Journal of the KIMST, Vol. 25, No. 4, pp. 372-380, 2022
ISSN 1598-9127(Print) · 2636-0640(Online)
DOI https://doi.org/10.9766/KIMST.2022.25.4.372

Research Paper 정보·통신 부문

대용량 위성영상 처리를 위한 FAST 시스템 설계

이영림*,1) · 박완용1) · 박현춘1) · 신대식1)

1) 국방과학연구소 국방인공지능기술센터

FAST Design for Large-Scale Satellite Image Processing

Youngrim Lee*,1) · Wanyong Park1) · Hyunchun Park1) · Daesik Shin1)

1) *The Defense AI Technology Center, Agency for Defense Development, Korea*

(Received 7 March 2022 / Revised 30 June 2022 / Accepted 29 July 2022)

Abstract

This study proposes a distributed parallel processing system, called the Fast Analysis System for remote sensing daTa(FAST), for large-scale satellite image processing and analysis. FAST is a system that designs jobs in vertices and sequences, and distributes and processes them simultaneously. FAST manages data based on the Hadoop Distributed File System, controls entire jobs based on Apache Spark, and performs tasks in parallel in multiple slave nodes based on a docker container design. FAST enables the high-performance processing of progressively accumulated large-volume satellite images. Because the unit task is performed based on Docker, it is possible to reuse existing source codes for designing and implementing unit tasks. Additionally, the system is robust against software/hardware faults. To prove the capability of the proposed system, we performed an experiment to generate the original satellite images as ortho-images, which is a pre-processing step for all image analyses. In the experiment, when FAST was configured with eight slave nodes, it was found that the processing of a satellite image took less than 30 sec. Through these results, we proved the suitability and practical applicability of the FAST design.

Key Words : FAST, Distributed Parallel Processing(분산병렬처리), Large-Scale Image Management(대용량 영상관리), HDFS(하둡 분산 파일 시스템), Apache Spark(아파치 스파크), Docker(도커)

1. 서 론

과학기술 발달과 원격 탐사 적용 가능 분야가 증가함에 따라 지구관측위성의 수가 늘어났다. 이에 따라 하루에 촬영되는 위성 영상의 수가 기하급수적으로 늘어나고 있다[15,17]. 우리는 매일 획득된 영상으로부터 기상예보, 농작물 모니터링, 자연재해 모니터링, 테러리즘 모니터링 등 다양한 분야에서 매일 정보를 추출하여 활용하고 있다[19-21]. 이러한 활용이 가능하기 위

* Corresponding author, E-mail: yrlee@add.re.kr

해서는 시간당 처리 용량 크고, 무중단 운용 가능한 시스템이 요구된다.

기존 연구들에서는 대용량 영상 고속처리 및 저장 관리를 위하여 분산 시스템에서 원격탐사데이터를 처리하고자 시도하였다. Sharma 등 연구에서는 하둡 맵리듀스(Hadoop Mapreduce)[2,4]를 사용하여 Landsat 영상으로부터 NDVI 영상을 생성하는 시스템을 개발하였다[22]. Sharma 연구의 단점은 입력 영상 데이터셋을 하나의 시퀀스 파일로 변환해야 하는 전처리단계가 필요하다는 것이다. Almeer는 3개 영상처리 알고리즘을 하둡 맵리듀스 프레임워크 상에서 구현하고 영상 크기별 처리시간을 단독 PC의 처리시간과 비교하여 제시하였다[1]. Chen은 원격탐사데이터 처리를 위해 클라우드 컴퓨팅과 WPS 명세[16]를 결합한 프레임워크를 제시하였다[3]. Chen의 연구는 분산처리시스템 구조 설계보다는 클라이언트, WPS 서버, 클라우드 컴퓨팅 클러스터 간 인터페이스 설계에 초점이 맞춰져 있다. Sharma, Almeer, Chen 연구는 하둡 맵리듀스 기반으로 수행되었다. 하둡 맵리듀스는 대용량 데이터의 일괄처리를 위해 만들어진 만큼 반복 연산과 실시간 처리에 어려움을 가진다[13,25].

이를 보완하기 위해 아파치 스파크(Apache Spark)[24]를 활용한 연구들이 발표된 바 있다. Rathone는 스파크/얀(Spark/YARN) 환경에서 동작하는 스트림 기반 프로그래밍 모델을 제시하였다[18]. Rathone은 제시한 모델로 구현된 프로그램의 실행 시간이 맵리듀스로 구현된 프로그램의 실행시간보다 훨씬 빠름을 보이고, 제시한 프로그래밍 모델이 스파크/얀 구성에 가장 최적화되어 있음을 증명하였다. Huang은 GPU, 스파크, 하둡 에코시스템 기반에서 도시 교통 모니터링 동영상을 실시간 분석 처리하는 시스템을 설계하였다[7]. Huang은 스파크를 실시간 데이터 수신/저장, 단위 작업의 실행 환경으로 사용하고, 하둡 에코시스템은 분산병렬환경으로 사용하고, GPU는 빠른 동영상 병렬처리에 사용하였다.

대용량 원격 탐사 데이터 처리를 위해 GPU를 활용한 병렬처리 시스템을 설계한 연구도 있다. 한희정[6]은 정지궤도 해양위성 2호(Geostaionary Ocean Color Imager-II; GOCI-II)를 위한 지상시스템(GOCI-II Ground Segment; G2GS)을 제안하였다. G2GS는 위성자료 수신부터 정밀 보정까지 수행하는 시스템으로 자료의 고속처리를 위해 GPU를 이용한 병렬처리 프로그래밍 프레임 워크를 제공하고, lustre 분산 자료 저장 시스템으로 자료 저장 관리를 수행한다. 해당 연구는 G2GS에서 동작하는 고성능 자료 처리를 위해서 GOCI 프로그래밍 프레임 워크 기반으로 자료 처리 알고리즘 코드를 작성해야 한다. 이는 기존 코드의 재활용성과 코드 범용성이 떨어뜨린다.

원격 탐사 데이터를 처리하는 응용프로그램들은 오픈 소스 프로그램을 포함하여 무수히 존재하지만, 이를 활용하기는 쉽지 않다. 이러한 응용프로그램은 특정 실행 환경과 의존성을 가진 라이브러리가 필요하기 때문이다. 이를 해결하기 위해 기존의 연구들은 가상화 기술을 사용하였다. Knoth는 지리적 정보 기반 객체 분석을 위한 오픈소스 프로그램과 데이터를 도커 컨테이너[14]로 패키지화하여 편의성, 이식성, 재생산성을 높였다[10]. Huo는 특정 고성능 컴퓨터 환경에서만 동작하는 의료영상처리 및 저장 플랫폼을 도커 컨테이너와 가상 머신 형태로 구현하였다. 이를 통해 해당 플랫폼의 이식성 및 이동성을 높였다[9].

본 논문은 특정 환경에 따라 영상처리 알고리즘을 재구현해야 하는 단점을 극복하면서 원격 탐사 데이터의 분석을 위한 필수적인 특징을 가진 분산병렬처리 시스템을 제안한다. 본 논문은 제안하는 시스템을 FAST(Fast Analysis System of remote sensing daTa)라 명명한다. 우리가 제안하는 시스템 FAST는 마스터/슬레이브 구조의 분산병렬처리와 버텍스/시퀀스/스테이지 개념 설계를 통해 단위 시간당 처리량이 크다. FAST는 도커 기반으로 단위작업을 처리함으로써 기존 코드의 재활용성이 높고, 단위 작업 오류에 강인하고, 유연한 시스템 확장 및 유지보수가 가능하다. 또한 FAST는 HDFS를 기반으로 대용량 영상 자료를 관리함으로써 자료 저장 공간의 무중단 확장이 가능하다. 본 논문은 이러한 장점을 가진 FAST 시스템 설계를 제시하고, FAST의 활용가능성을 보이기 위해 원본 위성영상을 정사영상으로 생성하는 실험을 수행하였다.

2. FAST 설계

2.1 FAST 구조

FAST은 Fig. 1과 같이 FAST 엔진(FAST engine), FAST 슬레이브(FAST slave), 버텍스 컨테이너(Vertex Container)로 구성되고, 하둡의 HDFS 기반으로 데이터를 관리하도록 설계하였다.

우리는 마스터/슬레이브(Master/Slave) 구조의 분산병렬처리 시스템으로 FAST를 설계하였다. 마스터 역할

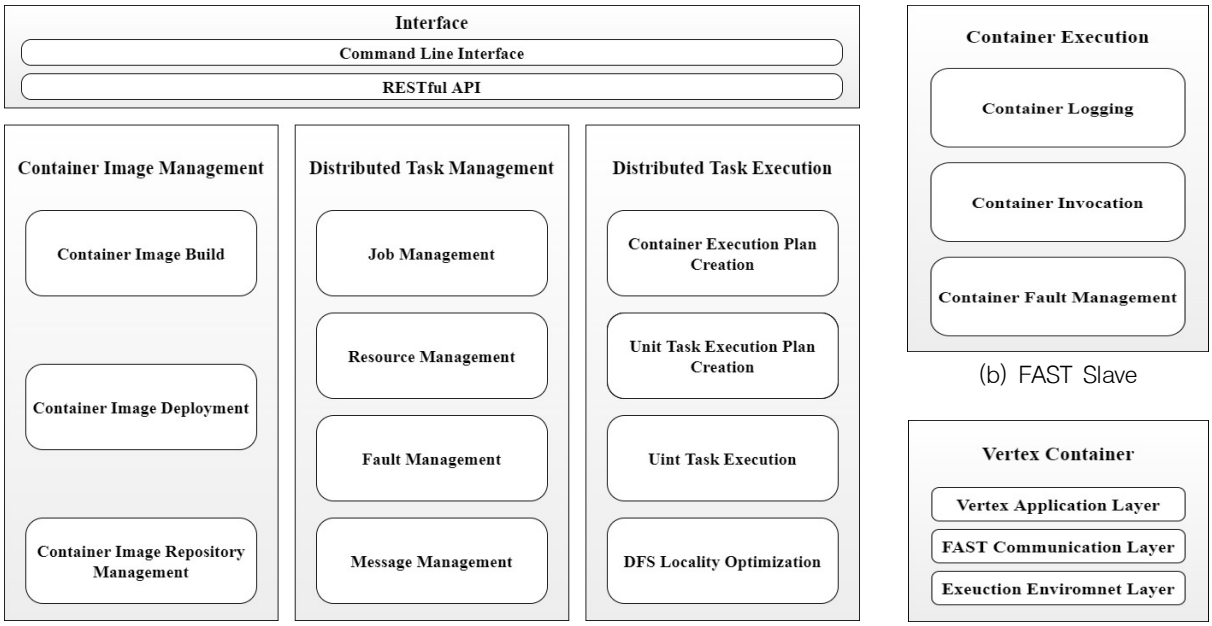

Fig. 1. FAST

은 FAST 엔진이 수행하고, 슬레이브 역할은 다수개의 FAST 슬레이브가 수행한다. FAST 엔진은 Spark 기반 응용프로그램으로써 주 장비(Master node)에 위치하여 전체 작업의 통제/관리/자원 관리를 담당한다. FAST 슬레이브는 부 장비(Slave node)마다 단일하게 동작하고, 단위작업을 수행하는 Docker 기반 다수개의 버텍스 컨테이너들을 실행 및 관리한다. 이러한 설계는 다수 장비의 자원을 효율적으로 활용하여 전체적인 성능을 향상한다.

FAST 엔진은 사용자/FAST 슬레이브와의 인터페이스를 담당하고 단위작업을 수행하는 버텍스 컨테이너 이미지들을 관리한다. 또한 전체적인 작업 관리 및 리소스 관리를 수행하고, 버텍스 컨테이너 실행 명령어를 생성하여, 최대한 버텍스 컨테이너에서 사용될 데이터가 위치한 FAST 슬레이브로 버텍스 컨테이너 실행 명령어를 전달한다. 이러한 설계는 데이터 로컬리티를 최대한 유지하도록 한다. 데이터 로컬리티는 데이터가 있는 곳에서 작업을 처리하는 개념이다. 단위작업이 데이터가 있는 곳에서 처리되면 데이터 입출력 시간이 네트워크를 통한 데이터 입출력 시간보다 줄어든다. 그 결과, 해당 작업의 처리시간이 빨라지고, FAST의 전체적인 성능을 향상한다.

FAST 슬레이브는 부 장비마다 단일하게 존재하며, 단위작업을 Docker 기반 버텍스 컨테이너로 실행, 로그 처리 및 장애 처리 기능을 가진다. 실질적인 단위작업을 수행하는 버텍스 컨테이너는 도커기반 컨테이너로 설계하였다. 버텍스 컨테이너(Vertex Container)는 실행환경 레이어(Execution Environment Layer), FAST 통신 레이어(FAST Communication Layer), 버텍스 응용프로그램 레이어(Vertex Application Layer)로 구성된다. 실행 환경 레이어는 도커 컨테이너 내에서 응용프로그램을 실행시킬 수 있는 환경이 설치된 레이어이다. 즉, 실행 환경 레이어는 경량화된 운영체제와 응용프로그램이 필요한 라이브러리와 응용프로그램이 위치한다. FAST 통신 레이어는 단위 작업의 컨테이너 내에 탑재되어 단위 작업 응용프로그램, FAST 슬레이브, FAST 엔진 간 인터페이스를 담당한다. 버텍스 응용 레이어는 실제 단위 작업을 수행하는 응용프로그램이 위치한다.

최소 작업 단위인 버텍스를 도커 기반 컨테이너로 실행 및 관리하는 설계는 크게 2가지 장점이 있다. 첫 번째는 기존 코드의 재활용성이다. 버텍스 컨테이너 내 실행 환경은 기존 일반적인 실행 환경과 동일하게 구성할 수 있다. 이를 통해 기존 코드들 재활용하여

버텍스를 생성하는 것이 가능하다. 이런 특징은 기존의 작업을 분산 병렬 처리 시스템으로 마이그레이션 해오는 비용을 줄일 수 있다. 두 번째는 소프트웨어/하드웨어 오류·고장(Fault)에 강인하다. FAST 시스템은 단위 작업을 도커 기반에서 수행한다. 가상화된 컨테이너 공간에서 단위 작업을 수행하기 때문에 단위 작업 시 오류가 발생하여도 컨테이너가 종료될 뿐이지 전체 시스템에는 영향이 없다. 또한 버텍스는 도커 컨테이너이기 때문에 단위 작업 응용프로그램 및 실행에 필요한 모든 환경을 자체적으로 가진다. 이러한 이유로 버텍스는 FAST 슬레이브 노드 어디에서나 동일하게 독립적으로 실행할 수 있다. 이러한 특징은 FAST 시스템이 클러스터 구성 노드의 추가 삭제를 통하여 하드웨어 고장에도 유연하고 즉각적으로 대처 가능하도록 한다.

2.2 데이터 관리

FAST은 대용량 수많은 위성영상을 포함한 필요 자료를 HDFS에 저장 관리한다. HDFS(Hadoop Distributed File System)는 저 사양 하드웨어에서 동작하는 분산 파일 시스템이다. HDFS는 대용량 파일을 저장하고 처리하기 위해서 개발된 아파치 하둡(Apache Hadoop) 프로젝트의 분산 파일 시스템이다. HDFS는 대용량 파일을 블럭 단위로 분할하여 여러 노드에 분산 중복 저장 관리한다.

HDFS를 통한 대용량 자료 관리 수행하는 설계는 두 가지 장점이 있다. 첫 번째는 좋은 응답성이다. FAST는 단위 작업이 동시에 동일한 블럭을 요청하더라도 해당 블럭이 다수 노드에 중복 분산 저장되어 있으므로 빠르게 해당 블럭을 각 단위 작업에 전달할 수 있다. 두 번째는 무중단 확장 가능성이다. FAST는 운영 중에도 노드를 추가하여 저장용량을 증설할 수 있다. 이는 지속적으로 데이터 축적이 필요한 원격탐사 자료를 관리하는 시스템에는 매우 중요한 장점이다.

2.3 작업 구성

FAST에서 동작하는 영상처리 응용프로그램은 단위 작업과 실행 순서로 구성된다. FAST에서는 단위 작업을 버텍스(Vertex)로, 단위 작업의 실행 순서를 시퀀스(Sequence)로, 실행되고 있는 단계를 스테이지(Stage)로 명명하였다(Fig. 2). FAST 상에서 동작하는 작업 전체 구성은 Fig. 2와 같이 기본적으로 파이프라인 형태를 가진다. 스테이지별 버텍스 구성은 크게 단일 구성, 다중 구성, 분기 구성으로 나뉘는데, 이는 스테이지에서 실행되는 버텍스 개수와 기능에 따라 달라진다. 단일 구성은 스테이지에 단일한 버텍스 1개가 실행되는 구성이다. 다중 구성은 스테이지에 해당 버텍스를 다수개 생성하여 동일하게 실행하는 구성이다. 이때 다수개의 버텍스들은 각기 다른 FAST 슬레이브 노드에서 동시에 실행된다. 분기 구성은 스테이지에 해당하는 버텍스의 특정 기능만을 실행하는 구성이다. 이러한 버텍스 구성은 스테이지별로 조합할 수 있다. 스테이지별로 버텍스 구성은 이전 스테이지에서 실행된 버텍스가 다음 스테이지의 버텍스로 전달하는 파라미터에 의해 결정된다. 즉 버텍스와 시퀀스를 작성하는 사용자가 작업의 전체 구성을 설계할 수 있다. Fig. 3은 다양한 작업 구성을 나타낸다. FAST은 다수개의 시퀀스를 동시에 실행 가능하다.

버텍스, 시퀀스, 스테이지 개념 설계는 작업 구성 용이성과 처리량을 높인다. 사용자는 작업을 버텍스, 시퀀스의 개념으로 작게 나누고, 스테이지 개념을 통해 병렬 처리를 쉽고 단순하게 설계할 수 있다. 이는 FAST 시스템의 구조 및 설계를 단순화하고, 사용자가 쉽고 용이하게 작업을 구성할 수 있게 한다. 다수개의 시퀀스를 동시 실행할 수 있도록 설계 구현함으로써 우리는 FAST 시스템 전체의 처리량을 높인다.

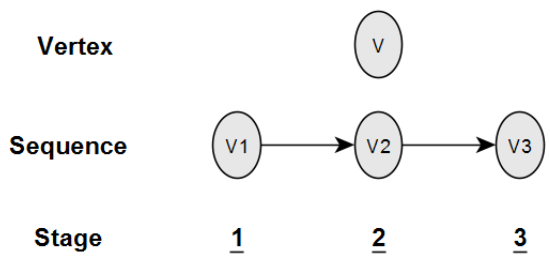

Fig. 2. Vertex, Sequence, Stage

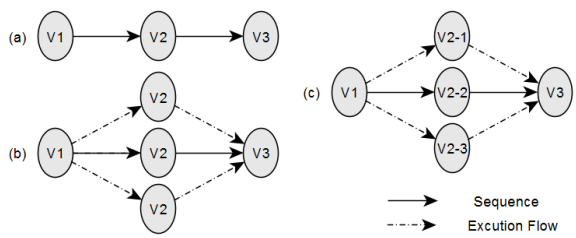

Fig. 3. Vertex Configurations: (a) Single, (b) Multiple, (c) Branch

3. 구 현

3.1 시스템 환경

FAST은 다음과 같은 필수 환경이 요구된다. FAST의 하드웨어적 환경은 주 장비(Master node) 1대와 부 장비(Slave node) 1대 이상이 필요하다. FAST 장비의 운영체제는 리눅스를 기반으로 한다.

주 장비에는 HDFS, 스파크, 얀(YARN[23]), 카프카(kafka[12]), 주키퍼(ZooKeeper[8])가 설치되며, 부 장비에는 HDFS, 스파크, 얀, 도커, 카프카가 설치된다. FAST를 위해 사용되는 오픈소스 프로그램들의 기능은 Table 1과 같다.

Table 1. Open source description

Name	Function
HDFS (Hadoop Distributed File System)	An open-source Java-based framework that enables distributed storage and management of large amounts of data
Apache Spark	An execution engine that is capable of analyzing and processing big data
Apache Kafka	A data stream management service in real time
ZooKeeper	coordination service for distributed application
YARN (Yet Another Resource Negotiator)	A task scheduling and cluster resource management
Docker	A container-based virtualization platform

FAST는 HDFS를 대용량 영상 데이터의 저장 및 관리를 위해 사용하고, 스파크를 분산처리 프레임워크로 활용한다. FAST는 카프카를 통제 메시지 전달 및 로그 관리를 위해 사용하고, 주키퍼와 얀은 분산처리 환경 관리를 위해 사용한다. 도커는 도커 컨테이너화된 단위 작업을 실행하기 위해 사용한다.

3.2 FAST 구현

FAST 엔진은 스파크의 응용프로그램으로 구현되어 주 장비에 상시 동작한다. FAST 슬레이브는 리눅스 응용프로그램으로 모든 부 장비에 적재된다. FAST 슬레이브는 버텍스의 실행 노드로 배정될 때 FAST 엔진에 의해 실행된다. 버텍스는 FAST 엔진에서 도커 컨테이너 이미지로 관리되고, FAST 슬레이브 노드 중 하나의 노드에서 실행된다.

우리는 FAST에서 버텍스의 실행 순서를 시퀀스로 관리할 수 있도록 구현하였다. 사용자는 작업을 버텍스와 시퀀스로 FAST 엔진에 등록하고, 등록된 시퀀스를 실행 요청하면, FAST은 등록된 실행 순서에 맞춰서 버텍스를 순차적으로 실행한다.

3.3 실행 흐름

버텍스 실행 흐름은 Fig. 4와 같다.

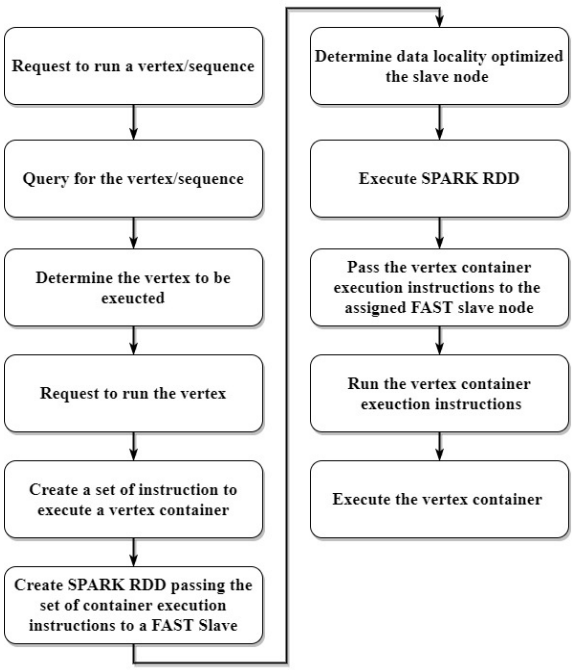

Fig. 4. Execution Flow

1) 사용자가 명령어 입력에서 버텍스나 시퀀스를 실행 요청한다.
2) FAST 엔진의 FAST 작업관리 모듈이 해당 버텍스/시퀀스를 조회한다.
3) FAST 작업 관리 모듈은 조회된 결과로 실행해야 하는 버텍스를 결정하고 분산 프로세스 실행 컴포넌트로 해당 버텍스 실행을 요청한다.

4) 분산 프로세스 실행 컴포넌트의 컨테이너 실행 계획 모듈은 해당 버텍스 컨테이너를 도커에서 실행하기 위한 명령어 집합을 생성한다.
5) 컨테이너 실행 계획 모듈은 컨테이너 실행 명령어 집합을 작업 실행 계획 수립 모듈에 전달한다.
6) 작업 실행 계획 수립 모듈은 FAST 슬레이브로 컨테이너 실행 명령어 집합을 전달하는 스파크 RDD (Resilient Distributed Dataset)를 생성한다.
7) 작업 실행 계획 수립 모듈은 생성된 스파크 RDD를 작업 분산처리 모듈로 전달한다.
8) 작업 분산처리 모듈은 해당 버텍스 컨테이너가 사용할 데이터의 위치를 DFS 로컬리티 최적화 모듈을 호출하여 조회한다.
9) 작업 분산처리 모듈은 조회된 노드에 해당 버텍스 컨테이너가 실행되도록 스파크 RDD를 실행한다.
10) 스파크는 지정된 노드의 FAST 슬레이브에 해당 버텍스 컨테이너 실행 명령어 집합을 전달하고, FAST 슬레이브는 전달받은 명령어 집합을 실행하여 버텍스 컨테이너를 도커 상에 구동시킨다.

버텍스 컨테이너에서 버텍스 처리가 완료되면, 버텍스는 RESTful API를 통해 FAST 작업 관리 모듈로 상태 정보와 다음 버텍스에 전달할 파라미터를 보낸다. FAST 작업관리 모듈이 파라미터를 전달받는 횟수와 파라미터 내용에 따라 해당 스테이지의 버텍스 구성이 달라진다. FAST 작업관리 모듈은 파라미터를 전달받을 때마다 다음 스테이지의 버텍스를 실행 요청한다. 이때 파라미터의 내용도 같이 전달한다. 시퀀스의 모든 버텍스 처리가 완료되면 FAST 작업관리 모듈은 시퀀스 상태 정보에 'DONE'을 기록하고 작업을 종료한다. 버텍스 컨테이너에서 버텍스 처리 중에 오류가 발생하면, 시퀀스 상태 정보에 에러 코드(ERROR)를 기록한다.

4. 실 험

우리는 대용량 위성영상 처리 속도를 확인하는 실험을 수행하였다. 우리는 실험을 통해 FAST가 대용량 위성영상 처리 및 분석하는 시스템 설계 목적에 적합함을 보이고자 한다.

본 실험은 FAST의 노드 수 변화에 따른 위성영상 처리속도를 측정하는 실험이다. 대상 위성영상 처리는 모든 위성 영상 분석의 필수적 과정인 원본 위성영상을 NITF(National Imagery Transmission Format)포맷의 정사영상으로 생성하는 것이다. 실험영상은 다양한 크기/센서종류/해상도를 가진, 최근에 가장 많이 활용되는 것으로 생각되는 종류의 위성영상 100장으로 구성하였다. 사용된 위성영상의 특징은 Table 4와 같다. 실험영상은 주로 TIF 파일로 HDFS에 저장되어 있고, 전체 총 용량은 63.25 GB이다. 사용된 하드웨어는 Table 3과 같고, 노드간의 네트워크 속도는 1 Gbps이다. 우리는 정사영상 생성작업을 버텍스 6개, 시퀀스 1개로 구현하였고, 버텍스 각각의 기능과 시퀀스 구성은 Fig. 5, 6과 같다. 우리는 FAST를 슬레이브 노드의 수 1, 3, 5, 7, 9 개로 구성하고, 구성별 5번의 반복 수행하여 시간을 측정하였고, 측정된 시간의 평균을 실험결과로 산출하였다.

- V1 : NITF metadata generation
- V2: NITF image generation of input image
- V3: Converting NITF image to TIFF
- V4: Splitter for ortho-rectified image generation
- V5: Patial ortho-rectified image generation
- V6: Mapper for generated ortho-rectifed image aggregation and making NITF image of ortho-rectified image

Fig. 5. The function of vertexes

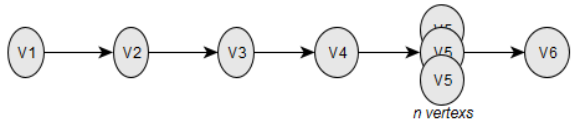

Fig. 6. Vertex Configuration

Table 2. Experimental results

Num. of Nodes	Total Processing Time(min)	Processing Time per 1 images(sec)
1	300.22	180.13
3	97.82	58.69
5	72.42	43.45
7	58.26	34.95
9	48.53	29.11

Table 3. H/W Specification

Master	Slave	
- Model : HP-DL360 G10 - CPU: Xeon-S 4114 2.2GHz 2EA - MEM :128GB - Disk : SAS 10K SFF SC DS HDD 7.2TB (1.2TB x 6EA) - NIC : 1G 4port	- Model : HP-DL360 G10 - CPU : Xeon-S 4114 2.2GHz 2EA - MEM :128GB - Disk : SAS 10K SFF SC DS HDD 24TB (6TB x 4EA) - NIC : 1G 4port	- Model : HP-DL360 G10 - CPU: Xeon-S 4114 2.2GHz 2EA - MEM :128GB - Disk : SAS 10K SFF SC DS HDD 7.2TB (1.2TB x 6EA) - NIC : 1G 4port
1EA	7EA	2EA

Table 4. The characteristics of test image

Satellite Name	Sensor	Band	Num. of Images	Ground Sample Distance(m)	File Size (average, MB)
GeoEye-1	EO	Panchromatic	43	0.4 / 0.5	1266 / 831
GeoEye-1	EO	Multispectral	41	1.6 / 2	318 / 210
KOMPSAT-2	EO	Multispectral	4	4	44
WorldView-1	EO	Panchromatic	3	0.5	1449
WorldView-3	EO	Panchromatic	2	0.3	2250
WorldView-3	EO	Multispectral	2	1.3	563
KOMPSAT-3	EO	Multispectral	1	0.7	129
Pleiades	EO	Multispectral	1	0.7	403
KOMPSAT-5	SAR	X	1	1.2	83
TerraSAR	SAR	X	1	3.3	1843
Airborne	μCASI	Hyperspectral VNIR	1	0.5	5836
100 images					

실험 결과는 Fig. 7, 8, Table 2와 같다. 본 실험에서 FAST의 슬레이브 노드 수가 증가할수록 FAST의 전체 처리 시간이 줄어드는 경향성을 보인다. 슬레이브 노드 수가 1개인 구성과 9개인 구성을 비교했을 때, 위성영상 1장당 처리시간이 6배 빨라졌다. 처리시간 변화량은 1개인 구성과 3개인 구성 사이에 큰 변화가 있고, 그 이후 구성에서는 일정한 정도의 처리시간이 감소함을 보인다. 이는 정사영상 생성 처리가 영상 읽기/쓰기를 동반하는 처리이기 때문에 네트워크를 통한 영상 입출력에서 발생하는 줄일 수 없는 오버헤드가 원인으로 판단된다.

9대 슬레이브 노드 구성 시 정사영상 생성시간이 위성영상 1장당 29.11초 소요된다. 이러한 처리 속도는 위성영상 1장의 지상 촬영영역을 15 km × 15 km (GEOEYE-1 기준)로 가정했을 때, FAST가 우리나라 전체면적(100401.285 km^2[11])을 촬영한 위성영상 446장을 3.6시간 안에 처리 가능함을 의미한다. 이는 FAST 시스템이 대용량 위성영상 처리 및 분석하는 시스템 목적에 맞는 처리 속도와 활용 가능성을 가졌음을 보여준다.

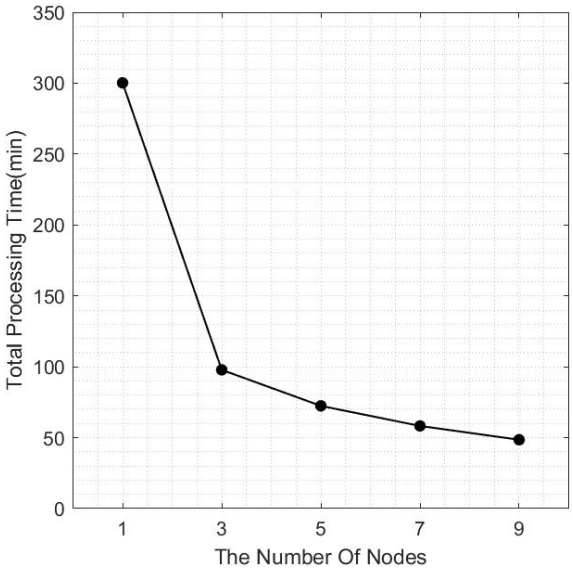

Fig. 7. Total Processing Time

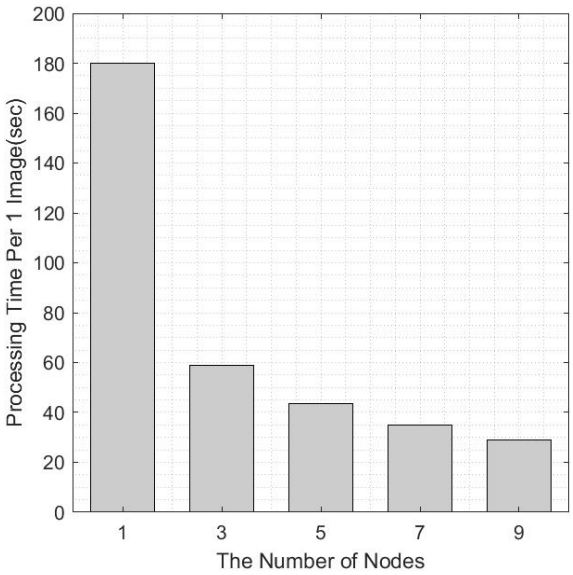

Fig. 8. Processing Time per 1 Image

5. 결 론

우리는 대용량 위성 영상처리에 적합한 분산병렬처리 시스템을 설계 및 구현하였다. 우리는 제안한 분산병렬처리 시스템의 성능을 확인하기 위하여 FAST 슬레이브 노드 수 변화에 따른 원본 위성영상 100장 63.25 GB를 정사영상으로 생성하는 시간을 측정한 실험을 수행하였다. 본 실험에서 위성영상분석의 꼭 필요한 처리 과정의 성능을 측정한 것은 기존의 연구들과는 확실한 차이를 나타낸다. 실험 결과로 위성영상 1장당 29.11초 소요됨을 보임으로써 제안하는 시스템이 대용량 위성영상 처리 및 분석 목적에 적합함과 충분한 활용 가능성을 제시하였다.

향후 연구로 우리는 위성영상 자료에 특화된 데이터 구조 및 HDFS 구성과 FAST의 최적화된 응용프로그램을 연구할 예정이다. 이를 통해 FAST의 처리 성능을 높이고, FAST 시스템의 활용성을 높이고자 한다.

References

[1] Almeer MH, Cloud Hadoop Map Reduce for Remote Sensing Image Analysis, Journal of Emerging Trends in Computing and Information Sciences, Vol. 3, No. 4, pp. 637-644, 2012.

[2] The Apache Software Foundation, “Apache Hadoop,” Apache Hadoop, 2006, https://hadoop.apache.org/, accessed 22 Apr 2022.

[3] Chen Z, Chen N and Yang C, Di L, Cloud Computing Enabled Web Processing Service for Earth Observation Data Processing, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, Vol. 5, No. 6, pp. 1637-1649, 2012.

[4] Dean J and Ghemawat S, MapReduce: Simplified Data Processing on Large Clusters, Communications of the ACM, Vol. 51, No. 1, pp. 107-113, 2008.

[5] Ghemawat S, Gobioff H and Leung ST, The Google File System, SIGOPS Operating Systems Review, Vol. 37, No. 5, pp. 29-43, 2003.

[6] Hee-Jeong H., Hyun Y., Jae-Moo H. and Young-Je P., “Systemic Design and Development of the Second Geostationary Ocean Color Ground Segment,” KIISE Transactions on Computing Practices, Vol. 25, No. 10, pp. 477-484, 2019.

[7] Huang W, Meng L, Zhang D and Zhang W, In-Memory Parallel Processing of Massive Remotely Sensed Data Using an Apache Spark on Hadoop

YARN Model, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, Vol. 10, No. 1, pp. 3-19, 2016.

[8] Hunt P, Konar M, Junqueira FP and Reed B, ZooKeeper: Wait-Free Coordination for Internet-Scale Systems, USENIX Annual Technical Conference, pp. 145-158, 2020.

[9] Huo Y, Blaber J, Damon SM, Boyd BD, Bao S, Parvathaneni P, Noguera CB, Chaganti S, Nath V, Greer JM, Lyu I, French WR, Newton AT, Rogers BP and Landman BA, Towards Portable Large-Scale Image Processing with High-Performance Computing, Journal of Digital Imaging, Vol. 31, No. 3, pp. 304-314, 2018.

[10] Knoth C and Nüst D, Reproducibility and Practical Adoption of Geobia with Open-Source Software in Docker Containers, Remote Sensing, Vol. 9, No. 3, p. 290, 2017.

[11] Statistics Korea, "The land area(G20)," Korean Statistical Information Service, 30 Sep 2021, https://kosis.kr/statHtml/statHtml.do?orgId=101&tblId=DT_2KAA101_G20&conn_path=I2, accessed 21 July 2022.

[12] Kreps J, Narkhede N and Rao J, Kafka: A Distributed Messaging System for Log Processing, Proceedings of the NetDB, Vol. 11, pp. 1-7, 2011.

[13] Liu X, Iftikhar N and Xie X, Survey of Real-Time Processing Systems for Big Data, Proceedings of the 18th International Database Engineering & Applications Symposium, pp. 356-361, 2014.

[14] Merkel D, Docker: Lightweight Linux Containers for Consistent Development and Deployment, Linux Journal, LLC 239, Article 2, 2014.

[15] Nibedita Mohanta, "How Many Satellites are Orbiting the Earch in 2021?," Geospatial World, 28 May 2021, https://www.geospatialworld.net/blogs/how-many-satellites-are-orbiting-the-earth-in-2021/, accessed 30 Jun 2022.

[16] OGC, OpenGIS Web Processing Service Specification (Version 1.0.0) OGC Standard No. 05-007r7, OGC, 2007.

[17] Planet Labs PBC, "Planet," Planet, http://planet.com, accessed 22 Apr 2022

[18] Rathore MM, Son H, Ahmad A, Paul A and Jeon G, Real-Time Big Data Stream Processing Using GPU with Spark Over Hadoop Ecosystem, International Journal of Parallel Programming, Vol. 46, No. 3, pp. 630-646, 2018.

[19] S. N. K. B. Amit, S. Shiraishi, T. Inoshita and Y. Aoki, "Analysis of Satellite Images for Disaster Detection," 2016 IEEE International Geoscience and Remote Sensing Symposium(IGARSS), pp. 5189-5192, 2016, doi: 10.1109/IGARSS.2016.7730352.

[20] S. P. R and R. U, "A Survey on Agriculture Monitoring with Satellite and Its Benefits," 2022 8th International Conference on Advanced Computing and Communication Systems(ICACCS), pp. 1854-1858, 2022, doi: 10.1109/ICACCS54159.2022.9785323.

[21] SERGEY S., TAMARA D., ANNA K. and EIDELSTEINC L., "Engaging the Public to Fight the Consequences of Terrorism and Disasters," IOS Press, 120, pp. 91-103, 2015.

[22] Sharma T, Shokeen V and Mathur S, Distributed Processing of Satellite Images on Hadoop to Generate Normalized Difference Vegetation Index Images, Proceedings of 2017 International Conference on Computing, Communication, pp. 1-5, 2017.

[23] Vavilapalli VK, et. al., Apache Hadoop Yarn: Yet Another Resource Negotiator, Proceedings of the 4th Annual Symposium on Cloud Computing(SOCC '13), Article 5, pp. 1-16, 2013.

[24] Zaharia M, et. al., Spark: Cluster Computing with Working Sets. Proceedings of 2nd USENIX Conf. Hot Topics Cloud Comput., pp. 10-10, 2010.

[25] Zhang Y, Gao Q, Gao L and Wang C, iMapReduce: A Distributed Computing Framework for Iterative Computation, Journal of Grid Computing, Vol. 10, No. 1, pp. 47-68, 2012.