JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 19, No. 9 : 2029~2036 Sep. 2015

하둡 성능 향상을 위한 VPT 개발 연구

양일등[1] · 김성열[2*]

A Development Study of The VPT for the improvement of Hadoop performance

Ill Deung, Yang[1] · Seong Ryeol, Kim[2*]

[1]Department of Computer & Information Engineering, Cheongju University, Cheongju-si 298, Korea
[2*]Department of Computer & Information Engineering, Cheongju University, Cheongju-si 298, Korea

요 약

하둡 MR(MapReduce)는 매퍼(Mapper)의 출력을 리듀서(Reducer)의 입력으로 전달하기 위해 파티션 함수(Partition Function)을 사용한다. 파티션 함수는 키에서 해쉬 값을 계산한 후 리듀서 개수로 나머지 연산을 수행하여 대상 리듀서를 결정한다. 기존 파티션 함수는 키의 편중도에 민감하여 잡이 균등하게 배분될 수 없었다. 잡이 균등하게 배분되지 못하면 특정 리듀서들의 처리 수행 시간이 길어져 전체 분산 처리 수행 성능에 영향을 주게 된다. 이에 본 논문은 VPT(Virtual Partition Table)을 제안하고 편중도가 심한 데이터에 VPT을 적용하여 실험을 수행 하였다. 적용된 VPT는 기존 파티션 함수와 대비하여 평균 3초 정도 성능향상이 발생하였으며, 데이터 처리량이 증가할수록 성능 향상 폭이 증가할 것으로 예상된다.

ABSTRACT

Hadoop MR(MapReduce) uses a partition function for passing the outputs of mappers to reducers. The partition function determines target reducers after calculating the hash-value from the key and performing mod-operation by reducer number. The legacy partition function doesn't divide the job effectively because it is so sensitive to key distribution. If the job isn't divided effectively then it can effect the total processing time of the job because some reducers need more time to process. This paper proposes the VPT(Virtual Partition Table) and has tested appling the VPT with a preponderance of data. The applied VPT improved three seconds on average and we figure it will improve more when data is increased..

키워드 : 하둡, 맵리듀스, 파티션함수

Key word : Hadoop, MapReduce, Partition Function

Received 02 July 2015, **Revised** 27 July 2015, **Accepted** 11 August 2015

*** Corresponding Author** Seong Ryeol, Kim(E-mail:srkim@cju.ac.kr, Tel:+82-43-229-8490)
Department of Computer & Information Engineering, Cheongju University, Cheongju-si 298, Korea

Open Access **http://dx.doi.org/10.6109/jkiice.2015.19.9.2029** print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright Ⓒ The Korea Institute of Information and Communication Engineering.

Ⅰ. 서 론

2004년 구글에서 구글MR(MapReduce)[1]를 발표한 이후 빅데이터 관련 솔루션들이 주목받기 시작했다. 구글 MR는 LISP의 Functional Programming[2]개념을 분산처리에 적용한 것으로 이후 많은 솔루션들이 개발되었다. 이중 자바 기반의 오픈소스인 하둡MR[3]이 구현되었으며 현재 페이스북[4] 야후 등에서 3,000노드 이상 운영되는 대형 클러스터에 사용되고 있다.

하둡MR은 오픈소스 구조로 분산처리가 필요한 사용자들에게 많은 관심을 받고 있으며 동작과정 이해도 용이하다. 하둡MR은 데이터가 저장되어 있는 노드에서 매퍼(Mapper)가 데이터를 처리하면 자동으로 리듀서(Reducer)로 전달되어 집계(Aggregate)처리를 수행한다. 이처럼 하둡MR은 사용자가 매퍼와 리듀서만 제공하면 나머지 분산처리에 관한 모든 것을 자동으로 처리한다.

이러한 동작 과정에는 파티션(Partition) 과정이 포함된다. 파티션 과정은 매퍼의 출력을 리듀서에게 전달하기 위해 대상 리듀서를 결정하는 것이다. 분산처리 성능은 사용자의 작업 시작 요청부터 결과확인 까지 모든 시간이 포함된다. 따라서 좋은 처리 성능을 얻기 위해서는 리듀서들에게 균등한 작업 배분을 해야 하며, 특정 리듀서에게 많은 작업이 할당되면 전체 작업 완료 시간에 영향을 주게 된다.

본 논문은 하둡MR 성능 향상을 위해 VPT(Virtual Partition Table)을 제안하고 이를 적용하여 처리 성능이 향상됨을 보인다.

Ⅱ. 관련연구

2.1. 하둡MR

하둡MR[3]는 구글MR를 기반으로 자바 기반 오픈소스로 구현되었다. 대용량 데이터가 분산 저장된 병렬 클러스터 환경에서 효과적으로 데이터를 처리할 수 있다. 하둡MR는 구글MR과 용어와 개념이 매우 유사하며 구글MR과 동일하게 마스터/슬레이브 구조를 가진다. 마스터는 태스크를 할당하고 제어하는 잡트래커(Jobtracker)이며 실제 태스크를 수행하는 슬레이브는 태스크트래커(Tasktracker)이다. 태스크트래커가 수행하는 태스크는 맵 태스크와 리듀스 태스크로 구분된다. 파일은 Split 블록으로 나누어져 GFS(Google File System)[5]와 유사한 HDFS(Hadoop Distributed File System)[6]에 저장된다. 저장된 데이터는 맵 태스크에 의해 처리된다. 맵 태스크가 출력한 중간 데이타는 리듀스 태스크의 입력이 되어 미리 정의된 규칙에 따라 원하는 데이터를 필터링하고 최종결과는 HDFS에 저장된다. 그림 1은 하둡MR의 동작과정을 나타낸다.

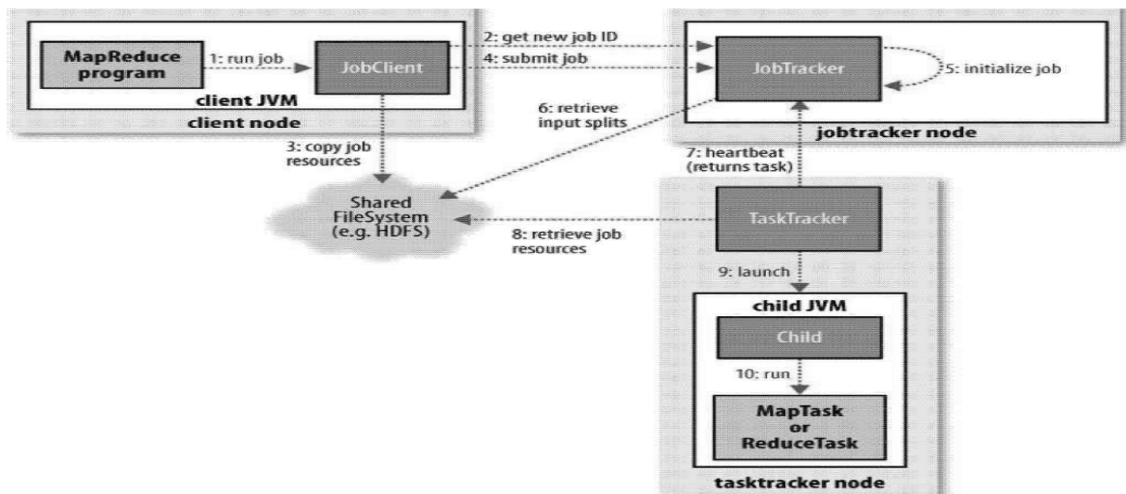

그림 1. 하둡 MR 동작과정[3]
Fig. 1 Processing steps of Hadoop MR[3]

사용자는 MR프로그램을 JobClient에 제출하고 Job Client는 공유 파일시스템에 Job을 저장한다.

저장된 잡(Job)은 잡트래커에 의해 관리 되며 각 노드에 설치된 태스크트래커에 의해 처리된다. 처리된 결과는 HDFS에 저장되고 사용자는 최종 결과를 확인한다.

2.2. MR Partition Function

맵퍼의 출력을 리듀스의 입력으로 전달하기 위해 하둡MR은 파티션 함수를 사용한다. 파티션 함수가 실행되는 곳은 그림 2에서 partition 부분이다.

파티션 함수는 매퍼가 출력한 데이터를 처리할 리듀서를 결정하기 위해 매퍼에게 입력되는 키 값에서 해쉬값을 계산하여 그 값을 전체 리듀서 개수로 나머지 연산한다. 따라서 입력된 키 값의 분포가 매우 중요 하며 키 값의 분포가 한쪽으로 편중될 경우 특정 리듀서에게 더 많은 작업이 할당된다[7, 8].

하둡 파티션 함수[1, 3]에 사용되는 수식은 다음 수식 1과 같다.

$$P = hash(key) \bmod R \quad (1)$$

P : The Result of Partition Function
key : The Output of a Mapper
R : The Number of Reducers

key는 매퍼에게 입력되는 키, R은 리듀서 개수, P는 결과 값이다. 수식 1은 주어진 키를 해쉬 함수의 입력으로 하고 그 출력을 R로 나머지 연산한 후 결과 P를 리턴한다.

P는 대상 리듀서의 번호가 된다. R 값은[3] 하둡 설정 파일에 설정된다. 따라서 기존 파티션 함수는 키값에 따라 특정 리듀서에게 많은 작업이 편중되어 할당될 수 있다.

Ⅲ. VPT(Virtual Partition Table)제안

3.1. VPT 제안

하둡MR 파티션 함수는 수식 1을 사용하여 대상 리듀서를 결정한다. 즉 R값이 작을수록 특정 P값이 집중될 확률이 높아지며 P값을 평균적으로 분포시키기 위해서는 R값을 크게 증가 시켜야 한다. 하지만 R값을 크게 증가 시키면 리듀서가 출력하는 결과 파일의 개수가 증가 하여 후처리 작업이 필요하다. 따라서 실제 R값을 유지하면서 P값을 평균적으로 분포 시키는 방법이 필요하다.

이에 본 논문은 VPT을 제안하며, 제안하는 VPT는 가상 R을 사용하여 특정 리듀서에 대한 집중도를 낮게 한다.

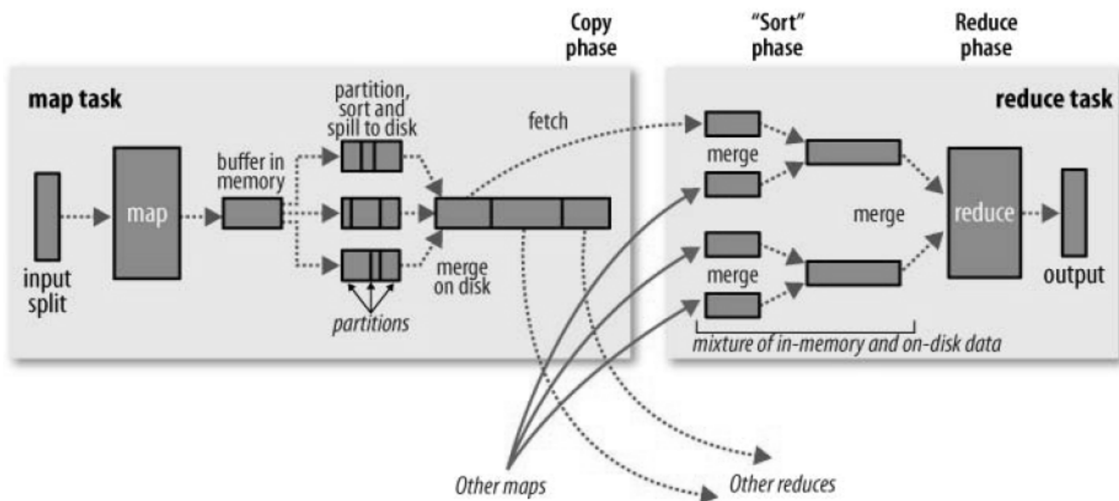

그림 2. 하둡 내부 동작 구조[3]
Fig. 2 Internal processing pathways of Hadoop[3]

VPT는 실제 파티션 테이블의 데이터가 저장된 가상 파티션 테이블을 사전 구축하고 매퍼가 VPT 파티션 함수를 사용하여 파티션을 수행한다. 이를 통해 R값을 증가 시키지 않고 대상 리듀서를 결정할 수 있다. VPT의 장점은 다음과 같다.

3.1.1. 사전 구축을 통한 런타임 시간 감소

VPT 데이터 구축은 하둡MR에 잡을 제출 전에 수행된다. 사전 구축된 VPT 데이터는 자바 컴파일러로 컴파일되어 JAR 파일에 포함된 후 압축되어 하둡MR에 잡으로 제출된다. 이후 동일 잡을 실행할 때 VPT 데이터를 재구축 할 필요 없이 기존 JAR 파일을 이용하여 잡을 처리한다.

이것은 런타임시에 제출된 잡의 개별 매퍼들이 각각 VPT 데이터를 생성하지 않고 사전 구축된 데이터를 지속적으로 사용 할 수 있는 장점을 제공한다. 대용량 데이터의 특성은 동일 패턴의 데이터가 반복적으로 발생되어 패턴을 분석한 이후에는 분석된 패턴을 지속적으로 사용 할 수 있다. 따라서 VPT 데이터를 사전 구축하여 동일한 패턴의 대용량 데이터에 지속적으로 적용할 수 있다.

또한 VPT 데이터 구축에 전체 샘플 데이터를 사용하지 않고 시스템에 설정된 Split 크기만큼 사용하기 때문에 VPT 데이터 생성 시간도 크지 않다. 키의 분할 여부에 따라 약간의 차이는 발생하지만 하둡 기본 Split 크기인 64M 데이터를 처리하는 시간은 약 7초 이하이다.

3.1.2. VPT 데이터 직접 참조

VPT는 데이터를 자바 클래스 형태로 JAR파일에 포함하여 배포한다. 따라서 각 매퍼는 JAR 파일에 포함된 VPT 데이터를 자바 소스 상에서 직접 참조 될 수 있는 장점을 제공하며 추가적인 네트워크 요구가 발생하지 않는다.

3.2. VPT 동작 설명

VPT는 전처리 단계와 VPT 데이터를 참조하는 VPT 파티션 함수 사용 단계로 나뉜다. 전처리 단계는 JAR파일을 하둡MR에 배포하기 전 일회 수행되며 데이터 유형이 변경되지 않으면 재구축 필요는 없다. 파티션 함수 사용 단계는 각 매퍼에서 VPT 데이터를 참조하여 R을 결정하며, 각 단계는 다음과 같다.

3.2.1. 전처리 단계

첫 번째는 전처리 단계로 샘플 데이터를 로드 하여 VPT 데이터를 생성한 후 자바 파일에 배열 형태로 저장한다. 그림 3은 전처리 단계를 나타낸 그림으로 전체적인 흐름은 HDFS에서 샘플 데이터를 패치한 후 기존 파티션 함수를 사용하여 VPT 데이터를 생성한다. 생성된 VPT 데이터는 자바 파일로 저장된다. 파티션된 데이터 저장 크기는 VPT 데이터 저장 크기와 같아야 한다.

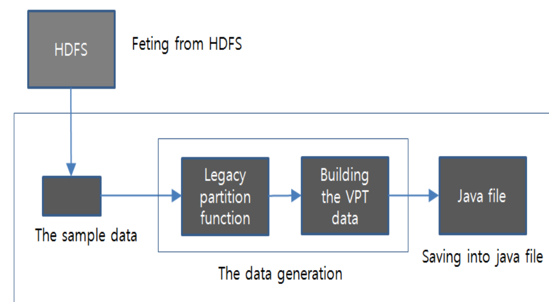

그림 3. VPT 전처리 단계
Fig. 3 VPT preprocessing parse

VPT 데이터는 자바 소스에서 직접 참조를 위해 static으로 선언 되어야 하며 자바 컴파일러가 컴파일 할 수 있는 최대 static 배열 크기는 약 1만개 이다. 따라서 샘플과 VPT 크기는 1만개로 설정한다. 이렇게 설정된 데이터는 각 키에 대한 HIT개수가 누적되어 저장된다. 이후 실제 리듀서 개수로 생성된 파티션 테이블에 데이터 값을 누적시키며 가장 적은 데이터 값을 갖는 파티션 테이블의 인덱스를 VPT에 저장한다. 이것은 데이터 전체크기만큼 수행된다. 저장된 VPT는 실제 리듀서의 인덱스가 저장되고 이후 VPT 파티션 함수를 통해 리듀서를 결정하게 된다.

다음은 VPT를 생성하는 생성코드의 일부이다.

```
int pt[ ] = new int[pt_cnt];
int sample[ ] = new int[VPT_SIZE];
int VPT[ ] = new int[VPT_SIZE];
for( int i = 0 ; i < sample.length ; i++ ) {
 sample[i] = 1;
```

```
}
for( int i = 0 ; i < sample.length ; i++ ) {
 sample[key[i].hashCode()%sample.length]++;
}
for( int i = 0 ; i < node_cnt ; i++ ) {
 pt[i] = 0;
}
int pt_idx = 0;
for( int i = 0 ; i < sample.length ; i++ ) {
 for( int j = 0 ; j < pt.length ; j++ ) {
  if( pt[j] < pt[pt_idx])
   pt_idx = j;
 }
  pt[pt_idx] = pt[pt_idx] + sample[i];
 VPT[i] = pt_idx;
}
```

pt_cnt :The number of real reducers
VPT_SIZE :The size of VPT array
key : The array of sample data

여기서 pt 배열 변수는 R의 누적 카운트를 저장 하는 배열, sample 배열 변수는 기본 파티션 함수의 결과 데이터에 대한 누적 카운트를 저장하는 배열, 그리고 VPT 변수는 최종결과가 저장되는 배열이다. VPT 배열의 값들은 자바 static 배열로 변환되어 자바 파일에 최종 저장된다. VPT 알고리즘 순서는 sample배열을 1로 초기화시키고 기존 파티션 함수를 실행시켜 각 값에 대한 누적값을 저장한다.

이때 기존 파티션 함수의 R은 sample 배열의 크기이다. 키 데이터를 저장하는 key배열에서 각 데이터에 대해 파티션 함수를 수행 하고 개별 키에 대한 HIT수를 sample 배열에 누적시킨다. 누적된 sample 배열 정보를 pt 배열에 누적시키며 가장 작은 값을 가지는 pt의 인덱스를 VPT 배열에 최종 저장한다. 마지막으로 VPT 배열 값을 기준으로 자바 staitc 배열 소스를 생성하고 자바 컴파일러로 컴파일 한 후 JAR형태로 압축한다. 이후 JAR 파일은 하둡MR에 잡으로 배포된다. 배포된 잡은 VPT 파티션 함수에서 VPT 데이터를 이용하여 해당 리듀서를 결정한다. 다음은 최종 저장된 VPT 데이터를 나타내며, 리듀서 개수가 6개인 경우 데이터는 0부터 5까지 해당 리듀서 인덱스 값을 저장되며, VPT 파티션 함수에 직접 참조된다.

```
public class a {
public static int[ ] VPT = {
   0,
   .......
   4,
   5,
   };
}
```

3.2.2. VPT 파티션 함수 사용 단계

VPT 데이타를 기준으로 리듀서를 결정하는 수식은 다음의 수식 2와 같다.

$$VP = VPT[hash(key) \bmod R] \qquad (2)$$

VP : The Result of VPT Partition Function
VPT : Virtual Partition Table
key : The Output of a Mapper
R : The length of VPT

VPT는 대상 리듀서 인덱스가 저장된 매핑 테이블, key는 매퍼에서 사용되는 키, R은 VPT의 크기이다. 전처리 작업으로 생성된 VPT staitc 배열에서 키값을 R로 나머지 연산하여 VP를 구한다. VP는 대상 리듀서 인덱스 번호 이다.

다음은 VPT를 이용한 파티션 함수 자바 코드로 하둡MR은 사용자가 정의한 파티션 함수를 잡 안에서 설정할 수 있다. 하둡MR Boot코드에 VPT 데이터를 참조하는 VPT 파티션 함수를 적용시킨다. VPT를 적용한 신규 파티션 함수는 키에 대한 hash 연산후 VPT static 배열 크기로 나머지 연산을 수행하고 그 인덱스에 저장된 실제 리듀스 인덱스를 반환하다. 따라서 VPT 파티션 함수는 추가적인 연산 리소스를 발생시키지 않는다.

```
public int getPartition(Text key, IntWritable value, int numPartitions)  {
  return a.VPT[key.hashCode( ) % a.VPT.length];
}
```

Ⅳ. 실험 및 평가

4.1. 실험환경

아파치 하둡 사이트[9]에서 제공하는 자료를 바탕으로 하둡을 설치하여 운영 중인 사이트들을 살펴보면 노드의 수가 많게는 1,000개 이상부터 작게는 5개 까지 다양한 크기의 하둡 클러스터가 운영되는 것을 알 수 있다. 이는 다수 노드를 운영하는 대형 클러스터 및 소수 노드를 운영하는 클러스터에서도 하둡이 활발히 사용되고 있음을 나타낸다. 따라서 실제 운영 환경에서 최소 운영 단위인 5대의 서버급 장비에 하둡을 설치하여 클러스터 구성하였다. 각 장비의 하드웨어 및 소프트웨어 구성은 표 1과 같다.

표 1. 실험환경
Table. 1 Test environment

OS	Ubuntu 12.04
CPU	Intel Core i3-3200 3.30Ghz
MEMORY	4G
HDD	500G SATA HDD
NETWORK	1G Fast Ethernet

총 5대의 장비중 4대는 DataNode와 TaskTracker를 실행에 사용하고, 1대에서는 NameNode와 JobClient 실행에 사용하였다. HDFS의 Replication 개수는 3으로 지정하였고, Block Size는 64M로 설정하였다. 높은 유사도 데이터와 일반 텍스트 파일 데이터를 각각 10G씩 HDFS에 저장한 후 하둡 MR 패키지에 포함되어 있는 워드카운트(WordCount) 프로그램을 사용하여 실험을 수행 하였다. 워드카운트 프로그램은 하둡 배포판 패키지에 기본 포함된 프로그램이다.

4.2. 실험 데이터

높은 유사도 데이터는 동일 키에 대한 R값의 편중이 발생하도록 생성된 데이터이며 love, flyn, loves, fyny1, love1 등이다. 높은 유사도 데이터를 생성하여 실험한 이유는 유사도가 낮은 데이터의 경우 기존 파티션 함수를 사용해도 리듀서의 분포가 한쪽으로 편중 되지 않기 때문에 일반 파티션 함수와 VPT를 적용한 VPT 파티션 함수와의 차이를 구분하기 어렵다. 이러한 높은 유사도 데이터 생성 코드는 그림 4와 같다.

```
int rn = get_random_number(7)
string word = 'abcdefg!@#$1234.....'
string seed_words = {"love", "like", "road",
                "would", "dis", "icon", "fly"}
string key = seed_words[rn]
rn = get_random_number(10)
rn = 10 - rn.length
for i=0 to rn
 int word_rn = get_random_number() %
   word.length
 key = key + word[word_rn];
end
return key
```

그림 4. 샘플데이타 생성코드
Fig. 4 Code of sample data generation

여기서 seed_words 배열에 prefix 단어를 임의 선택한 후 word배열에서 postfix 글자를 추가하여 높은 유사도 데이터를 생성한다. 일반 텍스트[10] 파일은 구글 검색에서 텍스트 교재를 랜덤 검색했으며 텍스트 교재의 특성상 높은 유사도 데이터와 유사 하게 R값이 편중 되어 있다.

4.3. 실험 평가

워드카운트 프로그램에 VPT를 적용하지 않은(Non-VPT) 파티션 함수와 VPT를 적용한 파티션 함수를 설정하고 각각 실험하였다. 실험은 각 리듀서 개수당 10회씩 행하여 평균을 구하였다.

다음의 그림 5는 높은 유사도 데이터에 대한 실험 결과이고 그림 6는 일반 텍스트 파일에 대한 실험 결과이다. 각 그래프의 Y축은 잡 제출부터 완료까지 총 수행 시간을 초단위로 나타내며 X축은 리듀서 개수이다. 각 리듀서 구간별 수행시간을 합하여 평균을 산출하였으며, 높은 유사도 데이터의 경우 Non-VPT 대비 VPT가 평균 2초 정도 실행 시간을 단축되었고, 일반 텍스트 데이터의 경우 평균 5초 정도 실행 시간이 단축되었다.

그러나 그림 5와 그림 6의 실험결과처럼 데이터의 성격에 따라 VPT의 성능이 동일하지 않으며 리듀서의 개수에 따라 성능 차이가 발생하였다. 또한 하드웨어 증설 없이 리듀서만 추가하면 성능 향상은 발생 되지 않

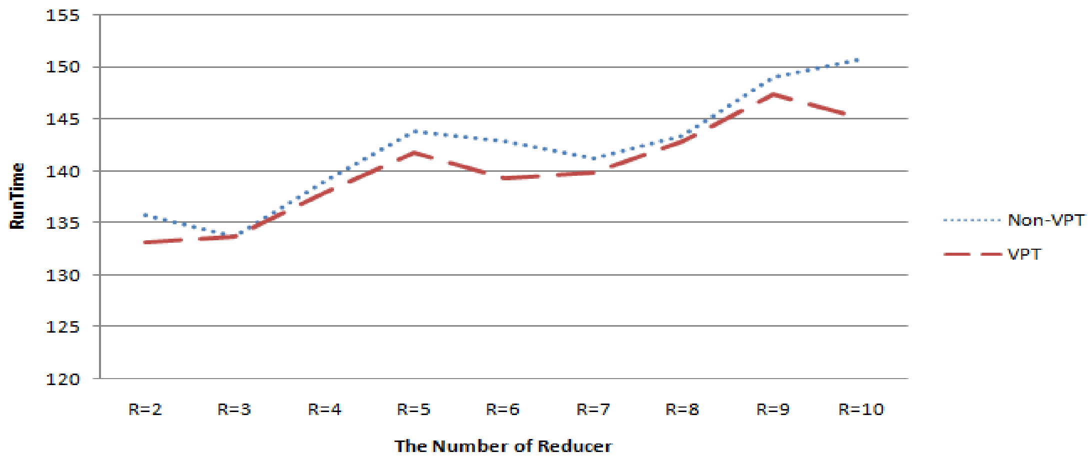

그림 5. 높은 유사도 데이터 실험 결과
Fig. 5 The result of testing of high similarity data

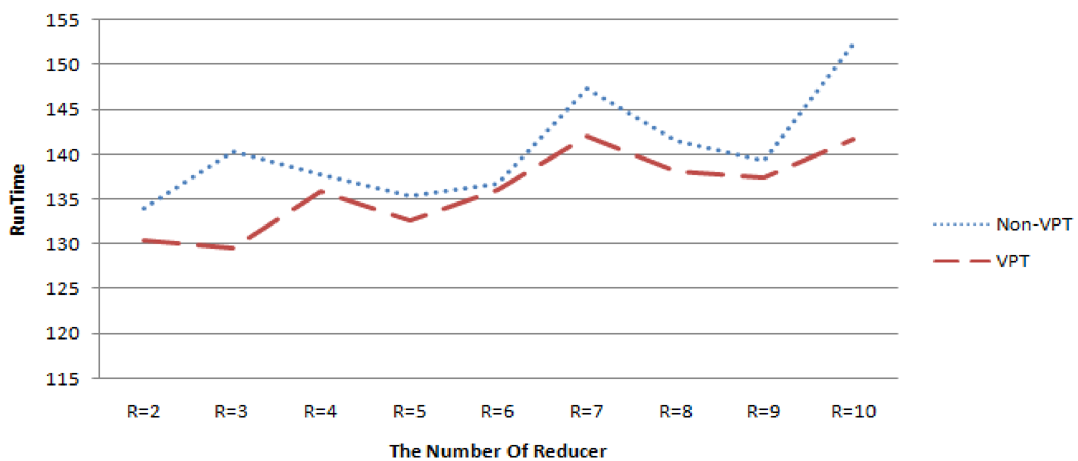

그림 6. 일반 텍스트 실험 결과
Fig. 6 The result of testing of plan text

았다.

그리고 VPT적용 MR이 Non-VPT적용 MR에 비해 모든 구간에서 상대적으로 성능이 매우 우수하지 못한데 이유는 데이터 분포에 따른 리듀서 재배정에서 데이터 편중이 발생했기 때문이다. 따라서 하드웨어 환경을 고려하여 최대의 성능을 이끌어 내는 리듀서 개수 튜닝이 필요하다는 것을 확인할 수 있다.

Ⅴ. 결론 및 향후 연구과제

하둡 MR는 매퍼의 출력을 리듀서의 입력으로 전달하기 위해 파티션 함수을 사용한다. 파티션 함수는 키에서 해쉬 값을 계산한 후 리듀서 개수로 나머지 연산을 수행하여 대상 리듀서를 결정한다. 기존 파티션 함수는 키의 편중도에 민감하여 잡이 균등하게 배분될 수 없었다. 잡이 균등하게 배분되지 못하면 특정 리듀서들의 처리 수행 시간이 길어져 전체 분산 처리 수행 성능에 영향을 주게 된다. 본 논문에서는 VPT을 제안하고

편중도가 심한 데이터들에 VPT을 적용하여 실험을 수행 하였다. 적용된 VPT는 기존 파티션 함수와 대비하여 평균 3초 정도 성능 향상이 발생 하였으며, 데이터 처리량이 증가할수록 성능향상 폭이 증가됨을 확인할 수 있었다. 또한 VPT를 적용해도 하드웨어의 물리적인 증설 없이 논리적인 리듀서만 증가시키는 것만으로는 성능 향상이 발생되지 않음을 확인할 수 있어 향후 연구는 최적의 리듀서 개수를 동적으로 계산하여 자동처리 가능한 EVPT (Extended Virtual Partition Table) 연구를 수행할 계획이다.

ACKNOWLEDGMENTS

This work was supported by the Research grant of Cheongju University in 2014-2015.

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *OSDI*, 2004, pp 137-150.

[2] David S. Touretzky, "*COMMON LISP: A Gentle Introduction to Symbolic Computation*", The Benjamin/Cummings Publishing Company, 1990.

[3] Tom White, “*Hadoop : The Definitive Guide*", OREILLY, 2011.

[4] Dhruba Borthakur and the eight members, “Apache Hadoop Goes Realtime at Facebook”, *SIGMOD’11*, June 12-16, 2011.

[5] Sanjay Ghemawat and the two members, "*The Google File System*", Google, 2003.

[6] Konstantin Shvachko and the three members, "The Hadoop Distributed File System", *IEEE*, 2010.

[7] Nandhini.C, Premadevi.P, “A Micro Partitioning Technique in MapReduce for Massive Data Analysis”, *International Journal of Innovative Research in Computer and Communication Engineering* Vol. 2, Issue 3, March 2014.

[8] Kenn Slagter and three members, "An improved partitioning mechanism for optimizing massive data analysis using MapReduce", *Springer Science Business Media New York 2013*, *J Supercomput* (2013) 66:539-555.

[9] http://wiki.apache.org/hadoop/PoweredBy

[10] http://www.gutenberg.org/ebooks/18525?msg=welcome_stranger

양일등(Ill Deung, Yang)

2002 청주대학교 컴퓨터 정보공학과 공학사
2004 청주대학교 컴퓨터 정보공학과 공학 석사
2010 청주대학교 컴퓨터 정보공학과 박사과정
2014 ~ 현재 (주)인터페이스 정보기술 부설 연구소 책임 연구원
※관심분야 : 웹 시스템 설계, 소프트웨어 공학, 분산처리

김성열(Seong Ryeol, Kim)

1982년 숭실대학교 전자계산학과 공학사
1987년 숭실대학교 대학원 전자계산학과 공학석사
1992년 숭실대학교 대학원 전자계산학과 공학박사
1982년 ~ 1984년 한국전력공사 전자계산소 근무
1984년 ~ 1990년 오산대학 전자계산과 교수
1997년 ~ 1998년 호주 QUT ISRC 객원 교수
1990년 ~ 현재 청주대학교 컴퓨터정보공학과 교수
※관심분야 : 웹 시스템 설계, 컴퓨터 보안, 소프트웨어공학