대의 데이터 노드로 구성된다. 스파크 클러스터도 하둡과 공유하므로 1대의 마스터 노드 14대의 슬레이브 노드로 구성된다. 카프카 클러스터는 총 3대로 구성되며 복제계수 3으로 설정하였으며 주키퍼를 활용하여 HA(High Availability)를 구성한다. 그림 15는 전체 시스템 아키텍처를 나타낸 것이다. 본 연구에서 제안하는 시스템이 전체 시스템에 어떻게 적용되는가를 보여주기 위한 그림이다.



[그림 15] 교통 스트리밍 데이터 수집・저장・처리를 위한 전체 시스템 아키텍처

실제 다량의 센싱(Sensing) 디바이스를 사용하기 어려움으로 인해(청주시내 교통 시스템의 동작에 영향을 줄 수 있으므로) 가상의 데이터 발생기(generator)를 활용하여 카프카에 데이터를 push 한다. 데이터 발생기는 실제

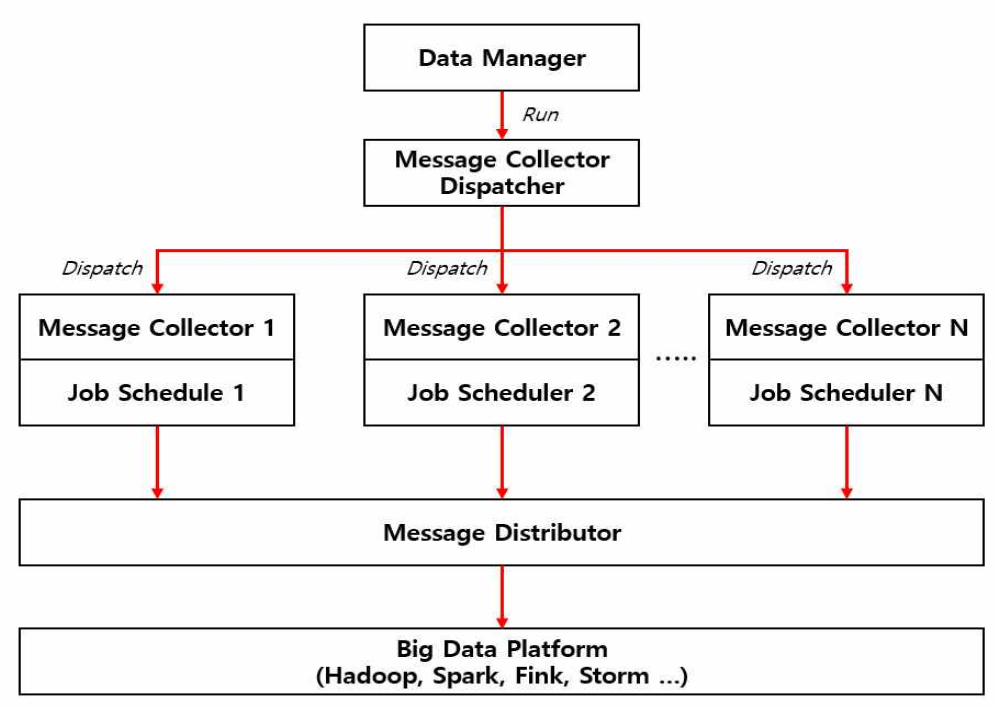
축적된 교통 빅데이터를 기반으로 데이터를 발생시킨다. Server System은 스트리밍 데이터를 수집·저장하기 위한 애플리케이션(데이터관리자: DataManager)과 스트리밍 데이터를 처리하기 위핸 애플리케이션(데이터분석기:DataAnalyzer)이 데몬 프로세스로 구동된다. Name은 하둡 클러스터의 네임노드, 스파크 클러스터의 마스터노드가 설치된 서버이다. 이 노드를 통해 데이터를 HDFS에 저장하거나 스파크 스트리밍을 통해 데이터를 처리한다. Web Application Server는 REST(Representational State Transfer) 서비스를 위한 웹 애플리케이션이 있으며, 시각화 애플리케이션으로부터 데이터 요청에 응답하기 위한 기능이 있다.

3.2 스트리밍 데이터 수집·저장 시스템

본 절에서는 스트리밍 데이터를 수집하고 스케줄러 기반 데이터 분배기를 활용한 스트리밍 데이터 저장 시스템에 대해 설명한다. 스트리밍 데이터 수집·저장 시스템은 데이터 관리자(Data Manager) 애플리케이션을 통해 동작한다. 데이터 관리자는 데이터 분배기를 통해 각 플랫폼 특성에 맞게 데이터 분배를 할 수 있다. 데이터 분배는 HDFS, DBMS, File System 등 다양한 플랫폼에 저장이 가능하며 각 플랫폼의 특징에 따라 개별 스케줄러를 통해 데이터 분배 제어가 가능하다. 데이터 관리자의 시스템 흐름도는 그림 16과 같다.

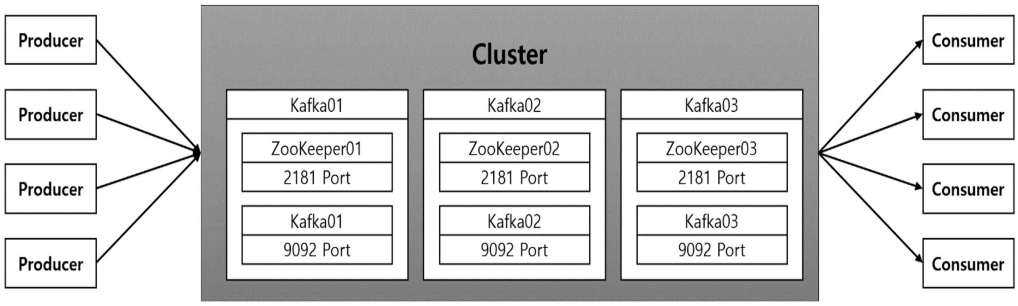
데이터 관리자는 Data Manager 데몬 애플리케이션으로 구동되며 Message Collector Dispatcher에게 Message Collector 스레드(thread) 관리 및 상태 모니터링 기능을 위임한다. Message Collector Dispatcher는 단일 객체

(Singleton)로 생성된다.



[그림 16] 데이터 수집·저장 애플리케이션(데이터 관리자) 시스템 흐름도

Message Collector는 카프카 토픽별로 1개씩 생성되는 스레드로 카프카로부터 해당 토픽의 메시지를 지속적으로 가져오는 역할을 수행한다. 데이터 생성기(Producer)로부터 발생한 데이터는 카프카에 저장되며 토픽별 Message Collector (Consumer)가 해당 메시지를 가져온다(그림 17).



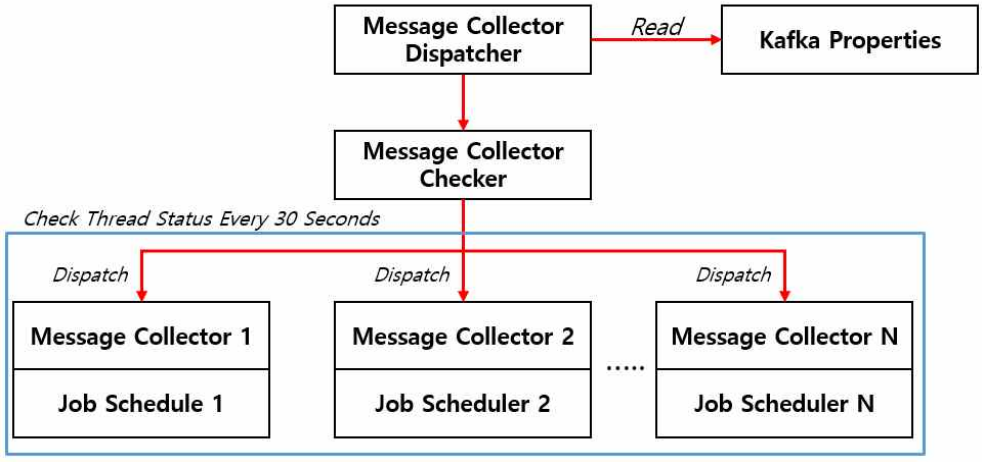
[그림 17] 카프카 클러스터 구조도

Message Collector가 수집한 메시지는 개별 스케줄러인 Job Scheduler에 의해 주기적으로 Message Distributor에게 전달된다. Message Distributor는 로컬 파일 시스템에 데이터를 모아두고 사용자가 설정한 주기마다 데이터를 빅데이터 저장·처리 플랫폼에 전달한다. 대표적으로 HDFS이다. HDFS에 주기적으로 배치 저장을 수행하는데 모아둔 스트리밍 데이터를 배치 처리하기 위함이다. 이상 데이터 관리자의 전체 흐름에 대한 설명이다.

데이터 관리자는 총 4개의 컴포넌트로 구성된다. 3.2.1절은 Message Collector Dispatcher, 3.2.2절은 Message Collector, 3.2.3절은 Job Scheduler, 그리고 3.2.4절은 Message Distributor에 대해 자세히 다룬다.

3.2.1 Message Collector Dispatcher

Message Collector Dispatcher는 Message Collector 스레드들을 관리하고 주기적으로 스레드 상태를 확인한다. 그림 18은 컴포넌트 구조도를 보여준다.



[그림 18] Message Collector Dispatcher 구조도

Message Collector Dispatcher 카프카 설정파일을 읽어 들여 카프카 토픽, 카프카 파티션, 그리고 카프카 클러스터 정보 등을 참조하여 초기화 작업을 수행한다. 이후 Checker를 통해 Message Collector 스레드의 상태를 주기적으로 점검한다. 기본 값으로 30초마다 스레드를 점검한다. 만약 스레드가 인터럽트와 같은 이유로 중단된다면, 기존 스레드 리소스를 반환하고 스레드를 재가동 시킨다.

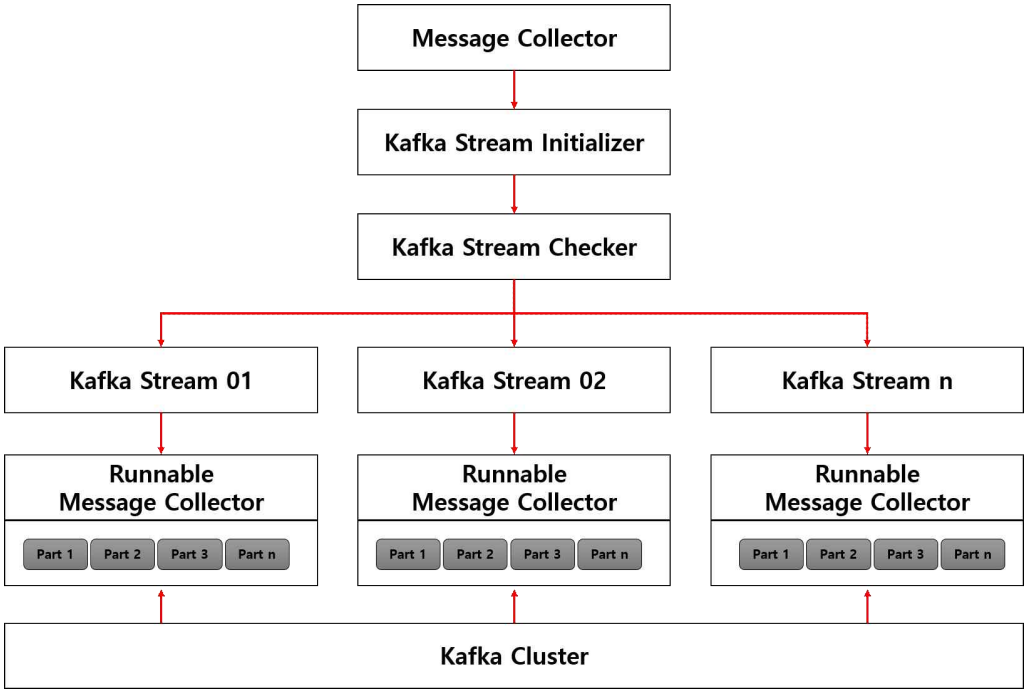
Message Collector Dispatcher는 단일(Singleton) 스레드로 동작한다. 애플리케이션 내에서 Message Collector Dispatcher는 단 1개만 생성된다. 이는 Message Collector를 구동시킬 수 있는 역할을 단 하나만 허용하기 위함이다. 스레드를 관리하는 기능을 갖는 모듈은 중복된 스레드 생성으로 예기치 못한 예외나 전체 시스템 불안정성을 야기하기 때문이다. 또한 Message Collector Dispatcher는 Daemon 속성을 가진 스레드이다. 즉, 데이터 관리자 애플리케이션이 비정상적으로 종료되거나 인터럽트로 인한 애플리케이션 정지가 아닌 이상 항상 살아 있는 스레드로 동작하는 것을

보장한다.

3.2.2 Message Collector

Message Collector는 카프카 토픽 별 메시지를 수집하는 역할을 한다. 각 토픽은 여러 개의 파티션으로 구성될 수 있는데(즉, 병렬처리가 가능함) 시계열 특성을 갖는 데이터는 단일 파티션으로 구성해야 한다. 따라서 본 연구에서 사용하는 교통 스트리밍 데이터는 시계열 특성을 보임으로 단일 파티션으로 구성한다.

그림 19는 Message Collector의 구조도를 보여준다. Message Collector는 Kafka Stream Initializer를 통해 토픽, 파티션 정보를 기반으로 생성해야하는 카프카 스트림과 파티션 스트림의 개수를 결정한다. 이 스트림 역시 스레드로 구동되며 Message Collector는 이 스레드들의 상태정보를 주기적으로 점검하여 문제가 발생한 스레드를 감지하여 재가동시킨다. 이런 역할을 하는 모듈이 Kafka Stream Checker이다. Kafka Stream Checker는 또한 초기에 Runnable Message Collector를 스트림마다 개별로 생성하여 카프카로부터 토픽 별로 메시지를 가져올 수 있게 한다.



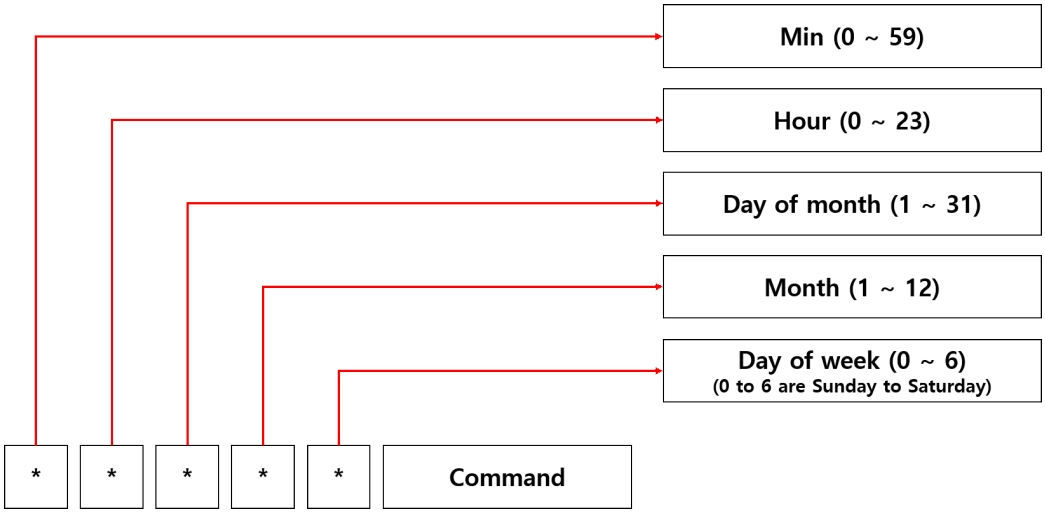
[그림 19] Message Collector의 구조도

Runnable Message Collector는 파티션 별로 스레드를 생성하고 각 스레드 별로 카프카로부터 메시지(데이터)를 가져온다(병렬 처리 수행). 각 파티션 스레드 역시 상태정보를 주기적으로 점검하는데 이 역할은 Runnable Message Collector 내부적으로 수행한다. 이 작업은 카프카에 데이터가 없을 때도 동작하는데 데이터가 없을 때 기본적으로 블록(Blocking) 상태가 된다.

Runnable Message Collector는 Instance Field로 Message Distributor 객체를 포함한다. Message Distributor는 Job Scheduler에 의해 빅데이터 저장 플랫폼으로 데이터를 전달하는 역할을 수행한다. Job Scheduler와 Message Distributor의 보다 자세한 내용은 다음 절에서 설명한다.

3.2.3 Job Scheduler

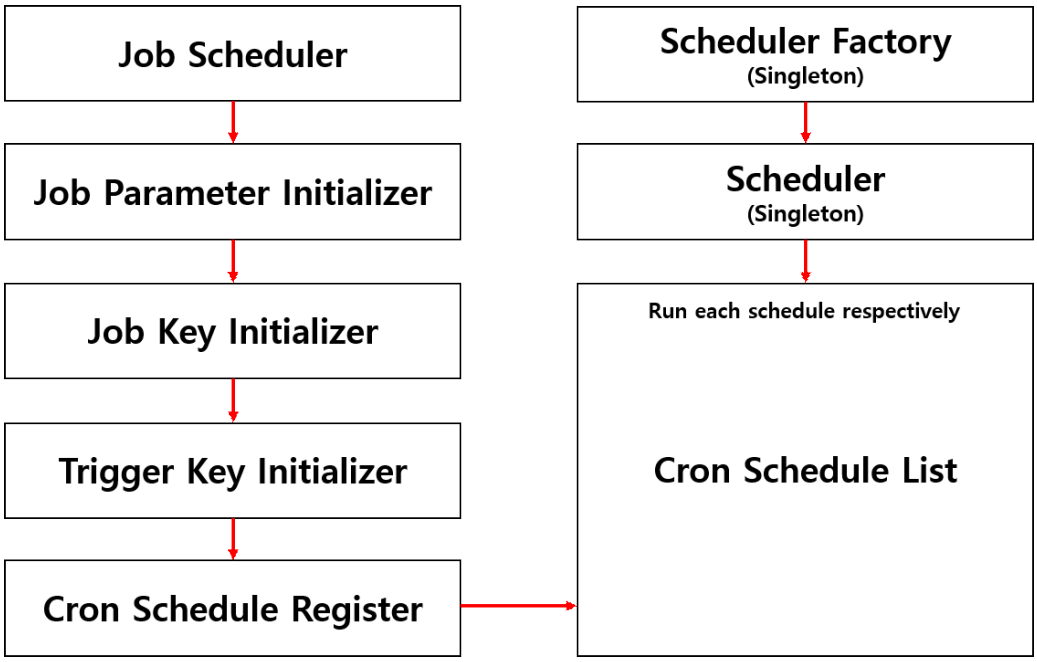
Job Scheduler는 유닉스/리눅스 운영체제의 소프트웨어 유틸리티인 Cron과 유사한 표현법으로 스케줄링 시간을 결정할 수 있다. Cron은 시간 기반 잡 스케줄러이다. 환경 설정을 통해 고정된 시간, 날짜, 주기적 간격으로 실행할 수 있도록 할 수 있다. 그림 20은 Cron 스케줄링 표현 방법을 보여준다.



[그림 20] Cron 스케줄링 표현법

Job Scheduler는 Message Collector 마다 개별적 설정이 가능하다. 그림 21은 Job Scheduler의 구조를 보여준다. Job Parameter는 Job Scheduler가 구동될 때 전달할 parameter를 설정한다. 예를 들어, 카프카로부터 가져온 메시지(데이터)와 토픽정보 등이 포함된다. Job Key는 Job을 식별하기 위한 식별자를 정하는 과정이다. 식별자뿐만 아니라 주기적으로 실행할 작업을 가진 작업 클래스를 설정하는 과정이다. Trigger

Key는 Cron Schedule을 실행할 때 구분하기 위한 식별자이다. 이 과정에서 Cron Schedule을 활용하여 실행 주기를 설정할 수 있다. 실행 주기가 설정되면 Cron Schedule List에 스케줄 등록을 한다.

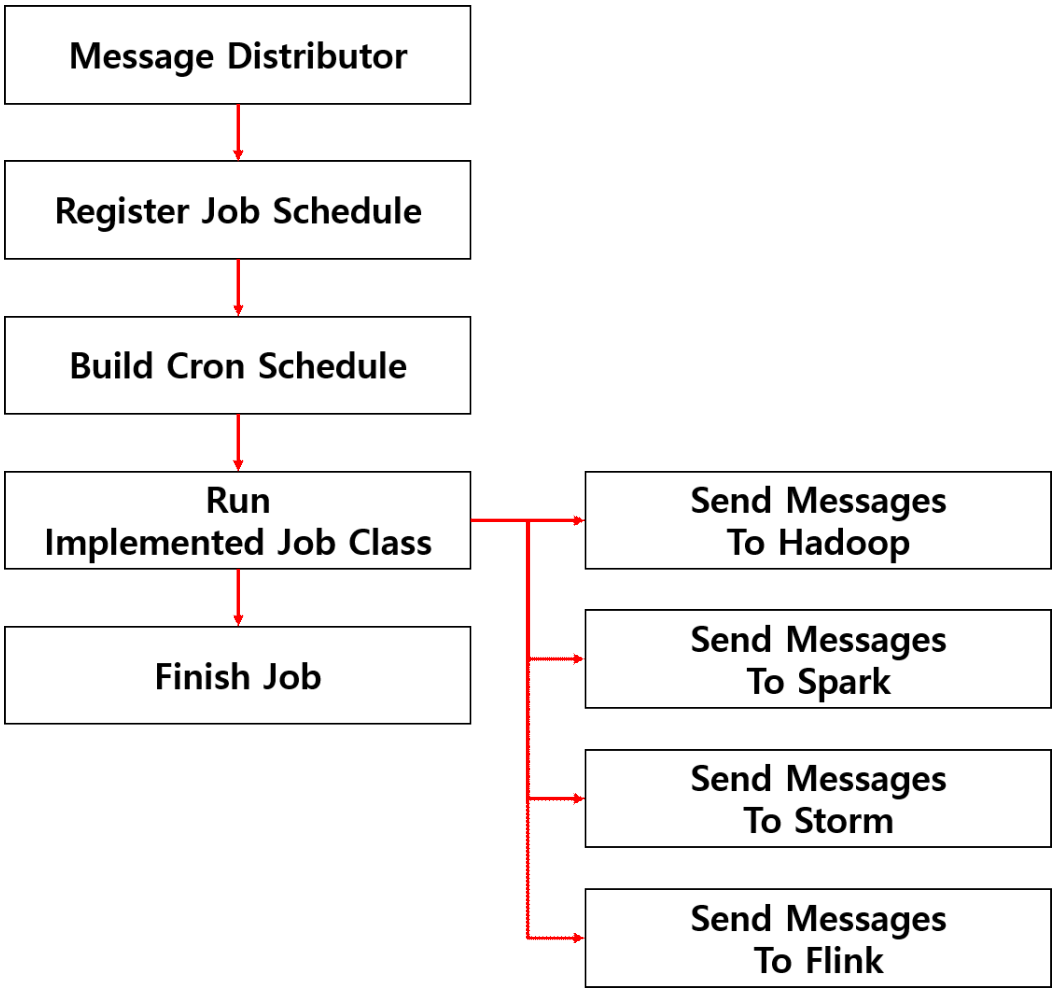


[그림 21] Job Scheduler의 구조도

Scheduler Factory와 Scheduler는 Singleton 객체로써 Job Scheduler로 등록된 Cron Schedule을 주기적으로 실행하는 역할을 한다. Scheduler는 Cron Schedule List에 등록된 작업(Job)을 개별적으로 실행한다. 본 연구에서 Job Scheduler는 Message Distributor를 위해서 사용된다.

3.2.4 Message Distributor

Message Distributor는 스트리밍 데이터를 빅데이터 저장 플랫폼에 전달하는 역할을 수행한다. Job Scheduler에 의해 호출되어 동작한다. 그림 22는 Message Distributor의 구조도를 보여준다.

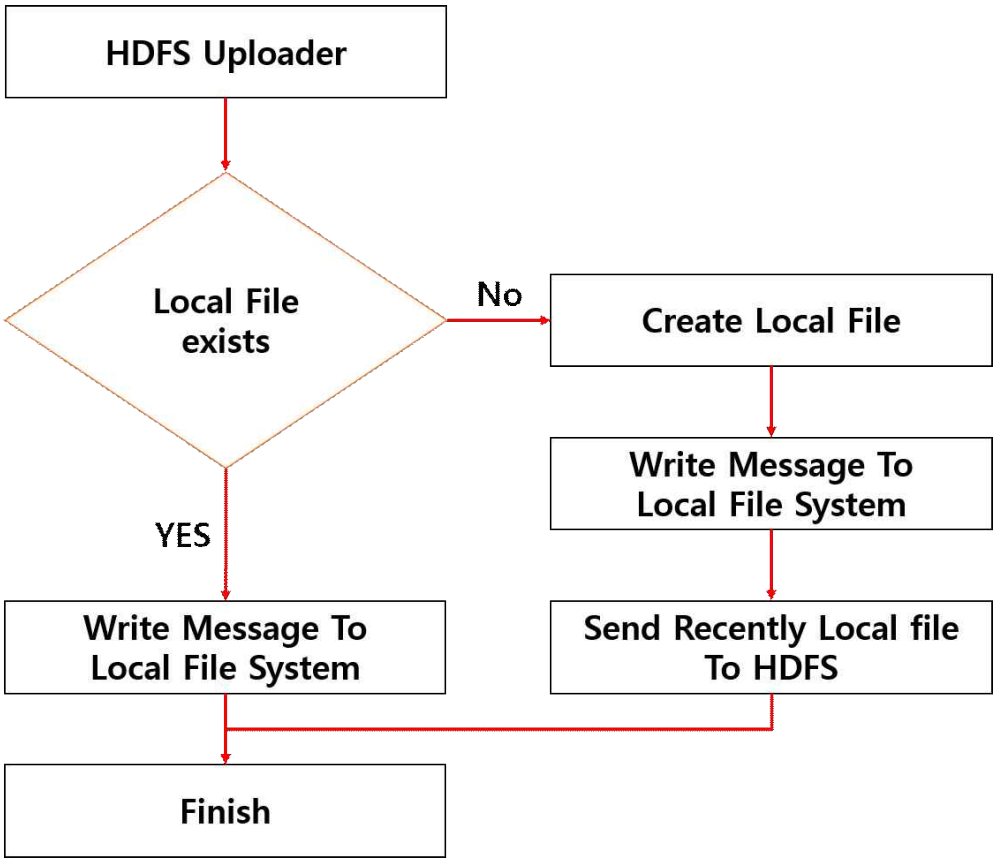


[그림 22] Message Distributor의 구조도

Message Distributor 스레드가 실행되면, 가장 먼저 스케줄을 등록한다. 스케줄은 단 1번만 등록이 되며 이후에 Job Class가 정해진 간격으로 실행된

다. Job Class에는 하둡, 스파크, 스톰, 그리고 플린크와 같은 데이터 저장 및 처리 플랫폼으로 스트리밍 데이터를 전송한다.

본 연구의 데이터 수집·저장 시스템에서는 하둡으로 스트리밍 데이터를 저장하는 역할을 한다. 데이터의 저장은 배치작업으로 수행되며 서버 로컬디스크에 일정시간 동안 스트리밍 데이터를 저장한다. 이후 스케줄러에 의해 Message Distributor 스레드가 실행되면 로컬 디스크에 저장된 스트리밍 데이터 집합을 HDFS에 전달한다. 이 역할을 하는 스레드를 HDFS Uploader라 한다. HDFS Uploader는 Message Distributor의 Send Message To Hadoop Method에서 실행된다(그림 23).



[그림 23] HDFS Uploader 구조도

HDFS Uploader는 기본 값으로 1시간 간격으로 HDFS에 파일을 전달한다. 로컬 파일의 존재 여부는 시간단위로 기록하는 파일이름으로 판별할 수 있다. 현재 날짜와 시간이 일치하는 파일명이 있다면 파일이 존재한다고 볼 수 있고 로컬 파일 시스템에 스트리밍 데이터를 계속 기록한다. 만약 파일이 존재하지 않는다면 마지막 시간에 기록된 로컬 파일을 HDFS에 전송한다. 이 작업은 스케줄러가 동작할 때 마다 작업을 수행한다.

3.2.5 스트리밍 데이터 수집·저장 시스템 구현 결과

본 절에서는 스트리밍 데이터 수집·저장 시스템인 데이터 관리자의 실행 과정에 대해 서술한다. 시스템은 빅데이터 플랫폼 위에서 동작하게 된다. 시스템을 실행하기 전에 반드시 하둡, 스파크, 카프카 클러스터를 동작시켜야 한다. 따라서 실행 확인 전에 각 클러스터의 동작을 먼저 확인한다. 이후에 데이터 관리자를 실행하고 HDFS에 데이터가 전달되는 결과를 확인한다.

(1) 하둡 클러스터 확인

Live Datanodes : 14

Node	Transferring Address	Last Contact	Admin State	Configured Capacity (TB)	Used (TB)	Non DFS Used (TB)	Remaining (TB)	Used (%)	Used (%)	Remaining (%)	Blocks	Block Pool Used (TB)	Block Pool Used (%)	Failed Volumes	Version
dn1	192.168.0.116:50010	2	In Service	10.91	0.48	0.00	10.42	4.44		95.56	21413	0.48	4.44	0	2.6.4
dn10	192.168.0.108:50010	1	In Service	4.50	0.46	0.23	3.81	10.12		84.79	19493	0.46	10.12	0	2.6.4
dn11	192.168.0.109:50010	1	In Service	4.50	0.46	0.23	3.81	10.25		84.66	19449	0.46	10.25	0	2.6.4
dn12	192.168.0.110:50010	2	In Service	4.04	0.46	0.21	3.37	11.50		83.41	19170	0.46	11.50	0	2.6.4
dn13	192.168.0.111:50010	0	In Service	4.50	0.47	0.23	3.80	10.36		84.56	19366	0.47	10.36	0	2.6.4
dn14	192.168.0.112:50010	1	In Service	4.50	0.47	0.23	3.80	10.34		84.57	19425	0.47	10.34	0	2.6.4
dn2	192.168.0.117:50010	2	In Service	10.91	0.49	0.00	10.42	4.51		95.49	21633	0.49	4.51	0	2.6.4
dn3	192.168.0.118:50010	2	In Service	10.91	0.49	0.00	10.42	4.52		95.48	21669	0.49	4.52	0	2.6.4
dn4	192.168.0.119:50010	0	In Service	10.91	0.48	0.00	10.43	4.36		95.64	21407	0.48	4.36	0	2.6.4
dn5	192.168.0.120:50010	1	In Service	10.91	0.46	0.00	10.45	4.19		95.81	21426	0.46	4.19	0	2.6.4
dn6	192.168.0.104:50010	0	In Service	4.50	0.46	0.24	3.81	10.12		84.65	18671	0.46	10.12	0	2.6.4
dn7	192.168.0.105:50010	0	In Service	4.50	0.46	0.23	3.81	10.14		84.78	19431	0.46	10.14	0	2.6.4
dn8	192.168.0.106:50010	2	In Service	4.50	0.46	0.23	3.81	10.28		84.63	19411	0.46	10.28	0	2.6.4
dn9	192.168.0.107:50010	2	In Service	4.50	0.47	0.23	3.80	10.35		84.57	19586	0.47	10.35	0	2.6.4

Hadoop, 2016.

[그림 24] 하둡 클러스터 실행 정보 (설치된 노드 조회결과)

(2) 스파크 클러스터 확인

Workers

Worker Id	Address	State	Cores	Memory
worker-20161110112829-192.168.0.106-50011	192.168.0.106:50011	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110112841-192.168.0.107-52746	192.168.0.107:52746	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110112846-192.168.0.105-43854	192.168.0.105:43854	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110112857-192.168.0.110-58630	192.168.0.110:58630	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110112917-192.168.0.112-60699	192.168.0.112:60699	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110112918-192.168.0.109-50230	192.168.0.109:50230	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110112918-192.168.0.111-39383	192.168.0.111:39383	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110112919-192.168.0.108-55469	192.168.0.108:55469	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110112922-192.168.0.104-46296	192.168.0.104:46296	ALIVE	8 (8 Used)	28.0 GB (4.0 GB Used)
worker-20161110193810-192.168.0.117-43746	192.168.0.117:43746	ALIVE	16 (16 Used)	54.0 GB (4.0 GB Used)
worker-20161110194851-192.168.0.120-42809	192.168.0.120:42809	ALIVE	16 (16 Used)	54.0 GB (4.0 GB Used)
worker-20161110195021-192.168.0.116-38935	192.168.0.116:38935	ALIVE	16 (16 Used)	54.0 GB (4.0 GB Used)
worker-20161110195206-192.168.0.118-35095	192.168.0.118:35095	ALIVE	16 (16 Used)	54.0 GB (4.0 GB Used)
worker-20161110195404-192.168.0.119-43447	192.168.0.119:43447	ALIVE	16 (16 Used)	54.0 GB (4.0 GB Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20161115010954-0077 (kill)	SparkSQL::192.168.0.121	152	4.0 GB	2016/11/15 01:09:54	hadoop	RUNNING	38 min

[그림 25] 스파크 클러스터 노드 및 스파크 애플리케이션 실행 정보

(3) 카프카 클러스터 확인

Details for the consumer group dbnis-atis-datamgr-collector

Groups List / dbnis-atis-datamgr-collector

Brokers

id	host	port
2	kafka02.bigdatacenter.org	9092
1	kafka01.bigdatacenter.org	9092
3	kafka03.bigdatacenter.org	9092

Consumer Offsets

Topic	Partition	Offset	logSize	Lag	Owner	Created	Last Seen
atis-avi		16574	16956	382			
	0	16574	16956	382		20 days ago	17 days ago
atis-bis		127239	127608	369			
	0	127239	127608	369		20 days ago	17 days ago
atis-dsrc		16166	16496	330			
	0	16166	16496	330		20 days ago	17 days ago
atis-iitp		0	0	0			
	0	0	0	0		20 days ago	20 days ago
atis-itrc		0	0	0			
	0	0	0	0		20 days ago	20 days ago

[그림 26] 카프카 클러스터의 브로커 및 토픽 정보

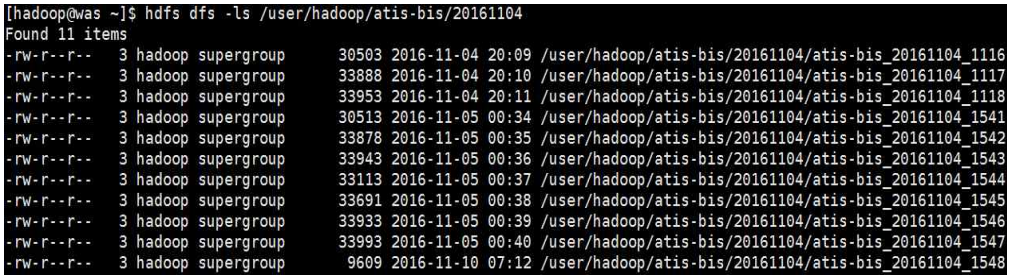
(4) 스트리밍 데이터 수집·저장 시스템 실행

카프카에서 수집된 교통 스트리밍 데이터를 빅데이터 플랫폼에 분배하기 위한 시스템으로 그림 27과 같이 구동한다. 시스템은 리눅스에서 구동된다.

```
[root@DBNIS-HW anthonykim]# java -jar DataManager-1.0.0.jar
INFO  org.springframework.context.support.AbstractApplicationContext.prepareRefresh(AbstractApplicationContext.java:581) - Refreshing org.springframework.context.suppor
INFO  org.springframework.beans.factory.xml.XmlBeanDefinitionReader.loadBeanDefinitions(XmlBeanDefinitionReader.java:317) - Loading XML bean definitions from class path
INFO  com.anthonykim.datamgr.DataManager.main(DataManager.java:27) - DataManager stated...
INFO  com.anthonykim.datamgr.DataManager.main(DataManager.java:33) - MessageCollectorDispatcher started...
INFO  com.anthonykim.datamgr.dispatcher.MessageCollectorDispatcherImpl.run(MessageCollectorDispatcherImpl.java:37) - MessageCollectorDispatcher started...
INFO  com.anthonykim.datamgr.DataManager.main(DataManager.java:36) - SparkStreamingDispatcher started...
INFO  com.anthonykim.datamgr.dispatcher.SparkStreamingDispatcherImpl.run(SparkStreamingDispatcherImpl.java:34) - SparkStreamingDispatcher started...
INFO  com.anthonykim.datamgr.collector.MessageCollectorImpl.run(MessageCollectorImpl.java:34) - Message Distributor atis-avi started with amount of partitions 1...
INFO  com.anthonykim.datamgr.collector.MessageCollectorImpl.run(MessageCollectorImpl.java:34) - Message Distributor atis-bis started with amount of partitions 1...
INFO  com.anthonykim.datamgr.collector.MessageCollectorImpl.run(MessageCollectorImpl.java:34) - Message Distributor atis-dsrc started with amount of partitions 1...
INFO  com.anthonykim.datamgr.collector.MessageCollectorImpl.run(MessageCollectorImpl.java:34) - Message Distributor atis-mybiCard started with amount of partitions 1...
INFO  com.anthonykim.datamgr.collector.MessageCollectorImpl.run(MessageCollectorImpl.java:34) - Message Distributor atis-iitp started with amount of partitions 1...
INFO  com.anthonykim.datamgr.collector.MessageCollectorImpl.run(MessageCollectorImpl.java:34) - Message Distributor atis-itrc started with amount of partitions 1...
```

[그림 27] 스트리밍 데이터 수집·저장 시스템 구동

시스템이 구동되면 스트리밍 데이터를 로컬 파일 시스템에 저장하고 1시간 간격으로 HDFS에 파일을 전달한다. 그림 28은 HDFS에 전달된 파일 리스트 목록을 보여준다. 테스트를 위한 결과이므로 기존 1시간에서 1분에 한 번씩 HDFS에 파일을 전달한다.



[그림 28] Data Manager에서 수집된 스트리밍 데이터의 HDFS로 파일 저장

HDFS에 저장된 파일 내용은 JSON(Java Script Object Notation) 형식의 스트리밍 데이터로 IoT 센서 데이터 생성기로부터 생성된 데이터이다. 실제 디바이스를 이용한 통신에서 JSON 형식이 많이 사용된다(그림 29).

[그림 29] HDFS에 파일로 저장된 JSON 형식의 교통 스트리밍데이터

3.2절에서는 스트리밍 데이터가 수집되어 HDFS에 저장되는 결과를 확인했다. HDFS 뿐만 아니라 스트리밍 데이터를 처리하는 플랫폼으로 직접 데이터를 보낼 수도 있다. 하지만 본 연구에서 사용하는 스트리밍 데이터 프로세서로 스파크를 사용한다고 했다. 스파크는 스파크 스트리밍이라는 스트리밍 데이터 처리에 특화된 API(Application Programming Interface)를 제공한다. 이 API를 활용하면 직접 데이터를 분배하지 않고도 스트리밍 데이터 셋을 마이크로-배치 처리할 수 있다. 스트리밍 데이터 수집・저장 시스템에서 쓰이는 스파크는 단지 맵리듀스 연산을 메인 메모리를 활용하여 고속 처리하기 위한 용도이다. 즉, 배치작업을 빠르게 처리하기 위함이다.

3.3 스트리밍 데이터 처리 시스템

스트리밍 데이터 처리 시스템은 스파크 스트리밍 API를 기반으로 구현하였다. 스파크 스트리밍은 스트리밍 데이터 수집·저장 시스템과는 다르게 카프카 클러스터로부터 스트리밍 데이터를 바로 가져올 수 있도록 설계되었다. 따라서 해결하고자 하는 문제에 대한 비즈니스 로직에 집중할 수 있다.

본 절은 스트리밍 데이터 처리에 필요한 데이터 모델을 설계하고 이 데이터 모델을 기반으로 처리 시스템을 설계며 마지막으로 처리 시스템 구현 결과에 대해 서술한다.

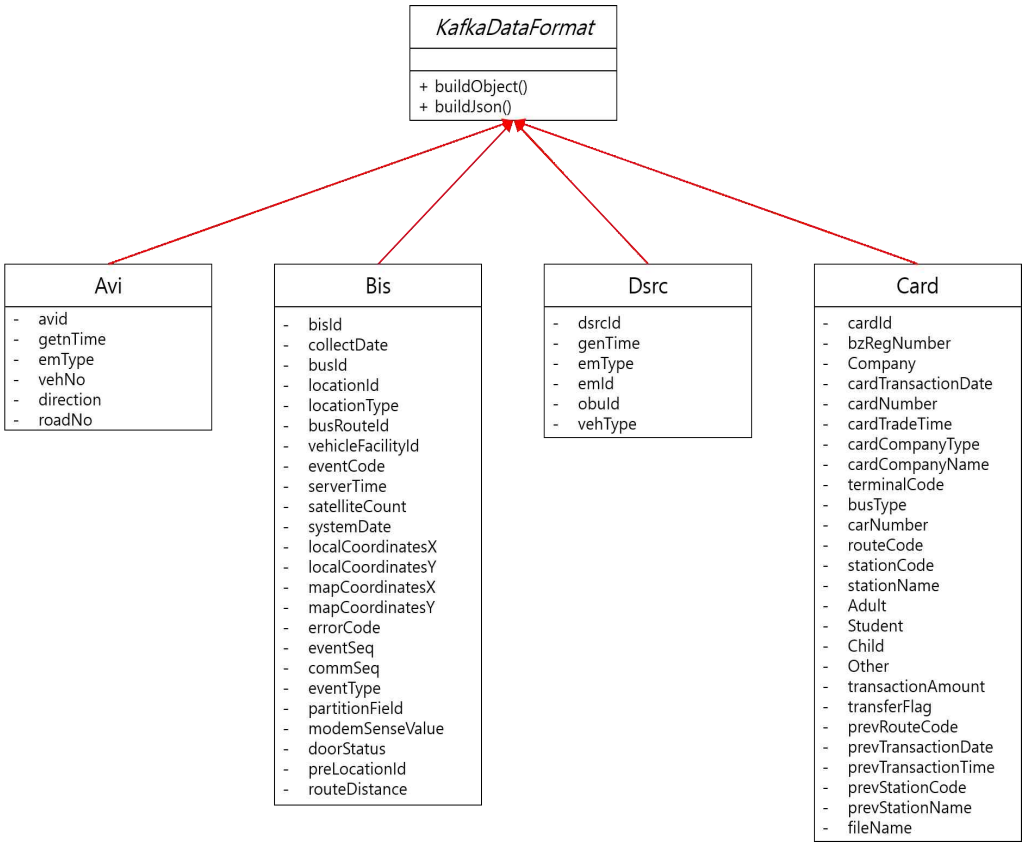
3.3.1 데이터 모델 설계

본 연구에서 사용되는 교통 스트리밍 데이터는 총 4가지 종류 AVI(Automatic Vehicle Identification), BIS(Bus Information System), DSRC(Dedicated Short Range Communications), 그리고 교통 CARD 데이터이다. 각 데이터에 대한 설명은 표2와 같다.

[표 2] 데이터 모델에 사용될 데이터 종류 정의

종류	데이터 설명
AVI	운행차량의 차량번호를 인식하는 방법으로 해당 도로의 속도 정보를 포함한다.
BIS	사용자에게 버스운행상황 정보를 제공하는 시스템의 데이터로 실시간 버스위치, 노선, 차량 배차 정보 등이 제공된다.
DSRC	차량의 이동정보를 수집하는 장치의 데이터로 차량 정보를 수집한다.
CARD	교통카드 데이터로 이용자의 승·하차 기록, 노선의 운영 정보, 환승·요금 기록 등을 포함하고 교통수단 또는 시설 이용자들의 세부적 이용 실태를 파악할 수 있는 데이터이다.

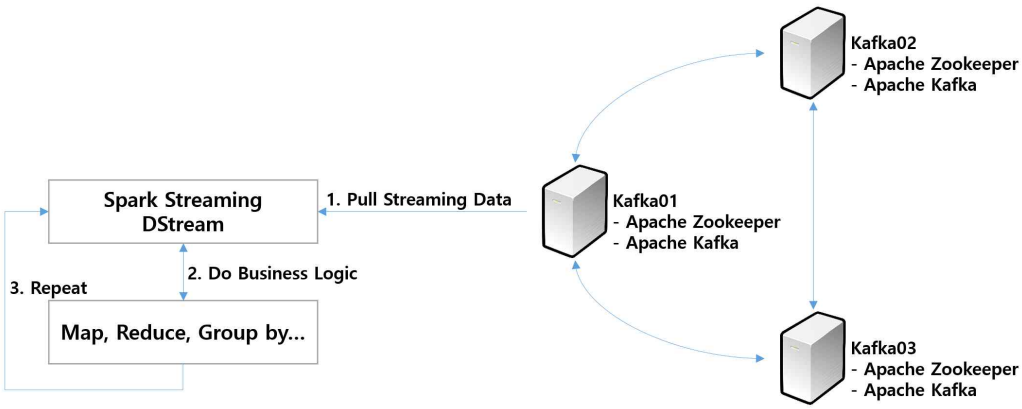
그림 30은 데이터 모델의 클래스 다이어그램이다. 클래스 다이어그램에서 getter와 setter 메서드는 생략한다. 추상 클래스인 KafkaDataFormat을 데이터 모델인 Avi, Bis, Dsrc, Card 클래스에서 상속하여 객체화를 위한 buildObject()와 데이터 전송을 위한 buildJson()을 재정의(Overriding) 하여 구현한다. 스트리밍 데이터는 JSON형식이므로 스파크 스트리밍에서 처리하기 위해 Object 형식으로 변환해야한다. 이 작업을 buildObject() 메서드를 사용해서 달성할 수 있다. 스파크 스트리밍에서 작업이 완료되면 다시 JSON 형식으로 데이터를 전달해야 한다. 이 작업은 buildJson() 메서드를 활용해 달성한다. 이 4가지 데이터 모델을 스파크 스트리밍에서 활용하여 스트리밍 데이터를 처리한다.



[그림 30] 데이터 모델의 클래스 다이어그램

3.3.2 스트리밍 데이터 처리 시스템 설계

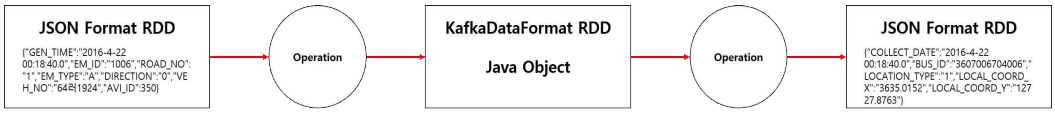
스트리밍 데이터 처리 시스템은 스파크 스트리밍 API를 기반으로 구현 되었다. 배치 처리 시스템과는 달리 카프카 클러스터로부터 데이터를 바로 가져올 수 있도록 스파크 스트리밍에서 API를 제공한다는 특징이 있다. 그림 31은 스트리밍 데이터 처리 시스템의 작업 과정을 보여준다.



[그림 31] 스트리밍 데이터 처리 시스템의 작업과정

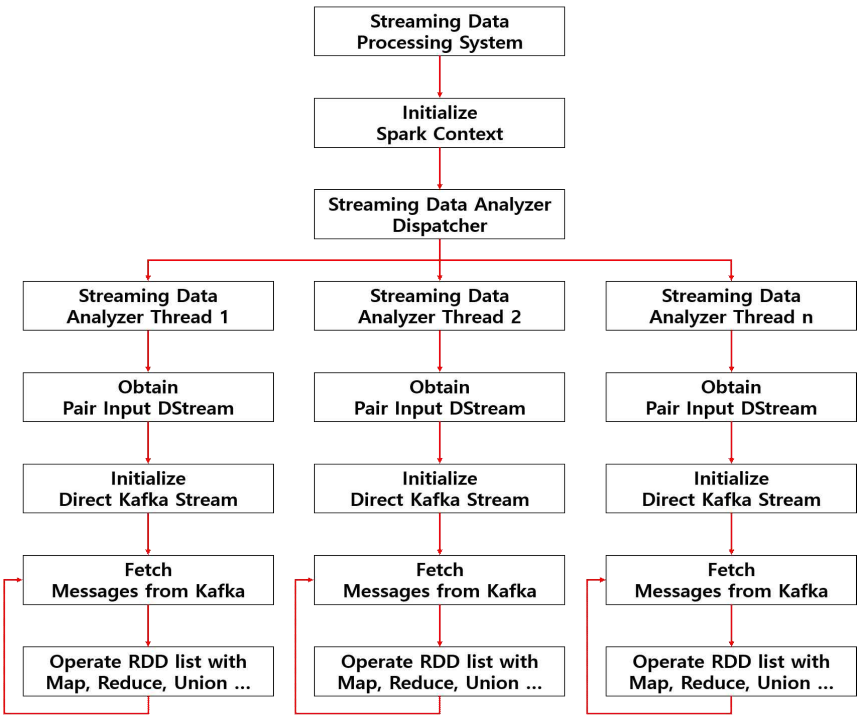
스파크 스트리밍의 DStream 객체를 통해 카프카 클러스터로부터 메시지를 마이크로-배치로 가져온다. 본 연구에서는 1초에 한번 씩 데이터를 처리한다. 스파크 스트리밍으로부터 가져온 데이터 셋은 JSON 형식으로 처음에 JSON 형식의 문자열 데이터를 RDD로 변환한다. 문자열 데이터는 데이터 모델의 buildObject() 메서드를 활용하여 객체 타입의 RDD로 재 변환한다. 스파크 스트리밍은 마이크로-배치로 데이터를 처리하므로 여러 객체를 포함하는 리스트 형태의 데이터 배열이다. 객체 타입의 RDD 리스트는 스파크 스트리밍

의 Map, Reduce, Union, Join, 그리고 Group by와 같은 연산을 수행하고 결과를 반환한다. 결과는 다시 JSON 형식으로 변환하기 위해 각 데이터 모델의 buildJson() 메서드 호출을 통해 변환되어 카프카 클러스터에 저장된다. 카프카 클러스터의 저장된 결과는 웹 애플리케이션을 통해 클라이언트에게 서비스될 수 있다. 스파크 스트리밍에서 RDD로 변환되는 과정은 그림 32와 같다. 스파크의 RDD는 필요한 연산에 따라 다양한 형식의 RDD로 변환할 수 있다.



[그림 32] 스파크 스트리밍에서 연산에 따른 RDD 변환 과정

그림 33은 스트리밍 데이터 처리 시스템의 동작 과정을 보여준다.



[그림 33] 스트리밍 데이터 처리 시스템의 동작과정

스트리밍 데이터 처리 시스템은 가장먼저 Spark Context 객체를 초기화한다. Spark Context 객체는 애플리케이션에 단 1개만 존재하는 단일(singleton) 객체이며 이후 Streaming Data Analyzer Dispatcher를 호출하여 카프카 토픽별로 스트리밍 데이터 분석 스레드를 생성한다. Dispatcher는 Analyzer Thread의 상태정보를 관리한다. 스레드에 인터럽트나 기타 예외로 인한 중단이 감지될 때 점유한 자원을 해제하고 즉시 스레드를 재 시작한다. 스레드 상태 점검은 매 60초마다 진행된다. 각 분석 스레드는 Spark Context 객체를 통해 DStream 객체를 얻어오게 되는데, 이 DStream 객체를 통해 카프카로부터 메시지(스트리밍 데이터) 셋을 얻어올 수 있다. 이후 각 DStream 객체는 Direct Kafka Stream을 통해 메시지 셋을 얻어오고 메시지 셋을 RDD로 변환하고 연산을 수행한다.

3.3.3 스트리밍 데이터 처리 시스템 구현 결과

그림 34는 BIS 데이터를 처리한 결과를 나타낸다. 노선 순서, 차량 번호, 정류장 이름과 같은 정보를 얻기 위해 해당 Bus ID를 기반으로 대중교통 정보(마스터 데이터)와 비교 연산을 통해 정보를 취합한다. 해당 내용은 카프카에 다시 저장되고 웹 애플리케이션을 통해 REST API로 클라이언트에 서비스된다.

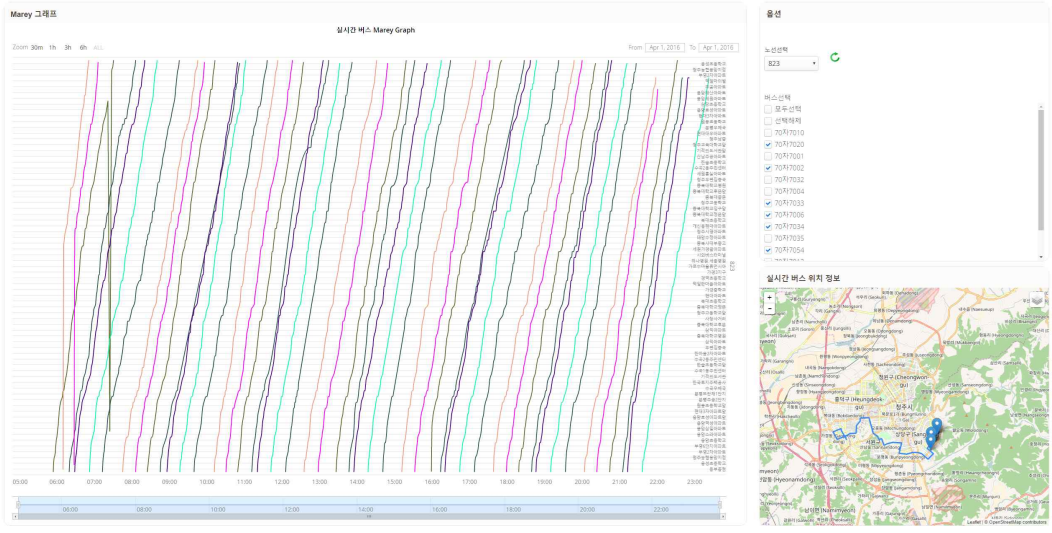
```
--- New RDD with 1 partitions and 1 records
-------------------------------------------
Time: 1481662253500 ms
-------------------------------------------
(null,{"CollectDate":"2014-11-20 05:33:17.0","MapCoordinatesY":"350488","SystemDate":
"2014-11-20 05:33:24.0","BusID":"3607006509204","MapCoordinatesX":"243615","ModemSens
eValue":"50","EventType":null,"LocationID":null,"BusRouteID":"3607004000201","EventSe
q":"0","DoorStatus":null,"RouteDistance":null,"CommSeq":"134","EventCode":"08","Parti
tionField":"3","PreLocationID":null,"VehicleFacilityID":"3607007025901","SatelliteCou
nt":"11","ServiceTime":"0","id":278,"ErrorCode":"00","LocationType":null,"LocalCoordi
natesX":"3639.2764","LocalCoordinatesY":"12729.315"})

--- New RDD with 1 partitions and 0 records
-------------------------------------------
Time: 1481662254000 ms
-------------------------------------------

--- New RDD with 1 partitions and 1 records
-------------------------------------------
Time: 1481662254500 ms
-------------------------------------------
(null,{"CollectDate":"2014-11-20 05:33:24.0","MapCoordinatesY":"338744","SystemDate":
"2014-11-20 05:33:27.0","BusID":"3607006704006","MapCoordinatesX":"238484","ModemSens
eValue":"60","EventType":null,"LocationID":"2740123000","BusRouteID":"3607004006801",
"EventSeq":"0","DoorStatus":null,"RouteDistance":null,"CommSeq":"110","EventCode":"01
","PartitionField":"3","PreLocationID":"2740139400","VehicleFacilityID":"360700703680
1","SatelliteCount":"8","ServiceTime":"0","id":279,"ErrorCode":"00","LocationType":"S
","LocalCoordinatesX":"3632.94","LocalCoordinatesY":"12725.8367"})
```


[그림 34] BIS 스트리밍 데이터 처리 결과

그림 35는 해당 데이터를 전달받는 웹 애플리케이션으로 실시간으로 버스의 정류장 별 버스 이동 시간을 그래프로 표현하고 지도에 버스 위치 정보를 동시에 표현하는 시공간 분석의 활용 예이다.



[그림 35] 스트리밍 처리 시스템을 활용한 시각화의 예
(이미지 출처: 충북대학교 경영정보학과 임원규)

그래프 X축은 시간 Y축은 정류장으로 버스가 정류장에 도착하는 시간을 그래프로 표현한 것이다. 지도는 현재 버스의 위·경도 좌표를 통해 표현한 것이다. 버스가 정류장 별 이동을 통해 위치 정보도 같이 표현해주는 시각화 예이다.

Ⅳ. 시스템 비교 및 평가

본 연구에서 제안한 스트리밍 데이터 수집·저장 시스템과 처리 시스템은 정성적 평가방법과 정량적 평가방법을 통해 성능 평가를 수행하였다. 정성적 평가 대상은 교통 빅데이터를 위한 기존 RDBMS로 구축된 시스템과 본 연구에서 제안하는 시스템의 수집능력, 저장능력, 처리능력을 기준으로 한다. 표 3은 정성적 평가방법을 정리한 내용이다.

[표 3] 기존 RDBMS 기반 시스템과 본 연구에서 제안하는 시스템의 비교

구분	기존 RDBMS 기반 시스템	제안하는 시스템
스트리밍 데이터 수집	미지원	지원
시스템 확장 방법	Scale-up (단일)	Scale-out (분산)
시스템 확장 비용	비교적 높음(고성능 서버)	비교적 낮음 (워크스테이션)
대용량 데이터 처리 프레임워크 지원여부	미지원	지원 (맵리듀스)
데이터 저장소 고가용성 지원여부	지원(DB 이중화)	지원 (하둡 2.0)
메인 메모리 기반 고속 데이터 처리 지원여부	미지원	지원 (스파크 2.0)
표준 SQL 지원	지원	부분적 지원

정량적 평가방법은 수집·저장·처리 시스템의 개별적 평가를 수행한다. 첫 번째, 수집 모듈은 카프카 클러스터의 메시지 수집 능력을 평가하는 방법으로 자체 평가를 수행했다. 두 번째, 저장 모듈은 하둡 클러스터를 평가하는데 빅데이터 플랫폼의 표준 벤치마크 도구인 TPCx-HS를 활용한다. TPC(Transaction Performance Council)는 DBMS(Database Management

System) 분야의 표준 벤치마크를 주도하는 기관으로 빅데이터 플랫폼의 성능 평가로 표준화된 방법을 제안하였다. TPCx-HS는 하둡 클러스터의 연산능력 평가를 위한 TeraSort와 저장 능력을 평가하기 위한 HSGen을 활용하여 클러스터의 종합적인 성능을 측정한다.

마지막 세 번째, 스트리밍 데이터 처리 능력을 평가하기 위해 본 연구에서 스트리밍 데이터 처리에 사용된 집계(Aggregation) 연산을 활용하여 단위시간당 처리량을 자체평가 한다.

(1) 스트리밍 데이터 수집능력 평가

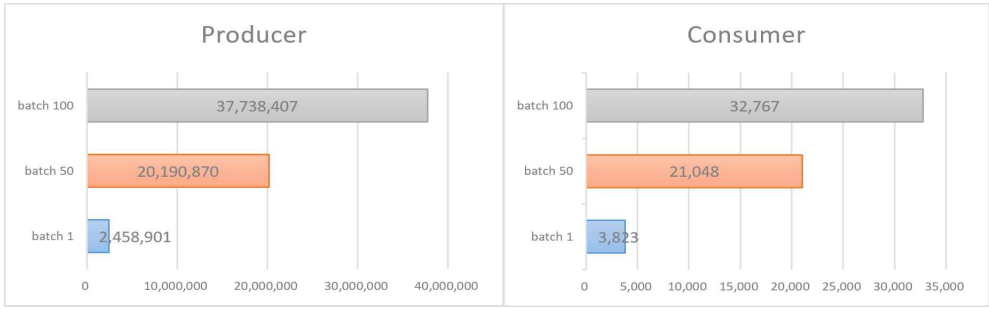
스트리밍 데이터 수집능력 평가는 2011년 Jay Kreps가 발표한 “Kafka: a Distributed Messaging System for Log Processing” 논문에서 카프카 클러스터의 테스트를 위해 배치 단위를 늘리면서 단위 시간당 데이터 수집능력을 평가한 내용을 참조하였다. 본 연구에서 수집을 위한 목적으로 개발된 데이터 생성기를 활용한다. 데이터 생성기는 스레드 개수를 늘려가면서 동시성 테스트가 가능하도록 설계되었으며 생성되는 데이터는 축적된 교통 스트리밍 데이터를 기반으로 한다.

[표 4] 스트리밍 데이터 수집능력 평가 결과 (60초간 처리된 데이터 건 수)

구분	Producer 테스트	Consumer 테스트
Kafka Batch 1	2,458,901 (40,918/sec)	3,823 (64/sec)
Kafka Batch 50	20,190,870 (336,514/sec)	21,048 (351/sec)
Kafka Batch 100	37,738,407 (628,873/sec)	32,767 (546/sec)

표 4는 성능평가 결과이다. 테스트 환경은 3대의 카프카 클러스터에서

진행되었다. 네트워크는 1Gbps 환경이며 각 클러스터 노드는 Intel i7 3.5GHz, 32GB 메인메모리, 4TB 크기의 Storage로 구성되며 RAID 구성은 하지 않았다. 데이터 생성기는 약 60초간 동작하게 되며 카프카 Producer 역할을 수행한다. Consumer는 단순히 카프카로부터 메시지를 가져가는 역할을 수행한다. 두 테스트 결과는 공통적으로 배치 처리할 메시지 크기가 커질수록 더 좋은 성능을 보이는 것으로 나타났다(그림 36). 하지만 배치처리 할 메시지 셋 크기가 증가할수록 실시간 처리능력이 떨어지게 되므로 적용 환경에 따라 배치크기를 적당히 결정해야 한다.



[그림 36] 스트리밍 데이터 수집 자체평가 결과 차트

(2) 스트리밍 데이터 저장능력 평가

스트리밍 데이터 저장능력은 TPCx-HS 도구의 HSGen을 사용하여 평가하였으며 100GB의 실수형 데이터를 무작위로 생성하고 하둡 클러스터의 복제 계수인 3에 따라 복제 본 2개를 생성하는데 걸리는 시간을 측정한다. 실험 환경은 하둡 클러스터 네임노드 1대와 데이터노드 14대로 구성하여 진행하였다. 네트워크는 1Gbps 환경이며 각 데이터 노드는 Intel Xeon E5 사양의 CPU, 64GB 메인 메모리, 그리고 RAID10 환경의 4TB Storage로 구성된다. 실험은 동일한 데이터 셋을 기반으로 총 2회 반복하였다. 결과는 표 5와 같다.

[표 5] 스트리밍 데이터 저장능력 평가 결과

구분	1회 테스트	2회 테스트
Real time	30.14 min	29.37 min
System time	11.84 sec	11.15 sec
User time	8.55 sec	8.93 sec

Real time은 작업이 시작되고 종료되기 까지 실제로 걸린 시간을 의미하고 System time은 커널 모드에서 동작한 시간, 그리고 User time은 유저 모드에서 동작한 시간을 의미한다. 실제 연산이 수행된 시간은 System time과 User time으로 전체 시간은 네트워크를 통한 데이터 전송시간 및 I/O 처리에 소요된 오버헤드로 보면 된다. HSGen 테스트 결과 100GB의 데이터를 하둡 클러스터에 저장하는데 2회 평균 약 29.76분이 소요되었다. 1분에 약 3.3GB, 1초에 약 56MB를 저장할 수 있었다. 이론적으로 1Gbps 환경은 약 125MB/S를 저장할 수 있었지만 클러스터에 2개의 복제본을 만드는데 걸리는 시간과 I/O 처리에 소요되는 오버헤드를 고려하였을 때 수긍할 만한 성능이라 할 수 있다. 하둡은 확장성(scale-out)에 자유로운 환경을 제공하는 만큼 추후에 클러스터 노드 확장을 통해 더 나은 성능을 이끌어 낼 수 있을 것으로 예상한다.

(3) 스트리밍 데이터 처리능력 평가

스트리밍 데이터 처리능력 평가는 BIS 데이터를 활용하여 노선 순서, 차량 번호, 정류장 이름과 같은 정보를 얻기 위해 해당 Bus ID를 기반으로 대중교통 정보(마스터 데이터)와 비교 연산을 통해 정보를

취합(Aggregation)하는 연산을 수행한다. 카프카 클러스터로 생성되는 데이터는 배치 사이즈 50으로 설정하여 진행했으며 초당 약 30만 건의 데이터를 카프카 클러스터에 생성한다. 스트리밍 데이터 처리 시스템은 카프카 Consumer로 동작하며 마이크로배치 처리 주기는 10초로 설정하였다. 스트리밍 데이터 처리 능력 평가 결과는 표 6과 같다.

[표 6] 스트리밍 데이터 처리능력 평가 결과 (10초간 처리된 데이터 건 수)

1회 테스트	2회 테스트	3회 테스트	4회 테스트	5회 테스트
2,341	2,289	2,377건	2,213	2,394

카프카로부터 가져온 메시지 셋은 초당 약 300건 내외이며 초당 약 230건의 데이터 처리 성능을 보였다.

Ⅴ. 결 론

5.1 요약 및 공헌

본 연구는 하둡 에코시스템을 활용하여 스트리밍 데이터의 수집·저장을 위한 시스템과 스트리밍 데이터 처리를 위한 시스템을 설계하고 구현하는 방법을 제시하였다. 하둡은 대용량 데이터를 저장하기 위한 HDFS(Hadoop Distributed File System)과 대용량 데이터를 배치 처리하기 위한 맵리듀스(MapReduce) 프레임워크가 제공된다. 하둡은 저장을 위한 시스템으로 사용되었으며 수집은 아파치 카프카를 이용한다.

스트리밍 데이터 수집·저장 시스템은 스케줄러 기반으로 데이터를 분배하고자 하는 빅데이터 플랫폼 마다 개별 스케줄러의 정책에 의해 분배된다. 빅데이터 플랫폼마다의 퍼포먼스와 특징 다르기 때문에 이를 위한 선택은 개별 스케줄러를 통해 데이터를 분배하는 것이다. 본 연구는 HDFS에 스트리밍 데이터를 배치 저장할 수 있게 시스템을 구성하였고, 그 외에 다른 저장소(Repository)에도 저장이 가능하다. 데이터 분배 모듈은 하나의 독립된 스레드로 동작한다. 시스템은 스레드 안전성을 보장하기 위해 주기적으로 스레드 상태를 점검하고 문제 발생 시 대처할 수 있게끔 설계되었다.

스트리밍 데이터 처리 시스템은 스파크 스트리밍 기반으로 운영된다. 다양한 스트리밍 처리 플랫폼이 존재하지만 스트리밍 데이터 처리 신뢰성이 높은 Exactly-once 정책을 지원하는 스파크를 사용함으로써 데이터의 중복과 데이터의 유실 방지를 어느정도 보장할 수 있었다. 스파크는 스파

크 스트리밍 API를 지원하는데 카프카와 바로 연계되어서 사용할 수 있다. 스파크 스트리밍 API를 활용해서 카프카로부터 토픽 별 메시지를 마이크로-배치로 가져옴으로써 메시지 셋을 스파크 연산을 통해 처리하고 결과를 카프카에 전달한다. 스파크 스트리밍 처리 모듈은 각 토픽 별로 생성되는 하나의 스레드이며, 이 스레드의 상태를 관리하는 Dispatcher를 두었다. 처리된 메시지 셋은 카프카에 저장되며 웹 애플리케이션을 통해 서비스가 가능하다.

스트리밍 데이터 수집·저장을 위한 시스템의 설계 방법을 제안함으로써 최근 화두가 되고 있는 IoT 환경에 데이터 수집에 응용할 수 있다. 또한 다양한 저장소에 데이터를 분배하기 위한 확장성 제공으로 하둡뿐만 아니라 다른 플랫폼에 데이터를 저장할 수 있는 인터페이스를 제공한다. 이 인터페이스를 구현함으로써 기존 시스템 구조를 변경하지 않고도 데이터 분배가 가능해진다.

스트리밍 데이터 처리 시스템은 스파크라는 뛰어난 범용적 고속 데이터 처리 플랫폼을 활용함으로써 스트리밍 데이터 처리에 높은 신뢰도를 보장함과 동시에 스파크가 제공하는 스파크 스트리밍 API 활용으로 비즈니스 로직에만 집중할 수 있게 되었다.

본 연구에서 제안하는 스트리밍 데이터 수집·저장·처리 시스템을 다른 분야에 활용함으로써 실시간 분석과 배치 분석을 위한 비즈니스 로직에만 집중할 수 있게 될 것으로 예상된다.

5.2 한계점 및 향후 연구

본 연구는 기존 개발된 빅데이터 플랫폼을 기반으로 스트리밍 데이터를 수집·저장·처리를 위한 시스템을 제안하였지만 모든 데이터에 최적화 된 시스템을 추구하지는 않았다. 즉, 데이터 특성에 맞게 빅데이터 플랫폼 엔진의 스케줄링 정책, 또는 데이터 처리 기법에 대한 연구가 추가로 진행될 필요가 있다. 또한, 교통 스트리밍 데이터 처리를 위한 스트리밍 데이터 처리 엔진 최적화 기법에 대한 연구도 향후 필요할 것으로 보인다.

참 고 문 헌

<국내문헌>

강윤희 (2014), “Apache Flume을 활용한 스트리밍 데이터 처리 시스템 프레임워크 설계,” 한국정보기술학회논문지, 12(11), 127-132.

강지우・손재기・안재훈 (2016), “Spark Streaming을 통한 ARP Spoofing 공격 감지와 대응 방법,” 대한전자공학회 학술대회, 194-195.

권순현・박동환・방효찬・박영택 (2015), “사물인터넷 환경에서 대용량 스트리밍 센서 데이터의 실시간 · 병렬 시맨틱 변환 기법,” 정보과학회논문지, 42(1), 54-67.

김영준・전용근・정일영 (2013), “IoT 서비스 지능화를 위한 디바이스 오브젝트화 및 오케스트레이션 메커니즘,” 한국통신학회논문지, 38(1), 19-32.

김직수・구엔 카오・김서영・황순욱 (2016), “하둡 기반 대규모 작업 배치 및 처리 기술 설계,” 정보과학회논문지, 43(6), 613-620.

박수현・류우석・홍봉희・권준호 (2012), “Hadoop기반 스트리밍 데이터 처리를 위한 스트림 분할 기법,” 한국정보과학회 학술발표논문집, 39(2C), 55-57.

박정희 (2016), “스트리밍 데이터에서의 기계학습,” 한국멀티미디어학회지, 20(3), 1-7.

복경수・육미선・노연우・한지은・김연우・임종태・유재수 (2015), “빅데이터 환경에서 스트림 질의 처리를 위한 인메모리 기반 점진적 처리 기법,” 한국콘텐츠학회논문지, 16(2), 163-173.

신승혁 (2015), “오픈소스 하드웨어를 이용한 경량 IoT 센서 게이트웨이에 관한 연구,” Journal of KIIT.

서원호・김현명 (2015), “테마 3: 사물인터넷 (IoT) 과 도시교통”, 대한지방행정공제회

정성민 (2016), “실시간 스트림 데이터 분석을 위한 시각화 가속 기술 및 시각적 분석 시스템,” 한국컴퓨터그래픽스학회.

정재화 (2016), “Beginning Hadoop: Learn fro basic to practical techniques,” 위키북스.

조완섭 (2015), 청주시 교통빅데이터 분석시스템, 연구보고서, 청주시청.

최도진 · 송석일 (2015), “Spark Streaming 기반의 그리드 색인을 이용하는 이동객체를 위한 연속 질의 기법,” 한국콘텐츠학회 종합학술대회 논문집, 67-68.

<국외문헌>

J. Dean · S. Ghemawat (2004), “MapReduce: simplified data processing on large clusters,” In Proc. conference on Symposium on Operating Systems Design & Implementation, 137-150.

Jay Kreps (2011), “Kafka: a Distributed Messaging System for Log Processing,” NetDB.

Jay Kreps · Neha Narkhede · Jun Rao (2011), “Kafka: a Distributed Messaging System for Log Processing,” 6th International Workshop on Networking Meets Databases.

M. Weiser (1991), “The Computer for the 21 Century,” Scientific American, 265(3), 94-104.

M. Weiser (1994), “Creating the Invisible interface,” UIST94 Proceedings ACM Symposium.

M. Weiser · J. Seely Brown (1996), “Designing Calm Technology,” PowerGrid Journal, 1(1).

하둡 에코시스템 기술을 적용한 교통 스트리밍 데이터의 저장 및 처리기술 설계 및 구현*

김 진 혁

충 북 대 학 교 대 학 원
빅데이터협동과정 빅데이터 전공

(지도교수 조 완 섭)

요 약

청주시 교통빅데이터 분석시스템은 청주시 전역에서 발생하는 BIS, ATMS, 교통카드 정보를 일단위로 수집하고 분석하는 배치 형태의 시스템이다. 본 연구는 청주시 교통빅데이터 분석시스템에서 스트리밍 빅데이터의 실시간 수집·저장 시스템과 처리 시스템을 설계하고 구현하였다. 교통 빅데이터 수집은 카프카를 사용하며, 배치형태의 저장과 처리를 위해 Hadoop를 활용한다. 또한 실시간 처리를 위해 스파크 스트리밍 기술을 활용한다. 또한 제안된 시스템의 성능을 평가하기 위해 카프카를 이용한 데이터 수집능력, 표준

* 2017년 2월 석사학위 논문.

화된 밴치마킹 방식인 TPCx-HS를 활용한 저장능력, 스트리밍 데이터 처리능력 세가지로 구분하여 성능을 평가하였다. 수집능력의 경우 메시지셋의 크기를 환경에 적절하게 결정하는 것이 성능에 큰 영향을 미치며, 저장능력의 경우에는 스캐일아웃 방식이므로 노드수의 증가에 따라 개선될 수 있으며, 처리능력은 초당 200건 정도 처리할 수 있어 청주시 교통데이터를 처리하는 데는 충분하며, 필요한 경우 노드수를 증가하면 된다.

키워드 : 스트리밍 데이터 프로세싱, 하둡 에코시스템, 스파크 스트리밍