Journal of Korea Multimedia Society Vol. 16, No. 5, May 2013(pp. 657-666)
http://dx.doi.org/10.9717/kmms.2013.16.5.657

맵리듀스에서 데이터의 유용성을 이용한 데이터 분할 기법

김 종 욱†

요 약

현대사회는 소셜 미디어, 비즈니스, 바이오 인포메틱스 같은 다양한 응용프로그램에서 지속적으로 생산되어 지고 있는 수많은 데이터의 빠른 유입으로 특징지어 지고 있다. 이에 따라 폭발적으로 증가하고 있는 대규모 데이터를 보다 효율적으로 분석하고 처리 할 수 있는 방법이 그 어느 때보다 강조 되고 있다. 지난 몇 년간 학계에서는 배치 지향 시스템 (batch oriented system) 환경 내에서 병렬 처리를 효과적으로 지원할 수 있는 맵리듀스 기법이 활발히 연구 되어 왔으며, 맵리듀스 기법은 다양한 분야에서 성공적으로 사용되고 있다. 그러나 이 기법은 데이터의 상대적 유용성 (data utility)을 고려하지 않기 때문에, 멀티미디어 응용프로그램 사용자의 특성 (즉, 높은 혹은 낮은 스코어를 가지는 몇몇 결과물에 관심을 가지는 사용자들의 특성)으로 인하여 효과적인 성능을 보여 주지 못하고 있다. 따라서 본 논문에서는 이러한 문제점을 해소하기 위해, 맵리듀스 상에서의 데이터 분할 방식을 제안한다. 또한, 제안된 분할 방식에 대한 성능 실험을 통하여 우리가 제안하는 데이터 분할 방식이 기존 방식보다 성능 향상을 자져올 수 있음을 보여준다.

Data Partitioning on MapReduce by Leveraging Data Utility

Jong Wook Kim†

ABSTRACT

Today, many aspects of our lives are characterized by the rapid influx of large amounts of data from various application domains. The applications that produce this massive of data span a large spectrum, from social media to business intelligence or biology. This massive influx of data necessitates large scale parallelism for efficiently supporting a large class of analysis tasks. Recently, there have been extensive studies in using MapReduce framework to support large parallelism. While this technique has produced impressive results in diverse applications, the same can not be said for multimedia applications where most of users are interested in a small number of results having high or low score. Thus, in this paper, we develop the data partitioning algorithm which is able to efficiently process large data set having different data utility. The experiment results show that the proposed technique provides significant execution time gains over the existing solution.

Key words: MapRedue(맵리듀스), Load Balance(부하 균형), Data Utility(데이터 유용성), Wasted Work(Wasted 워크)

1. 서 론

현대사회는 소셜 미디어, 비즈니스, 바이오 인포메틱스 같은 다양한 응용프로그램에서 지속적으로 생

※ 교신저자(Corresponding Author) : 김종욱, 주소 : Teradata Labs, El Segundo, CA, USA, 전화 : 1-602-705-2306, FAX : 1-602-705-2306, E-mail : jongwook.kim1004@gmail.cm

접수일 : 2013년 4월 30일, 수정일 : 20 년 월 일
완료일 : 2013년 5월 14일
†Teradata, Software Engineer

산 되어 지고 있는 수많은 데이터의 빠른 유입으로 특징지어 지고 있다. 그 예로, 오늘날 스마트기기의 발달은 사용자들로 하여금 소셜 네트웍 사이트를 통하여 다양한 데이터를 쉽게 생성하고 공유할 수 있게 하였으며, 민간뿐 아니라 공공기관이나 비즈니스 환경에 이르기까지 다양한 분야에서 방대한 양의 데이터를 기하급수적으로 양산하고 있다. 더욱이 비즈니스 환경에서 뿐만 아니라 공공분야에 이르기까지 미래 경쟁에서 우위를 차지하기 위해서는 이러한 대규모 데이터는 양은 물론 질과 다양성 측면에서 중요한 자원이 될 수 있음은 이미 통념으로 받아들여지고 있다. 이에 따라 폭발적으로 증가하고 있는 대규모 데이터를 보다 효율적으로 분석하고 처리 할 수 있는 방법이 그 어느 때보다 강조 되고 있다.

대용량의 데이터를 효율적으로 처리하기 위한 방법으로는 크게 병렬 데이타베이스 시스템과 맵리듀스 (MapReduce)[1] 기법 (예, Hadoop[2], Hive[3], Dynamo[4], Scope[5], PNUTS[6])이 활발하게 연구 진행되어 지고 있다. 특히, 맵리듀스 기법은 배치 지향 시스템 (batch oriented system) 환경 내에서 대용량의 데이터를 효율적으로 처리하기 위한 방식을 제공해 주고 있다. 이 기법에서 사용자는 맵(Map)과 리듀스(Reduce)라고 불리는 두 함수 (function)를 정의 한다 (그림 1). 맵 단계에서는 마스터 노드 (master node)가 전체 입력 데이터를 해시 함수를 이용하여 분할 한 후, 각각의 분할 된 데이터를 워커 노드 (worker node)로 분산 한다. 각각의 워커 노드들은 사용자가 정의한 맵 함수를 수행한 후, 정렬 (sorting) 작업을 거친 후 중간형태의 key/value로 구성 된 결과물을 생성 한다. 리듀스 단계에서는 맵 함수들에 의해 생성된 데이터를 해시함수 (hash function)를 사용하여 각각의 노드에 분산한 후 처리 한다. 리듀스 단계에서 해시함수를 사용하는 주된 목적은 같은 key를 가진 데이터가 동일한 노드에서 처리되어 질 수 있도록 하기 위함이다.

최근 들어 맵리듀스 기법은 데이터 마이닝, 정보처리 같은 다양한 분야에서 성공적으로 사용되고 있다[7,8]. 그러나 맵리듀스 기법에서 사용하는 데이터 분산 방식은 (즉, 마스터 노드가 입력 데이터의 유용성을 고려하지 않고 해시 함수를 사용하여 균등 분할하는 방식) 멀티미디어 응용프로그램 환경에서는 wasted work(즉, 질의 결과에 속하지 않는 데이터를 처리하느라 낭비 되어지는 가용 자원)으로 인하여 전체 성능을 저하 시킬 수 있다. 많은 멀티미디어 응용프로그램에서 사용자들은 (가장) 높은 혹은 낮은 스코어를 가지는 몇몇 결과물에 관심을 가지고 있다. 예를 들어, 두 배치 데이터(batch data) Actor_movie(a_name, m_id, m_title, score)와 Actress_movie(a_name, m_id, m_title, score)를 가정해 보자. 여기서 score (0≤ score ≤1)는 영화에 출연한 배우에 부여된 평점을 의미 한다.

- 질의 1: result ← select m.a_name, f.a_name
  from Actor_movie as m, Actress_movie as f
  where m.m_id = f.m_id and average(m.score, f.score) > 0.8

위 질의는 같은 영화에 출현한 두 쌍의 남·여 배우들 중 average(m.score, f.score)가 0.8 보다 큰 남·여 배우 쌍을 찾는다.

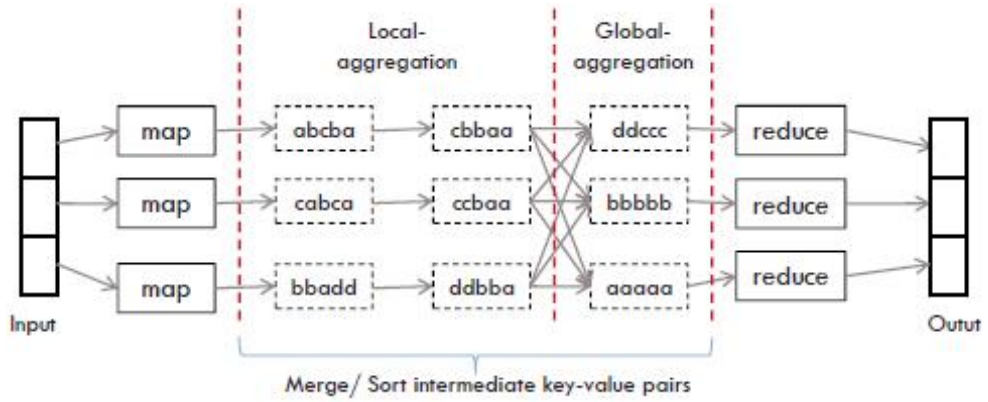

그림 1. 맵리듀스(MapReduce) 구성도

- 질의 2: result ← $TOP_{(1000,\ score)}$ (select m.a_name, f.a_name, product(m.score, f.socre) as score
  from Actor_movie as m, Actress_movie as f
  where m.m_id = f.m_id
  and sum(m.score, f.score) > 1.0)

위 질의는 같은 영화에 출현한 sum(m.score, f.score)가 1.0 보다 큰 두 쌍의 남·여 배우들 중 product(m.score, f.socre)가 가장 높은 1000쌍의 남·여 배우들을 찾는다.

본 예의 질의들은 2-way 조인 질의에 해당하면, 그림 2와 같이 2차원상에 표현할 수 있다. 그림 2-(a)에서 보는 바와 같이, 두 질의의 결과물들은 높은 스코어를 가지는 영역(즉, useful area)에서만 발견된다. 그러므로 데이터의 유용성 (useful area vs. non-useful area)을 이용하지 않는 데이터 분할·배분 기법은, 질의의 결과물에 포함되지 않는 데이터 영역(즉, non-useful area)을 처리하느라 가용자원을 낭비 한다는 문제점을 가지고 있다(그림 2-(b)).

1.1 논문의 초점과 공헌

본 논문에서는 멀티미디어 응용프로그램에서 일반적으로 사용하는 2-way 조인 질의를 맵리듀스 상에서 효율적으로 처리하기 위한 데이터 분할 방법을 연구 한다. 본 연구에서는, 그림 2에서 보여 지듯이, 각각의 데이터는 특정 열(column)에 대해 각기 다른 스코어 ($0 \leq score \leq max_{score}$)를 가지고 있으며, 스코어함수 (score function)는 많은 응용 프로그램에서 사용되는 단조 증가·감소함수 (예, average, product, euclidean, weighted sum)라 가정 한다. 본 연구에서 제시한 방법은 다음과 같이 요약 할 수 있다.

- 3.1장에서는 워커 노드에 할당된 두 데이터 소스를 조인할 때 발생하는 wasted work을 최소화하는 방법을 설명한다.
- 3.2장에서는 3.1장에서 제시한 방법을 이용하여 노드들 간의 부하 불균형 문제를 해소하는 데이터 분할 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 타겟 시스템인 RanKloud에서 제시한 데이터 분할 방식과 그 한계를 알아본다. 3장에서는 본 논문에서 제시하는 데이터 분할 방식을 설명하고, 4장에서는 우리가 제안한 알고리즘이 RanKloud보다 성능향상을 가져왔음을 실험적으로 보인다. 5장에서는 결론과 향후 연구 과제를 논한 후, 마지막으로 6장에서는 관련 연구들을 살펴본다.

2. 배경 지식

맵리듀스 상에서 데이터를 분할하는 가장 일반적인 방법은 해시 함수를 이용한 무작위적 분할(random split) 방식이다[1,2]. 이 기법은 구현 방식이 쉽고, 입력 데이터를 각각의 노드에 균등하게 배분할 수는 있다는 장점이 있다. 하지만, 이 기법은 데이터 영역의 유용성을 고려하지 않기 때문에 wasted work이 발생 할 수 있으며, 이로 인하여 성능의 저하

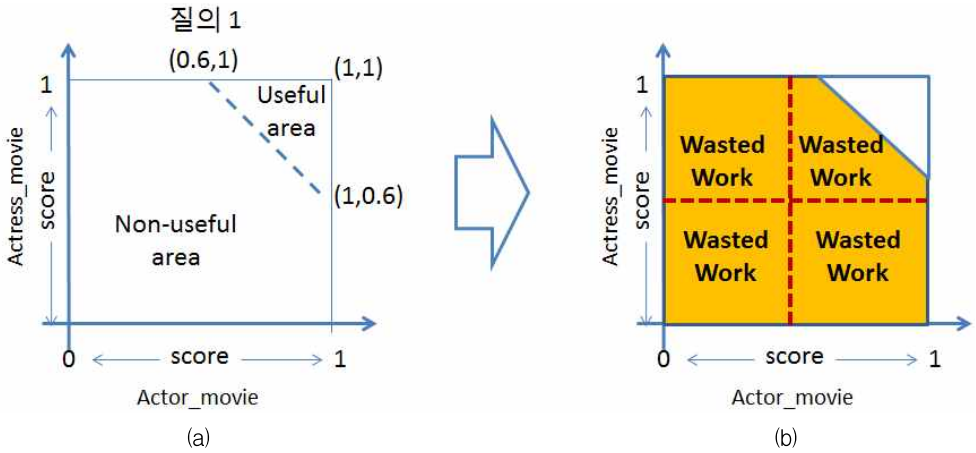

그림 2. (a) 데이터의 유용성 (useful area vs. non-useful area), (b) 데이터 유용성을 고려하지 않는 데이터 분할/배분 방식

를 초래 할 수 있다. 이러한 문제점을 해결하기 위하여, RanKloud[9,10]는 맵리듀스 상에서 데이터 영역의 유용성을 고려한 다음과 같은 데이터 분할·배분 방법을 제시 했다. 두 데이터 소스 R1, R2 와 사용자 질의가 주어 졌을 때, RanKloud는 전체 데이터 영역을 두 배타적인 영역 (useful area vs. non-useful area)으로 나누는 경계선 (boundary)을 구한다. 여기서 non-useful area는 질의에 대한 최종 결과물에 영향을 미치지 못하기 때문에, RanKloud에서는 non-useful area에 가용자원이 할당 되는 것을 최소화 하는 방법을 제안하였다. 또한 맵리듀스와 같은 병렬 시스템에서는 가장 느린 노드가 전체 시스템의 성능을 결정하기 때문에, RanKloud는 useful area에 속한 데이터가 균등하게 각각의 노드에 배분되어 노드들 간의 균형이 유지 되어질 수 있도록 한다. RanKloud의 데이터 분할·배분 방법은 n개의 워커노드가 주어졌을 때, 피봇 (pivot, 데이터 분할의 기준이 되는 축)을 기준으로 useful area를 n개의 균등한 데이터 슬라이스 (data slice)로 분할한 후, 각각의 워커노드에 배분 한다(그림 3). 이러한 데이터 분할 방식은 wasted work을 최소화 할 수 있기 때문에, 무작위적 분할 (random split) 방식보다 우수한 성능을 보인다[9,10]. 그러나 RanKloud에 제시한 기법은 각각의 슬라이스에 포함된 non-useful area의 상대적 비율을 고려하지 않기 때문에, 노드들 간의 부하 불균형을 초래할 수 있다는 단점을 가지고 있다.

3. 제안하는 데이터 분할 기법

본 논문에서 제안한 알고리즘은 RanKloud에서 제안한 기법을 발전시킨 기법이다. 제안하는 기법은 크게 두 단계로 나누어진다. 첫 번째 단계에서는 각각의 워커 노드에 할당된 데이터 슬라이스를 조인할 때 발생하는 wasted work을 최소화하는 방법을 제시한다. 두 번째 단계는 워커노드에서 데이터를 처리하는 방식을 고려하여, 노드들 간의 부하 불균형을 최소화하는 기법을 제안한다.

3.1 Wasted-work을 최소화 하는 조인 기법

각각의 워커 노드에 할당된 데이터 슬라이스는 일반적으로 중첩 반복 조인 (nested join)을 사용하여 조인을 수행할 수 있다. $R_{1,k}$, $R_{2,k}$를 k-번째 데이터 슬라이스에 해당하는 두 데이터 소스라고 가정하자. k-번째 워커노드에 속한 맵 함수는 $R_{1,k}$, $R_{2,k}$를 중첩 반복 조인기법을 사용하여 다음과 같이 조인을 수행할 수 있다.

- $R_{1,k}$를 내부 입력 데이터 소스, $R_{2,k}$를 외부 입력 데이터 소스라 가정한다. 중첩 반복 조인을 효율적으로 구현하기 위해 외부 입력 데이터는 메인 메모리(main memory)에 저장된다. 그러나 맵리듀스 상에서는 대용량의 입력 데이터로 인하여 일반적으로 전체 외부 입력 데이터가 메모리에 저장되어질 수 없다. 따라서 외부입력 데이터 $R_{2,k}$는 p개의 데이터 세그먼트(segment) $R_{2,k}^{1}$, $R_{2,k}^{2}$, ... $R_{2,k}^{p}$로 나눈 후, 반복적으로 다음을 수행한다.

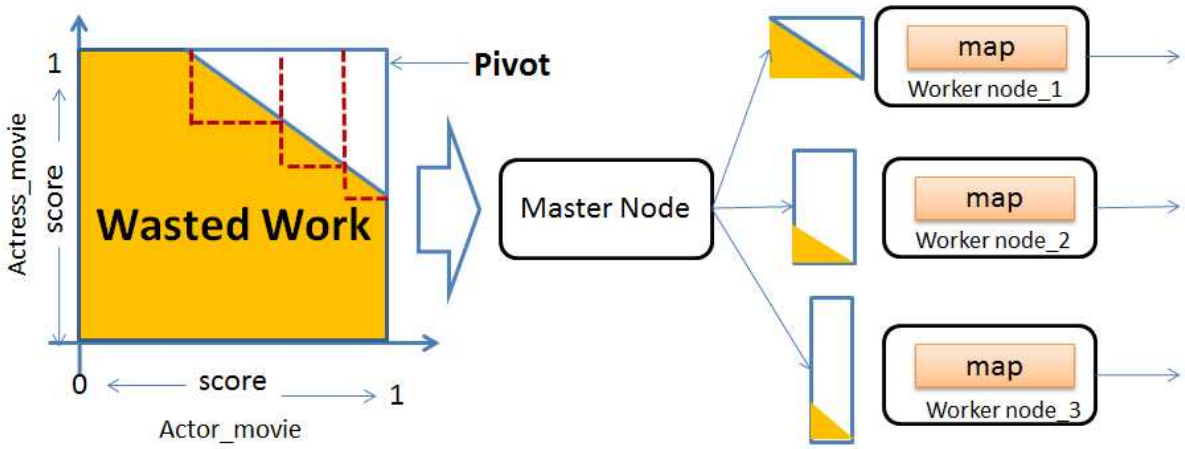

그림 3. 데이터 영역의 유용성을 이용한 분할 방식[9,10]: 워커 노드에 할당된 데이터 슬라이스는 useful area와 non-useful area를 동시에 포함한다.

- 중첩 반복 조인에서는 각각의 세그먼트에 ($R_{2,k}^{t}$, $1 \le t \le p$) 대해 다음을 수행한다. 메인 메모리(main memory)에 $R_{2,k}^{t}$를 저장 한다, 내부 입력 데이터 소스 $R_{1,k}$를 스캔하면서 조인컬럼에 해당하는 값을 가지고 메인 메모리에 저장된 $R_{2,k}^{t}$를 반복 스캔하며, 조인 조건을 만족하는지 검사한다.

중첩 반복 조인 방식은 구현이 쉽다는 장점이 있지만, wasted work이 발생할 수 있다는 단점을 가지고 있다. 그림 3에서 보여 지듯이, 각 워커 노드에 분산된 데이터 슬라이스는 유용한 영역(useful area)와 유용하지 않는 영역(non-useful area)를 동시에 포함하고 있다. 각각의 데이터 슬라이스에서 useful area와 non-useful area의 상대적 비율은 데이터 슬라이스의 위치와 스코어 함수의 종류에 따라 결정되어 진다. 예를 들어, 그림 3에서 보여 지듯이, 데이터 슬라이스가 피봇(pivot)에서 멀어질수록 non-useful area가 차지하는 상대적 비율이 증가 한다. 또한, 일반적으로 euclidean 함수는 product와 average 함수들에 비해 데이터 슬라이스에서 non-useful area가 차지하는 비율이 크다 (그림 4). Non-useful area가 차지하는 비율이 상대적으로 큰 데이터 슬라이스에서는 wasted work으로 인한 많은 양의 자원손실이 발생할 수 있다. 그러므로 본 장에서는 인덱스(index)를 사용하여, 가용자원의 손실을 최소화 할 수 있는 기법을 제시 한다.

B+ 트리는 파일시스템과 데이터베이스 시스템에서 가장 널리 사용되는 자료구조이다. B+ 트리는 데이터 전체에 대한 순차 접근을 지원하는 순차 데이터 파일 (sequential data fie)과 개별 데이터에 대한 직접 접근을 지원하는 인덱스 (index) 부분으로 구성된다[11]. 본 장에서는 B+ 트리의 순차 데이터 파일을 이용하여, 조인 시 발생하는 wasted work을 줄일 수 있는 기법을 다음과 같이 제안한다. k-번째 데이터 슬라이스에 해당하는 두 데이터 소스 $R_{1,k}$, $R_{2,k}$가 주어졌을 때, $R_{2,k}$를 피봇 (pivot)과 평행한 축에 해당하는 데이터 소스라 가정하자. B+ 트리구조를 이용하여 $R_{1,k}$, $R_{2,k}$의 조인은 다음과 같이 수행될 수 있다.

- $R_{2,k}$의 score 값들을 이용하여, B+ 트리의 인덱스 부분을 형성 한다. B+ 트리의 인덱스가 score를 이용하여 형성되었기 때문에, $R_{2,k}$는 score의 오름차순 (ascending order) 혹은 내림차순 (descending order)으로 정렬되어 순차 데이터 파일에 저장된다. 이 순차 데이터 파일은 $R_{2,k}$를 score에 따라 오름차순 또는 내림차순으로 접근할 수 있는 기능을 제공해 준다.
- 중첩 반복 조인과 유사하게, $R_{1,k}$에 속한 각각의 데이터에 대해 B+ 트리의 순차 데이터 파일을 스캔하면서, 조인 조건을 만족하는지 검사한다. 그러나 중첩 반복 조인과는 다르게, 스코어 함수 결과 값이 더 이상 조인 조건을 충족시키지 못하면, 순차 데이터 파일의 스캔을 중지한 후, $R_{1,k}$에 속한 다음 데이터를 처리 한다.

그림 5에서 보여 지듯이, B+ 트리를 사용하는 조인 기법은 non-useful area에 속한 데이터들 간의 조인을 수행하지 않기 때문에, wasted work이 발생하는 것을 방지 할 수 있다. 그러나 이 방식은 조인을

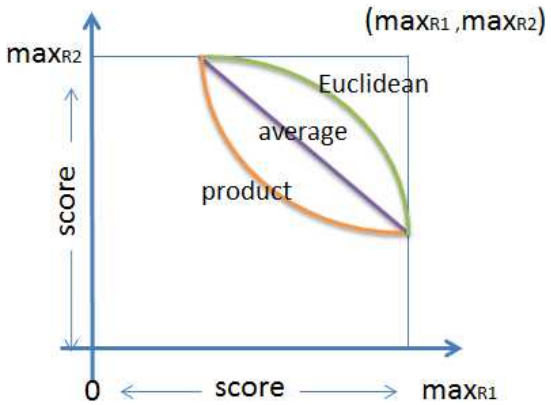

그림 4. 스코어 함수의 종류에 따른 경계선의 형태: 스코어 함수에 따른 wasted work의 비율은 product, average, euclidean 함수 순으로 증가한다.

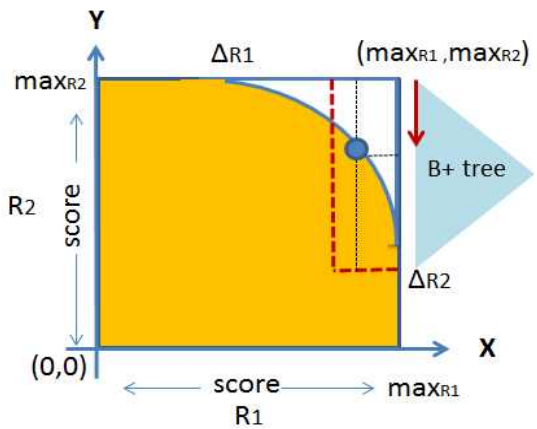

그림 5. B+ 트리를 이용한 wasted work을 방지하는 조인 방식

수행하기 위하여 B+ 트리를 형성해야 한다는 단점도 가지고 있다. 4.1장의 실험 결과는 인덱스를 이용한 조인이 중첩 반복 조인보다 우수하다는 것을 보인다. 이는 wasted work을 방지함으로써 얻어지는 성능향상이 인덱스를 형성하기 때문에 발생하는 성능저하보다 크다는 것을 의미한다.

3.2 Work-balanced 데이터 분할

3.1장에서 언급했듯이, 각각의 데이터 슬라이스에서 useful area와 non-useful area의 상대적 비율은 데이터 슬라이스의 위치에 따라 결정된다. 가령 그림 3의 예제에서, 각각의 워크노드에 할당된 데이터 슬라이스에서 non-useful area가 차지하는 비율은 워크노드 3, 워크노드 2, 워크노드 1의 순으로 증가한다. 이러한 non-useful area의 불균형 분포는 노드들 간의 워크 불균형을 초래 한다. 맵리듀스와 같은 병렬 시스템에서는 가장 느린 노드의 속도가 전체 시스템의 성능을 결정한다는 점을 고려하면, 노드들 간의 워크 균형(work balance)을 유지하는 것은 전체 성능과 밀접한 관련을 가지고 있다. 그러므로 본 장에서는 데이터 슬라이스의 non-useful area 비율을 고려한 데이터 분할 방식을 제안 한다.

그림 5에서 스코어 함수 $score(x,y)$가 주어졌을 때, 경계선에 해당하는 함수를 $y=f(x)$라 가정 한다. 이때 $y=f(x)$가 직선 $X=\max_{R1}$와 교차하는 점을 $(\max_{R1}, \Delta_{R2})$라 가정한다. 또한 교차점 $(\Delta_{R1}, \max_{R2})$는 $y=f(x)$가 직선 $Y=\max_{R2}$와 교차하는 점이라 가정한다. 이때 useful area의 전체 면적은 다음과 같이 계산할 수 있다.

$$\Theta_{useful} = (\max_{R1} - \Delta_{R1}) \times \max_{R2} - \int_{\Delta R1}^{\max_{R1}} f(x)\,dx$$

전체 데이터가 균등분포 (uniform distribution) 되었다고 가정 했을 때, n개의 워크 노드가 주어지면, 노드들 간의 work balance를 유지하기 위하여 전체 useful area를 $\frac{\Theta_{useful}}{n}$ 크기로 분할하여야 한다. 그림 6은 상기 방법으로 데이터를 분할하는 과정을 알고리즘으로 보여준다. 1번과 2번에서 B와 cur_partition_num은 각각 공집합 (Φ)과 0으로 초기화 된다. 또한 3번과 4번에서는 데이터 슬라이스의 X축 (즉,

Pseudo-code for data partitioning

Input : n (the number of worker nodes), $\max_{R1}$, $\max_{R2}$,
y=f(x) (boundary function)
output: B (a set of partition boundary)

1: B ← Φ
2: cur_partition_num ← 0
3: P_s ← $\max_{R1}$
4: P_e ← $\max_{R1}$
5: compute Δ_{R1}
6: compute Θ_{usfeul} using the method described in Section 3.2
7: **while (true)**
8: compute the smallest value of P_s which satisfies the following equation

$$((P_e - P_s) \times \max_{R2} - \int_{P_s}^{P_e} f(x)dx) \le \frac{\Theta_{useful}}{n}$$

9: B ← insert <(P_s, P_e), (f(P_e), $\max_{R2}$)>
10: P_e ← P_s
11: cur_partition_num ← cur_partition_num + 1
12: if (cur_partition_num = n)
13 break;
14: **end while**
15: **return** B

그림 6. 본 논문에서 제시한 데이터 분할 알고리즘

그림 5의 예제에서는 R_1)에 해당하는 시작점 (P_s)과 종료점 (P_e)을 max_{R1}로 초기화 한다. 5번과 6번에서는 본 장에서 설명한 방법을 이용하여 Δ_{R1}와 Θ_{useful}를 계산한다. 7번부터 14번까지는 반복적으로 데이터 영역을 분할 하면서 데이터 슬라이스를 계산하는 과정을 보여준다. 8번에서는 본 장에서 설명한 방식을 이용하여 데이터 슬라이스의 X축에 대항하는 시작점 P_s를 계산한다. 9번에서는 현재 데이터 슬라이스의 X축과 Y축의 시작점과 종료점에 해당하는 <(P_s, P_e), (f(P_e), max_{R2})>를 B에 삽입한다. 10번과 11번에서는 P_e와 cur_partition_num를 각각 업데이트 한다. 8번부터 11번까지의 과정은 조건 "cur_partition_num = n"를 만족할 때 까지 반복 수행 된다. 마지막으로 15번에서 데이터 슬라이스를 저장하고 있는 B를 반환한다.

4. 성능 평가

이 절에서는 제안하는 알고리즘의 효율성을 평가하기 위하여 기존 기법인 RanKloud 기법과 비교 실험하였다.

4.1 실험 방법

본 논문에서는 비교목적으로 다음의 세 가지 기법을 구현하였다.

- RK_nested: RanKloud에서 제시한 데이터 분할 방식을 중첩 반복 조인 (nested join)과 함께 사용하는 방법이다..
- RK_idx: RanKloud에서 제시한 데이터 분할방식과 3.1절에서 제시한 인덱스를 사용하여 조인을 수행하는 기법이다.
- WB_idx: 3.2절에서 제시한 work balance를 고려한 데이터 분할 기법이다.

표 1. 성능평가를 위한 환경변수들

파라미터	값
전체 데이터 개수	10K, 1000K
조인 선택도 (join rate)	10, 1000
스코어 함수	average
스코어 함수 결과 값 (Θ)	0.65~0.8

이 절에서 질의 시간에 대한 정확한 측정을 위하여 동일한 데이터를 기준으로 각각의 알고리즘에 대해 5회씩 반복측정 하였으며, 그 결과의 평균값을 사용하였다. 성능 평가를 위한 환경 변수들은 표 1과 같이 설정 하였다. 성능평가를 위한 데이터는 가상 데이터와 실 데이터를 사용하였다. 모든 실험은 3개의 워크노드를 사용하여 수행 되었다.

4.2 실험 결과(RK_nested vs. RK_idx)

그림 7에서는 가상 데이터를 가지고 RK_nested와 RK_idx의 성능을 비교 하였다. 실험에 사용된 가상 데이터는 각각 10K로 구성된 두 배치 데이터이며, 조인 선택도는 10으로 고정시켰다. 스코어 함수 결과 값 (Θ)은 0.65에서 0.8까지 변화시켰으며, 스코어 함수로는 average를 사용하였다. 그림 7은 가상 데이터에 대해서 RK_idx 기법이 RK_nested 기법보다 질의 처리속도가 빠른 것을 보여준다. 그림에서 보여 지듯이, Θ 값이 감소할수록 두 기법간의 성능 차이는 커짐을 알 수 있다. 그 이유 다음과 같다. Θ 값이 감소함에 따라 각각의 워커 노드에서 할당된 데이터 슬라이스의 크기가 증가한다. 이는 다시 각각의 데이터 슬라이스에서 non-useful area가 차지하는 비율이 증가하게 되며, 이로 인하여 wasted work이 증가하게 된다. 그러므로 중첩 반복 조인을 사용하는 기법은 wasted work을 효과적으로 방지하지 못하기 때문에, 이에 따른 성능저하를 보이게 된다. 이와 달리 RK_idx 기법은 wasted work를 최소화하기 때문에 non-useful area의 증가로 인한 성능저하가 RK_ nested 기법 보다 크지 않음을 그림 7을 통해 보여 준다.

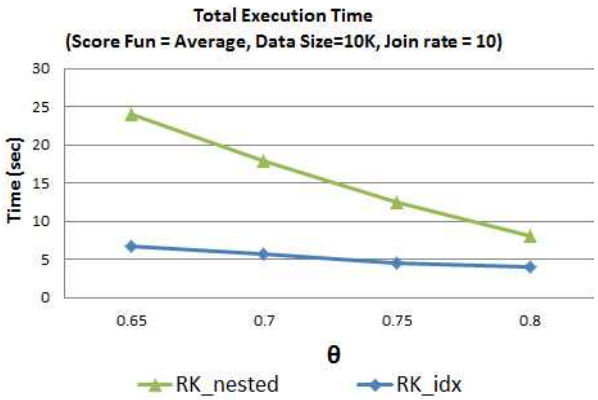

그림 7. 스코어 함수 결과 값 (Θ)에 따른 실행시간 성능 비교

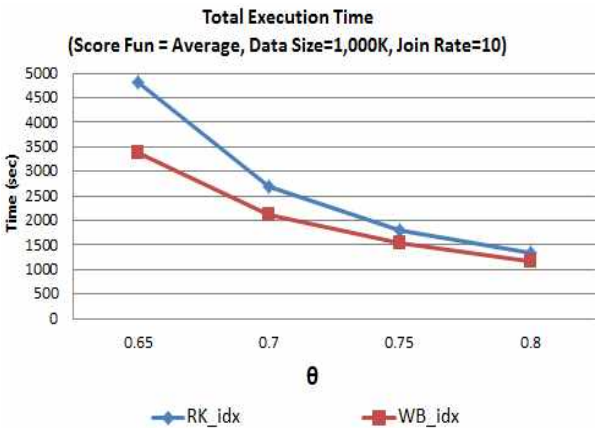

그림 8. 실 데이터를 이용하여 스코어 함수 결과 값 (Θ)에 따른 실행시간 성능 비교

4.3 실험 결과(RK_idx vs. WB_idx)

그림 8에서는 실 데이터를 가지고 RK_idx와 WB_idx의 성능을 비교 하였다. 실 데이터를 이용한 성능 평가를 위하여 IMDB (the Internet Movie Data-Base) [12]를 이용하여 각각 데이터의 크기가 1000K인 Actor_movie와 Actress_movie를 형성하였다. 이 실험에서는 조인 선택도가 ~10이 되도록 조인 조건을 선택 하였다. 또한 스코어 함수 결과 값 (Θ)은 0.65에서 0.8까지 변화 시켰으며, 스코어 함수로는 average를 사용하였다. 그림 8에서는 Θ 값의 변화에 따른 질의 처리시간을 평가하였다. 그림에서 보여 지듯이 Θ 값이 감소할수록 질의 처리 비용이 전반적으로 증가 하였다. 4.2절에서 언급했듯이, Θ 값이 감소함에 따라 각각의 워커 노드에서 분산된 데이터 슬라이스의 크기가 증가하기 때문에, 질의 처리시간은 증가하게 된다. 그럼에도 불구하고 제안하는 WB_idx 기법이 RanKloud에서 제안한 데이터 분할 방식보다 데이터 슬라이스의 크기에 따른 영향을 적게 받는다. 이는 본 논문에서 제안하는 기법은 노드들 간의 워크 밸런스 (work balance)를 고려하여 데이터를 분할하기 때문이다.

마지막으로, 그림 9에서는 실 데이터를 가지고 조인 선택도를 변화 시켰을 때 서로 다른 데이터 분할 기법간의 성능을 평가하였다. 이 실험에서는 데이터의 크기가 1000K인 Actor_movie와 Actress_movie를 이용하였다. 이 실험을 위하여 스코어 함수 결과 값 (Θ)은 0.8로 고정 하였고, average 함수를 사용하였다. 또한 조인 선택도 변화에 따른 성능을 평가하기 위하여 조인 선택도가 각각 ~10 과 ~1000가 되도록 조인 조건을 선택하였다. 조인 선택도가 증가함에 따라 질의 처리 비용이 전반적으로 증가하였음을 그림 9를 통해 알 수 있다. 그림 9의 실험 결과는 본 논문에서 제시한 WB_idx 기법이 RK_idx보다 조인 선택도의 변화에 영향을 적게 받고 있다는 것을 증명한다. 앞의 두 실험 결과를 통해 우리가 제안하는 wasted work을 최소화 하고 work balance를 고려한 데이터 분할 방식이 기존 방식보다 성능 향상을 가져올 수 있음을 알 수 있다.

그림 9. 실 데이터를 이용하여 조인 선택도에 따른 실행시간 성능 비교

5. 결론 및 향후 연구 계획

본 논문은 맵리듀스 환경에서 유용성이 다른 데이터들 간의 조인을 효율적으로 처리하는 새로운 방법을 제안하였다. 제안하는 기법은 조인 질의를 수행함으로써 발생하는 wasted work을 최소화하고, 맵리듀스에서 노드들 간의 work balance를 균형 있게 유지할 수 있는 데이터 분할/배분 방식이다. 실험결과는 본 논문에서 제안하는 기법이 기존의 RanKloud 방식보다 성능 향상을 자져올 수 있음을 보인다. 향후 연구 방향은 다음과 같다. 제안한 기법을 확장하여 맵리듀스 상에서 다양한 질의 (예, m-way 조인 질의, 스카이라인 질의[17])를 효율적으로 수행할 수 있는 포괄적 해법을 제시 하는 것이다. 또한 본 논문에서 제시한 방법은 데이터 분포로 인해 발생하는 노드들 간의 부하 불균형을 해결할 수 있는 방향으로 확장 연구 되어 질수 있다.

6. 관련 연구

맵리듀스를 이용하여 대규모의 데이터를 효율적으로 처리하기 위한 연구가 최근 몇 년간 집중적으로 수행되어 왔다. 맵리듀스는 사용자가 정의한 맵과 리듀스라 불리는 함수를 여러 노드에 분산 처리함으로써, 대규모의 데이터를 클라우드 환경 내에서 효율적으로 처리하고 있다. 그러나 노드들 간의 부하 불균형이 발생할 수 있다는 단점을 가지고 있으며, 이 문제점을 해결하기 위한 연구가 최근 들어 많이 수행되어 왔다. FAIR[13]에서는 노드에 할당된 작업 시간을 고려한 작업 스케쥴링 방법을 제안 하였다. [14]에서는 맵 함수 작업 후 발생할 수 있는 데이터 불균형 분포 문제점을 해소하기 위한 방법을 제시하였다. [14]에서 제안한 방식은 맵 함수에 의해 생성된 중간 단계의 key/value 데이터를 해시 함수에 의존하여 리듀스 함수에 분산 처리하지 않고, 데이터 분포와 밀도를 이용하여 리듀스에 배분한다. 일반적으로 맵리듀스 상에서 노드들간의 로드 불균형은 데이터 사이즈의 크기와 값에 의해 발생한다. 이 문제점을 해결하기 위하여, [15]에서는 사용자가 정의한 비용 함수 (cost function)를 이용하여 데이터를 분할하는 방식을 제안 했다. [16]에서는 데이터 분포를 이용하여 리듀스 함수에서 작업 시간을 예측하는 작업 비용 모델을 제시 했다. 이 작업 비용 모델에서 사용되는 변수 값들은 작업 수행 후 업데이트 되어 다음 작업을 위해 재사용 된다. 본 논문에서 제시한 데이터 분할 기법은 기존 방식들과 비교 하여 다음과 같은 차이점을 가지고 있다. 기존 방식들은 데이터의 분포 혹은 노드들에 할당된 작업시간을 이용하는 방법이지만, 본 논문에서 제안한 방식은 사용자의 질의에 따른 입력 데이터의 유용성을 이용하여 로드 불균형을 해소 하는 방식이다.

참 고 문 헌

[1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, *Symposium on Opearting Systems Design and Implementation*, pp. 137-150, 2004.)

[2] Yahoo, Hadoop, http://hadoop.apache.org, 2013

[3] Hive. http://wiki.apache.org/hadoop/Hive/ 2013.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapat, A. Lakshman, A. Pilchin, S. Sivasubramanian, P, Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," *Proc. the 21st ACM SIGOPS Symposium on Operating Systems Principles*, pp. 205-220, 2007.

[5] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets," *Proc. the International Conference on Very Large Data Bases*, pp. 1265-1276, 2008.

[6] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS: Yahoo!'s Hosted Data Serving Platform," *Proc. the International Conference on Very Large Data Bases*, pp. 1277-1288, 2008.

[7] B. Panda, J.S. Herbach, S. Basu, and R.J. Bayardo, "PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce," *Proc. the International Conference on Very Large Data Bases*, pp. 1426-1437, 2009.

[8] J. Lin. "Brute Force and Indexed Approaches to Pairwise Document Similarity Comparisons with MapReduce," *Proc. the international ACM SIGIR conference*, pp. 155-162, 2009.

[9] K.S. Candan, J.W. Kim, P. Nagarkar, M. Nagendra, and R. Yu. "RanKloud: Scalable Multimedia Data Processing in Server Clusters," *IEEE MultiMedia,* Vol. 18, Issue 1, pp. 64-77, 2011.

[10] R. Yu, M. Nagendra, P. Nagarkar, K.S. Candan, and J.W. Kim. "Data-Utility Sensitive Query Processing on Server Clusters to Support Scalable Data Analysis Services," *Lecture Notes in Business Information Processing* Vol. 74, pp 155-184, 2011.

[11] R. Raghu and G. Johannes, "Database Management Systems," *McGraw-Hill Higher Education*, 2nd edition, Boston, MA, 2000.

[12] Internet Movie Database, http://www.imdb.com/interfaces, 1990.

[13] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, *Job Scheduling for Multi-User MapReduce Clusters*, Technical Report No. UCB/EECS-2009-55, 2009.

[14] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi. "LEEN: Locality/Fairness- Aware Key Partitioning for MapReduce in the Cloud." *IEEE Second International Conference on Cloud Computing Technology and Science*, pp. 17-24, 2010.

[15] Y.C. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skew-resistant Parallel Processing of Feature-extracting Scientific User-defined Functions," *Proc. ACM Sympo. Cloud Computing*, pp. 75-86, 2010.

[16] B. Gurfler, N. Augsten, A. Reiser, and A. Kemper. "Handling Data Skew in MapReduce," *Proc. Cloud Computing and Services Science*, pp. 574-583, 2011.

[17] J.H. Kim and M. Kim, "A Filter Lining Scheme for Efficient Skyline Computation," Journal of Korea Multimedia Society, Vol. 14, n. 12, pp 1591-1600, 2011

김 종 욱

1994년 3월~2000년 8월 고려 대학교 전산과학과 학사

2000년 9월~2002년 8월 한국과학기술원 전산학과 석사

2004년 1월~2009년 12월 Arizona State University Computer Science 박사

2009년 10월~2010년 8월 Technicolor Member Research Staff

2010년 9월~현재 Teradata, Software Engineer