JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 23, No. 10: 1179~1187, Oct. 2019

실시간 데이터 처리를 위한 개방형 데이터 프레임워크 적용 방안

박순호[1*] · 김영길[2]

Application Of Open Data Framework For Real-Time Data Processing

Sun-ho Park[1*] · Young-kil Kim[2]

[1*]Principal Research Engineer, KL-Net, Seoul, 06246 Korea
[2]Professor, Department of BioMedical Engineering, Ajou University, Suwon, 16499 Korea

요 약

오늘날의 기술 환경에서 대다수의 빅 데이터 기반 애플리케이션 및 솔루션은 스트리밍 데이터의 실시간 처리를 기반으로 한다. 빅 데이터 스트림의 실시간 처리 및 분석은 빅 데이터 기반 애플리케이션 및 솔루션 개발에서 중요한 역할을 한다. 특히 해사 분야 데이터 처리 환경에서도 데이터의 폭발적 증대에 따른 대용량 실시간 데이터를 빠르게 처리 및 분석할 수 있는 기술 개발의 필요성이 가속화되고 있다. 따라서 본 논문에서는 다양한 빅 데이터 처리를 위한 오픈소스 기술 중에 적합한 오픈소스로 NiFi, Kafka, Druid의 특징을 분석하여 한국형 e-Navigation 서비스에서 해사 분야 서비스 분석에 필요한 외부 연계 필요 정보들을 상시 최신 정보로 제공할 수 있도록 실시간 데이터 처리를 위한 개방형 데이터 프레임워크 기술 적용의 기초를 마련하고자 한다.

ABSTRACT

In today's technology environment, most big data-based applications and solutions are based on real-time processing of streaming data. Real-time processing and analysis of big data streams plays an important role in the development of big data-based applications and solutions. In particular, in the maritime data processing environment, the necessity of developing a technology capable of rapidly processing and analyzing a large amount of real-time data due to the explosion of data is accelerating. Therefore, this paper analyzes the characteristics of NiFi, Kafka, and Druid as suitable open source among various open data technologies for processing big data, and provides the latest information on external linkage necessary for maritime service analysis in Korean e-Navigation service. To this end, we will lay the foundation for applying open data framework technology for real-time data processing.

키워드 : e-Navigation, 개방형 데이터 프레임워크, 빅 데이터, 스트리밍

Keywords : e-Navigation, Open Data Framework, Big Data, Streaming

Received 25 August 2019, **Revised** 27 August 2019, **Accepted** 25 September 2019

*** Corresponding Author** Sun-ho Park(E-mail:javaeye@klnet.co.kr, Tel:+82-2-2175-9446)
Principal Research Engineer, KL-Net, Seoul, 06246 Korea

Open Access http://doi.org/10.6109/jkiice.2019.23.10.1179 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

오늘날의 기술 환경에서 대다수의 빅 데이터 기반 애플리케이션 및 솔루션은 스트리밍 데이터의 실시간 처리를 기반으로 한다. 빅 데이터 스트림의 실시간 처리 및 분석은 빅 데이터 기반 애플리케이션 및 솔루션 개발에 중요한 역할을 한다.[1]

특히 해사 분야 데이터 처리 환경에서도 데이터의 폭발적 증대에 따른 대용량 실시간 데이터를 빠르게 처리 및 분석할 수 있는 기술 개발의 필요성이 가속화되고 있다. 또한 상황 인식 및 지능형 의사 결정을 지원하기 위해 스마트 IoT 장치에서 전송된 빅 데이터를 분석해야 하며, 지속적으로 생성되는 빅 데이터를 실시간으로 처리하려면 스트리밍 데이터 처리 기술이 필요하다.[2]

최근 빅 데이터 시스템도 비즈니스 요구사항에 따라 점점 발전하였고, 2011년 Hadoop 시스템 모습과 현재의 빅 데이터 시스템은 많은 변화가 있다. 빅 데이터를 처리하는 프레임워크로 흔히 Hadoop은 MapReduce를 사용하는데 MapReduce는 페타바이트(Petabyte) 이상의 데이터를 여러 노드로 구성된 클라우드 환경에서 병렬 처리하는 기법으로, 함수형 프로그래밍에서 일반적으로 사용되는 Map과 Reduce 방식을 사용해 데이터를 처리한다. MapReduce는 대량 데이터를 분산 처리할 수 있는 좋은 기법이지만, 배치 방식으로 데이터를 처리하기 때문에 실시간으로 데이터를 조회하기 어렵다.[3,4]

본 논문에서는 다양한 빅 데이터 처리를 위한 오픈소스 기술들 중에 적합한 오픈소스로 NiFi, Kafka, Druid의 특징을 분석하여 한국형 e-Navigation 서비스에서 해사 분야 서비스 분석에 필요한 기상정보, 해양안전정보, 선박운항정보 등의 외부 연계 필요 정보들을 상시 최신 정보로 제공할 수 있는 실시간 데이터 처리를 위한 개방형 데이터 프레임워크 기술 적용의 기초를 마련하고자 한다.

Ⅱ. 관련 연구

기존의 한국형 e-Navigation 서비스의 실시간 데이터 처리 시스템에서 Hadoop의 MapReduce가 빅 데이터 분석을 용이하게 해주는 장점이 있으나 복잡하고 다단계 처리가 필요한 업무나 상호적이고 신속한 대응이 필요한 처리 작업에는 적합하지 않은 상황이며, 분산 메모리 기반의 Spark는 Hadoop에 비해 10배 이상의 빠른 성능을 보여 주고 있다고 한다.[5] 하지만 Spark는 In-Memory 처리를 지원하기 때문에 데이터 처리 성능이 시스템 메모리에 의존적이다.[6] 따라서 기존 시스템을 개선하고 성능을 향상시키기 위해 제안하는 NiFi, Kafka, Druid 각각의 실시간 데이터 처리 기법에 대해 기본적인 이론과 특성을 살펴보고 프레임워크 적용 방법을 연구하였다.

2.1. NiFi

NiagaraFiles의 줄임말인 Apache NiFi는 소프트웨어 시스템 간의 데이터 흐름을 자동화하도록 설계된 Apache Software Foundation의 오픈소스 소프트웨어 프로젝트이다.[7]

NiFi는 클러스터 환경에서 실행되며 여러 네트워크를 통과하는 데이터가 어떻게 흘러가고 있는지를 파악해주며, 모든 데이터 소스와 대상 간의 데이터 이동을 쉽게 관리할 수 있도록 실시간 제어 기능을 제공한다. 또한 다양한 데이터 형식, 스키마, 프로토콜, 속도 및 크기의 서로 다른 분산 환경을 지원한다. 데이터베이스, 파일 시스템, 네트워크, 클라우드, 하둡, 검색엔진, SNS, 검색엔진, 암호화, 이미지 변환 등 150여 가지 이상의 구성 요소를 제공하며, 현재 글자 인식과 음석 인식 등 다양한 기능들을 공개하고 있다.[8]

NiFi는 엄격한 보안과 규정을 준수해야 할 요구사항이 있는 핵심적인 업무의 데이터 흐름에 사용할 수 있으며, 전체 데이터 프로세스를 시각화하고 변경 사항을 즉시 실시간으로 처리할 수 있다. 또한 일련의 이벤트로 정보 흐름을 처리할 수 있는 처리 엔진이며, 이벤트 프로세서 및 정보 컨트롤러로 작동하는데 배치 작업 보다는 대량의 실시간 데이터를 처리하는 데 적합하다.

가령 데이터가 일괄적으로 처리되도록 데이터가 완전히 로드되기를 기다리는 대신 NiFi를 사용하여 데이터베이스 SQL을 개별 행 단위로 처리할 수 있다. 이를 통해 최종 Consumer에게 데이터를 훨씬 풍부하게 제공할 수 있다.[8] 또한 이벤트 처리 중 손상된 값이 있는 경우 처리 중인 정보 일부분에만 영향을 미치며 하나의 큰 블록에서 데이터를 처리하는 경우와 동일한 방식으로 전체 프로시저에 영향을 미치지 않는다.

NiFi는 <그림 1>과 같이 Flowfile, Flowfile 프로세서 및 Connections로 구성되며, 정보 흐름을 일련의 사건으

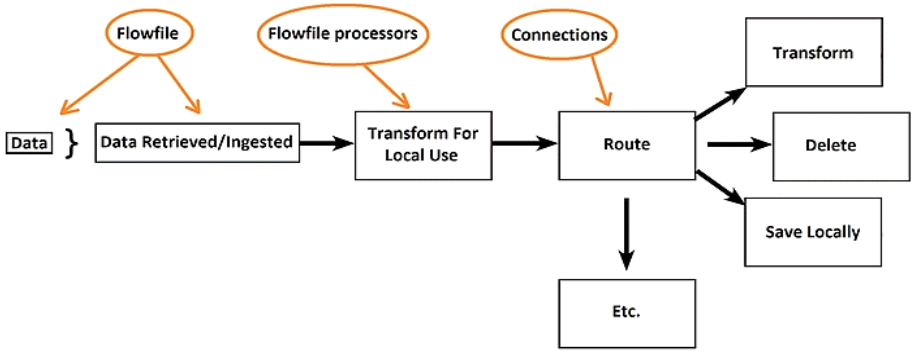

Fig. 1 NiFi Configuration

로 취급하여 파일 도착 및 검색, 파일 읽기, 파일 변경, 파일 이동 등 파일을 다른 곳으로 보내는 도구로 구성된다.[8]

2.1.1. Flowfile

Flowfile은 NiFi에서 데이터 단위를 나타내는데 Flowfile은 Flowfile 내용 및 Flowfile 속성으로 구성된다. 내용은 Flowfile이 나타내는 데이터로서 속성은 데이터에 대한 컨텍스트와 정보를 제공하는 특성을 가진다.

속성은 키와 값 형태로 데이터의 이동 및 저장 시 필요한 정보들로 구성되는데 각 Flowfile은 추가 작업을 사용하여 추가될 수 있는 기본 속성 세트로 구성된다.[8]

2.1.2. Flowfile 프로세서

Flowfile은 일반적으로 출력과 입력이 있는 자체 포함된 코드 세그먼트로서 Flowfile 프로세서는 NiFi에서 모든 작업을 수행한다. Flowfile이 생성되면 Flowfile Repository에 내용과 속성에 대한 위치가 저장된다. 이 속성에는 파일 이름, 작성 날짜 및 파일의 내용이 들어 있는 페이로드가 포함된다.

프로세서와 프로세서를 이동할 때마다 Flowfile의 복사본이 만들어져서 추적이 가능하며, 내용을 복사하지 않고 Flowfile 위치에 대한 포인트 정보만 복사한다.[8]

2.1.3. Connections

명령을 보내고 응답을 받으려면 연결이 필요한데 Flowfile은 프로세서 간에 이동해야 하며, Connections는 프로세서 간 이동 경로를 결정한다.

Connections은 Flowfile의 큐(Queue) 역할로서 Flowfile의 우선순위, 만료, 부하조절 기능을 한다. 우선 순위를 정해 다음 프로세서에게 제공할지 여부, 만료는 얼마까지만 큐에 머물지, 부하조절은 큐 용량에 따라 Flowfile 생성 속도를 조절한다.[8]

2.2. Kafka

Kafka는 메시지를 디스크 순차적으로 저장함으로 서버에 장애가 나도 메시지가 디스크에 저장되어 있으므로 유실 걱정이 없으며 디스크가 순차적으로 저장되어 있어 디스크 I/O가 줄어들어 성능이 빨라진다. 또한 분산 처리를 함에 있어 여러 개의 Partition을 서버들에 분산시켜 나누어 처리함으로 메시지를 상황에 맞추어 빠르게 처리할 수 있다.[6] 따라서 Kafka는 강력한 내구성, 확장성 및 결함 허용 지원 기능을 제공하는 분산 메시징 시스템으로 실시간 수집된 데이터를 분산 메시지 큐로 처리하여 데이터 손실을 방지한다.

데이터 Producer와 Consumer를 분리하는 분산형 메시지 버스로서 <그림 2>와 같은 구조로 대형 소프트웨어 시스템의 구성 요소 간 서비스의 신뢰성을 보장하기 위한 실시간 이벤트 처리 및 데이터 통합이 가능하다. 또한 성능에 영향을 미치지 않고 중단 없이 많은 수의 Consumer를 추가 할 수 있는 확장성이 뛰어나며, 이벤트를 더욱 효율적으로 처리함과 동시에 Producer와 Consumer 사이의 완충 역할을 한다.[9]

Kafka의 메시지 내구성이 높고 특정 기간 동안 메시지 및 이벤트 지속 다른 프레임 워크와의 통합이 용이하며 일반적으로 시스템 또는 애플리케이션 간에 데이터를 안정적으로 얻는 실시간 스트리밍 데이터 파이프라인 구축과 데이터 스트림을 변환하거나 이에 반응하는 실시간 스트리밍 애플리케이션 구축과 관련하여 광범위한 종류의 응용 프로그램에 사용된다.[9]

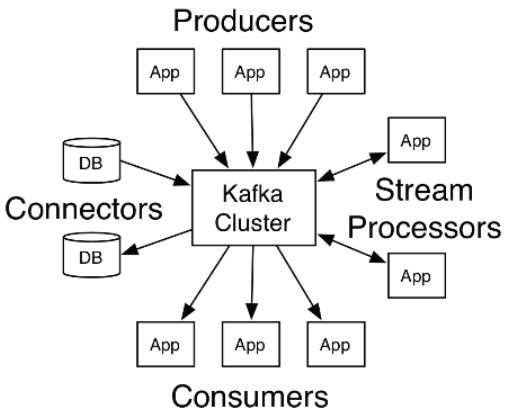

Fig. 2 Kafka Architecture

2.2.1. Producers

Producer는 데이터를 발생시키고 Kafka-Cluster에 적재하는 프로세스이다.

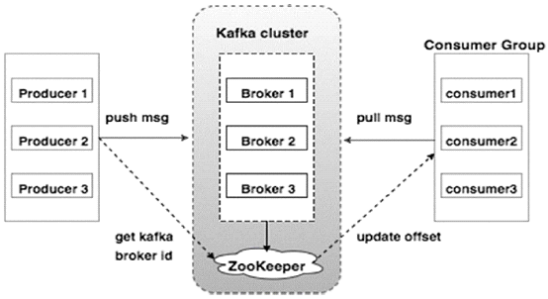

Fig. 3 Kafka Cluster

2.2.2. Kafka Cluster

Kafka 서버로 이루어진 클러스터로 <그림 3>와 같이 Kafka Cluster를 이루는 각 요소는 다음과 같다.

- Broker : Kafka Cluster는 일반적으로 균형을 유지하기 위해 여러 Broker로 구성되는데 Broker는 상태를 저장하지 않으므로 Zookeeper를 사용하여 클러스터 상태를 유지한다.[9]

하나의 Kafka Broker 인스턴스는 초당 수십만 건의 읽기 및 쓰기를 처리할 수 있으며 각 Broker는 성능에 영향을 주지 않고 메시지를 처리할 수 있다.[9]

- Zookeeper : Kafka Broker 관리 및 조정에 사용되며 주로 Producer와 Consumer 에게 Kafka시스템에 새 Broker가 있는지 또는 Broker 오류가 있는지를 확인하는데 사용된다.[9]

Broker 유무에 대해 Zookeeper가 수신한 통지에 따라 Producer와 Consumer는 결정을 내리고 다른 Broker와 작업을 조정한다.

2.2.3. Consumers

Consumer의 집합을 구성하는 단위로 Kafka에서는 Consumer Group으로서 데이터를 처리하며 Consumer Group 안의 Consumer 수만큼 파티션의 데이터를 분산 처리한다.[9]

2.2.4. Stream Processors

하나의 가공 공정을 나타내는 프로세서 토플로지 노드로 프로세서 API를 사용하면 한 번에 하나의 수신 레코드를 처리하는 임의의 스트림 프로세서를 정의하고 이러한 프로세서를 관련 상태 저장소와 연결하여 프로세서 토플로지를 구성한다.[9]

2.3. Druid

Druid는 고성능 실시간 분석 데이터베이스로 컬럼 지향의 분산 데이터 스토어로 막대한 양의 이벤트 데이터를 빠르게 흡수하고 데이터 상부에 낮은 레이턴시의 쿼리를 제공한다.

일반적으로 수십에서 수백 대의 서버 클러스터에 배포되며 초당 수백만 레코드의 수집 속도, 수조 개의 레코드 보존 및 1초 미만에서 몇 초의 쿼리 대기 시간을 제공한다. 또한 비공유 모델로 검색에 용이한 인덱스 포맷으로 실시간이나 배치로 데이터를 수집하여 OLAP (On-Line Analytical Processing) 질의를 처리할 수 있다.[10]

Druid는 <그림 4>와 같이 분산된 데이터베이스로 각 핵심 노드는 개별적으로 또는 공동으로 배포가 가능하며, 다른 노드의 작업에 영향을 미치지 않으면서 독립적으로 작동한다. 또한 효과적인 방식으로 수평으로 확장 가능하며 다중 테넌트 애플리케이션을 지원할 수 있는 기능도 있다.[10]

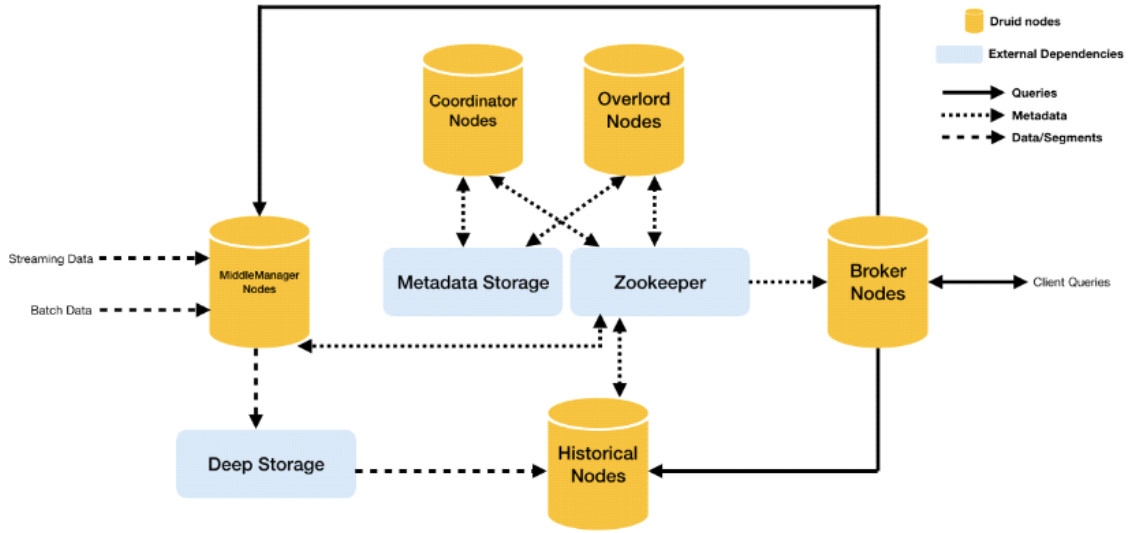

Fig. 4 Druid Architecture

2.3.1. Coordinator 노드

Coordinator 노드는 클러스터에서 데이터 가용성을 관리한다. 또한 Overload 노드와 Historical 노드를 감시하고 세그먼트가 Historical 노드에 균형을 잘 유지하고 각 실행마다 데이터베이스에서 사용 가능한 데이터 저장단위인 세그먼트의 목록을 클러스터의 현재 세그먼트와 비교한다.

주로 세그먼트 관리 및 배포를 담당하는데 Historical 프로세스와 통신하여 구성에 따라 세그먼트를 로드 또는 드롭하며 주기적으로 클러스터의 상태를 판단하여 예상되는 클러스터의 상태와 실제 클러스터의 상태를 비교함으로써 실제 상태와 비교하여 지금 클러스터의 상태를 결정한다.[11]

2.3.2. Overload(Real-time) 노드

Overload 노드는 실시간 데이터 스트림를 담당하며 데이터 수집 작업의 할당을 제어한다. 수집 작업을 MiddleManager에 할당하고 세그먼트 배포를 조정한다.

작업 수락, 작업 배포 조정, 작업 잠금 생성 및 상태 반환을 담당하며 로컬 또는 원격의 두 가지 모드 중 하나로 실행되도록 구성할 수 있다.[12]

로컬모드에서 Overload 프로세스를 실행할 때는 모든 MiddleManager 프로세스가 필요하며 일반적으로 간단한 작업에 사용되며 원격모드에서는 Overload 및 MiddleManager는 별도의 프로세스에서 실행되며 각각 다른 서버에서 실행할 수 있다. 인덱싱 서비스를 모든 Druid 인덱싱에 대한 단일 엔드 포인트로 사용하려는 경우 원격 모드를 사용한다.[12]

2.3.3. Broker노드

Broker 노드는 외부 클라이언트에서 쿼리를 요청 받고 이 쿼리를 중계하는 매개자로 Historical 노드와 MiddleManager 노드에 전달한다.

Historical 노드나 MiddleManager 노드에 직접 쿼리하기 보단 Broker 노드에 쿼리하는 것이 좋다. 분산 클러스터를 실행하려는 경우 쿼리를 라우팅하는 프로세스로 어떤 프로세스가 어떤 세그먼트에 있는지에 대해 Zookeeper에 개시된 메타 데이터를 이해하고 올바른 노드에 도달하도록 쿼리를 라우팅한다.[13]

2.3.4. Historical 노드

Historical 노드는 배치 처리를 담당하며 Deep Storage로부터 세그먼트를 다운 받아 처리하고 이 세그먼트에 대한 쿼리에 응답한다.

Zookeeper에 대한 지속적인 연결을 유지하고 새로운 세그먼트 정보를 위해 구성 가능한 Zookeeper 경로를 감시한다. Coordinator 노드와 직접 통신하지 않고 조정을 위해 Zookeeper에 의존한다.[14]

2.3.5. MiddleManager 노드

MiddleManager 노드는 Coordinator와 Overload 노드의 작업을 실행하며 새로운 데이터를 외부 데이터 소스로부터 받아서 클러스터에 배포한다.[15]

2.3.6. Zookeeper

Zookeeper는 클러스터 상태를 관리하기 위해 사용된다. Coordinator 및 Broker 노드를 관찰하여 어떤 세그먼트를 제공하고 있는지 확인할 수 있다.[16]

2.3.7. Deep Storage

Deep Storage는 세그먼트를 영구적으로 저장하기 위한 영역이다. 모든 Druid 서버가 액세스 할 수 있는 공유 파일 스토리지로 시스템에 수집된 모든 세그먼트가 저장된다. [17]

2.3.8. Metadata Storage

Metadata Storage는 Druid 클러스터가 작동하는데 필수적인 메타 데이터를 저장한다. 세그먼트 가용성 정보 및 작업 정보와 같은 다양한 메타 데이터를 보유한다. 일반적으로 PostgreSQL 또는 MySQL과 같은 전통적인 RDBMS와 유사하다.[18]

Ⅲ. 제안하는 개방형 데이터 프레임워크 적용 방안

3.1. 실시간 데이터 수집 및 처리

앞서 살펴본 바와 같이 NiFi와 Kafka는 데이터 파이프라인을 만드는 데 뛰어난 성능을 제공하고 수평으로 확장이 가능하며 사용자 정의 구성 요소를 통해 기능을 확장할 수 있는 플러그인 아키텍처를 제공한다.[19] 따라서 NiFi와 Kafka는 각자가 Producer 혹은 Consumer의 역할을 할 수 있다.

따라서 여러 가지 형태로 설정이 가능하나 한국형 e-Navigation 서비스 적용에서는 <그림 5>와 같이 NiFi가 Kafka의 Producer 역할을 하게 하여 NiFi는 스트림 데이터를 수집하고 처리하며 소프트웨어 시스템 간의 데이터 흐름을 자동화하도록 하고 Kafka는 실시간 데이터 파이프라인 및 스트리밍 App을 구축하여 스트리밍 데이터를 저장하도록 하였다. NiFi를 가장 효율적으로

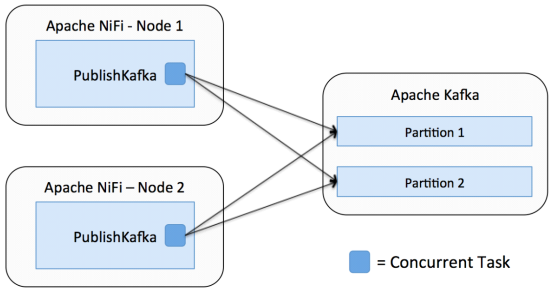

Fig. 5 NiFi as a Producer

사용하기 위해 모든 소스에서 데이터를 입력으로 생성하여 Kafka Broker에 전달하는 Kafka의 Producer 역할을 하도록 했다.

이런 기법으로 설정한 이유 중 가장 큰 장점은 NiFi의 PublishKafka라는 속성을 사용하여 일련의 프로세서를 끌어다 놓기만 하면 Publish 코드를 작성하지 않고도 Kafka에 데이터를 가져올 수 있으며 이 파이프라인을 시각적으로 모니터링뿐만 아니라 제어도 할 수 있다.

반대로 NiFi에는 Kafka브로커 및 그룹을 구성할 수 있는 ConsumerKafka 속성도 있다. 이럴 경우 NiFi와 Kafka는 Broker를 통해 Producer와 Consumer를 연결하는데 NiFi에서 데이터 흐름 논리의 대부분은 Producer와 Consumer 내부에 있지 않고 Broker에 있으므로 중앙 집중식 제어가 가능하다.

NiFi에는 Producer 및 Consumer 성능에 영향을 줄 수 있는 요소가 있는데 PublishKafka와 ConsumerKafka에 모두 “Message Demarcator” 라는 속성이 있다. Producer 측에서 구분자가 있는 경우 스트림 파일을 구분자 기준으로 각 메시지를 개별적으로 전달하는데 반해 이 속성을 비워 두면 PublishKafka는 스트림 파일을 단일 메시지로 전달하기 때문에 더 높은 처리량을 생성한다.

3.2. 실시간 저장

Kafka는 빠르고 확장 가능하며 내구성이 뛰어나고 스트림 전달을 위해 설계된 내결함성이 있는 메시징 버스로 NiFi로부터 전달된 이벤트를 수집하고 버퍼링한다. 여기서 Kafka는 스트림의 중앙 저장소처럼 사용되며, NiFi로부터 전달된 이벤트의 처리 및 분석을 위해 Druid로 라우팅 되기 전에 Kafka에 저장된다.

NiFi를 통해 수집된 데이터는 Kafka Topic이라 불리는 파이프라인에 데이터 스트림 레코드를 게시하고,

Kafka의 스트리밍 데이터를 사용하여 실시간 데이터 색인 작업을 위해 Druid로 전송된다.[20]

Druid는 초당 수백만 건의 이벤트 속도로 데이터를 수집할 수 있으며 이벤트가 발생한 직후 이벤트를 탐색하고 실시간 결과와 이벤트를 결합하는 데 사용되는 이상적인 스트리밍 분석 데이터 저장소로써 Kafka의 메시지 버스와 쌍을 이루어 고가용성과 유연성을 제공하여 Kafka와 Druid는 강력한 스트리밍 분석 어플리케이션 구축이 가능하다. [20]

Druid는 Kafka로부터 넘겨온 이벤트 데이터에 대한 OLAP 쿼리를 위해 설계된 데이터 저장소로서 실시간 데이터 처리를 수행하고 요청한 형태로 데이터를 인덱싱하며, 다차원에 대한 집계나 질의를 실시간으로 응답해준다.[21]

Kafka를 통해 Druid로 스트림 데이터를 넘기는 목적은 Kafka가 Druid로 들어오는 데이터의 버퍼 역할을 하기 때문이다. Druid의 Overlaod 노드는 데이터 Consumer로 데이터 스트림을 생성해 줄 Producer가 필요한데 Kafka가 여기에 위치하여 데이터 버스의 역할을 하게 된다. 그러면 Overload 노드에서 Kafka로부터 데이터를 가져간다.

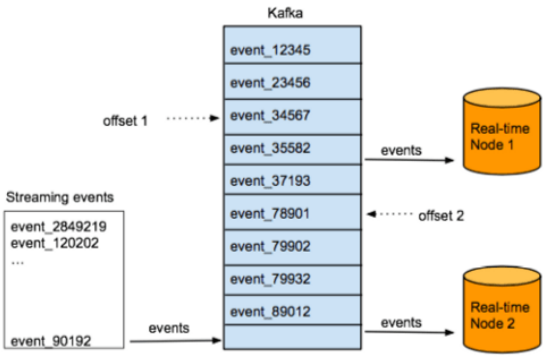

Fig. 6 Data Access between Kafka and Druid

Kafka는 <그림 6>과 같이 파일 오프셋으로 데이터를 접근하는데 이 파일 오프셋은 Consumer가 직접 관리할 수 있다. Druid의 Overload 노드는 이 오프셋을 직접 관리하여 전달에 실패할 경우를 대비한다. 만약 실패가 되었을 경우 이 오프셋을 통해 데이터를 빠르게 지속적으로 가져온다.

하지만 Druid의 한계점인 데이터 원본에 대한 저장을 하지 않으므로 Druid-HDFS-Storage 확장을 사용하여 Deep Storage로 Hadoop을 사용하도록 하였다.

이와 같이 한국형 e-Navigation 서비스를 위한 NiFi, Kafka, Druid의 통합 파이프라인은 복잡한 이벤트 처리 및 기계학습 작업을 처리하기 위해 실시간 데이터를 빠르게 집계하면서 탐색적 분석을 해야 하거나 데이터가 입력되는 동시에 실시간으로 분석, 항상 가용성을 보장하는 데이터 저장소가 필요한 경우에 적용이 가능하다

Ⅳ. 실험 및 평가

한국형 e-Navigation 서비스에서 해사 데이터의 실시간 데이터 처리를 위한 개방형 데이터 프레임워크 적용을 위해 제안하는 NiFi, Kafka, Druid 각각의 실시간 데이터 처리 기법을 적절하게 사용하여 데이터 수집, 데이터 처리, 데이터 저장의 역할로 <그림 7>과 같이 구성하였으며, 실험 환경은 <표 1>과 같으며, <표 2>와 같이 해양기상데이터, 선박입출항데이터, 선박운항데이터 테이블로 실험 및 평가를 진행하였다.

Table. 1 Test Environment

Equipment	Desc.
Server	Intel XEON Processor E5-1620 V4 128GB 2400MHzz DDR4 RDIMM ECC 1x2.5Inch 1TB SATA Class 20 SSD 2x4TB 3.5Inch SATA HDD NVIDIA Quadro M4000 8GB

Table. 2 Test Dataset

Data source	Dimension	row
SeaVantage	12	2,890,000
Ship Arr/Dept	76	2,500,000
Ship Sailing	125	3,600,000

제안하는 프레임워크의 우수성을 보이기 위해 실험환경에서 3개의 가상 노드를 구축하고 기존 시스템과의 데이터 쿼리 성능평가를 수행하였다. 기존 시스템과의 데이터 수집부터 쿼리까지의 전체 프로세스의 직접적인 비교에는 한계가 있어 실험을 위해 사용할 데이터셋을 HDFS 데이터 노드에 함께 배치하고 <그림 8>과 같이 실험 데이터에 대한 데이터 쿼리 처리 속도 면에서 비교해 봤을 때 제안하는 시스템의 경우 Timeseries 데이터에 대해 기존 시스템 보다 10배 이상의 응답성을 보

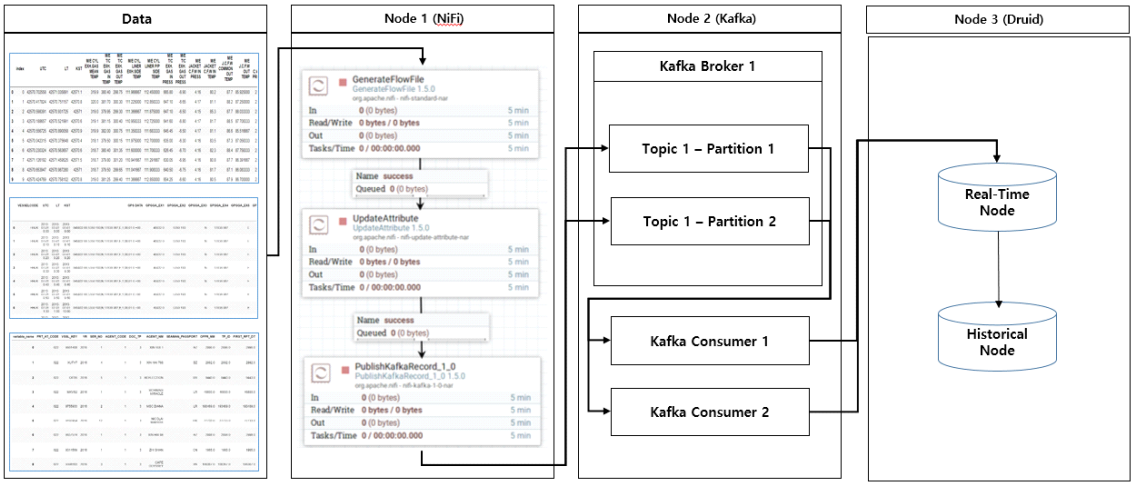

Fig. 7 Real-time data linkage processing diagram of maritime data

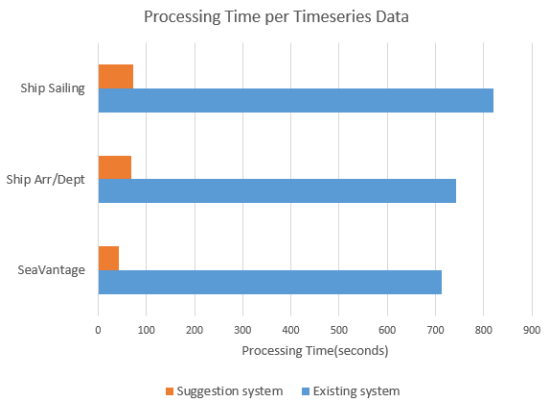

Fig. 8 Processing Time per Timeseries Data

여주고 있다. 여기서 Druid는 기본적으로 SQL을 지원하지 않기 때문에 RDruid를 사용하였고 Hadoop은 HiveQL을 사용하였다. 또한 실험을 위한 서버 구동환경의 제약적인 한계로 Druid를 단일 노드로 구성하였으나 좋은 성능을 내는 것을 확인하였다.

따라서 제안하는 프레임워크는 기존의 한국형 e-Navigation 서비스에서 대기 시간이 매우 짧은 쿼리에 서비스의 중점을 두고 수천 명의 사용자가 사용하는 응용 프로그램에 각 쿼리가 데이터를 탐색할 수 있을 정도로 빠르게 반환해야하는 경우에 매우 효과적이며, 수집하는 Timeseriese 데이터를 완전히 색인화하여 기존 Hadoop과 응용프로그램 사이의 중간 계층 역할을 할 수 있을 것으로 분석된다. 제안한 프레임워크를 기존 시스템에 적용 시 Hadoop에서 수집된 데이터의 원본을 모두 저장하고, 저장된 데이터를 로드하여 더 빠르게 엑세스할 수 있다.

Ⅴ. 결 론

본 논문은 한국형 e-Navigation 서비스의 해사 데이터의 실시간 데이터 처리를 개선하기 위해 NiFi, Kafka, Druid가 가지는 데이터 처리 기법을 적용하여 많은 양의 데이터를 실시간으로 처리할 수 있는 오픈소스 기반의 프레임워크를 제안하였다.

향후 연구로 제안하는 프레임워크와 데이터 분석 기법들의 활용하여 데이터 시각화 및 스트리밍 가시화를 통한 예측 분석이 가능한 연구를 진행할 계획이다.

ACKNOWLEDGEMENT

This research is a part of the project titled “SMART-Navigation project,” funded by the Ministry of Oceans and Fisheries.

REFERENCES

[1] F. Gurcan, and M. Berigel, “Real-Time Processing of Big Data Stream: Lifecycle, Tools, Tasks, and Challengs,” *2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Turkey, pp. 1-6, 2018.

[2] H. S. Jung, C. S. Yoon, and Y. W. Lee, “Cloud computing platform based real-time processing for stream reasoning,” *Sixth International Conference on Future Generation Communication Technologies (FGCT)*, Doublin, pp. 1-5, 2017.

[3] How to analyze real-time big-data [Internet]. Available: https://d2.naver.com/helloworld/694050.

[4] Real-time data feed processing and Apache Kafka for it [Internet]. Available: https://m.blog.naver.com/sundooedu/21230385470.

[5] K. S. Paik, “Multi-channel data connection and Real-time processing system desinged for Big Data collection,” *The Korea Contents Society*, 2016.

[6] S. Han, H. Chung, D. Ok, and Y. N, “Impact Analysis of Data Volume on Spark Performance,” *The Korean Institute of Information Scientists and Engineers*, 2016.

[7] Apache NiFi [Internet]. Available: https://en.wikipedia.org/wiki/Apache_NiFi.

[8] The Core concepts of NiFi [Internet]. Available: https://nifi.apache.org/docs.html.

[9] Kafka Introduction [Internet]. Available: https://kafka.apache.org/intro.

[10] What is Druid [Internet]. Available: https://druid.apache.org/docs/latest/design.

[11] Druid Coordinator Process [Internet]. Available: https://druid.apache.org/docs/latest/design/coordinator.html.

[12] Druid Overlord Process [Internet]. Available: https://druid.apache.org/docs/latest/design/overload.html.

[13] Druid Broker [Internet]. Available: https://druid.apache.org/docs/latest/design/broker.html.

[14] Druid Historical Process [Internet]. Available: https://druid.apache.org/docs/latest/design/historical.html.

[15] Druid MiddleManager Process [Internet]. Available: https://druid.apache.org/docs/latest/design/middlemanager.html.

[16] Druid Zookeeper [Internet]. Available: https://druid.apache.org/ocs/latest/dependencies/zookeeper.html.

[17] Druid Deep Storage [Internet]. Available: https://druid.pache.org/docs/latest/dependencies/deep-storage.html.

[18] Druid Metadata Storage [Internet]. Available: https://druid.pache.org/docs/latest/dependencies/metadata-storage.html.

[19] H. Isah, and F. Zulkernine, “A Scalable and Roburst Framework for Data Stream Ingestion,” Cornell University, arXiv: 812.04197, 2018.

[20] F. Yang, (2016, June). Building a Streaming Analytics Stack with ApacheKafka and Druid. *Confluent* [Online]. Available:https://www.confluent.io/blog/building-a-streaming-analytics-stack-with-apache-kafka-and-druid.

[21] F. Yang, E. Tschetter, and X. Leaute, “Druid-A Real time Analytical Data Store,” 2014.

박순호(Sun-Ho Park)

2009년 고려대학교 정보통신공학 석사
2017년 아주대학교 의용공학 박사과정
2005년 ~ 현재 KL-Net 수석연구원
※관심분야 : 오픈소스 시스템(OSS), 스트리밍 데이터 분석 플랫폼, 클라우드 스택, 인공지능

김영길(Young-Kil Kim)

1978년 고려대학교 전자공학과 학사
1980년 한국과학기술원 석사
1984년 ENST(프랑스) 박사
1984년 ~ 현재 아주대학교 전자공학과 교수
※관심분야 : 임베디드 시스템, 초음파 의료기, Mobile 의료정보 시스템, RFID Platform