Journal of The Korea Society of Computer and Information
Vol. 18, No. 11, November 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.11.23

MapReduce 작업처리시간 단축을 위한 선 정렬 기반 태스크 스케줄링 기법

박정효*, 김준상*, 김창현*, 이원주**, 전창호*

Pre-arrangement Based Task Scheduling Scheme for Reducing MapReduce Job Processing Time

Jung Hyo Park*, Jun Sang Kim*, Chang Hyeon Kim*, Won Joo Lee** Chang Ho Jeon*

요 약

본 논문에서는 MapReduce 작업처리시간을 줄일 수 있는 선 정렬 기반 태스크 스케줄링 기법을 제안한다. 태스크와 그 태스크가 처리할 데이터가 동일 노드에 존재하지 않으면 해당 태스크는 다른 노드로부터 데이터를 전송받아 처리한다. 이때 전송시간으로 인해 MapReduce의 작업처리시간이 증가하는 문제점이 발생한다. 이러한 문제점을 해결하기 위해 본 논문에서는 두 단계로 태스크를 스케줄링한다. 첫 번째 단계에서는 데이터 지역성이 높은 순으로 태스크를 노드 리스트에 정렬한다. 두 번째 단계에서는 데이터의 위치정보를 이용하여 태스크들이 데이터 지역성을 높일 수 있도록 교환하여 스케줄링한다. 본 논문에서는 제안한 스케줄링 기법의 성능평가를 위해 소규모 Hadoop 클러스터를 구현하여 실험하였다. 제안한 기법을 적용하였을 때 작업처리시간이 약 18% 감소하였으며 데이터가 저장된 노드에 할당되지 않은 태스크 수는 약 25% 감소하였다.

▸ Keywords : Hadoop, MapReduce, 데이터 지역성

Abstract

In this paper, we propose pre-arrangement based task scheduling scheme to reduce MapReduce job processing time. If a task and data to be processed do not locate in same node, the data should be transmitted to node where the task is allocated on. In that case, a job processing time increases owing to data transmission time. To avoid that case, we schedule tasks into two steps. In the first step, tasks are sorted in the order of high data locality. In the second step, tasks are exchanged to improve their data localities based on a location information of data. In performance evaluation,

•제1저자 : 박정효 •교신저자 : 이원주
•투고일 : 2013. 10. 23, 심사일 : 2013. 11. 2, 게재확정일 : 2013. 11. 20.
* 한양대학교 컴퓨터공학과(Dept. of Computer Science & Engineering, Hanyang University ERICA Campus)
** 인하공업전문대학 컴퓨터정보과(Dept. of Computer Science, Inha Technical College)

we compare the proposed method based Hadoop with a default Hadoop on a small Hadoop cluster in term of the job processing time and the number of tasks sorted to node without data to be processed by them. The result shows that the proposed method lowers job processing time by around 18%. Also, we confirm that the number of tasks allocated to node without data to be processed by them decreases by around 25%.

▸ Keywords : Hadoop, MapReduce, Data Locality

I. 서 론

스마트 기기의 보급 확산, 소셜 네트워크의 활성화, 마이크로소프트의 Azure[1], KT의 Ucloud[2], Google App Engine[3]과 같은 클라우드 서비스의 확대로 사용자들은 방대한 데이터를 생산하고 있으며, 그 규모는 기하급수적으로 증가하고 있다. 구글에서는 이와 같은 대용량 데이터를 처리하기 위해 MapReduce를 제안하였다[4]. MapReduce를 사용할 경우, 개발자는 시스템 내부의 데이터 분할이나 노드 간의 통신을 직접 구현할 필요가 없다. 따라서 병렬/분산 시스템에 대한 프로그래밍 경험이 없는 개발자도 MapReduce를 사용하기 쉽다[5]. 또한 MapReduce는 내고장성을 보장하기 때문에 신뢰할 수 없는 컴퓨터로 구성된 클러스터에서도 병렬처리를 할 수 있는 특징이 있다.

아파치에서도 Hadoop의 일부로서 MapReduce를 계속 발전시키고 있다. 아파치의 Hadoop은 오픈소스 기반 플랫폼으로 클라우드 컴퓨팅을 위한 플랫폼이다. Hadoop의 MapReduce는 그림 1과 같이 한 개의 Job Tracker와 여러 개의 Task Tracker로 구성된다.

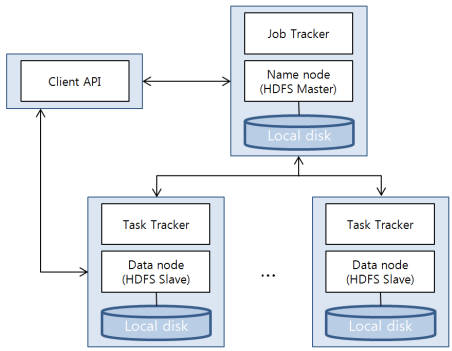


그림 1. MapReduce 구조
Fig. 1. MapReduce Architecture

그림 1에서 Job Tracker는 Client API로부터 작업 요청을 받으면 작업 명령을 Task Tracker들에게 분산시켜 수행한다. Task Tracker는 클러스터 내의 데이터 노드에 위치하고 있으며, HDFS(Hadoop Distributed File System)를 통해 태스크 처리를 위한 파일의 입출력을 수행한다.

Hadoop은 기본적으로 데이터의 복사본을 노드, 랙, 오프 랙(off-rack)에 저장하고 노드 레벨 데이터 지역성(locality), 랙 레벨 데이터 지역성, 오프 랙 레벨 데이터 지역성으로 구분하여 운용한다[6]. MapReduce는 이 세가지 데이터 지역성을 고려하여 처리할 데이터가 저장된 노드에 최대한 근접하도록 태스크를 할당한다. 이때 태스크는 처리할 데이터가 저장된 노드가 유휴 상태이면 태스크를 할당하고, 그렇지 않으면 동일한 랙의 유휴 노드에게 할당한다. 동일한 랙에 유휴 노드가 없다면 다른 랙의 유휴 노드에게 태스크를 할당한다. 만약 데이터가 저장된 노드에 태스크를 할당하지 않으면 처리할 데이터를 전송받아야 한다. 이것은 데이터의 전송을 위해 검색, 접근 등의 과정을 발생시켜 작업처리시간을 증가시키는 문제가 있다. 이러한 문제점을 해결하기 위해 본 논문에서는 불필요한 데이터 전송횟수를 줄임으로써 작업처리시간을 단축할 수 있는 선 정렬 기반 태스크 스케줄링 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 소개하고 3장에서는 본 논문에서 제안하는 선 정렬 기반 태스크 스케줄링 기법에 대하여 설명한다. 4장에서는 성능평가 및 분석결과를 제시하고 5장에서 결론을 맺는다.

II. 관련 연구

MapReduce의 작업처리시간을 줄이기 위해 다양한 연구가 진행되어 왔다. Cristina L. Abad[7]은 데이터 복사 정책을 통해 작업처리시간을 줄이는 방법을 제안했다. 데이터에 대한 접근횟수를 계산하고 접근이 빈번한 데이터의 복사본 수

를 증가시켜 데이터를 전송하는 오버헤드를 감소시킨다. 하지만 스토리지의 용량이 부족할 때만 복사본을 제거하기 때문에 불필요하게 많은 복사본을 유지하는 것이 단점이다.

J. Xie[8]은 데이터 배치 정책을 통해 작업처리시간을 줄이는 방법을 제안했다. 클러스터에 속한 각 노드들의 성능에 따라 데이터의 수를 다르게 배치한다. 성능이 좋은 노드에게 데이터를 많이 배치하여 더 많은 태스크를 처리하도록 한다. 이 방법은 모든 노드에게 데이터의 수를 다르게 배치하기 때문에 스토리지의 용량을 균형적으로 사용할 수 없다는 단점이 있다. 또한 데이터가 많이 배치된 노드에서 결함이 발생하면 데이터들의 대규모 전송이 발생하여 작업처리시간이 증가한다.

C. Tian[9]은 태스크의 유형을 분류하고 각 유형의 태스크를 빠르게 처리할 수 있는 노드에게 태스크를 할당하는 방법을 제안했다. CPU 중심의 태스크는 CPU 성능이 좋은 노드에 할당하고, I/O 중심의 태스크는 읽기/쓰기 속도가 빠른 노드에 할당한다. 이 방법은 태스크의 유형을 파악하기 위해 샘플 태스크를 실행시키기 때문에 샘플 태스크가 완료될 때까지 대기하는 시간이 필요하다는 단점이 있다.

X. Zhang[10]은 노드 간의 데이터 전송시간을 고려하여 작업처리시간을 줄이는 방법을 제안했다. 이 방법은 노드에서 실행 중인 태스크의 진행상황과 처리할 데이터의 전송시간을 비교하여 태스크를 할당할 것인지 보류할 것인지를 결정한다. 태스크 할당 시점에서 태스크를 할당할 것인지 보류할 것인지를 판별하기 때문에 한 번에 하나의 태스크만 데이터 지역성을 고려하게 된다. 또한 처리할 데이터의 전송시간은 미리 측정한 값을 사용하기 때문에 처리할 데이터의 크기를 변경하면 데이터 전송시간도 다시 측정해야 하는 단점이 있다.

Hadoop의 MapReduce도 태스크를 할당하는 시점에 데이터 지역성을 고려한다[11]. 데이터 노드의 태스크 할당 요청이 있을 경우에만 태스크의 데이터 지역성을 검사하기 때문에 모든 태스크의 데이터 지역성을 고려하지 못한다. 따라서 데이터 지역성을 검사하지 않은 태스크들은 할당 시점에서 한 번만 데이터 지역성을 향상시킬 수 있는 기회가 주어진다는 단점이 있다.

III. 선 정렬 기반 태스크 스케줄링

본 논문에서는 MapReduce의 작업처리시간을 줄일 수 있는 선 정렬 기반 태스크 태스크 스케줄링 기법을 제안한다. 제안하는 기법은 작업처리시간을 줄이기 위해 모든 태스크의 데이터 지역성을 높여 추가적인 데이터 전송 횟수를 줄인다. 이 스케줄링 기법은 두 단계로 태스크를 스케줄링 한다. 첫 번째 단계에서는 모든 태스크를 데이터 지역성이 높은 순으로 노드 리스트에 정렬한다. 두 번째 단계에서는 처리할 데이터를 가진 노드들과 정렬된 태스크들을 대상으로 태스크들이 더 높은 데이터 지역성을 갖도록 교환하여 스케줄링 한다. 이러한 태스크 스케줄링 과정에서 Time Left[12]와 데이터의 위치정보를 이용한다.

1. Time Left

Time Left는 태스크를 할당받은 데이터 노드가 남은 작업량을 처리하는데 걸리는 시간을 의미한다. Time Left는 데이터 노드가 작업을 완료하고 새로운 태스크를 요청하는 순서를 결정하는 역할을 한다. 그 값은 다음 수식 (1)로 구한다.

$$\text{Time Left} = (1-\text{Progress Score}) / \text{Progress Rate} \quad (1)$$

수식 (1)에서 Progress Score는 태스크가 처리할 데이터의 입력율을 의미한다. 데이터 블록 전체를 읽었으면 Progress Score는 1이 되고 20%를 읽었다면 0.2가 되는 것이다. Progress Rate는 데이터 노드의 단위 시간당 데이터 처리율을 의미한다. 예를 들어 데이터 노드가 3초 동안 한 블록의 데이터를 30% 읽었다면 이 데이터 노드의 Progress Rate는 10이 된다.

2. 데이터 위치정보

데이터 위치정보는 데이터 노드의 호스트 네임을 의미한다. Hadoop은 기본적으로 데이터를 여러 개의 블록으로 나누어 다수의 데이터 노드에 분산 저장한다. 이때 각 블록은 원본을 포함한 3개의 복사본을 가진다. 그 중 2개의 복사본은 동일한 랙에 저장하고 나머지 하나는 다른 랙에 저장한다. 따라서 태스크는 복사본의 위치에 따라 노드 레벨 데이터 지역성, 랙 레벨 데이터 지역성, 오프 랙 레벨 데이터 지역성 중 하나의 데이터 지역성을 만족한다. 데이터 위치정보는 태스크들이 갖는 데이터 지역성을 확인하는데 사용한다.

3. 제안하는 태스크 스케줄링 기법

제안하는 태스크 스케줄링 기법은 먼저 Time Left를 이용하여 데이터 노드들의 태스크 요청순서를 예측한다. 그리고 요청 순서에 따라 노드 레벨 데이터 지역성을 최대한 만족하도록 태스크를 맵핑한다. 일부 노드 레벨 데이터 지역성을 만족하지 못하는 태스크들의 데이터 지역성을 향상시키기 위해 맵핑된 태스크를 재검사한다. 데이터 지역성이 낮은 태스크들은 데이터 노드와 노드 레벨 데이터 지역성을 만족하도록 할

당순서를 교환한다. 먼저 교환할 태스크의 데이터를 저장한 데이터 노드를 검색한다. 검색된 데이터 노드가 이미 노드 레벨 데이터 지역성을 만족하도록 태스크가 맵핑된 상태이고, 그 데이터 노드의 태스크가 처리할 데이터가 노드 레벨 데이터 지역성을 만족하지 못한다면 이 두 태스크들을 교환한다. 교환된 두 태스크는 자신이 필요로 하는 데이터가 존재한 노드에게 할당되기 때문에 노드 레벨 데이터 지역성을 만족시킨다. 교환할 수 없는 태스크는 아직 태스크가 맵핑되지 않은 데이터 노드들에게 차례로 맵핑한다. 모든 태스크들의 할당순서가 결정되면 Job Tracker는 순서대로 Task Tracker에게 태스크를 할당한다.

그림 2는 선 정렬 기반 태스크 스케줄링 기법을 의사코드로 표현한 것이다.

```
waitTaskList;  // 노드에 할당되지 않은 태스크 리스트
nodeList;      // 클러스터에 연결된 노드 리스트

// 태스크 요청 순서대로 노드를 정렬
nodeList=NodeSort(nodeList);

// 노드 레벨 지역성을 만족하도록 태스크 맵핑
for i=0 to length(nodeList)
   node=nodeList.get(i);
   node.task=FindNodeLevelTask(nodeList, node,
                               waitTaskList);
end for

// 노드 레벨 지역성을 높이기 위한 태스크 교환
for i=0 to length(waitTaskList)
   task = waitTaskList.get(i);

   // 노드 리스트에서 태스크가 맵핑된 노드를 검색
   nodeWithTask = FindNodeWithTask(nodeList, task);

   // 태스크 수행에 필요한 데이터를 가진 노드 검색
   nodeWithData = FindNodeWithData(nodeList, task);

   timeLeftDataNode = TimeLeft(nodeWithData);
   timeLeftTaskNode = TimeLeft(nodeWithTask);

   if timeLeftDataNode < timeLeftTaskNode then
      Swap(task, nodeWithData.task);
   end if
end for
```

그림 2. 제안한 태스크 스케줄링 기법의 의사코드
Fig. 2. Pseudo-code of Proposed Task Scheduling Scheme

그림 2에서 FindNodeLevelTask()함수는 태스크 선 정렬을 수행한다. 이 함수는 임의 노드에 태스크를 할당하여 노드 레벨 데이터 지역성을 만족하면 해당 태스크를 반환한다. 그렇지 않을 경우 nodeList에서 태스크가 맵핑되지 않은 어떤 노드에 맵핑되어도 노드 레벨 데이터 지역성을 만족시키지 못하는 임의의 태스크를 반환한다. 그리고 노드 당 하나의 태스크만 할당하기 위해 한번 반환한 태스크는 재반환하지 않는다. 태스크 교환 과정에서는 태스크 수행에 필요한 데이터가 저장된 노드의 Time Left보다 태스크가 맵핑된 노드의 Time Left가 크고 처리해야하는 데이터가 있다면 두 노드에 맵핑된 태스크를 교환하여 데이터 지역성을 향상시킨다.

IV. 성능평가

제안한 태스크 스케줄링 기법은 기존의 방법[7, 8, 9, 10]과 달리 태스크 할당순서를 변경하여 MapReduce의 작업처리시간을 줄인다. 본 논문에서는 성능평가 척도로 Non-node Local Task 수와 작업처리시간을 사용한다. Non-node Local Task 수는 노드 레벨 데이터 지역성을 만족시키지 못하는 태스크 수를 의미한다. 본 논문에서는 전체적으로 노드 레벨 데이터 지역성을 높이기 위해 태스크를 교환할 때 데이터 전송에 따른 오버헤드를 알아보기 위해 Non-node Local Task 수를 사용한다. 기존의 방법[7, 8, 9, 10]에서 성능평가 척도로 사용한 작업처리시간은 전체 작업완료시간을 의미한다.

본 논문에서는 제안한 기법의 성능을 평가하기 위해 소규모 Hadoop 클러스터를 구현하여 사용한다. Hadoop 클러스터에서 데이터 정렬 프로그램인 Terasort[13]를 실행하여 Non-node Local Task 수와 작업처리시간을 구한다.

1. 실험 환경

본 논문에서 사용한 소규모 Hadoop 클러스터의 시스템 환경은 표 1과 같다.

표 1. 시스템 환경
Table 1. System Environment

노드 수	네임노드 : 1대
	데이터 노드 : 5 ~10대
노드성능	CPU : Intel Core 2 Duo E7500 2.93GHz, 2.94GHz
	Memory : 2GB
	Storage : 300GB
네트워크 대역폭	100Mbps
운영체제	Ubuntu 12.04 LTS
Hadoop	Hadoop-1.0.4

Hadoop 클러스터는 1대의 네임노드와 10대의 데이터 노드로 구성한다. 각 노드의 운영체제는 Ubuntu 12.04 LTS이고, Hadoop 버전은 1.0.4이다. 또한, 각 노드는 외부 트래픽의 간섭을 없애기 위해 독립된 네트워크상의 동일한 스위치에 연결한다. 기본 Hadoop은 Haoop 버전 1.0.4에서 제공하는 FIFO 스케줄러를 사용한다. FIFO 스케줄러는 Job Tracker에 있는 스케줄링 알고리즘으로 사용자가 Hadoop을 따로 수정하지 않으면 기본으로 실행된다. FIFO 스케줄러는 작업 대기 큐에서 가장 먼저 입력된 태스크를 수행시키는 동작을 반복한다.

표 2는 성능평가에 사용한 데이터 정보를 나타낸 것이다.

표 2. 처리할 데이터 정보
Table 2. Data Information for processing

입력 데이터 크기	Map 태스크 수	Reduce 태스크 수	블록 크기
1GB	16	10	64MB
2GB	32	10	64MB
3GB	48	10	64MB

Teragen[13]을 사용해 한 블록의 크기가 64MB인 다양한 크기의 데이터를 생성했다. 데이터의 복사본 수에 의한 데이터 지역성 영향을 없애기 위해 복사본의 수는 3개로 고정했다.

2. 결과 분석

Hadoop 클러스터의 데이터 노드 수가 5일 때, 입력 데이터의 크기에 따른 Terasort 프로그램을 실행하고 Non-node Local Task 수를 측정한 결과는 그림 3과 같다.

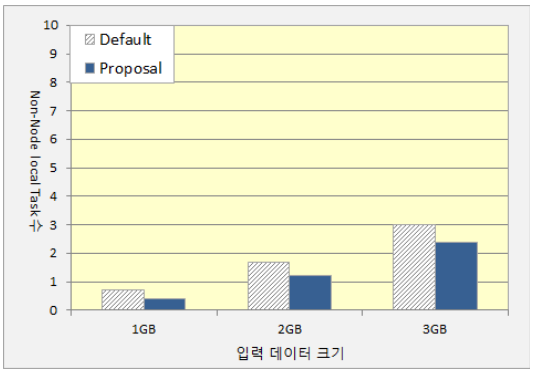


그림 3. Non-node Local Task 수 (데이터 노드 수=5)
Fig. 3. The Number of Non-node Local Task(Data node=5)

그림 3에서 입력 데이터 크기가 커질수록 기본 Hadoop과 제안한 기법의 Non-node Local Task 수가 증가함을 볼 수 있다. 특히, 입력 데이터 크기가 3GB일 때 기본 Hadoop과 제안하는 방법 모두 Non-node Local Task의 수가 급증하는 것은 MapReduce의 Straggler 때문이다. MapReduce의 Job Tracker는 Task Tracker에게 주기적으로 Heart Beat 신호를 보낸다. 이 Heart Beat 신호에 대한 응답이 없거나, 태스크의 작업이 60초 이상 진행되지 않으면 그 Task Tracker가 있는 데이터 노드를 Straggler로 선정하고 Task Tracker에서 실행중인 작업을 다른 노드에서 실행하게 한다. 이때 새롭게 실행되는 태스크의 경우 대부분 Non-node Local Task로 실행되기 때문에 Non-node Local Task의 수가 증가한다. 이러한 Straggler에 의한 Non-node Local Task 수의 증가에도 불구하고 제안한 기법의 Non-node Local Task의 수는 기본 Hadoop에 비해 약 20% 감소함을 볼 수 있다.

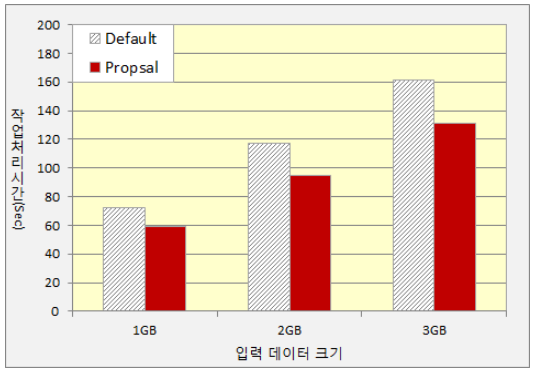


그림 4. 작업처리시간(데이터 노드=5)
Fig. 4. Job Processing Time(Data node=5)

그림 4는 데이터 노드 수가 5일 때, 입력 데이터의 크기에 따른 작업처리시간을 측정한 결과이다. 입력 데이터의 크기가 커질수록 전체 작업량이 증가하기 때문에 작업처리시간 또한 선형적으로 증가함을 볼 수 있다. 제안하는 기법은 Non-node Local Task 수를 감소시켜 데이터 전송 횟수를 줄임으로써 작업처리시간을 단축한다. 제안한 기법의 작업처리시간은 기본 Hadoop에 비해 약 18% 감소했음을 볼 수 있다.

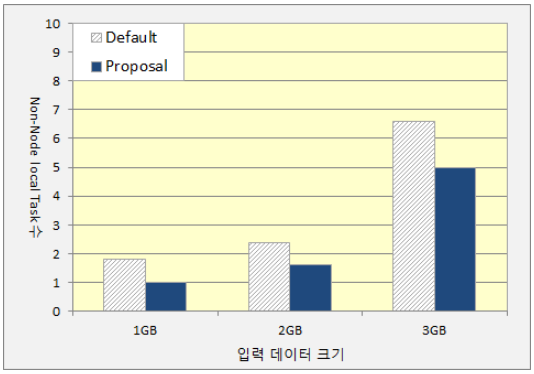


그림 5. Non-node Local Task 수 비교(데이터 노드=10)
Fig. 5. The Number of Non-node Local Task(Data node=10)

그림 5는 Hadoop 클러스터의 데이터 노드 수가 10일 때, Non-node Local Task 수를 측정한 결과이다. 그림 5를 살펴보면 제안한 기법의 Non-node Local Task의 수는 기본 Hadoop에 비해 약 25% 감소함을 볼 수 있다. 또한, 데이터 노드 수가 5일 때 결과인 그림 3에 비해 Non-node Local Task 수가 더 많음을 볼 수 있다. 이것은 입력 데이터 배치와 관련이 있다. 데이터 노드의 수가 증가하면 노드 당 데이터의 수는 감소하기 때문에 노드 레벨 데이터 지역성을 만족시킬 가능성이 낮아진다. 따라서 Non-node Local Task 수는 증가하게 된다.

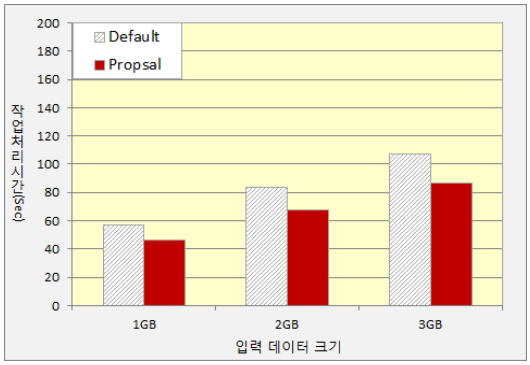


그림 6. 작업처리시간(데이터 노드=10)
Fig. 6. Job Processing Time(Data node=10)

그림 6은 데이터 노드의 수가 10일 경우, 작업처리시간을 측정한 결과이다. 제안한 기법의 작업처리시간은 기본 Hadoop에 비해 약 18% 감소했음을 볼 수 있다. 그림 4와 그림 6을 비교해보면 데이터 노드의 수=10일 경우 데이터 노드의 수=5일 경우에 비해 작업처리시간이 감소함을 알 수 있다. 데이터 노드의 수=10일 경우, 하나의 작업을 분산 처리하는 노드가 많기 때문에 작업처리시간이 짧아진다.

V. 결 론

본 논문에서는 MapReduce의 작업처리시간을 줄일 수 있는 선 정렬 기반 태스크 스케줄링 기법을 제안하였다. 이 기법은 클러스터에 속한 데이터 노드들의 태스크 진행상황을 평가하고 태스크 요청순서를 예측하여 태스크를 스케줄링한다. 특히, 데이터 지역성이 낮은 태스크들은 노드 레벨 데이터 지역성을 만족시키도록 교환하여 태스크들의 데이터 지역성을 높인다. 태스크들의 데이터 지역성이 높아지면 데이터 전송 횟수가 감소함에 따라 작업처리시간도 함께 감소한다.

본 논문에서는 소규모 Hadoop 클러스터를 구현하여 제안한 스케줄링 기법의 성능을 평가하였다. 그리고 제안한 스케줄링 기법의 성능이 기본 Hadoop에 비해 우수함을 검증하였다. 제안한 기법에서는 태스크 교환 과정을 통하여 전체 태스크들의 노드 레벨 데이터 지역성을 향상시킴으로써 노드 레벨 데이터 지역성을 만족시키지 못하는 태스크 수를 약 25% 감소시켰다. 이러한 결과는 데이터 전송 횟수를 줄이기 때문에 작업처리시간을 단축하는 효과도 함께 얻을 수 있었다. 제안한 기법은 기본 Hadoop에 비해 작업처리시간을 약 18% 단축하였다.

참고문헌

[1] Microsoft Azure, http://www.microsoft.com/windowsazure/Whitepapers/introducingwindowsazureplatform.

[2] KT Ucloud,. http://home.ucloud.olleh.com/guide/guide.kt.

[3] Google App Engine, https://developers.google.com/appengine/docs/whatisgoogleappengine.html.

[4] K. Lee, H. Choi, B. Moon, Y. Lee, and Y. Chung, "Parallel Data Processing with MapReduce : A Survey," In Proceedings of ACM SIGMOD, Vol . 4, Issue 3, pp. 11-20, Dec. 2012.

[5] J. Dean and S. Ghemawat, "MapReduce :Simplified Data Processing on Large Clusters," In Proceeding of the 6th USENIX Symposium on

Operating Systems Design and Implementation, pp. 107-113, Jan. 2008.

[6] J. Lee, H. Yu, E. Lee, "Data Replication Technique for Improving Data Locality of MapReduce," In Proceeding of the KIISE Korea Computer Congress 2012, Vol. 39, No. 1(A), pp. 218-220, Jun. 2012.

[7] C. L. Abad, Y. Lu, and R. H. Campbell "DARE:Adaptive Data Replication for Efficient Cluster Scheduling," IEEE CLUSTER, 2011 IEEE International Conference, pp. 159-168, Sep. 2011.

[8] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares and X. Qin, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters," The 24th IEEE International Symposium on Parallel&Distributed processing:Workshops and Phd Forum, pp. 1-9, April 2010.

[9] C. Tian, H. Zhou, Y. He, and L. Zha, "A Dynamic Scheduler for Heterougeneous Workloads," The 8th International Conference on Grid and Cooperative Computing, pp. 218-224, Aug. 2009.

[10] X. Zhang, Y. Feng, S. Feng, J. Fan and M. Zhong, "An Effective Data Locality Aware Task Scheduling Method for MapReduce Framework in Heterogeneous Environments," In Proceedings of the Internatinal Conference on Cloud and Service Computing, pp. 235-242, Dec. 2011.

[11] Z. Guo, G. Fox, and M. Zhou, "Investigation of Data Locality in MapReduce," In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cludster, Cloud and Grid Computing, pp.419-426, May 2012.

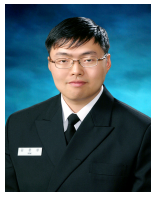
[12] M. Zaharia, A. Konwinski, A. D. Joseph, R.Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," In Proceedings of 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI, Vol. 8, No. 4, pp. 29-42, Dec. 2008.

[13] O. O'Malley, "TeraByte Sort on Apache Hadoop", Yahoo, available online at: http://sortbenchmark.org/Yahoo-Hadoop.pdf, May 2008.

저 자 소 개

박 정 효

2012: 경일대학교
컴퓨터공학부 공학사.
현 재: 한양대학교
컴퓨터공학과 석사과정.
관심분야: 클라우드 컴퓨팅, 병렬처리,
성능분석
Email:whdkgod55@hanyang.ac.kr



김 준 상

2003: 한양대학교
전자컴퓨터공학부 공학사.
2005: 한양대학교
컴퓨터공학과 공학석사.
2008-2012: 해군사관학교
전산과학과 전임강사
현 재: 한양대학교
컴퓨터공학과 박사과정
관심분야: 병렬처리, 성능분석,
그리드 컴퓨팅,
클라우드 컴퓨팅
Email : kjspbe@hanyang.ac.kr

김 창 현

2008: 경일대학교
컴퓨터공학부 공학사.
2010: 한양대학교
컴퓨터공학과 공학석사.
현 재: 한양대학교
컴퓨터공학과 박사과정
관심분야: 센서네트워크, 성능분석,
그리드컴퓨팅,
클라우드 컴퓨팅
Email : ctcquatre@hanyang.ac.kr

이 원 주

1989: 한양대학교
전자계산학과 공학사.
1991: 한양대학교
컴퓨터공학과 공학석사.
2004: 한양대학교
컴퓨터공학과 공학박사.
현 재: 인하공업전문대학
컴퓨터정보과 교수.
관심분야: 병렬처리시스템,
모바일컴퓨팅, 성능분석,
클라우드컴퓨팅
Email: wonjoo2@inhatc.ac.kr

전 창 호

1977: 한양대학교 전자공학과 학사.
1982: Cornell University
컴퓨터공학과 석사.
1986: Cornell University
컴퓨터공학과 박사.
1977-1979: 전자통신연구소 연구원.
현 재 : 한양대학교
전자컴퓨터공학부 교수.
관심분야: 병렬처리시스템, 성능분석,
Grid컴퓨팅,
클라우드컴퓨팅
Email: chj5193@hanyang.ac.kr