특집 02

HadoopX : Hadoop MapReduce 기반 순환 처리 시스템

홍승태 · 윤민 (전북대학교), 최동훈 (한국과학기술정보연구원), 조희승 · 장재우 (전북대학교)

목 차 »

1. 서 론

최근 문서의 디지털화, 웹/SNS의 발전으로 인하여 데이터 규모가 폭발적으로 증가하고 있으며, 이에 따라 빅데이터의 중요성이 강조되고 있다. 빅데이터는 기존의 IT 기술로는 저장/관리/분석이 어려울 정도로 데이터의 생성 속도가 빠르고, 다양한 종류로 구성된 대용량 데이터를 의미한다. 따라서 빅데이터에 대한 효율적인 분석 및 마이닝을 수행하기 위한 MapReduce 기술에 대한 연구가 활발히 수행되고 있다. MapReduce는 2004년 Google에서 제안한 소프트웨어 플랫폼으로써, 분산 컴퓨팅 환경에서 빅데이터 처리를 지원하기 위한 목적으로 개발되었다[1]. 이러한 MapReduce 기술을 구현한 대표적인 분산 컴퓨팅 시스템으로는 Apace의 Hadoop MapReduce가 존재한다[2].

한편, 대부분의 빅데이터 분석 응용은 특정 작업의 반복적인 수행을 요구하지만, Hadoop MapReduce는 이러한 순환 처리를 자체적으로 지원하지 못하는 문제점이 존재한다. 즉, Hadoop MapReduce는 한 단계(phase)의 MapReduce 과정을 통해 결과를 도출하는 비순환 처리(non-iterative) 프로세싱 구조를 따르기 때문에, 두 단계 이상의 MapReduce 과정을 통해 결과가 도출되는 순환 처리(iterative) 응용에 최적화된 성능을 제공하지 못한다. 이에 따라 빅데이터에 대한 순환 처리의 중요성이 강조되고 있으며, MapReduce 순환 처리 기법에 대한 연구가 활발히 수행 중에 있다[3,4]. 그러나 기존 순환 처리 기법을 이용하여 순환 처리 응용을 병렬 처리하려면 MapReduce 프로그래밍 모델 및 전체 데이터 구조에 대한 숙련된 지식이 요구됨에 따라, 비전문개발자가 사용하기에는 어려운 실정이다.

따라서 본 논문에서는 비전문 개발자를 위한 Hadoop MapReduce 기반 순환 처리 기법(이하 HadoopX)을 제안한다. HadoopX에서는 비전문 개발자가 효율적으로 순환 처리 응용을 사용할

수 있도록, 매 수행 단계에서 반복적으로 사용되는 불변(invariant) 데이터의 자동 검출 기능을 수행한다. 아울러, 반복적인 입력 데이터의 캐싱을 위한 Map Input Cache 및 MapReduce의 shuffle 단계에서의 네트워크 전송 비용 감소를 위한 Reduce Input Cache를 사용자 설정에 맞게 선택적으로 제공함으로써 캐싱 성능을 극대화한다. 마지막으로 제시하는 기법은 순환 처리 기능을 모듈화하여 제공함으로써, 대표적인 병렬 처리 기법인 Hadoop의 버전 변경에 따른 내부 수정을 최소화하였다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 대표적인 MapReduce 기반 분산 데이터 처리 기법에 대하여 살펴보고, 3장에서는 Hadoop MapReduce 기반 순환 처리 기법을 제안한다. 4장에서는 다양한 응용을 통해 성능평가를 수행하고 이에 대한 분석을 수행한다. 마지막으로 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

빅데이터 분석을 위한 대표적인 분산 병렬 처리 기법은 Hadoop[2], HaLoop[3], Twister[4]가 존재한다. 첫째, Apache 그룹에서 개발한 Hadoop MapReudce는 Google의 MapReduce와 동일한 구조와 기능을 제공하며, 오픈 소스 기반의 분산 병렬 처리 기법으로써 현재 대용량 데이터의 처리를 위한 대표적인 시스템이다. MapReduce는 (key, value) 기반의 데이터를 분산 처리하는 모델로, Map 함수는 입력 데이터를 (key, value) 쌍으로 추출하여 중간 값 집합을 생성한다. Reduce 함수는 중간 key값을 가지는 모든 중간 값들을 통합하여 최종 출력 값으로 저장한다.

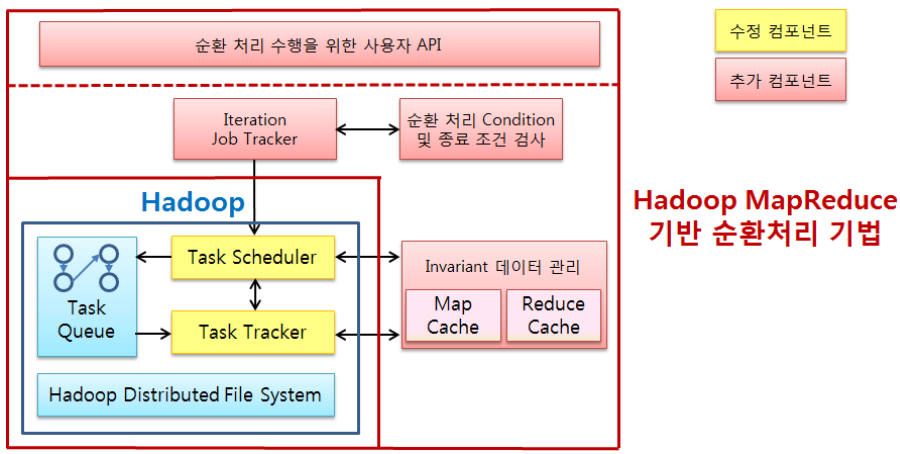
둘째, Washington 대학의 Y. Bu et al.가 제안한 HaLoop[4]은 마스터 노드의 Loop control 모듈을 통해 자체적으로 반복 연산 수행을 지원한다. 사용자가 순환 연산의 종료 조건을 시스템에 입력하면, HaLoop은 종료 조건을 지속적으로 검사하면서 이를 만족하는 동안 반복적으로 연산을 수행한다. HaLoop은 매 수행단계에서 반복적으로 사용되는 invariant 데이터를 캐싱 및 인덱싱하여 재사용하기 때문에, 데이터 전송 오버헤드 및 I/O 비용을 감소시킨다. 그러나 HaLoop은 각 응용에서 사용자가 invariant 데이터를 직접 설정 및 관리해주어야 하기 때문에, 순환 처리 기법에 대한 숙련된 지식을 요구하는 문제점이 존재한다. 아울러, HaLoop은 순환 처리를 위하여 기존 Hadoop MapReduce의 핵심 컴포넌트를 수정하였기 때문에, Hadoop의 지속적인 버전 변경에 신속히 반영할 수 없는 문제점이 존재한다.

셋째, Indiana 대학의 J. Ekanayake et al.은 MapReduce의 순환 처리를 위한 Twister[5]를 제안하였다. Twister는 멀티 코어의 CPU를 지닌 하나의 컴퓨터를 클러스터로 구성하고, 모든 MapReduce의 job을 in-memory 상에서 수행함으로써, 빠른 수행시간을 제공한다. 아울러, 데이터 타입을 정적(static) 및 동적(dynamic)으로 나누어, 순환 처리 수행 시 정적 데이터를 캐시에 저장함으로써 순환 처리에 대한 I/O 통신 오버헤드를 크게 감소시킨다. 그러나 Twister는 순환 처리를 위하여 독자적인 프레임워크를 제공하기 때문에, 기존 Hadoop MapReduce 응용을 활용할 수 없다.

3. Hadoop MapReduce기반 순환 처리 기법

3.1 시스템 구조

본 논문에서 제안하는 Hadoop MapReduce 기반 순환 처리 기법의 시스템 구조는 (그림 1)과



(그림 1) Hadoop MapReduce 기반 순환처리 기법 시스템 구조

같다. 제안하는 순환 처리 기법은 기존 Hadoop 위에 Add-on 형태로 제공되며, 순환 처리 기능을 모듈화함으로써 Hadoop 내부의 수정을 최소화한다. 제안하는 Hadoop MapReduce 기반 순환처리 기법의 추가 컴포넌트로는 Iteration Job Tracker, Invariant 데이터 관리 모듈, 순환 처리 condition 및 종료 조건 검사 모듈이 존재한다. 첫째, Iteration Job Tracker는 사용자로부터 주어진 순환처리 job에 대한 통합적인 관리 기능을 수행한다. 아울러 다수의 job을 step으로 관리하여 map 및 reduce 과정에 대한 초기 설정 비용을 최소화한다. 둘째, Invariant 데이터 관리 모듈은 Invariant 데이터에 대한 자동 검출 및 관리 기능을 수행한다. 또한, 반복 작업을 수행하면서 검출된 Invariant 데이터를 캐싱함으로써 불필요한 네트워크 트래픽이 발생하는 것을 방지한다. 셋째, 순환 처리 condition 및 종료 조건 검사 모듈은 이전 수행단계에서의 결과와 현재 수행단계의 결과 비교를 통해 반복 작업의 종료 여부를 검사한다.

3.2 Invariant 데이터 자동 검출

제안하는 기법에서는 사용자가 어떠한 데이터가 invariant 데이터인지 정확한 정보가 없더라도, 순환 처리 수행 중에 자동적으로 검출함으로써 사용자의 사용성(usability)을 극대화한다. 이를 위해, 제안하는 기법에서는 partitioner 기반의 data locality 정책을 사용함으로써, 각 수행단계마다 동일한 key의 데이터가 지속적으로 동일한 노드에 할당되도록 한다. 이를 통해, 이전 수행단계와의 데이터 비교가 가능하도록 함으로써, 매 수행단계에서 각 노드에 할당되는 데이터들의 변화 추세를 파악하는 것이 가능하다. 이 때, 이전 수행단계의 데이터에 대한 빠른 검색을 위해 key별로 그룹화를 수행하여 다수의 파일로 나누어 저장하고, 각 파일을 위한 hash 기반의 인덱스를 생성한다. 이를 통해, invariant 데이터 검출을 위해 탐색해야 하는 데이터의 양을 크게 감소시킬 수 있다.

Invariant 데이터 자동 검출의 전체 수행과정은 다음과 같다. 첫째, 순환 프로세싱의 첫 수행단계에서 Mapper는 전송받은 split 데이터를 <key, value>쌍으로 추출하여 각 Reducer에게 전송한다. Reducer는 전송받은 <key, value>쌍의 데이터를 입력 데이터의 경로(data_source)별로 나누어 캐싱한다. 첫 번째 수행단계에서는 초기 모든 데

이터를 invariant 데이터로 간주하기 때문에, Mapper 로부터 전송받은 모든 데이터를 캐싱한다. 이때, 캐싱 탐색 비용을 감소시키기 위하여 key별로 그룹화를 수행하여 다수의 파일로 나누어 저장하고, 각 파일을 위한 hash 기반의 인덱스를 생성한다. 해당 인덱스는 <key, index>를 저장하며, 여기서 key는 데이터의 key 값을 의미하고, index는 각 key 값에 해당하는 데이터가 파일 내 저장되어 있는 위치를 의미한다. 아울러, 해당 수행단계에서는 데이터의 변화 추세를 파악하기 위하여 Reducer에서 연산된 모든 결과를 다음 수행단계로 전송한다. 둘째, 두번째 수행단계에서 Mapper는 이전 수행단계의 Reducer 출력 결과를 입력받는다. 첫 번째 수행단계와 마찬가지로 Mapper는 전송받은 데이터를 Reducer에게 전송한다. Reducer는 전송받은 데이터에 대해 data_source를 판단하여, 이전 수행단계에서 저장한 캐싱 데이터 중 상응하는 캐싱 파일 및 hash 인덱스를 선택한다. 해당 인덱스를 검색하여 전송받은 데이터가 저장되어 있는 index를 확인하고, 선택한 파일 내 해당 위치에 접근하여 전송받은 데이터와 동일한 데이터가 있는지 검사한다. 만약 동일한 데이터가 없다면 변경된(variant) 데이터로 간주하여 해당 데이터의 data_source에 해당하는 데이터를 캐싱 파일에서 삭제한다. 만약, 동일한 데이터가 존재할 경우, invariant 데이터로 간주하여, 출력결과를 다음 수행단계로 전달하지 않고 작업 machine 내에서 캐싱을 유지한다. 이를 통해 invariant 데이터가 재전송되는 것을 방지함으로써 불필요한 네트워크 트래픽을 감소시킨다. 마지막으로, 세번째 이상의 수행단계에서는 캐싱된 invariant 데이터를 제외한 데이터만을 Mapper에서 입력받게 되며, 이를 Reducer로 전송한다. Reducer에서는 해당 machine에 캐싱된 invariant 데이터와 전송된 데이터를 통하여 연산을 수행하고, 다음 수행단계로 invariant 데이터를 제외한 연산 결과만을 전송한다.

3.3 Map Input Cache 및 Reduce Input Cache

MapReduce에 사용되는 순환처리 응용은 사용되는 데이터 및 연산 과정이 서로 상이하기 때문에, 각 순환처리 응용에 대한 최적화된 캐싱 정책이 필수적이다. 이를 위해 제안하는 기법에서는 반복적인 입력 데이터의 캐싱을 위한 Map Input Cache, MapReduce의 shuffle 단계에서의 네트워크 전송 비용 감소를 위한 Reduce Input Cache를 제공한다. 먼저 Mapper는 job의 초기 수행단계에서 Input 데이터를 Hadoop의 InputFormat, RecordReader, InputSplit를 통하여 입력받는다. 이때, k-means 응용과 같이 수행단계마다 동일한 데이터를 입력받는 경우, Map Input Cache를 통하여 입력 데이터를 캐싱한다. 캐싱된 데이터는 해당 슬레이브 노드의 로컬 디스크에 저장됨으로써, 다음 수행단계에서 HDFS에서의 데이터 입력을 통한 네트워크 전송 오버헤드를 감소시킨다.

한편, Mapper의 중간 연산 결과는 shuffle 단계를 거쳐 Reducer로 전달된다. 이때, 이전 수행단계에서 해당 슬레이브 노드에 캐싱된 Reduce Input Cache가 존재할 경우, 해당 캐싱 데이터를 불러와서 Mapper로부터 전달된 데이터와 함께 연산을 수행한다. 이때, Reduce Input Cache는 Invariant 데이터 관리 모듈을 통하여 해당 노드에 유지되며, 이를 통해 매 수행단계에서 반복적으로 사용되는 Invariant 데이터에 대한 전송 오버헤드를 감소시킨다. 이때, shuffle을 통해 전달되는 Mapper의 중간 연산 결과는 hash 기반 partitioner를 통하여 이전 수행단계와 동일한 노드에 할당된다. 아울러, Reducer의 연산결과는 다

음 수행단계의 입력으로 전달되며, 이는 하둡 파일 시스템(HDFS)를 통하여 전달된다. 이때, Mapper Input Cache가 존재할 경우, 이전 수행단계에서의 결과와 Map Input Cache를 응용의 설정에 따라서 입력 데이터로 사용된다.

4. 성능평가

4.1 성능평가 환경

본 장에서는 제안하는 기법의 효율성을 입증하기 위해, Hadoop 및 HaLoop과의 성능 비교를 수행한다. 성능평가는 HaLoop 1.2버전과 Hadoop 0.20.0 버전을 사용하여 수행한다. 이는 HaLoop이 Hadoop 0.20.0 버전을 기반으로 순환 처리와 관련된 사항을 확장한 시스템이기 때문에, 순환처리 관련 성능 비교를 수행하기 위하여 해당 버전의 Hadoop과 성능 비교를 수행한다. 클러스터는 총 7개의 노드로 구성되며, 이 중 1개의 노드는 네임노드로써 마스터(master) 역할을 수행하며, 6개의 노드는 데이터노드로써 슬레이브 노드의 역할을 수행한다. 클러스터를 구성하는 각 노드의 성능은 <표 1>과 같다. 아울러, 전체 Reduce Task의 수는 6개, 순환처리 작업의 반복횟수(iteration)는 10으로 고정하였으며, 다른 시스템 환경변수들은 기본 값으로 설정하였다.

성능평가는 대표적인 순환 처리 응용인 PageRank 알고리즘[5], Descendant 질의[3], k-means 알고리즘[3]의 수행 시간 비교를 통해 수행한다.

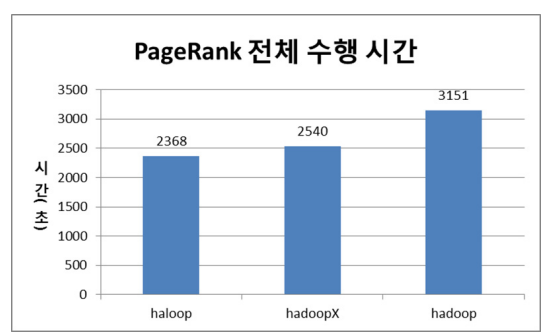
〈표 1〉 클러스터 노드 성능

항목	성능
CPU	Intel i5 Xeon 2.5Ghz
Memory	2GB
OS	Redhat Linux 4.12-44 64bit

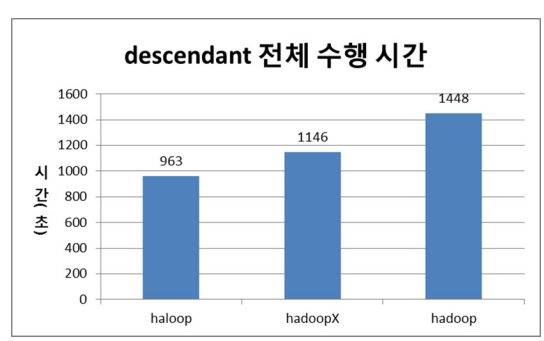
PageRank 알고리즘 및 Descendant 질의 성능평가는 1GB 크기의 Livejournal short data[6]를 이용하여 수행하였으며, k-means 알고리즘 성능평가는 391MB 크기의 NASA의 위성 데이터 중에서 해조류 데이터[7]를 이용하여 수행하였다.

4.2 Reduce Input Cache 성능평가

(그림 2)는 Reduce Input Cache에 대한 성능평가 결과를 나타낸다. Reduce Input Cache의 성능평가를 위하여 PageRank 알고리즘 및 Descendant 질의에 대한 HaLoop, HadoopX, Hadoop의 전체 수행 시간을 비교하였다. 성능평가 결과 HadoopX는 HaLoop과 유사한 성능을 보이고 있으며, Hadoop에 비해 약 24% 성능이 향상됨을 알 수 있다. 이는 HadoopX가 Reduce Input Cache를 통



(a) PageRank 알고리즘



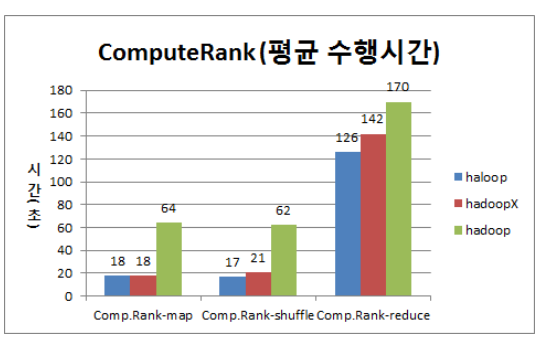
(b) Descendant 질의

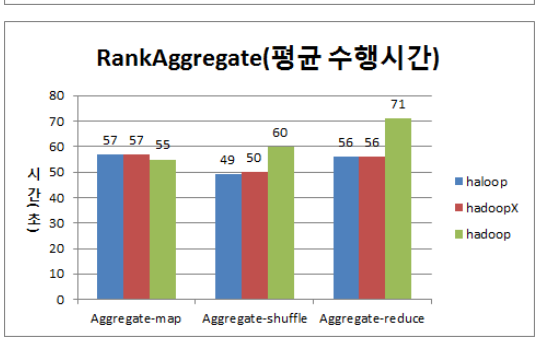
(그림 2) Reduce Input Cache 성능평가

해 shuffle 과정에서 불필요한 네트워크 전송 비용을 감소시켰기 때문이다.

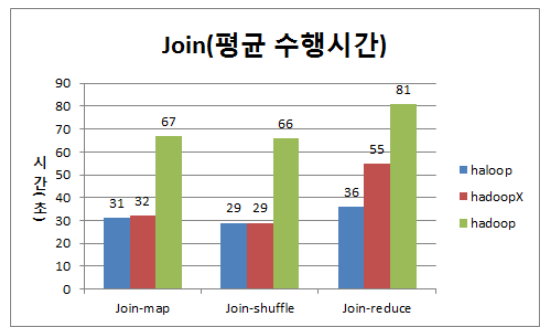
(그림 3)은 PageRank 알고리즘의 순환처리 작업(job)에 대한 세부 수행시간을 나타낸다. PageRank 알고리즘은 비순환처리 작업과 순환처리 작업으로 구성되며, 순환처리 작업은 ComputeRank와 Rank-Aggregate 작업으로 이루어진다. ComputeRank 작업은 이웃 노드와의 연결성을 통하여 각 노드의 rank를 계산하며, RankAggregate 작업은 이전 수행단계의 rank 연산 결과를 병합한다. 그림에서와 같이 HaLoop 및 HadoopX는 순환처리 작업 측면에서 Hadoop에 비해 빠른 처리 시간을 나타내는 것을 확인할 수 있다. 한편, 비순환처리 작업의 경우 세 시스템 모두 유사한 수행시간을 나타낸다.

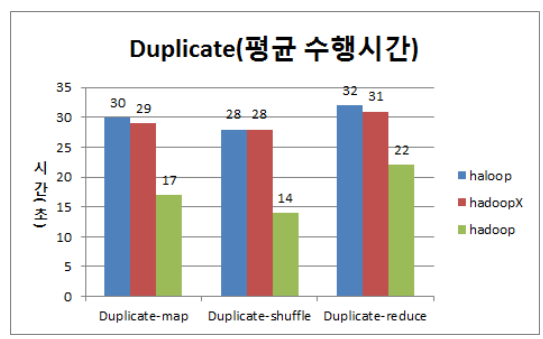
(그림 4)는 Descendant 질의의 순환처리 작업의 평균 수행시간을 나타낸다. Descendant 질의





(그림 3) PageRank 알고리즘의 순환처리 작업에 대한 세부 비교



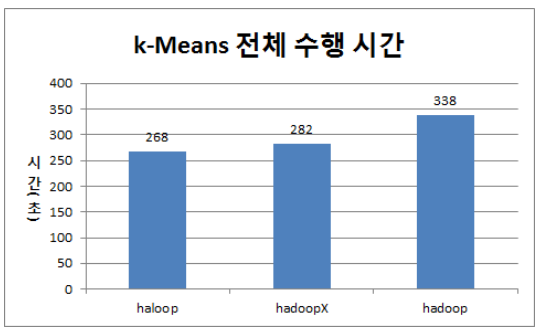


(그림 4) Descendant 질의의 순환처리 작업에 대한 세부 비교

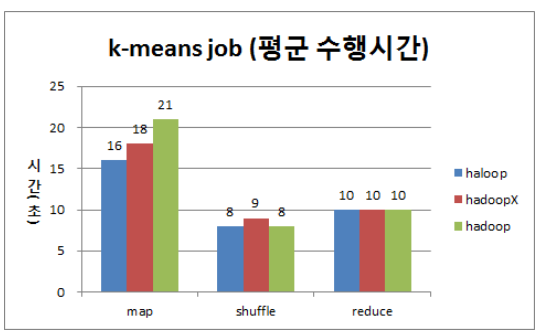
는 순환처리 작업인 Join과 Duplicate 작업으로 구성된다. Join 작업은 iteration 수에 따라 해당 노드의 이웃 노드를 검색하며, Duplicate 작업은 검색된 이웃 노드 중에서 중복된 노드를 제거한다. 성능평가 결과, Join 작업에서는 HaLoop과 HadoopX가 Hadoop에 비해 빠른 처리 속도를 나타내는 반면, Duplicate 작업에서는 Hadoop에 비해 낮은 처리 속도를 나타낸다. 이는 HaLoop과 HadoopX의 경우, 각 노드의 연결 정보를 invariant 데이터로 캐싱함으로써, 중복된 노드의 수가 Hadoop에 비해 증가하였기 때문이다.

4.3 Map Input Cache 성능평가

(그림 5(a))는 Map Input Cache에 대한 성능평가 결과를 나타낸다. Map Input Cache의 성능평가를 위하여 k-means 알고리즘에 대한 HaLoop,



(a) k-means 알고리즘 전체 수행시간



(b) k-means 순환처리 작업 세부 수행시간

(그림 5) Map Input Cache 성능평가

HadoopX, Hadoop의 전체 수행 시간을 비교하였다. 성능평가 결과 HadoopX는 HaLoop과 유사한 성능을 보이고 있으며, Hadoop에 비해 약 20% 성능이 향상되었음을 알 수 있다. (그림 5(b))는 k-means 알고리즘의 순환처리 작업에 대한 세부 수행시간을 나타낸다. 그림에서와 같이 Reduce 측면에서는 모두 동일한 성능을 나타내며, Map 측면에서 HaLoop과 HadoopX가 Hadoop에 비해 빠른 처리 속도를 나타낸다. k-means 알고리즘은 하나의 순환처리 작업으로 구성되며, 매 수행단계 마다 동일한 데이터를 입력받는다. 따라서 k-means 응용과 같이 매 수행단계마다 동일한 데이터를 입력받는 경우, Map Input Cache를 통해 불필요한 네트워크 전송 비용을 감소시킬 수 있다.

5. 결 론

최근 빅데이터에 대한 분석 기술이 각광받게 됨에 따라, 빅데이터를 수많은 서버들에 분산 저장하고 관리하기 위한 분산 데이터 처리 기법에 대한 연구가 활발히 진행되고 있다. 그러나, Hadoop 및 HaLoop과 같은 기존 연구들은 반복적인 수행을 위한 다양한 순환 처리 작업에 대해 효율적으로 지원하지 못하는 문제점이 존재한다. 이러한 문제점을 해결하기 위해, 본 논문에서는 다양한 순환 처리 응용을 위한 자동적인 invariant 데이터 검출을 지원하는 Hadoop MapReduce 기반 분산 시스템을 제안하였다. 제안하는 시스템은 Invariant 데이터 자동 검출 및 캐싱 기능을 지원함으로써, 비전문개발자를 위한 고사용성의 순환 처리 기법을 제공한다. 아울러, Map Input Cache 및 Reduce Input Cache를 통하여 불필요한 네트워크 전송 비용을 감소시킨다. 성능평가 결과, 제안하는 기법이 Hadoop에 비해 20% 의 성능 향상을 보임으로써, 효율적인 순환 처리 시스템임을 검증하였다.

향후 연구로는 과학기술 빅데이터의 효율적인 처리를 위해, SciDB와 같은 기존 과학기술 빅데이터 처리 시스템과의 연동을 수행하는 것이다.

참 고 문 헌

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", Operating Systems Design and Implementation (OSDI), pp. 137-150, 2004

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

[3] Y. Bu, B. Howe, M. Balazinska and M. D. Ernst, "HaLoop: Efficient Iterative Data Processing on Large Clusters", VLDB, 2010.

[4] J. Ekanayake et al., "Twister: A Runtime for Iterative MapReduce", ACM International Symposium on High Performance Distributed Computing (HPDC), 2010.
[5] L. Page et al., "The PageRank Citation Ranking: Bringing Order to the Web", Technical Report. Stanford InfoLab, 1999
[6] Stanford Large Network Dataset Collection, http://snap.stanford.edu/data/index.html
[7] NASA, OceanColor Data, http://oceandata.sci.gsfc.nasa.gov/MODISA/L3SMI/

저 자 약 력

홍 승 태
이메일 : dantehst@jbnu.ac.kr

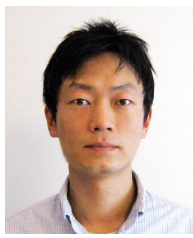
- 2008년 전북대학교 전자정보공학부 학사
- 2010년 전북대학교 컴퓨터공학과 석사
- 2010년~현재 전북대학교 전자정보공학부 박사과정
- 관심분야 : 공간 데이터베이스, 클라우드 컴퓨팅, 데이터 암호화, 사용자 접근제어

윤 민
이메일 : myoon@jbnu.ac.kr

- 2008년 전북대학교 컴퓨터공학과 학사
- 2011년 전북대학교 컴퓨터공학과 석사
- 2011년~현재 전북대학교 컴퓨터공학과 박사과정
- 관심분야 : 데이터베이스, 빅데이터, 분산처리 시스템

최 동 훈
이메일 : choid@kisti.re.kr

- 21981년 2월 서울대학교 계산통계학과 졸업(학사)
- 1983년 2월 한국과학기술원 전산학과 졸업(석사)
- 1989년 6월 Northwestern University 전산학과 졸업(박사)
- 1983년 2월~1986년 8월 한국증권전산㈜
- 1989년 8월~1992년 2월 한국국방연구원 선임연구원
- 1992년 3월~1999년 2월 동덕여자대학교 부교수
- 2005년 2월 현재 한국과학기술정보연구원 책임연구원
- 관심분야 : 데이터베이스, 고성능 컴퓨팅



조 희 승

이메일 : heeseung@jbnu.ac.kr

- 2000년 서강대학교 컴퓨터공학과학사
- 2010년 한국과학기술원 전산학과박사
- 2010년~현재 전북대학교 IT정보공학과 조교수
- 관심분야 : 가상화시스템, 클라우드 컴퓨팅, 빅데이터

장 재 우

이메일 : jwchang@jbnu.ac.kr

- 1984년 서울대학교 전가계산기공학과 학사
- 1986년 한국과학기술원 전산학과 석사
- 1996년~1997년 Univ. of Minnesota, Visiting Scholar
- 2003년~2004년 Penn State Univ., Visiting Scholar
- 1991년~현재 전북대학교 IT정보공학과 교수