대용량 데이터의 분산 처리를 위한 클라우드 컴퓨팅 환경 최적화 및 성능평가

Optimization and Performance Analysis of Cloud Computing Platform for Distributed Processing of Big Data

홍 승 태[*] 신 영 성[**] 장 재 우[***]
Seung Tae Hong Young Sung Shin Jae Woo Chang

요 약 최근 IT 분야에서 인터넷을 기반으로 IT 자원들을 서비스 형태로 제공하는 클라우드 컴퓨팅에 대한 관심이 증대되고 있으며, 이에 따라 대규모 데이터를 수많은 서버들에 분산 저장하고 관리하기 위한 분산 데이터 처리 기법에 대한 연구가 활발히 진행되고 있다. 한편 GIS 기술의 성장과 더불어 급격히 증가하고 있는 공간 데이터를 효율적으로 활용하기 위해서는, 클라우드 컴퓨팅을 이용한 대용량 공간 데이터의 분산 처리가 필수적이다. 이를 위해 본 논문에서는 대표적인 분산 데이터 처리 기법에 대해 살펴보고, 분산 데이터 처리 기법 성능 개선을 위한 최적화 요구사항을 분석한다. 마지막으로 Hadoop 기반 클러스터를 구축하고 이를 통해서 분산 데이터 처리 기법의 성능 최적화에 대한 성능평가를 수행한다.

키워드 : 클라우드 컴퓨팅, 대용량 데이터, 분산 처리

Abstract Recently, interest in cloud computing which provides IT resources as service form in IT field is increasing. As a result, much research has been done on the distributed data processing that store and manage a large amount of data in many servers. Meanwhile, in order to effectively utilize the spatial data which is rapidly increasing day by day with the growth of GIS technology, distributed processing of spatial data using cloud computing is essential. Therefore, in this paper, we review the representative distributed data processing techniques and we analyze the optimization requirements for performance improvement of the distributed processing techniques for a large amount of data. In addition, we uses the Hadoop and we evaluate the performance of the distributed data processing techniques for their optimization requirements.

Keywords : Cloud Computing, Big Data, Distributed Processing

1. 서 론

최근 IT 분야에서 클라우드 컴퓨팅에 대한 연구가 활발히 진행되고 있다. 클라우드 컴퓨팅이란 인터넷을 기반으로 하여 IT 자원들을 서비스 형태로 제공하는 컴퓨팅을 의미한다[13]. 클라우드 컴퓨팅은 2006년 Google의 크리스토프 비시글리아가 CEO인 에릭 슈미츠에게 처음으로 제안하였으며, 이후 경제 전문지 및 대표적인 글로벌 기업의 CEO들이 잇달아 클라우드 컴퓨팅을 차기 주력 비즈니스 아이템으로 지목하면서 클라우드 컴퓨팅은 차세대 인터넷 비즈니스의 핵심 분야로 부각되고 있다[14]. 국외에서는 이미 많은 연구와 개발이 진행 중이며, 대표적인 예로 Amazon Elastic Compute Cloud, IBM Blue Cloud, 그리고 Google App Engine 등이 있다[18]. 국내에서도 IBM, HP, 마이크로소프트 등 글로벌 기업의 한국지사가 대기업 계열 IT 서비스 업체와 함

† 본 논문은 중소기업청에서 지원하는 2011년도 산학연공동기술개발사업(00046773-1)의 연구수행으로 인한 결과물임을 밝힙니다.

* 전북대학교 전자정보공학부 박사과정 sthong@dblab.chonbuk.ac.kr
** 전북대학교 전자정보공학부 석사과정 shinys@dblab.chonbuk.ac.kr
*** 전북대학교 전자정보공학부 교수 jwchang@chonbuk.ac.kr(교신저자)

께 국내 클라우드 컴퓨팅 보급을 구체화 하고 있으며, KT, SK 텔레콤 등 통신 사업자도 사업 준비를 서두르면서 클라우드 컴퓨팅에 대한 관심을 높이고 있다[15].

한편 최근 GPS(Global Positioning System), LIDAR (Laser Imaging Detection And Ranging) 등과 같은 GIS 기술의 획기적인 발전으로 인하여 공간 데이터가 급격히 증가하고 있다[16, 17]. 이는 복잡한 지형지물의 2D/3D 형태정보 및 다양한 속성정보를 포함하는 공간 데이터의 특징으로 인하여, 대부분의 공간 데이터의 경우 데이터의 볼륨이 매우 크기 때문이다. 따라서 이러한 대용량 공간 데이터를 효율적으로 활용하기 위해서는, 클라우드 컴퓨팅을 활용한 대용량 공간 데이터의 분산 처리에 대한 연구가 필수적이다. 클라우드 컴퓨팅을 제공하기 위해서는 막대한 물리적인 IT 인프라 구축뿐만 아니라, 데이터를 효율적으로 활용할 수 있는 분산 데이터 처리 기법들이 요구된다. 이에 따라 데이터의 저장과 관리를 위한 분산 데이터 저장 기법과 데이터의 병렬 처리를 위한 분산 병렬 처리 기법에 대한 연구가 활발히 진행되고 있다. 그러나 현재 실제 시스템에서 분산 데이터 처리 기법의 성능을 최적화하기 위한 주요 시스템 환경변수 및 분산 기법에 대한 연구는 미흡한 실정이다. 이에 따라 본 논문에서는 대표적인 분산 데이터저장 기법과 분산 병렬 처리 기법에 대해 살펴보고, 성능 개선을 위한 주요 시스템 환경변수 및 분산 기법에 대하여 분석한다. 아울러 분산 파일 시스템과 분산 병렬 처리 기법을 이용하여 클라우드 컴퓨팅을 위한 클러스터를 구축하고, 다양한 실험을 통하여 분산 데이터 처리 기법의 성능 최적화에 대한 성능평가를 수행한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 대표적인 분산 데이터 처리 기법인 Apache의 HDFS (Hadoop Distributed File System)[5]와 Hadoop MapReudce[3] 및 Berkeley 대학의 MapReduce Online[7]에 대하여 살펴보고, 3장에서는 Hadoop의 주요 시스템 환경변수와 함께 대표적인 분산 기법인 Sampler에 대하여 분석한다. 4장에서는 Hadoop을 이용한 클러스터 구축 방법을 설명하고, 아울러 구축한 클러스터를 통하여 주요 시스템 환경변수 및 Sampler 기법에 따른 분산 데이터 처리 기법의 성능평가를 수행한다. 마지막으로 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련연구

클라우드 컴퓨팅을 위한 대표적인 분산 데이터 저장 기법과 분산 병렬 처리 기법은 Google의 GFS (Google File System)[10]와 MapReduce[9]가 있으며, 이를 바탕으로 Apache에서는 오픈 소스 기반의 Hadoop[1] 프로젝트를 진행 중이다. Hadoop 프로젝트는 대용량 데이터의 처리를 위한 대표적인 오픈 소스 기반 분산 컴퓨팅 프레임워크이다. Hadoop은 크게 분산 파일 시스템인 HDFS와 분산 병렬 처리 기법인 MapReduce로 구성되며, 추가적으로 분산 데이터베이스인 HBase 등을 사용자가 선택적으로 구성하여 사용할 수 있다. 본 장에서는 Hadoop 기반 분산 컴퓨팅 프레임워크의 기본 파일 시스템인 HDFS에 대해 살펴본 후, Hadoop의 MapReduce와 함께 기존 MapReduce를 개선한 Berkeley 대학의 MapReduce Online에 대하여 살펴본다. 아울러, 대표적인 분산 병렬 처리 기법인 MapReduce와 MapReduce Online에 대한 비교 분석을 통하여 두 분산 병렬 처리 기법의 특징에 대해 살펴본다.

2.1 Hadoop Distributed File System(HDFS)

분산 파일 시스템은 대용량 데이터를 저장하고 관리하기 위해 많은 서버들에 데이터를 나누어 저장하고 관리하는 파일 시스템이다. 클라우드 컴퓨팅에서의 분산 파일 시스템은 단순히 데이터를 저장하고 관리하는 것뿐만이 아니라, 데이터를 저장하고 있는 하드웨어의 장애에 유연하게 대처하고 서비스가 요구하는 충분한 성능도 보장해야 한다. 최근 클라우드 컴퓨팅이 부각되면서 기존의 분산 파일 시스템 기술들이 다양한 형태로 활용되고 있으나, 클라우드 컴퓨팅에서 대용량 데이터를 위한 분산 데이터 저장 기법으로써 활용되고 있는 분산 파일 시스템은 그리 많지 않다. 그 중 대표적인 분산 파일 시스템인 HDFS는 오픈 소스 소프트웨어 개발 프로젝트인 Hadoop에서 분산 컴퓨팅 프레임워크를 지원할 목적으로 개발된 분산 파일 시스템이다. HDFS는 현재 Amazon, IBM, Yahoo 등과 같은 글로벌 IT 기업들의 클라우드 컴퓨팅 플랫폼의 기반이 되는 분산 파일 시스템으로 가장 널리 활용 되고 있다. HDFS는 GFS를 기반으로 개발된 분산 파일 시스템으로써, 플랫폼간의 이식성을 보장하기 위해 자바를 사용하여 구현되었다.

HDFS는 그림 1과 같이 GFS와 동일한 구조와 기능을 제공한다. HDFS는 네임스페이스를 관리하고 클라이언트의 파일에 대한 접근을 통제하는 단일 네임노드(NameNode)와 데이터 저장소를 관리하는 수많은 데이터노드(DataNode)들로 구성된다. HDFS에서 하나의 파일은 GFS의 청크와 동일한 개념의 블록들로 이루어지며 데이터노드들에 분산되어 저장된다. 마지막 블록을 제외하고는 한 파일을 구성하는 모든 블록들은 동일한 크기를 가지며 데이터노드의 장애에 대처하기 위해 블록들의 복제본이 여러 데이터노드들에 중복되어 저장된다. 파일마다 블록의 크기와 복제본의 수를 조절할 수 있으며 복제본의 수는 파일을 생성할 때 지정할 수도 있고 나중에 변경하는 것도 가능하다.

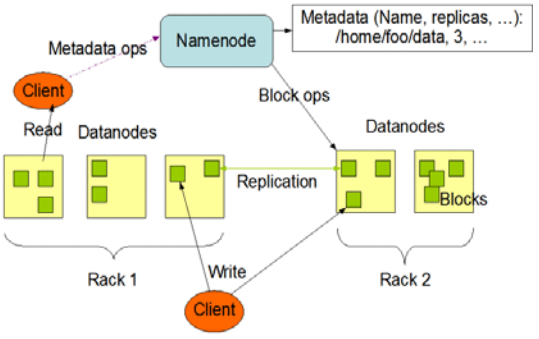


그림 1. HDFS 구조

2.2 Hadoop MapReduce

기존 분산 병렬 처리 기법은 고성능의 컴퓨팅이 요구되는 분야의 응용들을 빠르게 처리하기 위한 HPC(High Performance Computing)에 초점을 맞추고 있어 데이터량이 많은 경우엔 적용에 문제가 있다. 따라서 대규모 분산 병렬 처리를 위해서는 데이터 증가에 따른 확장성을 제공할 수 있어야 하며, 노드간 데이터 이동에 따른 네트워크 트래픽을 최소화할 수 있도록 업무 분산이 필요하다. MapReduce는 이와 같은 요구사항을 고려하여 대용량 데이터 집합을 처리하기 위해 만들어진 기법이다. 이를 위하여 Google에서는 2004년 MapReduce를 발표 하였으며, 이를 기반으로 Hadoop에서는 오픈 소스 기반의 분산 병렬 처리 기법인 Hadoop MapReduce를 개발하였다. Hadoop의 MapReudce는 Google의 MapReduce와 동일한 구조와 기능을 제공하며, 오픈 소스 기반의 분산 병렬 처리 기법으로써 현재 대용량 데이터의 처리를 위한 대표적인 기술로 각광받고 있다. Hadoop MapReduce는 현재 아마존, 야후 등 주요 글로벌 기업에서 주요 프로젝트로 적극 지원하고 있으며, 야후의 클라우드 컴퓨팅 테스트베드에도 탑재되어 활용되고 있다[8]. 아울러, Hadoop MapReduce를 기반으로 대용량 텍스트 정보 검색 시스템인 Lucene[2], 야후의 MapReduce 기반 데이터플로 언어 처리 기술인 PigLatin[12]과 같은 다양한 연구가 진행됨으로써, 현재 Hadoop MapReduce는 대용량 데이터의 분산 병렬 처리 기법의 표준으로써 인정받고 있다.

MapReduce는 (key, value) 기반의 데이터를 분산 처리하는 모델로, 그림 2와 같이 입력 데이터 소스를 기반으로 Map 태스크를 수행하여 중간 결과를 생성하고 이를 입력으로 Reduce 태스크를 수행하여 최종 결과를 산출하는 2단계로 구성된다. 이때 개발자는 Map 함수와 Reduce 함수를 정의한다. Map 함수는 입력된 (key, value) 쌍들을 처리하여 중간값 집합을 생성한다. Reduce 함수는 중간 key값을 가지는 모든 중간값들을 통합하여 최종 출력값으로 저장한다.

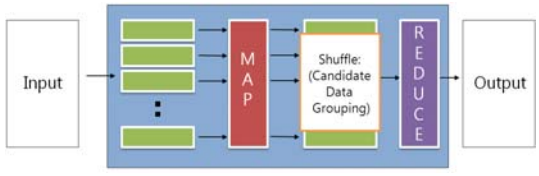
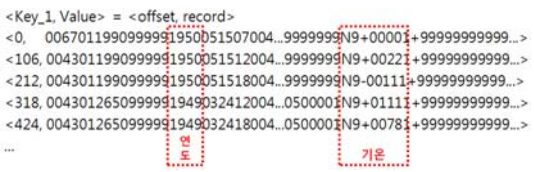


그림 2. MapReduce 실행 모델

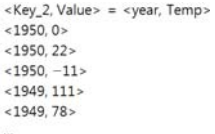
한편, MapReduce는 로그 분석 또는 데이터 필터링 등과 같은 데이터 분석 등의 용도로 주로 사용되고 있으며, 특히 단일 컴퓨터 환경으로는 일정 이상의 성능 향상이 어려운 대용량 데이터에 대한 분산 처리의 표준 기술로써 인정받고 있다. MapReduce를 이용한 대용량 데이터 분석의 대표적인 예로 미국의 NCDC(National Climatic Data Center)의 수천만 개의 기후 데이터에 대한 분산 병렬 처리가 있다. 그림 3은 데이터 파일로부터 각 연도별로 가장 높은 기온을 측정하는 MapReduce 수행 과정의 예를 나타낸다. 그림 3.(a)는 전체 수행 과정을 나타내며, 각 세부 수행 과정은 다음과 같다. 첫째, 하나의 레코드가 라인 단위로 저장된 입력 데이터 파일에서 (offset,

record)쌍을 추출한다(그림 3.(b)). 둘째, 각 레코드에서 연도별 기온을 추출하여 (key, value)쌍을 생성한다(그림 3.(c)). 셋째, 두 번째 Map 과정에서 결과가 많을 경우, Reduce 과정에서 병합시 처리 비용을 감소시키기 위해 각 연도별로 데이터를 그룹화한다(그림 3.(d)). 넷째, 모든 데이터노드로부터 후보집합을 병합하여 최종 결과를 반환한다(그림 3.(e)).

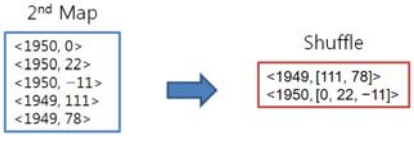
(a) 전체 수행 과정



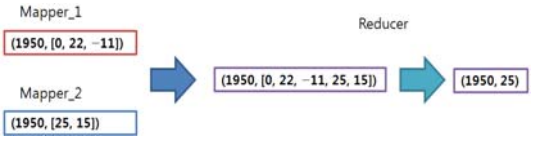
(b) 1st Map 과정



(c) 2st Map 과정



(d) Shuffle 과정



(e) Reduce 과정

그림 3. MapReduce 수행 예제

2.3 MapReduce Online

MapReduce Online은 실시간 분산 병렬 처리를 지원하기 위해 Berkeley 대학에서 개발한 파이프라인(pipeline) 기반 분산 병렬 처리 기법이다. MapReduce Online은 기본적으로 MapReduce와 동일한 프로그래밍 모델을 제공하며, 파이프라인을 통하여 연산 결과를 실시간적으로 전송함으로써 전체 데이터 처리 시간을 단축시킨다. 아울러 파이프라인을 통한 실시간 데이터 집계와 같은 연속 질의(continuous query)를 지원함으로써 실시간 이벤트 모니터링 및 스트림 처리를 제공한다.

MapReduce Online의 동작 과정은 다음과 같다. 첫째, 네임노드는 입력 파일들을 특정 크기로 분할하고, 분할된 M개의 조각을 클러스터의 Mapper에게 할당한다. 둘째, 각 Mapper는 할당 받은 Map 태스크에 필요한 조각들을 로드하여, Map 함수를 수행하고 중간 결과값을 저장한다. 셋째, Mapper에 저장된 중간 결과값이 일정 시간 간격으로 파이프라인을 통하여 Reducer에 전송되며, Reducer는 전송된 중간 결과값을 로드하여 Reduce 함수를 수행하고 사용자에게 중간 결과값을 반환한다. 마지막으로 모든 Map과 Reduce 태스크가 완료되면 네임노드는 최종 결과를 사용자 프로그램에 전송한다.

그림 4는 데이터 파일로부터 각 월별로 가장 높은 기온을 측정하는 MapReduce Online 수행 과정의 예를 나타낸다. 그림에서 Mapper는 월별 기온을 추출하여 (key, value)쌍을 생성하고 이를 파이프라인을 통하여 Reducer에게 지속적으로 전송한다. Reducer는 전송된 (key, value)쌍에 대하여 병합을 수행하여 임시적인 중간 결과값을 사용자에게 반환한다. 최종적으로 Mapper로부터 모든 (key, value)쌍이 Reducer로 전송될 경우 최종 결과를 사용자에게 반환한다.

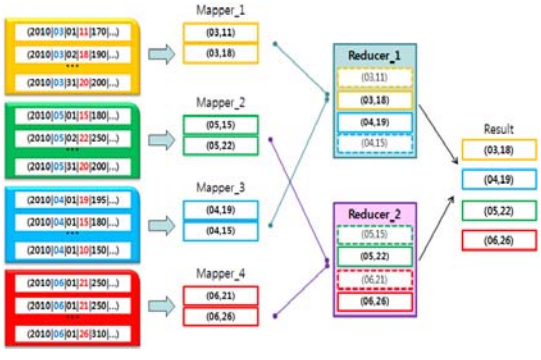


그림 4. MapReduce Online 수행 예제

2.4 MapReduce와 MapReduce Online 비교 분석

그림 5는 MapReduce의 데이터 연산 결과 전송 과정을 나타낸다. 우선 네임노드는 각 태스크들의 스케쥴을 조정하며, 각 Mapper에 입력 파일들을 특정 크기로 분할하여 할당한다. Mapper는 Map 함수를 수행하고 중간 결과값을 저장하고, Reducer들은 Mapper에서 중간 결과값을 로드하여 Reduce 함수를 수행한다.

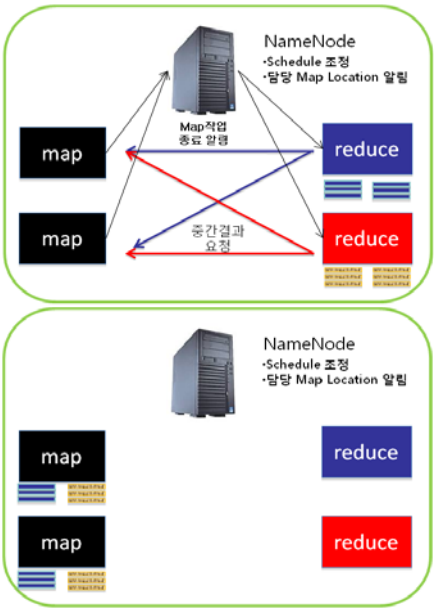


그림 5. MapReduce에서의 데이터 연산 결과 전송

그림 6은 MapReduce Online의 데이터 연산 결과 전송 과정을 나타낸다. MapReduce와 동일하게 네임노드는 각 태스크들의 스케쥴을 조정하며, 각 Mapper에 입력 파일들을 특정 크기로 분할하여 할당한다. 아울러, Mapper와 Reducer사이에 파이프라인을 설정함으로써 연산 결과를 실시간적으로 전송함으로써 실시간 데이터 집계 처리를 지원한다. MapReduce Online에서는 Mapper에 저장된 중간 결과값이 일정 시간 간격으로 파이프라인을 통하여 Reducer에게 전송하며, Reducer는 전송된 중간 결과값을 로드하여 Reduce 함수를 수행하고 사용자에게 중간 결과값을 반환한다. 이를 통해 사용자는 MapReduce에 비해 좀 더 빠르게 결과값을 확인할 수 있을 뿐만 아니라, 최종 결과를 확인하기 이전에 일정 비율의 중간 결과를 미리 확인할 수 있다. 이와 같이 MapReduce Online은 파이프라인을 통하여 연산 결과를 실시간적으로 전송함으로써 MapReduce에 비해 전체 데이터 처리 시간을 단축시킬 뿐만 아니라, 실시간 데이터 집계와 같은 연속 질의를 지원한다. 따라서 본 논문에서는 대표적인 분산 데이터 처리 기법 중에서 MapReduce Online에 대해 살펴본다.

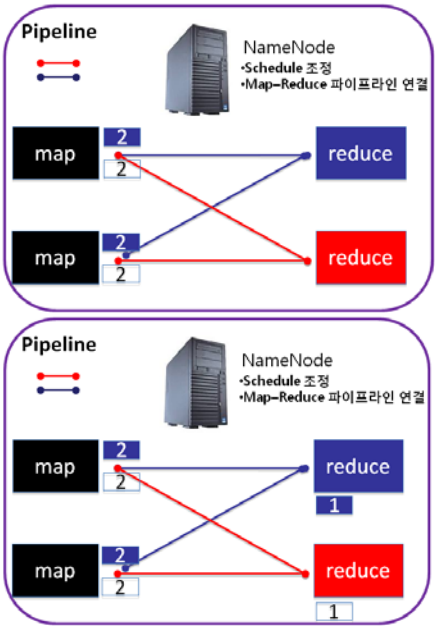


그림 6. MapReduce Online에서의 데이터 연산 결과 전송

3. 분산 데이터 처리 기법 분석

현재 MapReduce Online에는 190여개가 넘는 시스템 환경변수가 존재하며, 사용자는 이 중에서 성능에 직접적인 영향을 주는 주요 시스템 환경변수에 대한 세부적인 정보가 부족한 실정이다. 또한, 대용량 데이터의 효율적인 검색 및 정렬을 위해서는 각 노드에 일정한 기준으로 데이터를 분할하여 처리하는 것이 중요하다. 이는 특정 노드에 데이터가 편중된다면, 분산 처리의 효율성이 저하될 뿐만 아니라 전체적인 데이터 처리 시간이 증가하기 때문이다. 따라서 동일한 시스템 환경에서 MapReduce Online을 이용한 대용량 데이터 분산 처리의 성능을 최적화하기 위해서는, 시스템 환경에 적합한 최적의 환

경 설정과 함께 데이터를 각 노드에 일정한 기준으로 분할하는 분산 기법에 대한 연구가 필수적이다. 이를 위하여 본 장에서는 사용자가 전체적인 시스템 환경과 프로그램 구조를 모르더라도 쉽게 변경이 가능할 뿐만 아니라 성능에 큰 영향을 미칠 수 있는 시스템 환경변수와 함께, 샘플 데이터를 추출하여 데이터의 경향을 분석함으로써 각 노드에 일정하게 데이터를 분할하는 Sampler 기법에 대해 살펴본다.

3.1 MapReduce Online 주요 시스템 환경변수

MapReduce Online에서 사용하는 분산 데이터 저장 기법인 HDFS에서는 기존의 파일 시스템과 다르게 디스크 탐색 시간을 최소화하기 위하여 블록(block)을 통하여 데이터를 관리한다. HDFS와 같은 분산 파일 시스템에서 블록을 사용함으로써 다음과 같은 이점을 가진다. 첫째, 같은 디스크에서 불필요한 블록을 생성하지 않음으로써 데이터 저장에 대한 최적화를 유도한다. 둘째, 블록을 사용함으로써 어떠한 데이터가 저장되었는지에 대한 데이터 추상화(abstraction)가 가능하다. 마지막으로, 여러 블록들 간의 상호 보완을 통하여 데이터 손실이 발생하더라도 복구가 가능하다. 표 1은 MapReduce Online에서 HDFS와 관련되어 주로 사용되는 시스템 환경변수와 그에 대한 기본값을 나타낸다.

표 1. HDFS 환경변수

변수	타입	기본값	설명
dfs.replication	int	3	Block 사본 수
dfs.block.size	long	67108864	HDFS에서 사용되는 Block의 기본 크기

MapReduce Online의 성능 향상을 위해서는 물리적인 디스크와의 반복적인 데이터 전송을 최소화해야한다. 이는 MapReduce가 입력 데이터 소스를 기반으로 Map 함수를 수행하여 중간 결과를 디스크에 저장하고, 디스크에 저장된 중간 결과를 다시 Reduce 함수에 전송하는 2단계 구조이기 때문이다. 따라서 디스크와의 반복적인 데이터 전송을 최소화하기 위해서는, Map과 Reduce의 데이터 입/출력 버퍼로 가능한 많은 메모리를 할당해야 한다. 하지만, 기본적으로 Map과 Reduce 함수들이 동작하는데 있어서도 충분한 메모리가 확보되어 있어야 한다. 따라서 Map과 Reduce를 수행할 때 가능한 한 메모리를 적게 사용해야, 데이터 입/출력 버퍼에 많은 메모리를 할당함으로써 MapReduce Online의 성능을 향상시킬 수 있다. 표 2는 MapReduce Online에서 입/출력 버퍼와 관련되어 주로 사용되는 시스템 환경변수와 그에 대한 기본값을 나타낸다.

표 2. I/O 환경변수

변수	타입	기본값	설명
io.sort.mb	int	100	Map 출력을 정렬하는 동안 사용할 메모리 버퍼의 크기(단위 : MB)
io.sort.record.percent	float	0.05	Map 출력의 저장 레코드 경계를 나타내기 위해 예약된 io.sort.mb의 비율 남는 공간은 맵 출력 레코드 자신들을 위해 사용
io.sort.factor	int	10	파일을 정렬할 때 한 번에 병합할 최대 스트림의 수 Map과 Reduce에서 모두 사용

또한, Map과 Reduce Task의 수도 시스템의 성능과 직접적인 연관이 있는 주요 시스템 환경변수이다. Map Task의 수는 일반적으로 사용자가 기본값을 설정하며 입력파일의 크기에 따라 시스템에 의해 값이 증가할 수 있다. 반면, Reduce Task의 수는 사용자에 의해 결정되며, 최종 출력 파일의 수는 Reducer의 수에 따라 결정된다. 아울러 대용량 데이터를 처리함에 따라 실제 데이터 처리 시간보다 네트워크 통신비용이 크게 증가할 경우, Map 출력들을 압축함으로써 네트워크 통신비용을 감소시켜 전체적인 처리 시간을 감소시킬 수도 있다. 표 3은 MapReduce Online에서 Map과 Reduce의 동작과 관련되어 주로 사용되는 시스템 환경변수와 그에 대한 기본값을 나타낸다.

마지막으로 전체적인 시스템 환경과 프로그램 구조를 모르더라도, 쉽게 변경이 가능할 뿐만 아니라, 성능에 큰 영향을 미칠 수 있는 시스템 환경변수를 MapReduce의 성능 분석에 대한 연구[4, 10]를 참고하여 표 4와 같이 선정하였다. 첫째, Block의 크기이다. 기본적으로 Block의 크기는 64MB로 설정되어 있으며, 현재는 디스크 드라이브의 성능 향상으로 인하여 대부분의 시스템에서는 128MB로 설정하여 사용한다. 둘째, Map Task 및 Reduce Task의 수이다. 일반적으로 Task의 수가 많고, 하나의 노드에서

표 3. Map/Reduce 환경변수

변수	타입	기본값	설명
mapred.tasktracker.map.tasks.maximum	int	2	TaskTracker가 처리할 수 있는 최대 Map Task의 수
mapred.map.task	int	2	입력 파일을 처리하는데 필요한 총 Map Task의 수
mapred.compress.map.output	boolean	false	Map 출력들을 압축
mapred.map.output,compression.codec	Class name	org.apache.hadoop.io.compress.DefaultCodec	Map 출력에 사용할 압축 코덱
mapred.tasktracker.reduce.tasks.maximum	int	2	TaskTracker가 처리할 수 있는 최대 Reduce Task의 수
mapred.reduce.task	int	1	전체 Reduce Task의 수
mapred.reduce.parallel.copies	int	5	Map 출력들을 Reducer에 복사하기 위해 사용되는 스레드의 수
mapred.reduce.copy.backoff	int	300	Job 실패를 선언하기에 앞서 Reducer가 하나의 Map 출력을 재전송받기 위해서 대기할 수 있는 최대 시간(단위 :초)
mapred.job.shuffle.input.buffer.percent	float	0.70	Shuffle의 복사 단계 동안 Map 출력 버퍼에 할당될 전체 힙 크기의 비율
mapred.job.shuffle.merge.percent	float	0.66	출력들의 병합과 디스크로의 분할을 시작하기 위한 Map 출력 버퍼의 한계 사용 비율
mapred.inmem.merge.threshold	int	1000	출력들의 병합과 디스크로의 분할을 시작하기 위한 Map 출력 한계 개수
mapred.job.reduce.input.buffer.percent	float	0.0	Reduce 동안 Map 출력들을 메모리에 보유하기 위해 사용될 전체 힙 크기의 비율

표 4. MapReduce Online 주요 시스템 환경변수

변수	기본값	추천값	설명
dfs.block.size	64	128, 256	HDFS에서 사용되는 Block의 기본 크기 (단위:MB)
mapred.map.task	2	노드 수	입력 파일을 처리하는데 필요한 총 Map Task의 수
mapred.reduce.task	1	노드 수	전체 Reduce Task의 수
io.sort.factor	10	10~500	파일을 정렬할 때 한 번에 병합할 최대 스트림의 수

동시에 다수의 Task를 수행할 경우 성능이 향상된다. 그러나 Map과 Reduce Task의 수와 동시에 실행할 수 있는 최대 Task의 수는 시스템의 CPU 성능에 따라 최적값이 다르다. 예를 들어, 8 코어 CPU를 가진 10개의 노드에서 200개의 Map Task를 수행할 때, 1개의 코어에서 1개의 Map Task를 수행할 수 있다고 가정하자. 이 때, 하나의 노드에서 최대한 처리할 수 있는 Map Task의 수(mapred.tasktracker.map.tasks.maximum)를 2로 설정할 경우, 동시에 실행 가능한 Map Task의 수는 20개(노드 수 * 최대 Map Task 수)이다. 이 경우, 동시에 20개의 Map Task만 처리하고 나머지 180개의 Map Task가 대기하게 될 뿐만 아니라, 나머지 6개의 코어를 효율적으로 활용하지 못하게 된다. 이와 같은 경우, 하나의 노드에서 최대한 처리할 수 있는 Map Task의 수를 8로 설정할 경우, 동시에 80개의 Map Task를 수행할 수 있게 됨으로써 CPU를 최대한 효율적으로 사용할 수 있게 된다. 반대로, 싱글 코어 CPU를 가진 노드에서 동시에 많은 수의 Task를 수행하게 한다면, 오히려 CPU의 부하로 인하여 처리 속도가 저하될 것이다. 따라서 분산 병렬 처리 기법의 성능을 최적화하기 위해서는 시스템의 사양과 Task의 수를 고려하여 최적의 값을 설정하는 것이 중요하다. 셋째, 최대 동시 병합 스트림의 수이다. 일반적으로 Map 측면에서 가장 좋은 성능을 발휘할 때는 적은 수의 파일이 디스크로 분할될 때다. 만약 Map 출력 파일들의 크기를 측정할 수 있다면, io.sort.* 환경변수들을 적절히 설정하여 분할 파일들의 수를 최소화할 수 있다. 아울러 Reduce 측면에서 가장 좋은 성능을 발휘할 때는 중간 데이터 전체가 메모리에 있을 때이다. 하지만, 기본적으로 이러한 현상은 나타나지 않는다. 그 이유는 Reduce 함수가 모든 메모리

를 예약하기 때문이다. 따라서 Reduce 함수가 메모리를 적게 사용하도록 유도한다면 시스템의 성능을 향상시킬 수 있다. 즉, Reduce 함수가 적은 메모리를 요구한다면, 디스크로의 이동 횟수를 최소화하기 위하여 Reduce를 수행할 동안 Map 출력들을 메모리에 보유하기 위해 사용될 전체 메모리 크기의 비율을 증가시킴으로써 전체적인 처리 시간을 감소시킬 수 있다.

3.2 MapReduce Online 데이터 분산 기법

MapReduce Online에서는 샘플 데이터 추출을 통하여 실제 데이터를 각 노드에 일정한 기준으로 분할하는 데이터 분산 기법을 제공한다. 본 절에서는 이러한 데이터 분산 기법의 특징과 함께 그 종류에 대해서 살펴본다. MapReduce Online의 데이터 분산 기법은 효율적인 데이터의 검색 및 정렬을 위해서, 샘플 데이터를 추출하여 데이터의 경향을 분석하는 Sampler와 선정된 분할 기준점을 통하여 실제 데이터를 각 노드에 분할하는 Partitioner로 이루어진다. Sampler에서는 Map 함수를 통하여 분할된 각 분할 파일(spilt)에서 샘플 데이터를 추출하여 데이터의 경향을 분석하고, 분석한 내용을 바탕으로 분할 기준점을 선정하여 샘플 데이터 인덱스를 구성한다. 이를 통하여 Partitioner는 실제 데이터를 Reduce를 실행하는 노드(reducer)에 지정된 영역의 데이터만을 전송함으로써, 실제 데이터를 일정한 기준으로 분할하게 된다. 그림 7은 이러한 데이터 분산 기법의 처리 과정을 나타낸다.

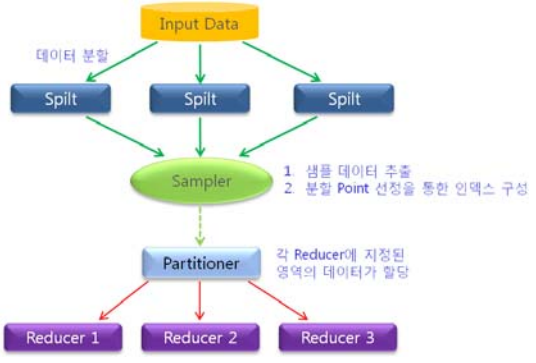


그림 7. MapReduce Online 데이터 분산 처리 과정

기본적으로 MapReduce Online에서는 3가지의 Sampler를 제공한다. 첫째, Random Sampler는 각 분할 파일의 임의 지점으로부터 샘플을 추출한다. 이를 위하여 각 분할 파일의 위치를 임의로 변경한 후, 각 분할 파일에서 임의의 위치에 있는 레코드를 샘플로 추출한다. 그림 8은 이러한 Random Sampler의 처리 과정의 예제를 나타낸다. 둘째, Spilt Sampler는 각 분할 파일에서 정해진 값에 따라 처음 n개의 샘플을 추출한다. 추출 샘플 수 n은 분할 파일의 수와 전체 샘플의 수로 결정된다. 그림 9는 4개의 분할 파일이 있을 경우, 각 분할 파일에서 처음 2개의 레코드를 샘플로 추출하는 과정의 예제를 나타낸다. 마지막으로, Interval Sampler는 각 분할 파일의 레코드를 일정한 간격으로 추출한다. 그림 10은

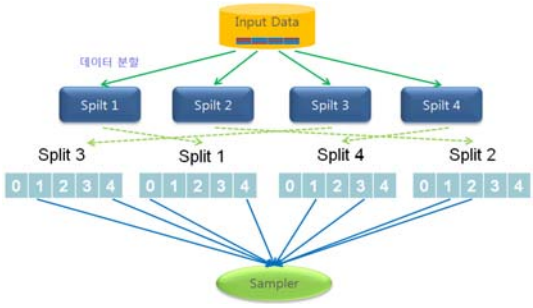


그림 8. Random Sampler 처리 예제

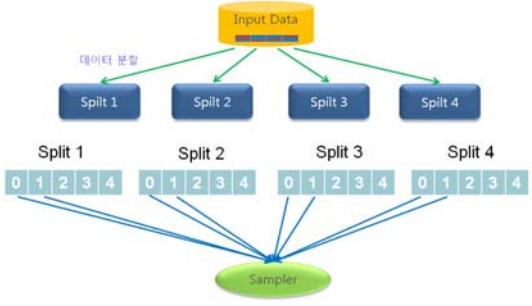


그림 9. Spilt Sampler 처리 예제

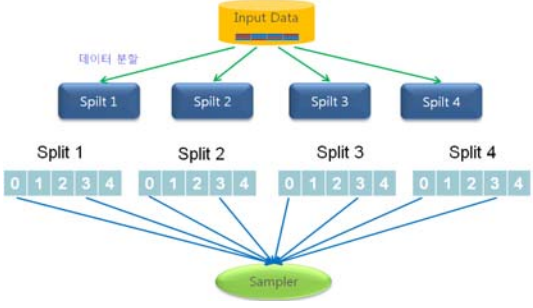


그림 10. Interval Sampler 처리 예제

각 분할 파일에서 Interval이 3일 경우, 정해진 Interval에 따라 샘플을 추출하는 과정의 예제를 나타낸다.

표 5는 MapReduce Online에서 제공하는 데이터 분산 기법 중에서, 샘플 데이터 추출을 통하여 실제 데이터를 각 노드에 일정하게 분할하기 위한 Sampler의 주요 특징과 함께 차이점을 나타낸다. 첫째, Random Sampler는 각 분할 파일의 임의 지점으로부터 샘플을 추출함으로써, 데이터의 형식에 구애받지 않고 어느 정도의 일정 성능을 나타낼 수 있다. 둘째, Spilt Sampler는 각 분할 파일에서 순차적으로 일정 샘플을 추출하며, 추출하는 파일의 수는 (전체 샘플 수/최대 분할 파일 수)이다. 한편, Spilt Sampler는 각 분할 파일에서 순차적으로 샘플을 추출하므로, 정렬된 데이터 상에서 상위 집단에 대한 결과를 추출할 때에만 효율적이다. 마지막으로, Interval Sampler는 각 분할 파일로부터 일정한 간격으로 샘플을 추출하며, 추출 기준은 (샘플 선정 비율/최대 분할 파일 수)이다. 따라서 Interval Sampler와 같은 경우, 정렬된 데이터에 대해서는 데이터의 분포 경향 파악에 있어서 가장 좋은 성능을 나타낸다.

표 5. Sampler 종류에 따른 특징 및 차이점 분석

변수	설명	추출기준	특징
Random Sampler	각 분할 파일의 임의 지점으로부터 샘플 추출	임의	모든 데이터에 대하여 일정 성능 보장
Spilt Sampler	각 분할 파일에서 순차적으로 일정 샘플을 추출	전체 샘플수 / 최대 분할파일수	정렬된 상위 집단에 대한 데이터만을 추출할 때 효율적
Interval Sampler	각 분할 파일로부터 일정한 간격으로 샘플 추출	샘플선정 비율 /최대 분할파일수	정렬된 데이터에 대해서 샘플 추출 시 효과적

4. 성능평가 및 분석

본 장에서는 HDFS와 MapReduce Online을 이용하여 Hadoop 기반 클러스터를 구축하고 이를 통하여 MapReduce Online의 주요 시스템 환경변수 및 Sampler의 종류에 따른 분산 데이터 처리 기법의 성능평가를 수행한다. 이를 위하여 HOP(Hadoop Online Prototype)[6] 0.2 버전을 설치하였다. Berkeley 대학의 HOP는 HDFS와 MapReduce Online으로 구성되어 있으며, Hadoop과 기본적으로 동일한 프로그래밍 환경을 제공한다. 또한 전체 클러스터의 총 노드 수는 21개이며, 이 중 1개의 노드가 네임노드로써 마스터(master) 서버의 역할을 수행하며, 20개의 노드가 데이터노드로써 슬레이브(slave) 노드의 역할을 수행한다. 아울러, TCP/IP 프로토콜로 통신하는 다수의 노드로 하나의 클러스터를 구성하는 완전분산방식으로 클러스터를 구축하였다. 클러스터를 구성하고 있는 각 노드들의 환경은 표 6과 같다.

표 6. 클러스터 노드 환경

항목	성능
CPU	Intel Xeon 2.5Ghz
Memory	2GB
OS	Redhat Linux 4.12-44

4.1 성능평가 예제 프로그램 시나리오

MapReduce Online의 성능평가를 수행하기 위하여 다음과 같이 데이터를 생성하였다. 첫째, 대용량 텍스트 데이터이다. 우선 영문 소설 및 성경으로부터 데이터를 수집하여 각 데이터에서 문장 추출에 의한 문장 집합(sentence pool)을 생성한다. 이를 통하여 10자리의 임의의 사용자 ID를 나타내는 Key와 함께, 문장 집합을 통하여 140자의 텍스트 데이터를 생성하여 Value로 저장한다. 그림 11은 이러한 텍스트 데이터 생성 및 저장 과정을 나타낸다. 둘째, 멀티미디어 데이터 검색을 위한 특징 벡터 데이터이다. 멀티미디어 데이터는 색상, 형태, 움직임, 소리 등 다양한 정보가 복합된 데이터를 말한다. 예를 들어 이미지, 동영상, 음악 등의 데이터가 있다. 이와 같은 데이터는 키워드와 같은 단순 텍스트만으로는 원하는 멀티미디어 데이터 검색이 어렵다. 따라서 이러한 멀티미디어 데이터를 검색하기 위해서는 다른 방법이 필요하다. 멀티미디어 데이터는 색상, 형태, 움직임 등의 저수준 특징 정보를 추출할 수 있다. 이러한 특징 정보를 고차원 특징 벡터라 한다. 따라서 멀티미디어 데이터를 검색하기 위하여, 특정 멀티미디어 데이터와 유사한 고차원 특징 벡터를 이

용한다. 이러한 벡터 데이터를 생성하기 위해 실제 멀티미디어에서 추출한 벡터 데이터 샘플을 이용한다. 그림 12는 이미지 및 비디오 데이터로부터 특징 벡터를 생성하는 과정을 나타낸다.

그림 11. 텍스트 데이터 생성 과정

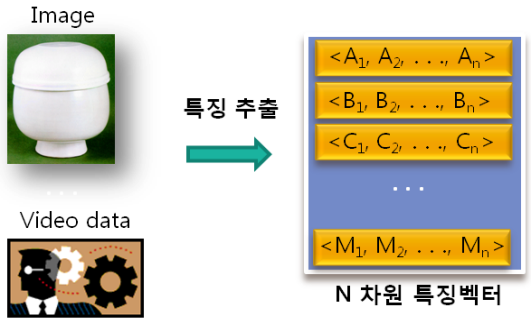


그림 12. 벡터 데이터 생성 과정

아울러, MapReduce Online의 텍스트 및 벡터 데이터에 대한 성능평가를 위하여 다음과 같이 예제 프로그램 시나리오를 작성하였다. 첫째, MapReduce Online을 이용한 텍스트 데이터 검색 프로그램이다. 텍스트 데이터 검색을 위하여 본문 내용 중 지정된 문자열이 포함된 레코드의 Key와 Value를 반환하는 본문 검색을 수행한다. 그림 13은 지정된 문자열을 검색하여 해당 레코드의 Key와 Value를 반환하는 본문 검색을 수행하는 텍스트 데이터 검색 과정을 나타낸다. 본문 검색은 지정된 레코드만을 생성 또는 반환하여 주므로 별도의 Reduce 과정이 필요하지 않으며, Map 과정만을 통하여 수행된다. 둘째, MapReduce Online을 이용한 텍스트 및 벡터 데이터 정렬 프로그램이다. 데이터 정렬을 위하여 각 Reducer의 수만큼 분할된 최종 결과 파일이 전체적으로 정렬되도록 TotalOrderPartitioner를 이용하여 전체 정렬을 수행하였다. 그림 14는 Sampler와 Partitioner를 이용하여 각 Reducer에서 정렬된 최종 결과 파일이 연결되어 전체적으로 정렬된 결과를 유지하도록 하는 전체 정렬의 수행 과정을 나타낸다.

그림 13. 텍스트 데이터 검색 프로그램 처리 과정

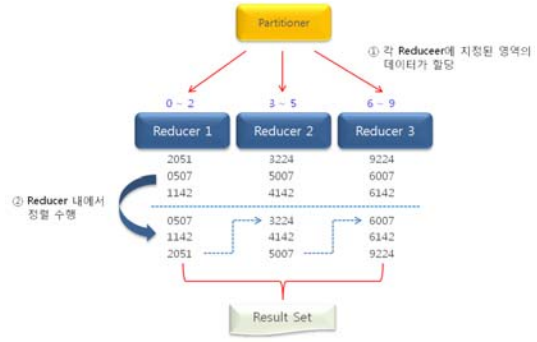


그림 14. 텍스트 및 벡터 데이터 정렬 프로그램 처리 과정

4.2 분산 병렬 처리 기법 성능비교

MapReduce Online의 성능평가에 앞서, MapRed--uce와 MapReduce Online의 텍스트 데이터에 대한 저장, 검색 및 정렬 부분에서의 성능 평가를 통하여 성능을 비교하였다. 성능평가를 위하여 5GB의 텍스트 데이터를 생성하고, 본문 검색을 수행하는 텍스트 데이터 검색 프로그램과 전체 정렬을 수행하는 텍스트 데이터 정렬 프로그램을 수행하였다. 아울러, 전체 Map Task의 수는 20개로 고정하였으며, 다른 시스템 환경변수들은 기본값으로 설정하였다. 성능평가 결과 MapReduce Online이 MapReduce에 비해 데이터 저장 측면에서 6%, 정렬 측면에서 16%, 검색 측면에서는 38% 처리 시간이 향상되었음을 알 수 있었다. 그림 15는 MapReduce와 MapReduce Online의 텍스트 데이터의 저장, 검색 및 정렬에서의 성능 비교 결과를 나타낸다.

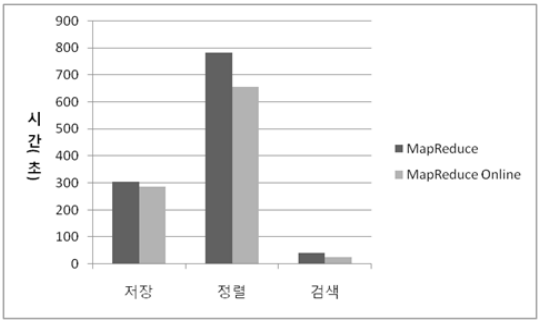


그림 15. MapReduce와 MapReduce Online 성능비교

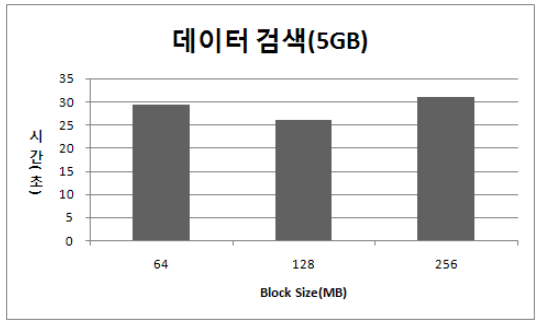
4.3 주요 시스템 환경변수에 따른 성능평가

본 절에서는 앞에서 분석한 내용을 바탕으로, 전체적인 시스템 환경과 프로그램 구조를 모르더라도 쉽게 변경이 가능할 뿐만 아니라, 성능에 큰 영향을 미칠 수 있는 주요 시스템 환경변수에 대해서 데이터 검색 및 정렬 시간에 대한 성능평가를 수행한다. 표 7은 성능평가 항목에 대한 설명과 함께 측정 범위를 나타낸다. 아울러 성능평가를 위해 앞서 설명한 텍스트 데이터 생성 프로그램을 이용하여 100GB의 텍스트 데이터를 생성하고, Map Task와 Reduce Task의 수는 하나의 Task에서 처리 가능한 최대 데이터량(약 2GB)을 고려하여 최소 80개로 설정하였다. 또한, 주요 시스템 환경변수에 따른 성능평가에서는 기본적으로 제공되는 Random Sampler를 사용하였다. 성능평가 항목은 첫째, Block의 크기에 따른 데이터 검색 시간, 둘째, 전체 Map Task 수에 따른 데이터 검색 시간, 셋째, 전체 Reduce Task 수에 따른 데이터 정렬 시간, 넷째, 최대 동시 병합 스트림 수에 따른 데이터 검색 및 정렬 시간, 마지막으로 주요 시스템 환경변수의 복합적인 변화에 대한 데이터 검색 및 정렬 시간 성능평가이다.

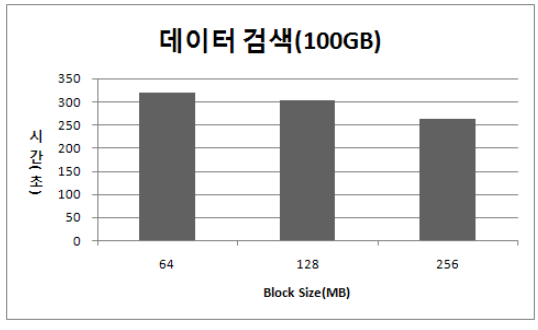
표 7. 성능평가 항목

변수	기본값	측정 범위	설명
dfs.block.size	64	64,128, 256	HDFS에서 사용되는 Block의 기본 크기 (단위:MB)
mapred.map. task	2	80,160, 320	입력 파일을 처리하는데 필요한 총 Map Task의 수
mapred.reduce .task	1	80,160, 320	전체 Reduce Task의 수
io.sort.factor	10	10,20, 40	파일을 정렬할 때 한 번에 병합할 최대 스트림의 수

첫째, Block의 크기(dfs.block.size)에 따른 데이터 검색 시간 측정을 위하여, Block의 크기를 64, 128, 256MB로 증가하면서 총 10회의 평균 시간을 측정하였다. 아울러 데이터의 크기에 따른 성능을 분석하기 위하여, 5GB와 100GB의 데이터를 생성하여 성능평가를 수행하였다. 데이터의 크기가 5GB인 경우 Map Task의 수는 20개이며, 100GB의 경우 Map Task의 수는 앞서 설명한 바와 같이 하나의 Map Task에서 처리 가능한 최대 데이터량을 고려하여 80개로 설정하였다. 다른 시스템 환경변수들은 기본값으로 고정하였다. 그림 16은 Block의 크기에 따른 데이터 검색 시간을 나타낸다. 그림에서와 같이 데이터의 크기가 5GB인 경우 Block의 크기가 128MB일때 가장 좋은 성능을 나타내며, 데이터 크기가 100GB인 경우는 Block의 크기가 256MB일때 가장 좋은 성능을 나타낸다. 이는 데이터의 크기가 작은 경우에는 비교적 작은 크기로 Block의 크기를 설정해야 최적의 성능을 나타내며, 데이터의 크기가 클 경우에는 Block의 크기를 크게 설정할수록 디스크의 탐색 시간이 최소화됨에 따라 성능이 좋게 나타



(a) 데이터 검색 시간(5GB)



(b) 데이터 검색 시간(100GB)

그림 16. Block 크기에 따른 데이터 검색 시간

남을 알 수 있다. 한편, Block 크기의 기본값인 64MB인 예전 디스크 드라이브의 데이터 전송률을 기준으로 설정된 값으로, 현재 대부분의 시스템에서는 디스크 드라이브의 성능 향상으로 인하여 Block의 크기를 최소 128MB로 설정하여 사용한다.

둘째, 전체 Map Task의 수(mapred.map.task)에 따른 데이터 검색 시간 측정을 위하여, 데이터의 크기를 100GB로 고정하고 Map Task의 수를 80, 160, 320개로 증가하면서 총 10회의 평균 데이터 검색 시간을 측정하였다. 성능평가를 위하여 다른 시스템 환경변수들은 기본값으로 고정하고, 앞서 나타난 실험결과를 바탕으로 Block의 크기를 최적값인 256MB로 설정하였다. 그림 17은 전체 Map Task의 수에 따른 데이터 검색 시간을 나타낸다. 그림에서와 같이 Map Task의 수가 160개 일 때 최적의 성능을 나타냄을 알 수 있었으며, Map Task의 수가 320개 일 때 오히려 데이터 검색 시간이 증가하였음을 알 수 있다. 이는 Map Task의 수가 과도하게 많아짐에 따라 시스템 부하에 의하여 전체 처리 시간이 증가하였기 때문이다. 즉, Map Task의 수를 시스템의 성능에 의한 일정 한도보다 높게 설정할 경우, 디스크로의 반복적인 접근에 의하여 데이터 전송 비용이 증가하기 때문이다. 따라서 시스템의 전체적인 성능을 고려하여 Map Task의 수를 설정하는 것이 필요하다.

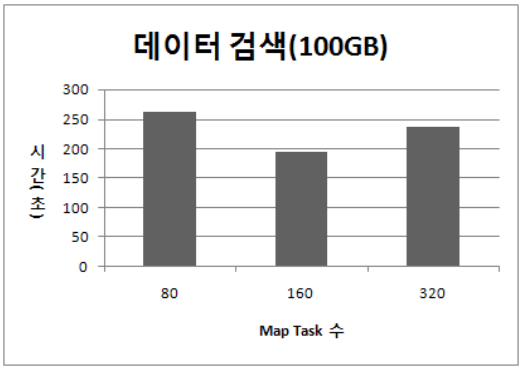


그림 17. Map Task수에 따른 데이터 검색 시간

셋째, 전체 Reduce Task의 수(mapred.reduce.ta--sk)에 따른 데이터 정렬 시간 측정을 위하여 데이터의 크기를 100GB, Map Task의 수를 80개로 고정하고, Reduce Task의 수를 80, 160, 320개로 증가하면서 총 10회의 평균 데이터 정렬 시간을 측정하였다. 아울러 두 번째 성능평가와 마찬가지로 다른 시스템 환경변수들은 기본값으로 고정하고, Block의 크기를 256MB로 설정하였다. 그림 18은 전체 Reduce Task의 수에 따른 데이터 정렬 시간을 나타낸다. Map Task수에 따른 성능평가 결과와 마찬가지로 Reduce Task의 수가 Map Task의 수가 160개 일때 최적의 성능을 나타냄을 알 수 있었으며, Reduce Task의 수가 320개 일 때 성능이 급격히 저하됨을 알 수 있다. 이는 데이터 정렬 프로그램의 경우 Map Task와 Reduce Task를 모두 사용하기 때문에, Reduce Task가 과도하게 많아짐에 따라 데이터 전송비용이 크게 증가하였기 때문이다. 즉, 고정된 중간 처리 데이터를 여러 Reduce Task를 거쳐서 분할하여 처리할 경우, 한 번에 여러 개의 Reduce Task를 수행할 수 없다면 전체적인 처리 시간이 증가할 수 있음을 나타낸다.

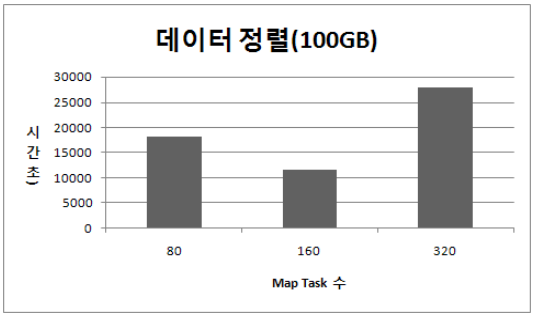
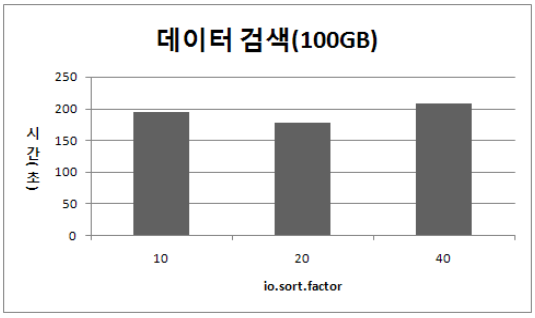


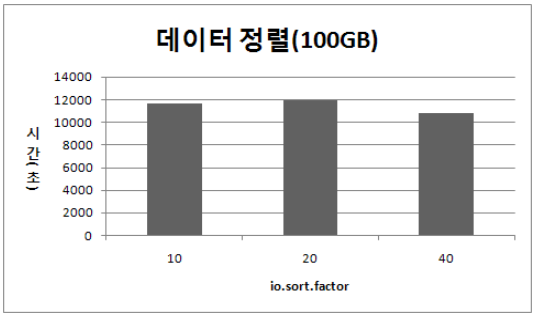
그림 18. Reduce Task수에 따른 데이터 정렬 시간

넷째, 파일을 정렬할 때 한 번에 병합할 최대 스트림의 수(io.sort.factor)에 따른 데이터 처리 시간 측정을 위하여, 스트림의 수를 기본값인 10부터 20, 40으로 증가하면서 총 10회의 평균 데이터 검색 및 정렬 시간을 측정하였다. 또한, 앞서 나타난 실험결과를 바탕으로 100GB의 데이터에서 Block의 크기는 256MB로 고정하였으며, 데이터 검색시 Map Task의 수는 160개, 데이터 정렬시의 Map Task의 수와 Reduce Task의 수는 각각 80, 160으로 설정하였다. 그림 19는 최대 동시 병합 스트림 수에 따른 데이터 검색 및 정렬 시간을 나타낸다. 성능평가 결과, 데이터 검색 측면에서는 최대 스트림의 수가 20일 때 가장 좋은 성능을 나타내며, 데이터 정렬 측면에서는 최대 스트림의 수가 40일 때 가장 좋은 성능을 나타

낸다. 이는 시스템의 성능뿐만이 아니라, 프로그램에 사용되는 데이터의 형태뿐만 아니라, 프로그램의 구조 및 목적에 따라 전체적인 데이터 처리 성능이 달라질 수 있음을 보여준다. 한편, 최대 동시 병합 스트림의 수에 따른 성능평가 결과, Map Task의 수와 Reduce Task의 수에 따른 성능평가 결과와 달리 최대 동시 병합 스트림의 수는 성능에 크게 차이를 주진 않는 것으로 나타난다. 이는 MapReduce의 성능 분석에 대한 연구[4]에서도 나타난 결과로, Map과 Reduce Task수가 최적의 값 이상일 경우, 최대 동시 병합 스트림의 수는 성능에 크게 차이를 나타내지 않는다.



(a) 데이터 검색 시간



(b) 데이터 정렬 시간

그림 19. 최대 동시 병합 스트림 수에 따른 데이터 처리 시간

마지막으로 여러 시스템 환경변수의 복합적인 변화에 따른 전체적인 시스템의 성능 변화 측정을 위하여, 주요 시스템 환경변수의 복합적인 변화에 대한 데이터 검색 및 정렬 시간에 대한 성능평가를 수행한다. 성능평가를 위해 100GB의 데이터에서 본문 검색을 수행하는 텍스트 데이터 검색 프로그램과 전체 정렬을 수행하는 텍스트 데이터 정렬 프로그램을 수행하였다. 또한 성능평가를 위하여 Block의 크기는 최적값인 256MB로 고정하고, 분석할 시스템 환경변수를 제외한 나머지 환경변수들은 기본값으로 고정하였다. 그림 20은 전체 Map Task 수와 최대 동시 병합 스트림 수에 따른 데이터 검색 시간을 나타낸다. 데이터 검색 측면에서는 전체 Map Task 수와 최대 동시 병합 스트림의 수를 증가시켜가면서 총 10회의 평균 데이터 검색 시간을 측정하였다. 데이터 검색 측면에서는 전체 Map Task의 수가 160개이고 최대 동시 병합 스트림 수가 20일 때 최적의 성능을 나타내었다. 이에 비해, 전체 Map Task 수가 80개인 경우 최대 동시 병합 스트림 수가 20개일 때, Map Task 수가 320개인 경우 최대 동시 병합 스트림 수가 10개일 때 최적의 성능을 나타내었다.

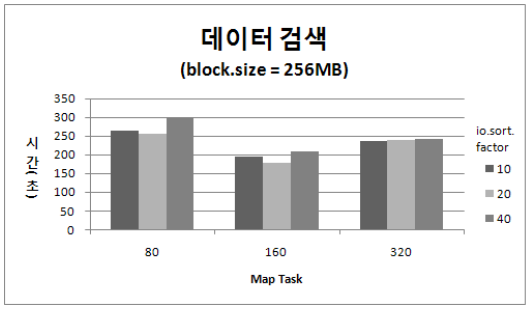


그림 20. 전체 Map Task 수와 최대 동시 병합 스트림 수에 따른 데이터 검색 시간

그림 21은 전체 Reduce Task 수와 최대 동시 병합 스트림 수에 따른 데이터 정렬 시간을 나타낸다. 데이터 정렬 측면에서는 전체 Map Task의 수는 80개로 고정하고, 전체 Reduce Task 수와 최대 동시 병합 스트림의 수를 증가시켜가면서 총 10회의 평균 데이터 정렬 시간을 측정하였다. 데이터 정렬 측면에서는 전체 Reduce Task의 수가 160개이고 최대 동시 병합 스트림 수가 40일 때 최적의 성능을 나타내었으며, 전체 Reduce Task의 수가 320개이고 최대 동시 병합 스트림 수가 10일 경우 데이터 정렬 시간이 가장 길게 측정되었다. 이를 통하여 시스템 환경변수가 복합적으로 변경될 경우, 고정된 특정 값에서 최적의 성능을 나타내지 않음을 알 수 있다. 또한, 앞서 나타난 실험결과와 마찬가지로 Map 혹은 Reduce Task 수에 비해 최대 동시 병합 스트림 수는 성능에 크게 영향을 미치진 않음을 알 수 있다.

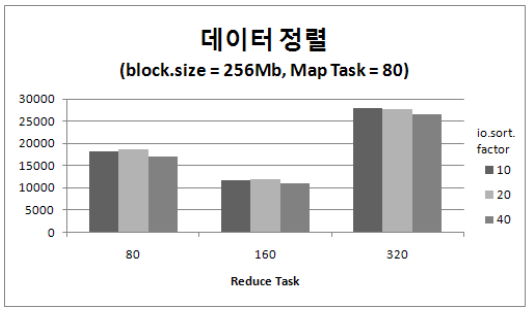
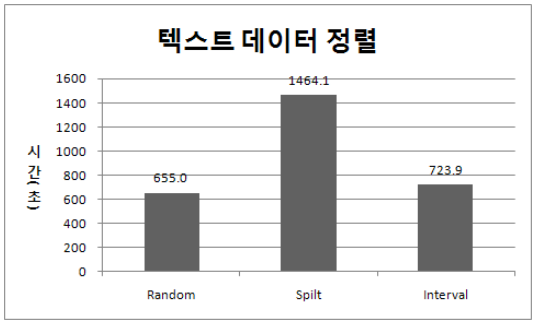


그림 21. 전체 Reduce Task 수와 최대 동시 병합 스트림 수에 따른 데이터 정렬 시간

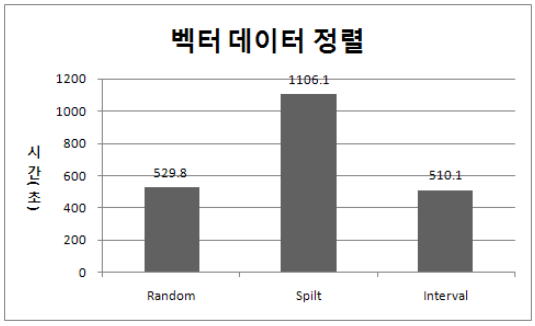
4.4 Sampler에 따른 데이터 정렬 성능평가

본 절에서는 Sampler의 종류에 따른 데이터 정렬 시간에 대한 성능평가를 수행한다. 성능평가를 위해 5GB의 텍스트 데이터와 함께 10차원의 벡터 데이터를 Value로 가지는 5GB의 벡터 데이터를 생성하였다. 한편, 데이터의 형태에 따라서 Task의 수에 따른 성능차이가 발생할 수 있으므로, Map Task의 수와 Reduce Task의 수는 각각 20으로 설정하였다. 이는 시스템 환경변수에 의한 성능차이보다는 Sampler의 종류에 따른 성능변화를 분석하기 위함이다. 아울러, Block의 크기는 5GB의 데이터에서 최적의 값을 나타내는 128MB로 고정하였으며, 추출하는 전체 샘플의 수는 10000개로 설정하였다. 성능평가 대상은 MapReduce Online에서 기본적으로 제공하는 Random, Spilt, Interval Sampler이다.

그림 22는 각 Sampler에 따른 데이터의 정렬 시간을 나타내며, 그림 23은 각 Sampler에 따른 데이터의 노드 편중도를 나타낸다. 성능평가 결과, 텍스트 데이터의 경우 Random Sampler가 가장 좋은 성능을 보여주고 있으며, 벡터 데이터의 경우 Interval Sampler가 Random Sampler에 비해 다소 좋은 성능을 나타내고 있다. 이는 텍스트 데이터에 비해 벡터 데이터의 Value 크기가 작고 정렬 기준이 상대적으로 간단함에 따라, 좀 더 간단한 처리방법으로 샘플 데이터를 추출하는 Interval Sampler가 다소 좋은 성능을 나타냄을 알 수 있다. 아울러, 데이터의 종류와 무관하게 두 경우 모두 Spilt Sampler가 가장 좋지 않은 성능을 나타내고 있으며, 이는 Spilt Sampler는 데이터를 효율적으로 분할하지 못하였기 때문에 그림 23과 같이 마지막 노드로 대부분의 데이터가 편중되었기 때문이다. 이를 통하여, 처음 레코드부터 순차적으로 일정 데이터만을 추출하여 분할 기준점을 선정하는 것은 데이터의 전체적인 분포를 판별할 수 없음을 알 수 있다.



(a) 텍스트 데이터 정렬 시간

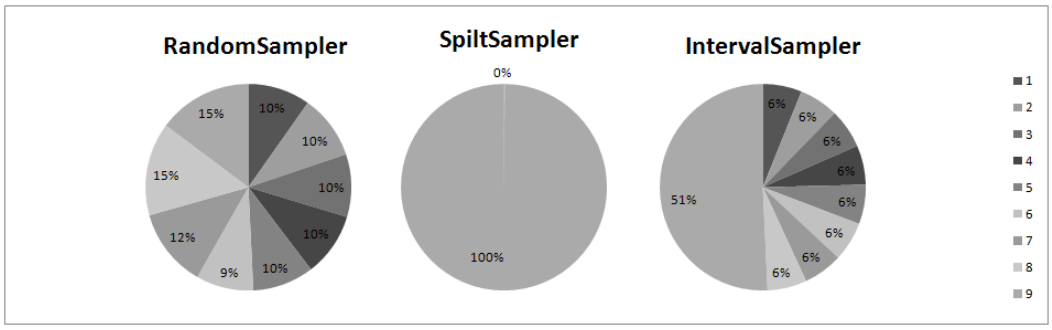


(b) 벡터 데이터 정렬 시간

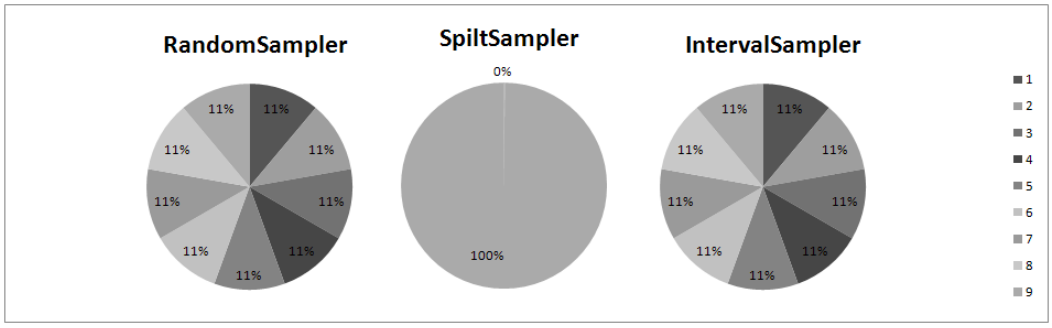
그림 22. Sampler에 따른 데이터 정렬 시간

4.5 분산 병렬 처리 기법 성능 최적화 요구사항 분석

본 절에서는 성능평가 결과를 바탕으로 MapReduce Online의 성능 개선을 위한 최적화 요구사항에 대하여 살펴본다. 첫째, MapReduce Online의 성능을 개선하기 위해서는, 시스템 환경에 최적화된 주요 시스템 환경변수의 설정이 필요하다. 성능평가 결과, Map Task 수와 Reduce Task 수가 성능에 가장 큰 영향을 주는 것을 확인할 수 있었으며, 이를 위해서 노드 수와 시스템의 성능에 따라 Map과 Reduce Task 수를 적절히 조절하는 것이 필요하다. 각 노드에서 4개의 멀티 코어 CPU를 탑재한 20개의 노드를 사용하는 본 성능평가 환경에서는, Map Task와 Reduce Task 수를 160개로 설정하는 것이 최적의 성능을 나타내었다. 따라서 Map Task 수와 Reduce Task의 수는 최소한 노드 수와 동일하게 설



(a) 텍스트 데이터의 노드 편중도



(b) 벡터 데이터의 노드 편중도

그림 23. Sampler에 따른 데이터의 노드 편중도

정하여야 하며, 멀티 코어와 같이 병렬 처리가 가능한 CPU를 사용하는 시스템 환경에서는 CPU의 성능에 따라 2배수 이상의 값을 설정하는 것이 성능을 향상 시킬 수 있다. 아울러, Block 크기의 경우, 일반적으로 128MB로 설정하며, 대용량의 데이터를 처리할 경우 256MB 이상의 크기로 설정함으로써 다소 성능을 향상 시킬 수 있다. 둘째, 데이터의 효율적인 분산 처리를 위해서는 각 노드에 일정한 비율로 데이터를 분할하는 것이 중요하다. 따라서 샘플 데이터를 추출하여 데이터의 분포 경향을 분석함으로써, 실제 데이터를 각 노드에 일정하게 분할하도록 하는 것이 필요하다. 이를 위해 MapReduce Online에서는 샘플 데이터 추출을 위한 3가지의 Sampler를 제공하며, 성능평가 결과 텍스트 데이터의 경우 Random Sampler가 가장 좋은 성능을 나타냈으며, 벡터 데이터의 경우 Interval Sampler가 Random Sampler에 비해 다소 좋은 성능을 나타냈다. 반면, Spilt Sampler의 경우, 마지막 노드에 대부분의 데이터가 편중됨에 따라 가장 좋지 않은 성능을 나타냈다. 따라서 일반적으로는 Random Sampler를 사용하는 것이 성능 향상에 도움이 되며, 벡터 데이터와 같이 상대적으로 크기가 작고 비교 연산이 간단한 데이터의 경우에는 Interval Sampler를 사용하는 것도 성능을 향상 시킬 수 있다.

5. 결론 및 향후 연구

본 논문에서는 대표적인 분산 데이터 저장 기법인 Apache의 HDFS에 대해 살펴보고, Hadoop MapReudce 및 Berkeley 대학의 MapReduce Online에 대하여 분석하였다. 아울러 분산 병렬 데이터 처리 기법의 성능 최적화를 위한 주요 시스템 환경변수와 함께 효율적인 데이터의 검색 및 정렬을 위한 Sampler 기법에 대하여 살펴보았다. 이를 기반으로 Hadoop 기반 클러스터를 구축하고, 주요 시스템 환경변수 및 Sampler 기법에 따른 분산 데이터 처리 기법의 성능평가를 수행하였다. 주요 시스템 환경변수의 성능평가 결과, MapReduce Online은 Map-

Reduce에 비해 데이터 저장 측면에서 6%, 정렬 측면에서 16%, 검색 측면에서는 38% 처리 시간이 향상되었음을 알 수 있었다. 아울러, MapReduce Online의 주요 시스템 환경변수 중에서 Map Task 수와 Reduce Task 수가 성능에 가장 큰 영향을 주는 것을 확인할 수 있었으며, Map과 Reduce Task 수는 최소한 노드 수와 동일하게 설정하여야 하며 CPU의 성능에 따라 2배수 이상의 값을 설정해야 성능을 향상 시킬 수 있음을 확인할 수 있었다. 한편, Sampler의 종류에 따른 데이터 정렬 시간 성능평가 결과에서는 Random Sampler가 가장 좋은 성능을 나타냈으며, 벡터 데이터와 같은 경우 Interval Sampler가 Random Sampler에 비해 다소 좋은 성능을 나타냈다. 향후 연구는 GIS 응용에서 실제 사용하는 공간 데이터를 사용하여 분산 처리 연구를 수행하는 것이다.

참 고 문 헌

[1] Apache Software Foundation, Apache Hadoop: http://hadoop.apache.org/.

[2] Apache Software Foundation, Apache Lucene: http://lucene.apache.org/.

[3] Apache Software Foundation, Hadoop Map-Redce: http://hadoop.apache.org/mapreduce.

[4] Shivnath Babu, 2010, “Towards Automatic Optimization of MapReduce Programs”, ACM Symposium on Cloud Computing, pp. 137-142

[5] D. Borthakur, 2009, HDFS Architecture: http://hadoop.apache.org/common/docs/r0-.20.0/hdfs_design.pdf.

[6] T. Condie, N. Conway, Hadoop Online Prototype: http://code.google.com/p/hop.

[7] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy and R. Sears, 2010, “MapReduce Online”, Networked Systems Design and Implementation, pp. 21-21.

[8] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan and R. Sears, 2010, “Benchmarking cloud serving systems with YCSB”, ACM Symposium on Cloud Computing, pp. 143-154.

[9] J. Dean, S. Ghemawat, 2004, “MapReduce : Simplified Data Processing on Large Clusters”, Operating System Design and Implementation, pp. 10-10.

[10] S. Ghemawat, H. Gobioff and Shun-Tak Leung, 2003, “The Google File System”, Symposium on Operating Systems Principles, pp.29-43.

[11] D. Jiang, B. C. Ooi, L. Shi and S. Wu, 2010, “The performance of MapReduce: an in-depth study”, VLDB, vol. 3, Issue 1-2, pp. 472-483.

[12] C. Olston, B. Reed, U. Srivastava, R. Kumer and A. Tomkins, 2008, “Pig latin: a not-so-foreign language for data processing”, ACM SIGMOD, pp. 1099-1110.

[13] 민영수, 김홍연, 김영균, 2009, “클라우드 컴퓨팅을 위한 분산 파일 시스템 기술”, 한국정보과학회지, 제27권, 제5호, pp. 86-94.

[14] 민옥기, 김학영, 남궁한, 2009, “클라우드 컴퓨팅 기술 동향”, ETRI 전자통신동향분석, 제24권, 제4호, pp 1-13.

[15] 성병용, 2009, “국내 기업의 클라우드 컴퓨팅 동향 및 전략”, 한국소프트웨어진흥원 정책리포트, 2009년, 7월호, pp. 6-25.

[16] 이기영, 김동오, 신중수, 한기준, 2008, “대용량 공간 데이터의 효율적인 검색을 위한 공간미들웨어의 개발”, 한국공간정보시스템학회 논문지, 제10권, 제1호, pp. 1-14.

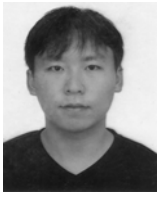
[17] 이동규, 이경민, 정석호, 이성호, 류근호, 2010, “대용량 공간 데이터로부터 빈발 패턴 마이닝”, 한국공간정보시스템학회 논문지, 제12권, 제1호, pp. 49-56.

[18] 정재호, 2008, “클라우드. 컴퓨팅의 현재와 미래, 그리고 시장전략”, 한국소프트웨어진흥원 정책리포트, 2008년, 10월호, pp. 56-85.

논문접수 : 2011.05.17
수 정 일 : 1차 2011.07.22 / 2차 2011.08.08
심사완료 : 2011.08.11

홍 승 태

2008년 전북대학교 전자정보공학부 (공학사)
2010년 전북대학교 전자정보공학부 (공학석사)
2010년~현재 전북대학교 전자정보공학부 박사과정
관심분야는 공간 데이터베이스, 센서 네트워크, 클라우드 컴퓨팅



신 영 성

2010년 전북대학교 전자정보공학부 (공학사)
2011년~현재 전북대학교 전자정보공학부 석사과정
관심분야는 공간 데이터베이스, 클라우드 컴퓨팅, 소셜 네트워크

장 재 우

1984년 서울대학교 전자계산기공학과 (공학사)
1986년 한국과학기술원 전산학과 (공학석사)
1991년 한국과학기술원 전산학과 (공학박사)
1996년~1997년 Univ. of Minnesota, Visiting Scholar
2003년~2004년 Penn State Univ., Visiting Scholar.
1991년~현재 전북대학교 컴퓨터공학과 교수
관심분야는 공간 네트워크 데이터베이스, 하부저장구조, 센서 네크워크