http://dx.doi.org/10.5392/JKCA.2016.16.07.667

분산 컴퓨팅 환경에서 효율적인 유사 조인 질의 처리를 위한 행렬 기반 필터링 및 부하 분산 알고리즘

Matrix-based Filtering and Load-balancing Algorithm for Efficient Similarity Join Query Processing in Distributed Computing Environment

양현식*, 장미영**, 장재우*
전북대학교 IT정보공학과*, 전북대학교 컴퓨터공학과**

Hyeon-Sik Yang(gustlr1222@jbnu.ac.kr)*, Miyoung Jang(brilliant@jbnu.ac.kr)**,
Jae-Woo Chang(jwchang@jbnu.ac.kr)*

요약

하둡 맵리듀스와 같은 분산 컴퓨팅 플랫폼이 개발됨에 따라, 기존 단일 컴퓨터 상에서 수행되는 질의 처리 기법을 분산 컴퓨팅 환경에서 효율적으로 수행하는 것이 필요하다. 특히, 주어진 두 데이터 집합에서 유사도가 높은 모든 데이터 쌍을 탐색하는 유사 조인 질의를 분산 컴퓨팅 환경에서 수행하려는 연구가 있어 왔다. 그러나 분산 병렬 환경에서의 기존 유사 조인 질의처리 기법은 데이터 전송 비용만을 고려하기 때문에 클러스터 간에 비균등 연산 부하 분산의 문제점이 존재한다. 본 논문에서는 분산 컴퓨팅 환경에서 효율적인 유사 조인 처리를 위한 행렬 기반 부하 분산 알고리즘을 제안한다. 제안하는 알고리즘은 클러스터의 균등 부하 분산을 위해 행렬을 이용하여 예상되는 연산 부하를 측정하고 이에 따라 파티션을 생성한다. 아울러, 클러스터에서 질의 처리에 사용되지 않는 데이터를 필터링함으로서 연산 부하를 감소시킨다. 마지막으로 성능 평가를 통해 제안하는 알고리즘이 기존 기법에 비해 질의 처리 성능 측면에서 우수함을 보인다.

■ 중심어 : | 빅데이터 | 분산 병렬 컴퓨팅 | 유사 질의 처리 | 필터링 | 부하 분산 |

Abstract

As distributed computing platforms like Hadoop MapReduce have been developed, it is necessary to perform the conventional query processing techniques, which have been executed in a single computing machine, in distributed computing environments efficiently. Especially, studies on similarity join query processing in distributed computing environments have been done where similarity join means retrieving all data pairs with high similarity between given two data sets. But the existing similarity join query processing schemes for distributed computing environments have a problem of skewed computing load balance between clusters because they consider only the data transmission cost. In this paper, we propose Matrix-based Load-balancing Algorithm for efficient similarity join query processing in distributed computing environment. In order to uniform load balancing of clusters, the proposed algorithm estimates expected computing cost by using matrix and generates partitions based on the estimated cost. In addition, it can reduce computing loads by filtering out data which are not used in query processing in clusters. Finally, it is shown from our performance evaluation that the proposed algorithm is better on query processing performance than the existing one.

■ keyword : | Distributed Computing Environment | Big Data | Similarity Join Query Processing | Filtering | Load Balancing |

* 본 논문은 미래창조과학부 및 정보통신산업진흥원의 서울어코드활성화지원사업의 연구결과로 수행되었음(R0613-16-1037) 2016년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.2013R1A1A4A01010099).

접수일자 : 2016년 03월 04일
수정일자 : 2016년 04월 18일
심사완료일 : 2016년 04월 19일
교신저자 : 장재우, e-mail : jwchang@jbnu.ac.kr

I. 서 론

최근 SNS 및 모바일 기기가 보급됨에 따라 종래와는 비교할 수 없을 만큼 거대한 데이터가 생산, 유통, 소비되고 있다. 기존의 단일 컴퓨터 자원으로는 이러한 데이터를 효과적으로 저장, 분석, 가공할 수 없기 때문에 대용량 데이터 처리를 위한 분산 병렬 플랫폼에 대한 연구가 활발하게 진행되었다. 특히, 하둡(Hadoop)[1]과 같은 맵리듀스(MapReduce) 프레임워크[2]는 다수의 범용 컴퓨터에서 대용량 데이터의 분산 병렬 처리를 수행하기 위해 고안되었다. 맵리듀스는 데이터를 키(key)와 값(value)으로 구성한 한 쌍의 데이터로 변환하여 다수의 클러스터 상에서 분산 병렬 처리를 수행하는 모델로서, 맵(Map)과 리듀스(Reduce) 두 단계의 함수를 작성함으로써 쉽고 간단하게 병렬 프로그램을 작성할 수 있는 장점을 가지고 있다. 따라서, 맵리듀스 기반 데이터 처리 프레임워크는 대용량 데이터에서 의미 있는 데이터를 빠르게 선별하여 서비스를 제공하기 위한 데이터 처리 모델로서 다양한 분야에서 사용되고 있다.

이러한 데이터 처리 플랫폼의 변화에 따라, 기존 단일 컴퓨터상에서 수행되는 질의 처리 기법을 분산 병렬 환경에서 효율적으로 수행하기 위한 연구의 필요성이 대두되었다. 특히, 주어진 데이터 집합에서 유사도가 높은 모든 데이터 쌍을 탐색하는 유사 조인은 데이터 마이닝[3], 웹페이지 부정 클릭 방지[4], 문서 클러스터링[5], 표절 방지[6] 등 매우 다양한 영역의 응용에서 필수적으로 사용된다. 유사 조인 질의 처리를 맵리듀스에서 수행하는 연구로 Yasin N. Silva의 연구[7]와 Ahmed Metwally의 V-Smart-join[8]이 대표적이다. 그러나 이들 기법은 별도의 파티셔닝 기법 없이 질의를 수행하여 소수 클러스터에 부하가 집중되는 문제점을 지닌다.

한편, Alper와 Mirek이 제안한 1-Bucket-Theta[9]는 이러한 문제를 해결하기 위해 질의 대상인 데이터 집합을 균등한 크기로 분할, 전송하여 분산된 클러스터 상에서 유사 조인을 병렬적으로 처리하는 행렬기반 알고리즘을 제안하였다. 따라서 맵리듀스 기반 분산 병렬 처리의 대표적인 문제점인 소수의 클러스터에 작업 부하 쏠림 현상에 의한 전체 수행 시간이 지연되는 문제점을 해결하였다. 그러나 1-Bucket-Theta는 클러스터에 할당되는 데이터를 균등하게 분할하는 것에 초점을 맞추고 있기 때문에, 다음과 같은 문제점을 지닌다. 첫째, 실제로 조인에 사용되지 않는 데이터가 클러스터에 전송되는 문제가 발생한다. 둘째, 하나의 데이터가 다수의 클러스터에 중복 전송되는 경우가 빈번하게 발생한다. 마지막으로, 클러스터 당 상이한 연산 부하를 지니는 문제점이 발생한다. 이러한 문제점은 유사 조인이 처리되는 시간을 증가시키며 결과적으로 전체 질의 처리 수행 효율을 감소시킨다.

따라서 본 논문에서는 유사 조인을 위한 행렬 기반 균등 부하 분산 기법을 제안한다. 제안하는 기법의 특징은 다음과 같다. 첫째, 행렬 기반 필터링 기법을 이용하여 질의 처리 시 조인에 사용되는 데이터만을 선별하여 클러스터에 전송한다. 이를 통해 불필요한 전송 비용을 감소시킨다. 둘째, 클러스터 간 데이터 중복을 최소화한다. 마지막으로, 균등 부하 분산 기법을 통해 각 클러스터의 작업 할당량을 균등하여 전체 수행시간을 단축한다. 이를 통해 클러스터의 작업 부하 쏠림 현상을 방지하고 각 클러스터가 대기 시간 없이 질의를 수행함으로서 작업 효율을 증가시킨다.

본 논문의 구성은 다음과 같다. 2장에서는 대표적인 데이터 병렬 처리 플랫폼인 하둡 맵리듀스 및 기존 유사 조인 질의처리 알고리즘인 1-Bucket-Theta에 대해 기술한다. 3장에서는 제안하는 행렬 기반 균등 부하 분산 기법과, 제안하는 유사 조인 질의처리 알고리즘을 기술한다. 4장에서는 성능평가를 통해 제안하는 기법의 우수성을 증명한다. 마지막으로 5장에서는 결론 및 향후 연구에 대해 기술한다.

II. 관련 연구

본 장에서는 대표적인 분산 병렬 프레임워크인 맵리듀스 및 유사 조인 기법에 대해 기술한다.

1. 맵리듀스(MapReduce)

맵리듀스는 대용량 데이터의 효율적인 병렬 프로세싱을 지원하는 대표적인 데이터 처리 프레임워크이다. 맵리듀스는 하나의 작업(job)을 여러 개의 태스크(task)

로 나누어 분산 병렬 처리를 수행하는 자바 기반의 프로그래밍 모델로써, 원본 데이터를 키(key)와 값(value)으로 구성한 한 쌍의 데이터로 변환하여 분산 병렬 처리를 수행한다. [그림 1]과 같이, 맵리듀스의 수행 과정은 크게 3단계로 구성된다. 이는 입력 데이터를 처리하는 맵(Map)단계, 맵 단계의 산출물을 중간 결과물로 취합하는 셔플(Shuffle)단계, 취합된 데이터를 병합하여 최종 결과물을 생성하는 리듀스(Reduce)단계이다. 맵리듀스 상에서 질의 처리는 입력 데이터가 맵에 병렬 입력되어 중간 결과를 생성한 후, 셔플 단계를 통해 관련 데이터를 수집해 이를 리듀스에 전송하면 리듀스가 최종 결과를 산출하는 방식이다. 이때, 개발자는 맵 클래스와 리듀스 클래스를 독립적으로 정의함으로써, 분산 병렬 처리하는 환경을 손쉽게 구현할 수 있다. 또한, 맵리듀스는 부하 조절 정책(Load balancing scheme)을 기반으로 가용한 노드에 태스크를 할당한다. 그러나 대용량 데이터 처리 시 소수의 키에 데이터가 밀집된 경우 작업량이 동등하게 분포되지 않아 전체 처리 시간이 지연되는 문제점이 존재한다.

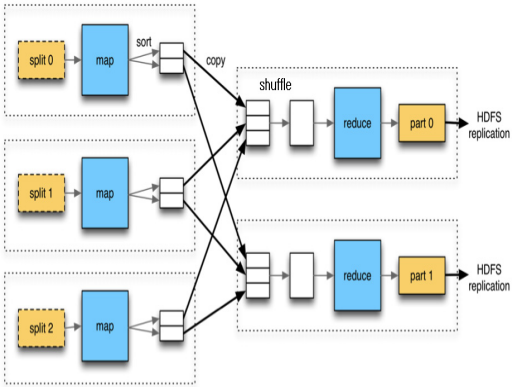


그림 1. 맵리듀스 프레임 워크

2. 유사 조인

유사 조인(Similarity Join)은 주어진 데이터 집합에서 정의된 유사도 기준보다 높은 유사도를 지니는 데이터 쌍을 모두 탐색하는 질의처리 이다. 유사 조인은 데이터 마이닝, 웹페이지 부정 클릭 방지, 문서 클러스터링, 표절 방지, 영상 표본화 등 매우 다양한 영역의 응용에서 필수적으로 사용된다. 데이터 간 유사도 기준값은 유사도 측정 함수에 의해 정의되는데, 대표적인 유사도 측정 함수로는 유클리드 거리 측정이 존재한다. 유클리드 거리 값은 직교 좌표계로 나타낸 점 상에서 유클리드 노름(Norm)을 사용해 계산하는 것으로 두 점 p, q 사이의 거리를 <식 1>을 이용하여 계산한다.

$$p = (p_1, p_2, \ldots, p_n) \qquad \cdots\cdots\text{<식 1>}$$
$$q = (q_1, q_2, \ldots, q_n)$$
$$\|p-q\| = \sqrt{(p-q) \cdot (p-q)} = \sqrt{\|p\|^2 + \|q\|^2 - 2p \cdot q}$$

분산 병렬 환경에서 유사 조인 질의를 수행하는 대표 연구로는 Alper Okcan가 제안한 1-Bucket-Theta가 존재한다. 이 알고리즘은 일차원 데이터 집합 S와 T 사이에서 주어진 유사도 임계값(θ, theta)을 만족하는 모든 데이터 쌍을 효율적으로 탐색하기 위해 데이터를 분할하여 각 클러스터에게 균일한 양의 입력 데이터를 전송한다. 맵 함수는 전체 데이터 집합을 파티션 단위로 분할하고, 임의의 지역 아이디를 각 데이터에 할당한다. 아울러, 지역 아이디를 기반으로 각 클러스터에 데이터를 전송한다. 리듀스 함수는 미리 정의된 파티션 내의 지역 아이디를 가진 모든 데이터를 전송받고, 실제 조인 작업(Theta-join)을 수행한다.

Ⅲ. 효율적인 유사 조인을 위한 행렬 기반 부하 분산 알고리즘

본 장에서는 제안하는 기법의 연구 동기 및 전체 시스템 구조에 대해 설명하고, 제안하는 행렬 기반 필터링 기법 및 균등 부하 분산 알고리즘에 대해서 자세히 기술한다.

1. 연구 동기

맵리듀스 상에서 유사 조인을 처리하는 기존 1-Bucket-theta 기법은 질의 대상인 데이터 집합을 균등한 크기로 분할, 전송하여 분산된 클러스터 상에서 유사 조인을 병렬적으로 처리한다.

Algorithm	Map(Partitioning)
Input : tuple x ∈ SUT	
/*Partitioning*/	
1:	if x ∈ S then
2:	matrixRow = random(1,|S|)
3:	for all regionID in matrixRow
4:	Output(regionID, (x, "S"))
5:	else
6:	matrixCol = random(1,|T|)
7:	for all regionID in matrixCol
8:	Output(regionID, (x, "T"))

그림 2. 1-bucket-theta의 분할 알고리즘

[그림 2]는 1-Bucket-theta의 데이터 분할 알고리즘으로서, 데이터 집합 S와 T 상의 튜플을 입력 받아 행렬에서 무작위로 선정된 클러스터로 전송하는 역할을 한다. 만약 데이터가 S 집합에 속한다면(1 line), 무작위 열을 택한 후(2 line) 해당 열에 설정된 모든 클러스터로 데이터를 전송한다(3~4 lines). 만약 데이터가 T집합에 속한다면(5 lines), 무작위 행을 택한 후(6 lines) 해당 행에 설정된 모든 클러스터로 데이터를 전송한다.

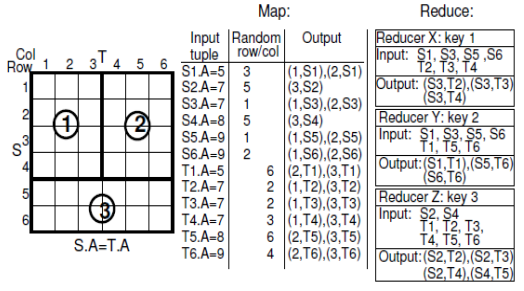


그림 3. 1-Bucket-Theta의 파티션 기법

예를 들면, [그림 3]는 3개의 클러스터를 이용하여 두 개의 데이터 집합 S, T를 분할하는 수행 예제를 나타낸다. S, T가 다음과 같이 분포할 때, 데이터 간 거리(d) 1을 만족하는 모든 데이터 쌍 (S_i, T_i)를 탐색하는 1-Bucket-Theta 알고리즘은 다음과 같이 수행된다.

$S=\{S_1=5, S_2=7, S_3=7, S_4=8, S_5=9, S_6=9\}$

$T=\{T_1=5, T_2=7, T_3=7, T_5=8, T_6=9\}$,

첫째, 전체 데이터의 분포 및 각 클러스터 당 할당되는 파티션 수를 설정한다. [그림 2]에서는 3개의 클러스터 및 3개의 리듀스 함수 파티션이 각각 설정되었다. 둘째, 맵 함수에서는 입력 데이터에 대해 지역 아이디를 임의로 할당한다. S1의 경우, 전체 데이터 분포 1~6 사이에서 임의의 지역 아이디 3을 할당받았다. S 데이터 집합의 3열은 행렬에서 1, 2번 파티션과 교차하므로 S1은 2개의 키 값을 지니도록 중복되어 중간 데이터{(1, S1), (2, S1)}를 생성한다. 동일한 방법으로 모든 입력 데이터에 대해 임의의 아이디를 할당하고, 중간데이터를 생성한다. 이를 기반으로, 첫 번째 클러스터에서 구동되는 리듀스 함수 X는 $S_X=\{S_1, S_3, S_5, S_6\}$와 $T_X=\{T_2, T_3, T_4\}$를 전송 받는다. 리듀스 함수 X는 S_X와 T_X 사이에서 거리 임계값 d=1을 만족하는 모든 데이터 집합을 탐색하여, $\{\{S_3, T_2\}, \{S_3, T_3\}, \{S_3, T_4\}\}$을 결과로 반환한다. 리듀스 함수 Y, Z는 각각 파티션 구조 2, 3내의 지역 아이디를 가진 모든 데이터에 대해 동일한 작업을 수행하며, 리듀스 함수 X, Y, Z의 모든 출력을 취합하여 유사 조인의 최종 결과를 반환한다.

1-Bucket-Theta는 무작위 파티션 선정 방법을 통해 부하를 분산하지만 데이터 중복 전송률이 높고, 클러스터 간 유사 조인 수행 작업량이 균일하지 않아 전체 질의처리 효율이 저하되는 문제점을 지닌다. [그림 2]의 예제에서 총 12개의 데이터가 리듀스 함수에 총 22번 전송되어 전체 데이터 중 83%가 중복 전송되었으며, 리듀스 X~Z에서 각각 3번, 6번, 6번의 유사 조인이 수행되었다. 데이터 중복 전송 및 미사용 데이터 전송은 전송 부하를 증가시키고, 클러스터 간 균일하지 않은 연산 부하는 처리 시간을 증가 시켜 유사 조인의 처리 효율을 저하시키는 문제점을 지닌다.

이를 해결하기 위해서는 실제 조인에 수행되는 데이터를 분석하여 작업량을 균등하게 분배하고, 불필요한 데이터 및 중복 데이터의 전송을 최소화하기 위한 기법의 연구가 필수적이다. 따라서 본 논문에서는 맵리듀스 환경에서 수행되는 행렬 기반 균등 부하 분산 알고리즘을 제안한다.

2. 시스템 구조 및 전체 질의처리 흐름

본 논문에서 제안하는 시스템 구조 및 질의처리 전체

구조는 [그림 4]과 같다. 제안하는 기법은 데이터 중복 및 불필요한 데이터 전송을 방지하기 위해, 행렬 기반으로 데이터 분포 및 조인 영역을 분석하여 균등 부하 분산을 수행한다. 이를 위해, 하둡 분산 파일 시스템(HDFS)에 전체 데이터 정보 및 행렬 기반 분할 정보를 저장하고, 이를 기반으로 2단계 맵리듀스를 통해 유사 조인 질의를 수행한다.

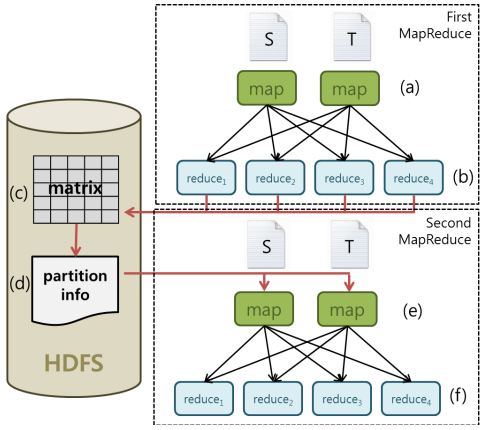


그림 4. 제안하는 질의 처리 전체 구조

유사 조인 질의를 수행하기 위한 2단계 맵리듀스는 데이터 분포 분석, 조인 행렬 생성, 데이터 필터링, 균등 부하 분산, 유사 조인의 5단계로 구성된다. 제안하는 알고리즘의 수행 과정은 다음과 같다. 첫째, 1단계 맵리듀스의 맵 함수는 데이터 집합 S와 T에서 무작위로 데이터를 선별하여 각 클러스터의 리듀스 함수에 전송한다(a). 둘째, 1단계 리듀스 함수는 전송받은 데이터의 분포를 분석하기 위해 HDFS에 저장된 행렬에 데이터를 사상한다(b). 셋째, HDFS상에 기록된 행렬 정보를 이용해 실제 조인에 사용되는 데이터만을 선별하고, 선별된 데이터를 고려하여 균등 부하 분산을 수행한다. 조인 데이터 분할 결과는 HDFS에 저장된다(c, d). 넷째, 2단계 맵리듀스의 맵 합수는 HDFS에 저장된 파티션 정보를 이용하여 각 데이터가 전송되어야 할 리듀스 ID를 탐색하고, 이를 데이터의 키로 선정한다. 따라서, 모든 데이터를 <리듀스 ID, 원본데이터 값>의 형태로 변환하여 중간 데이터를 생성한다(e). 마지막으로, 2단계 맵리듀스의 리듀스 함수는 실제 조인에 사용될 데이터만을 전송받아 유사 조인 알고리즘을 수행하여 최종 결과를 생성한다(f).

3. 행렬 기반 균등 부하 분산 알고리즘

본 절에서는 분산 병렬 환경에서 효율적인 유사 조인을 위한 데이터 필터링 및 균등 부하 분산 알고리즘을 제안한다. 제안하는 알고리즘은 2단계 맵리듀스를 통해 수행되며, i)데이터 분포 분석, ii)조인 행렬 생성, iii)데이터 필터링, iv)균등 부하 분산, v)유사 조인 수행의 5단계로 구성된다. 1단계 맵리듀스는 조인을 수행하기 위한 준비 단계 (수행단계 1-4)이며, 2단계 맵리듀스는 마지막 조인 알고리즘을 수행(수행단계 5)한다.

수행단계 1. 데이터 분포 분석

조인 데이터 선별 및 균등 부하 분산을 수행하기 위해, 데이터 분포에 대한 분석이 필수적이다. 이를 위해, 제안하는 기법은 데이터를 읽고 HDFS에 저장된 행렬에 데이터를 삽입하여 데이터 분포를 파악한다. 유사 조인에 사용되는 데이터 집합은 임의로 분할되어 각각 맵 함수에 전송되며, 맵 함수는 할당된 데이터를 <Data Source, {Data id, value}> 형식으로 변환하여 리듀스 함수에 전송한다(A-B). 리듀스 함수는 데이터 소스별로 이를 집계하여 데이터를 정렬 및 분석하고(C), 만약 다른 리듀스 함수가 동일한 데이터 값에 대한 튜플 수를 이미 기록하였다면 합산하여 HDFS에 기록한다(D).

[그림 5]는 1차원 자연수 범위 값(1~10)을 가지는 두 개의 데이터 집합 S, T상에서 데이터 분포 정보를 수집하는 예제를 나타낸다. S1 데이터의 경우, 맵 함수를 통해 <S, {1, 1}> 형식으로 변환되어, 리듀스 X에 할당된다. T11데이터의 경우, 맵 함수를 통해 <T, {11, 4}> 형식으로 변환되어 리듀스 Z에 할당된다. 리듀스 함수 X는 S 데이터 값을 기준으로 해당 값을 가지는 튜플의 수를 측정하여 S정보 파일에 이를 저장하게 되고, 리듀스 함수 Z는 T11을 포함하여 데이터 값 4를 가지는 튜플의 수를 세어 T 정보 파일에 이를 저장하게 된다. 최종적으로 리듀스 함수 X, Y, Z가 [그림 4-D]와 같이 S, T정보 파일에 모든 튜플에 대한 정보를 값으로 정렬하

여 저장한다. 생성된 S, T 정보 파일은 수행단계2에서 조인 행렬 생성 시 데이터 분포 정보를 제공하는데 사용된다.

Input tuple	Output tuple
S_1.A=1	(S 1 1)
S_2.A=3	(S 2 3)
S_3.A=7	(S 3 7)
S_4.A=4	(S 4 4)
S_5.A=9	(S 5 9)
S_6.A=10	(S 6 10)
S_7.A=2	(S 7 2)
S_8.A=3	(S 8 3)
S_9.A=8	(S 9 8)
S_{10}.A=1	(S 10 1)
S_{11}.A=2	(S 11 2)
...	

(A) S data Map

Input tuple	Output tuple
T_1.A=2	(T 1 2)
T_2.A=1	(T 2 1)
T_3.A=4	(T 3 4)
T_4.A=6	(T 4 6)
T_5.A=2	(T 5 2)
T_6.A=9	(T 6 9)
T_7.A=3	(T 7 3)
T_8.A=1	(T 8 1)
T_9.A=4	(T 9 4)
T_{10}.A=6	(T 10 6)
T_{11}.A=4	(T 11 4)
...	

(B) T data Map

Reduce X	
Input tuple	(S 1 1)(T 10 6)(T 5 2)(S 3 7) ...
Output tuple	(S 1 1)(S 3 7)(S 5 9) ... (T 1 2)(T3 4)(T 4 6) ...

Reduce Y	
Input tuple	(S 2 3)(S 4 4)(T 6 9)(T 10 6) ...
Output tuple	(S 2 3)(S 4 4)(S 11 6) ... (T 6 9)(T 10 6)(T 13 2) ...

Reduce Z	
Input tuple	(T 11 4)(S 9 8)(T 8 1)(S 5 9) ...
Output tuple	(S 5 9)(S 9 8)(S 12 8) ... (T 8 1)(T 11 4)(T 19 2) ...

(C) Reduce

S
1 2
2 3
3 4
4 8
5 4
6 2
7 3
8 1
9 2
10 5

T
1 3
2 1
3 4
4 5
5 6
6 7
7 9
8 6
9 4
10 2

(D) A number of each value

그림 5. 데이터 분포 분석을 위한 첫번째 맵리듀스

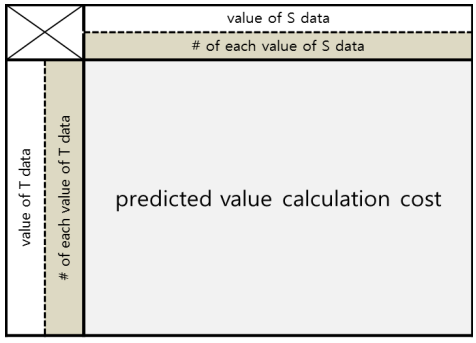
수행단계 2. 조인 행렬 생성

수행단계 1에서 생성된 데이터 분포 정보를 기반으로 조인 행렬을 생성하고, 조인에 수행되는 데이터를 파악하여 연산 부하를 예측한다. 조인 행렬은 [그림 6]과 같이 S 데이터 값 및 T 데이터 값을 각 행/열의 키 값으로 설정하고, 행과 열이 교차하는 곳에는 해당 데이터 쌍을 조인하기 위한 부하 예측값을 저장한다.

$$N_s(\alpha) = \{N(S_x) \mid S_x.A = \alpha, S_x \in S\} \quad \cdots\text{<식 2>}$$
$$N_t(\beta) = \{N(T_x) \mid T_x.A = \beta, T_x \in T\}$$
$$\text{if}(a > 1, b > 1)$$
$$\text{Matrix}_{ab} = N_s(\text{Matrix}_{1b}) \times N_T(\text{Matrix}_{a1})$$

이때, 연산 부하 예측값은 <식 2>를 이용하여 계산한다. 식에서 $N_s(\alpha)$는 데이터 집합 S에서 값 α를 가지는 튜플의 개수이며, $N_t(\beta)$는 데이터 집한 T에서 값 β를 가지는 튜플의 개수이다. Matrix_{ab}는 연산 부하 예측값으로서 a행 1열의 데이터와 1행 b열의 데이터가 조인될 시 몇 번의 조인이 발생하는 지를 계산한다.

예를 들어, [그림 4-D]에서 S, T 정보 파일을 읽어 S 데이터 집합 중 값이 1인 튜플이 2개, T데이터 집합 중 값이 1인 튜플이 3개 존재함을 알 수 있다. <식 2>를 이용하면 S데이터 집합 중 값이 1인 튜플과 T 데이터 집합 중 값이 1인 튜플이 조인에 사용될 경우 6번의 조인 연산이 발생함을 예측할 수 있다.



		1	2	3	4	5	6	7	8	9	10
		2	3	4	8	4	2	3	1	2	5
1	3	6	9	12	24	12	6	9	3	6	10
2	1	2	3	4	8	4	2	3	1	2	5
3	4	8	12	16	32	16	8	12	4	8	20
4	5	10	15	20	40	20	10	15	5	10	25
5	6	12	18	24	48	24	12	18	6	12	30
6	7	14	21	28	56	28	14	21	7	14	35
7	9	18	27	36	72	36	18	27	9	18	45
8	6	12	18	24	48	24	12	18	6	12	30
9	4	8	12	16	32	16	8	12	4	8	20
10	2	4	6	8	16	8	4	6	2	4	10

그림 6. 조인 행렬 구조

수행단계 3. 데이터 필터링

수행단계 2에서 생성한 조인 행렬을 이용하여 데이터 필터링을 수행한다. 이를 위해, 각 행렬에 저장된 데이터 쌍의 유사도를 계산하여 미리 정의된 임계값을 만족하지 못하는 데이터 셀을 필터링 함으로써, 연산에 필요한 데이터만을 부하 분산 알고리즘에 전송한다. [그림 7]은 유클리드 거리 함수를 이용한 필터링 알고리즘이다.

Algorithm 1. Filtering	
Input : Matrix mtx, int theta	
Output : Matrix mtx	
/*Filtering*/	
1:	***for*** $each\, s_i = mtx$.getS(),
2:	$each\, t_j = mtx$.getT()
3:	**if** ($\lvert s_i - t_j \rvert \leq$ theta) **then**
4:	mtx(i,j).isJoining()
5:	**else** mtx(i,j).isFilteredOut()
6:	return mtx;

그림 7. 필터링 알고리즘

제안하는 필터링 알고리즘은 수행단계2에서 생성한 행렬과 사용자 질의에 의해 지정된 조인 임계값(Θ, theta)를 입력 받아, 필터링이 완료된 행렬을 반환한다. 이를 위해 S, T의 데이터를 정렬하여 행렬에 삽입하고, 모든 데이터 쌍 에 대해 (1-2 lines) 두 값의 차 가 임계값보다 작은 값을 지니는 모든 데이터 쌍을 탐색한다(3 line). 만약 추출된 두 데이터 쌍이 Θ의 범위에 포함된다면 두 쌍은 조인에 참여됨을 표시하며(4 line), 포함되지 않는다면 두 쌍이 조인에 참여되지 않음을 표시한다(5 line). 행렬의 모든 데이터 쌍 에 대해 본 작업이 완료되면 필터링 완료된 행렬을 반환하고 종료한다(6 line).

[그림 8]은 [그림 6]의 행렬에서 유사도 임계값 Θ≤3이 주어졌을 때, 불필요한 데이터를 필터링한 결과를 나타낸다. 모든 데이터에 대해 유사 조인을 시도하는 기존 1-Bucket-Theta 과 달리, 제안하는 기법은 필터링 기법을 이용하여 중복 데이터 전송 및 불필요한 데이터 전송률을 낮춘다.

		1	2	3	4	5	6	7	8	9	10
		2	3	4	8	4	2	3	1	2	5
1	3	6	9								
2	1	2	3	4							
3	4		12	16	32						
4	5			20	40	20					
5	6				48	24	12				
6	7					28	14	21			
7	9						18	27	9		
8	6							18	6	12	
9	4								4	8	20
10	2									4	10

그림 8. 필터링된 행렬

이를 통해 조인 행렬 상에서 실제 유사 조인에서 사용되지 않는 셀을 필터링하고, 불필요한 연산 및 데이터 전송 비용을 감소시킬 수 있다.

수행단계 4. 균등 부하 분산

본 단계에서는 조인 행렬을 이용하여 각 클러스터에 균일한 연산 작업량을 할당하기 위한 균등 부하 분산을 수행한다. 제안하는 알고리즘은 필터링이 완료된 행렬 상에서 예측된 연산 부하 값을 이용하여 분산 처리를 위한 데이터 파티션을 생성한다. 데이터 파티션은 조인에 사용되는 S, T 데이터 집합 중 각각의 데이터 집합에 대해 중복을 허용할 경우, 소요되는 데이터 중복 비용을 계산하여 최소한의 비용을 지니는 데이터 집합을 선택한다.

제안하는 균등 부하 분산 알고리즘은 [그림 9]과 같다. 먼저 S 데이터를 중복 허용한 파티션 pS와 T 데이터를 중복 허용한 pT 파티션을 리듀스 수만큼 분할하고, 하나의 파티션에 할당되는 부하 예측값을 구한다(1-3 lines). S데이터에 대한 중복을 허용하는 경우, 모든 파티션에 대해 행렬의 연산 부하 예측 값을 균일하게 포함시키기 위해 하나의 파티션에 들어갈 연산 부하 예측 값이 평균에 근접할 때까지 현재 파티션 행렬에서 한 열씩 연산 부하 예측 값을 추가한다(5-14 lines). T 데이터 중복을 허용한 파티션에서는 행 단위로 확장하여 최종 파티션을 선정하고, 파티션 별 연산 부하 예측 값을 계산한다(16-25 lines). 마지막으로, pS와 pT 파티

션 중 표준편차가 작은 분할값을 최종 분할 영역으로 선정하여 각 데이터의 분할 정보를 저장한다(26-27 lines). 반환된 최종 파티션 정보는 <{Data Source, value} {Partition_num, ...}> 형태로 HDFS에 저장된다.

```
Algorithm 2.          Load-balancing
Input : Matrix mtx, int numberOfPartitions
Output : PartitionInfo[] prtInfo
/*make partition S side and T side*/
1:   Integer sum = mtx.getSumByRow();
2:   Integer avg = suml/numberOfPartitions
3:   PartitionInfo[numberOfPartitions] pS, pT;
4:   Integer index=2;
5:   For i=0 to numberOfPartitions-1
6:      if(CheckClose(pS[i].getValue,
7:                    mtx.getRow(index), avg))
8:        pS[i].add(mtx.getRow(index));
9:        if(index<mtx.getRowLength())
10:         index++;
11:     else  i++;
12:  While( index<mtx.getRowLength())
13:  pS[numberOfPartitions].add(
14:                   mtx.getRow(index++));
15:  index=2;
16:  For i=0 to numberOfPartitions-1
17:     if(CheckClose(pT[i].getValue,
18:                   mtx.getCol(index), avg))
19:       pT[i].add(mtx.getCol(index));
20:       if(index<mtx.getColLength())
21:         index++;
22:     else  i++;
23:  While( index<mtx.getColLength())
24:  pT[numberOfPartitions].add(
25                   mtx.getCol(index++));
/* select more uniform partition*/
26:  if(Stddev(pS) < Stddev(pT))  return pS;
27:  else                        return pT;
```

그림 9. 균등 부하 분산 알고리즘

[그림 10]는 수행단계 4에서 필터링한 [그림 8]의 데이터에 대해 균등 부하 분산을 수행하는 과정을 나타낸다. 예제에서 사용된 데이터를 기반으로 예측된 연산 부하 값은 총 474이며, 이상적인 분할이 이루어진다면 하나의 리듀스 파티션에 158의 연산 부하가 할당되어야 한다. pS에서 첫 번째 파티션을 만들기 위해 열 단위로 파티션을 증가시킨다. 첫 번째 파티션은 이상적인 연산 부하와 실제 포함된 연산 부하가 가장 적을 때까지 열 단위로 파티션을 증가하게 되며, 이 경우 최종적으로 1-4열을 포함해 이상적인 연산 부하와 34 차이가 나는 192의 연산부하를 포함하게 된다. 두 번째, 세 번째 파티션도 동일한 수행을 거쳐 pS가 완성되고, 행 단위로 파티션을 추가하는 pT에도 동일한 과정을 반복한다. pS의 파티션들은 192, 116, 146의 연산 부하를 가져 표준 편차 38.27의 분포를, pT의 파티션들은 164, 147, 136의 연산 부하를 가져 표준 편차 14.1의 분포를 가지게 된다. 표준 편차가 적을수록 분포가 고르기 때문에, pT가 더 균일한 분포의 파티션임을 알 수 있으며, 균등 부하 분산 알고리즘은 pT를 분할 축으로 선정하고 종료한다.

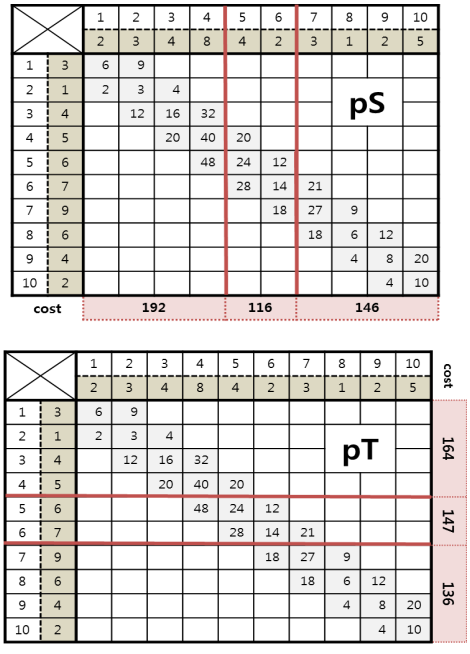


그림 10. 균등 부하분산을 위한 데이터 분할

수행단계 5. 유사 조인

2차 맵리듀스에서는 맵 함수를 이용하여 S 데이터와 T 데이터를 읽고, HDFS에 저장된 파티션 정보를 이용해 모든 데이터의 분할 영역 ID를 키로 하는 중간 데이

터를 생성한다. 각 리듀스 함수는 자신에게 할당된 분할 영역별 데이터를 전송받아 실제 유사 조인을 수행한다. [그림 11]는 파티션 정보를 이용해 맵 함수가 리듀스 함수에 데이터를 전송하는 예제를 나타낸다. 맵 함수(A, B)는 HDFS에서 데이터를 읽고 해당 데이터가 전송되는 리듀스를 파티션 정보(C)에서 데이터 값으로 검색하며, S1의 경우 데이터 값이 1이기 때문에 파티션 정보의 <S 1 (1)>이 검색된다. 이는 데이터 값이 1인 S 집합의 데이터는 리듀스 1에 전송됨을 의미하며, 중간 데이터 튜플 (S 1 1)의 형태로 변환되어 리듀스 1에 전송된다. (S4.A=4)의 경우, 검색된 파티션 정보는 <S 4 (1,2)>이기 때문에 튜플 (S 4 4)가 리듀스 1과 리듀스 2에 중복 전송된다. 모든 전송이 완료되면 리듀스 함수(D)는 전달 받은 데이터 집합에서 실제 유사 조인을 수행하고 최종 결과물을 생성한다.

Input tuple	Output tuple
S_1.A=1	(S 1 1)
S_2.A=3	(S 2 3)
S_3.A=7	(S 3 7)
S_4.A=4	(S 4 4)
S_5.A=9	(S 5 9)
S_6.A=10	(S 6 10)
S_7.A=2	(S 7 2)
S_8.A=3	(S 8 3)
S_9.A=8	(S 9 8)
S_{10}.A=1	(S 10 1)
S_{11}.A=2	(S 11 2)
…	

(A) S data Map

Input tuple	Output tuple
T_1.A=2	(T 1 2)
T_2.A=1	(T 2 1)
T_3.A=4	(T 3 4)
T_4.A=6	(T 4 6)
T_5.A=2	(T 5 2)
T_6.A=9	(T 6 9)
T_7.A=3	(T 7 3)
T_8.A=1	(T 8 1)
T_9.A=4	(T 9 4)
T_{10}.A=6	(T 10 6)
T_{11}.A=4	(T 11 4)
…	

(B) T data Map

S 1 (1)
S 2 (1)
S 3 (1)
S 4 (1,2)
S 5 (1,2)
S 6 (2,3)
S 7 (2,3)
S 8 (3)
S 9 (3)
S 10 (3)
T 1 (1)
T 2 (1)
T 3 (1)
T 4 (1)
T 5 (2)
T 6 (2)
T 7 (3)
T 8 (3)
T 9 (3)
T 10 (3)

(C) Partition Info

Reduce 1	
Input tuple	(S 1 1)(T 2 1)(T 3 4)(S 4 4) …
Output tuple	(4:1 7 10 11 19 24 38 46 … …

Reduce 2	
Input tuple	(S 4 4)(T 4 6)(S 7 2)(T 10 6) …
Output tuple	(12:2 4 8 18 22 42 61 77 … …

Reduce 3	
Input tuple	(T 6 9)(S 6 10)(S 9 8)(T 12 8) …
Output tuple	(6: 5 6 9 17 23 28 55 56 … …

(D) Reduce

그림 11. 파티션 정보를 이용한 유사 조인

Ⅳ. 성능 평가

본 장에서는 기존 분산 병렬 프레임워크 기반 유사조인 질의처리 기법과 제안하는 기법의 질의 처리 성능 평가를 수행한다. 성능평가 실험 환경은 [표 1]과 같으며, 모든 질의는 주어진 데이터 집합 상에서 설정된 θ 임계값 범위 안에서 각 데이터별 유사도를 측정하는 셀프 유사 조인이다. 성능평가는 제안하는 맵리듀스 기반 유사조인 질의처리 알고리즘과 기존 Yasin N. Silva의 연구인 맵리듀스 기반 유사 조인 알고리즘(이하 EMSJ)[7], Ahmed Metwally의 연구인 V-Smart-join[8], Alper와 Mire의 연구인 1-Bucket-Theta[9]를 질의처리 시간, 균등 부하 분산 측면에서 비교 수행한다.

표 1. 실험 환경 매개변수

항목	성능
CPU	2.9GHz Quad-Core Intel Core i5
Memory	4GB
O/S	ubuntu 12.4

아울러, 실험 데이터는 GSTM(Generate Spatio Temporal Data알고리즘[TSN99]을 사용하였으며 [그림 12]처럼 균일 분포(a), 정규 분포(b), 비대칭 분포(c), 실제 데이터 집합(d)을 1차원 사상하여 사용하였다.

균일 분포 데이터 집합에서 데이터는 모든 영역에 균일하게 분포되어 있기 때문에, 추가적인 부하 분산 기법 없이도 클러스터 간 부하가 균일하게 할당된다. 정규 분포 데이터 집합에서 데이터는 평균 0.5, 분산 0.1의 수치로 중앙에 밀집되어 있다. 비대칭 분포 데이터 집합에서 데이터는 비대칭도 1의 수치로 좌측에 밀집되어 있으며, 추가적인 부하 분산 기법이 없다면 클러스터 간 부하가 상이하게 할당된다. 실제 데이터 집합은 미국 북동부 지역 postal address 기반 실제 데이터[10]로서, 극도로 밀집되어 있는 분포를 보이며 추가적인 부하 분산 기법이 없다면 클러스터간 부하가 매우 상이하게 할당된다.

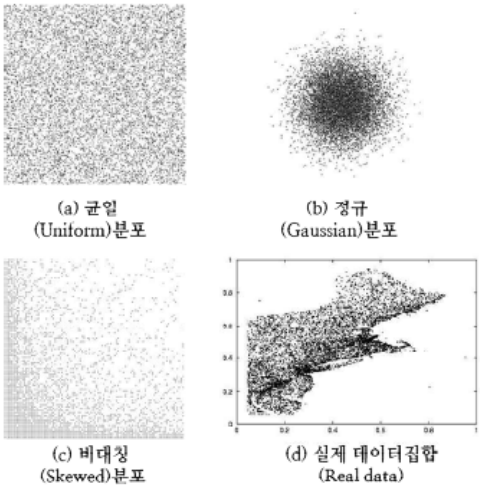


그림 12. 성능평가를 위한 데이터 집합과 분포

[그림 13-16]는 Reduce 개수 5, Theta value 10 설정하에 기존연구와 제안하는 기법의 질의처리 시간 성능을 비교한 것이다.

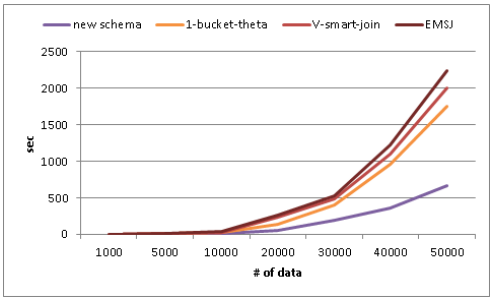


그림 13 . 균일 분포 데이터 기반 크기에 따른 성능 평가

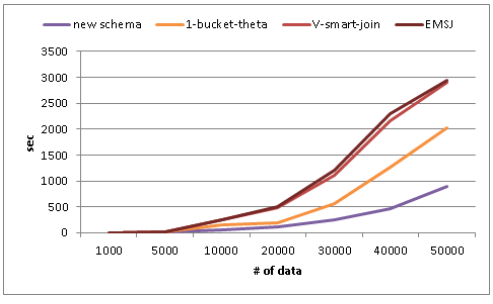


그림 14 . 정규 분포 데이터 기반 크기에 따른 성능 평가

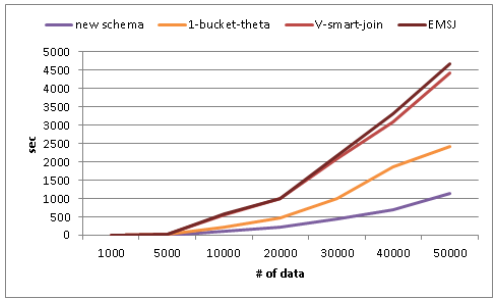


그림 15. 비대칭 분포 데이터 기반 크기에 따른 성능 평가

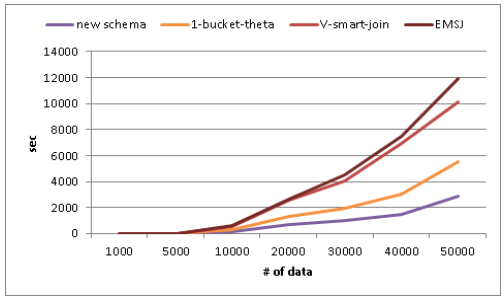


그림 16. 실제 데이터 기반 크기에 따른 성능 평가

제안하는 기법은 필터링 및 부하 분산을 위한 행렬을 생성하고 균등한 부하를 가진 파티션을 선정하는 추가 오버헤드가 존재하기 때문에, 레코드 5000개 미만의 성능평가에서는 기존 기법과 유사한 성능을 나타낸다. 그러나 해당 오버헤드는 레코드가 증가함에 따라 급속도로 감소한다.

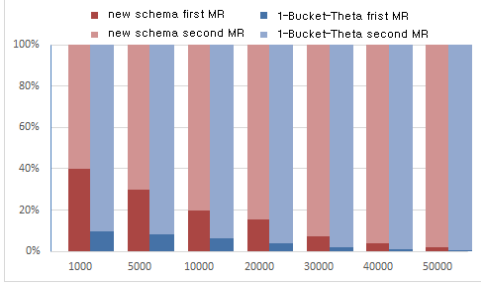


그림 17. 제안하는 기법의 오버헤드 평가

[그림 17]은 균일 분포 기반 질의처리 시간 성능평가 [그림 13]에서 필터링 및 균등 부하 분산 알고리즘의 처

리 시간이 전체 소요 시간에 미치는 영향을 측정한 것이다. 이를 위해, 각 기법의 첫 번째 맵리듀스(파티션 생성), 두 번째 맵리듀스(실제 조인 수행) 소요 시간의 비율을 비교하였다. EMSJ와 V-smart-join은 부하 분산이 고려된 파티션을 생성하지 않기 때문에 제외하였다. 레코드 1000개의 적은 데이터의 집합의 경우, 제안하는 기법의 첫 번째 맵리듀스 수행시간은 전체 수행시간의 40%를 차지하지만, 레코드 개수가 증가함에 따라 1%(레코드 5000개의 경우)까지 급격히 감소함을 알 수 있다. 이는 데이터 집합이 증가함에 따라 유사 조인을 수행하는 두 번째 맵리듀스 수행시간은 지수적으로 증가하는데 반해 행렬을 생성하고 분할하는 첫 번째 맵리듀스 수행시간은 선형적으로 증가하기 때문이며, 행렬 생성 및 부하 분산 수행 오버헤드가 전체 수행시간에 큰 영향을 주지 않는 다는 것을 알 수 있다. [그림 13-16]를 통해 모든 데이터 집합에서 10,000개 이상의 레코드 사용 시 제안하는 기법이 우수한 성능을 나타냄을 알 수 있다. 특히 [그림 13]의 균일 분포 데이터 집합 기반 성능평가의 경우, 데이터 분포가 균일하여 제안하는 기법의 균등 부하 분산 알고리즘이 부하 분산에 큰 효력을 발휘하지 못함에도 불구하고 우수한 성능을 보이는데, 이는 기존 기법은 조인될 가능성이 있는 모든 데이터를 클러스터에 전송하여 조인을 시도하는 반면 제안하는 기법은 필터링 기법을 통하여 클러스터에 전송되는 데이터양을 감소시키기 때문이다.

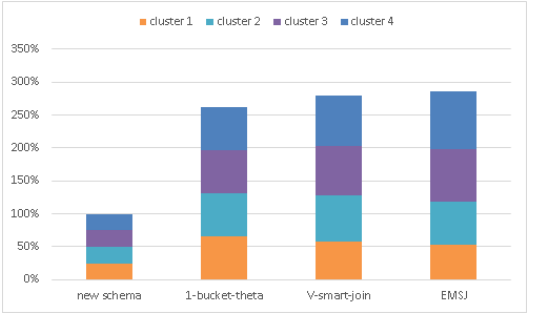


그림 18. 균일 분포 데이터 기반 클러스터 간 연산 부하

[그림 18]은 [그림 13]의 균일 분포 기반 데이터 크기에 따른 성능평가에서, 레코드 50000개 사용 시 각 클러스터에서 계산된 유사 조인 횟수를, 제안하는 기법을 기준으로 백분율 한 것이다. 각 클러스터 당 할당된 부하가 크게 상이하지 않음에도 불구하고, 제안하는 연구는 필터링을 통해 클러스터가 기존 연구보다 적은 부하의 연산만 수행하여 높은 성능 향상을 보여준다.

또한 [그림 15]의 실제 데이터 집합에서 제안하는 기법은 균등 부하 분산 알고리즘을 통한 파티션 구조를 생성하기 때문에 기존 기법보다 매우 좋은 성능을 보임을 알 수 있다. [그림 18]은 [그림 16]의 실제 데이터 기반 성능평가에서, 5만개 레코드에 대해 각 클러스터에 할당된 레코드 수를 비교한 것이다.

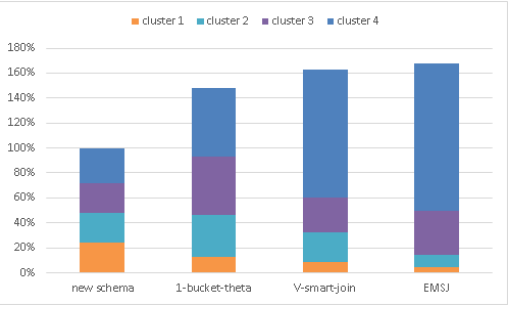


그림 19. 실제 데이터 기반 클러스터 간 연산 부하

실제 데이터 집합은 데이터가 편중되어 분포되었기 때문에 균등 부하 분산 알고리즘 없이는 클러스터간 매우 상이한 연산 부하가 할당된다. EMSJ와 V-smart-join는 별도의 파티션 기법이 없기 때문에 cluster 4에 데이터가 크게 밀집되어 전송되었으며, 1-Bucket-Theta는 무작위 알고리즘을 이용한 부하분산을 수행하였지만 클러스터 당 연산 부하를 고려하지 않았기 때문에 여전히 cluster 4에 부하가 밀집되어 있다. 제안하는 기법은 행렬 기반 부하 분산 알고리즘을 통해 연산 부하를 분산함으로서, 각 클러스터에 할당된 부하가 매우 균등하게 나타남을 알 수 있다. EMSJ와 V-Smart-join은 부하 분산을 고려한 파티션 기법을 제안하지 않았기 때문에, 이후 성능평가는 1-Bucket-Theta와 제안하는 연구의 성능 비교를 통해 제안하는 연구의 우수성을 증명한다.

유사도 임계값(θ)에 따른 성능 평가 결과는 [그림 19-22]과 같다. 제안하는 기법은 필터링 기법을 통해

실제 조인에 사용되는 데이터만을 전송하기 때문에, 기존 기법보다 더 우수한 성능을 보인다.

리듀스 개수에 따른 성능 평가 결과는 [그림 23-26]과 같다. 제안하는 기법은 특히 밀집도가 높은 데이터 집합일수록 기존 연구에 비해 좋은 성능을 보인다. 이는 기존 연구의 무작위 부하 분산 기법은 리듀스 수가 증가함에 따라 각 리듀스에 중복 전송되는 데이터 수가 선형적으로 증가하는 반면, 제안하는 연구는 필터링 기법을 통해 데이터 중복 전송을 방지하기 때문이다.

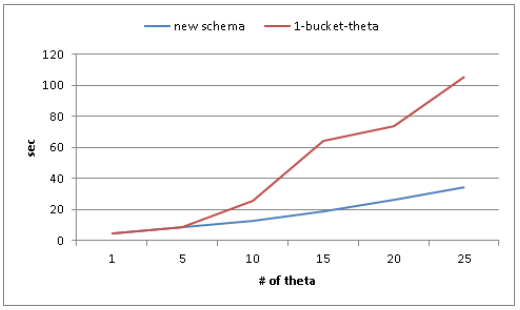


그림 20. 균일 분포 기반 theta에 따른 성능 평가

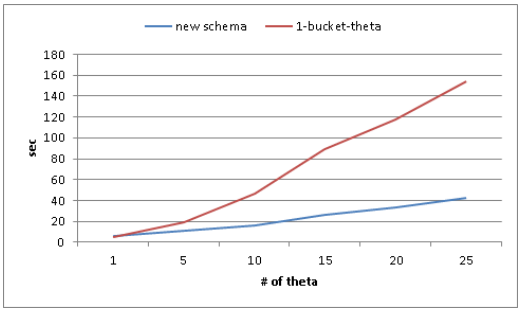


그림 21. 정규 분포 기반 theta에 따른 성능 평가

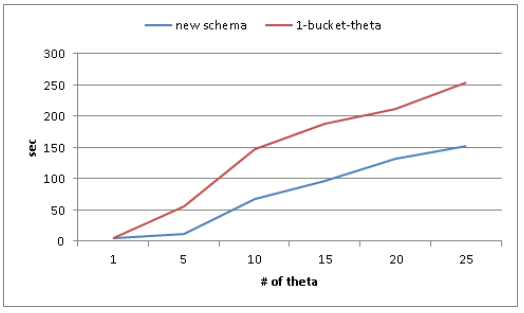


그림 22. 비대칭 분포 기반 theta에 따른 성능 평가

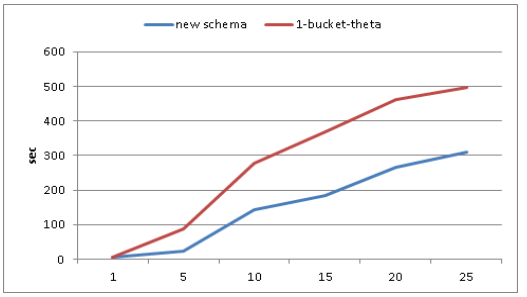


그림 23. 실제 데이터 기반 theta에 따른 성능 평가

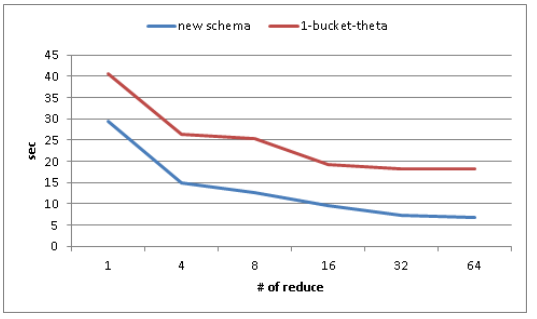


그림 24. 기반 reduce 수에 따른 성능 평가

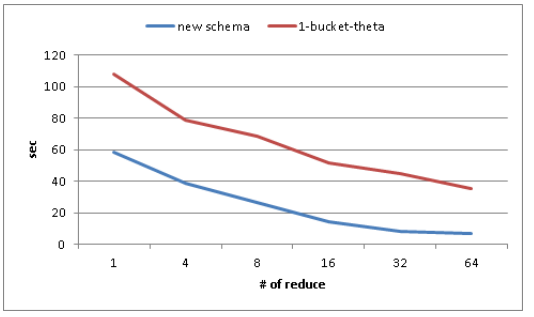


그림 25. 기반 reduce 수에 따른 성능 평가

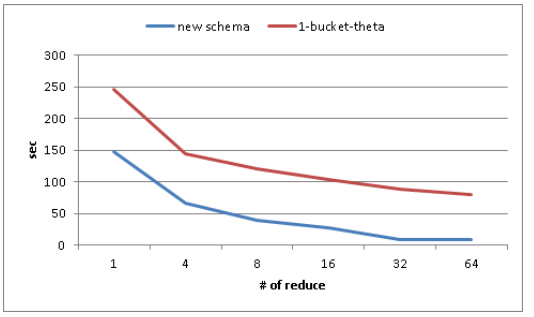


그림 26. 기반 reduce 수에 따른 성능 평가

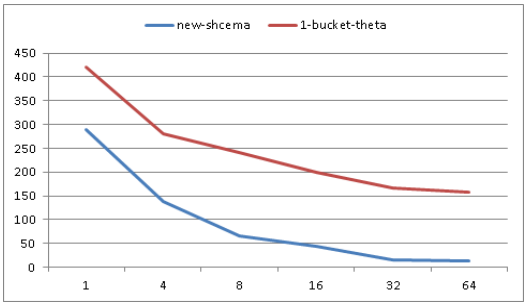


그림 27. 기반 reduce 수에 따른 성능 평가

V. 결론 및 향후연구

본 논문에서는 분산 병렬 환경에서 효율적인 유사 조인을 위한 행렬 기반 필터링 및 균등 부하 분산 알고리즘을 제안하였다. 제안하는 기법은 클러스터의 부하를 균등하게 분산하기 위해 행렬 구조를 사용하며, 사용되지 않는 데이터를 사전에 필터링한다. 이를 통해 불필요한 전송 비용을 제거하고 클러스터 간 균등한 부하를 할당함으로써 질의 처리 효율을 증가시킨다. 또한 기존 유사 질의처리 연구와 성능 비교를 통해 제안하는 기법이 질의 처리 시간 측면에서 우수함을 보였다.

향후 연구 방향은 제안하는 기법을 top-k 및 k-means 질의를 지원하는 알고리즘으로 확장하여 연구하는 것이다.

참 고 문 헌

[1] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler “The hadoop distributed file system,” Mass Storage Systems and Technologies (MSST), pp.1-10, 2010.

[2] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: simplified data processing on large clusters,” Communications of the ACM, Vol.51, Issue.1, pp.107-113, 2010.

[3] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik, “A primitive operator for similarity joins in data cleaning,” Data Engineering, p.5, 2006.

[4] A. Metwally, D. Agrawal, and A. El Abbadi, “DETECTIVES: DETEcting Coalition hiT Inflation attacks in adVertising nEtworks Streams,” Proceedings of the 16th WWW International Conference on World Wide Web, pp.241-250, 2007.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, “Syntactic clustering of the web,” Computer Networks, pp.1157-1166, 1997.

[6] T. C. Hoad and J. Zobel, “Methods for identifying versioned and plagiarized documents,” JASIST, Vol.54, Issue.3, pp.203-215, 2003.

[7] Yasin N. Silva and Jason M. Reed, “Exploiting MapReduce-based similarity joins,” Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp.693-696, 2012.

[8] Ahmed Metwally and Christos Faloutsos, “V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors,” Proceedings of the VLDB Endowment, Vol.5, No.8, pp.704-715, 2012.

[9] Alper Okcan and Mirek Riedewald, “Processing theta-joins using MapReduce,” Proceedings of the 2011 ACM SIGMOD International Conference on Management of data ACM, pp.949-960, 2011.

[10] http://chorochronos.datastories.org/

저 자 소 개

양 현 식(Hyeon-Sik Yang) 준회원

▪ 2016년 2월 :전북대학교 IT정보공학과(공학사 과정)

<관심분야> : 빅데이터, 분산 병렬 컴퓨팅, HTM

장 미 영(Miyoung Jang) 정회원

- 2009년 2월 : 전북대학교 컴퓨터공학과(공학사)
- 2011년 2월 : 전북대학교 컴퓨터공학과(공학 석사)
- 2016년 2월 : 전북대학교 컴퓨터공학과(공학 박사)

<관심분야> : 공간 데이터베이스, 데이터베이스 아웃소싱, 위치 정보 보호, 궤적 데이터 마이닝

장 재 우(Jae-Woo Chang) 정회원

- 1984년 2월 : 서울대학교 컴퓨터공학(공학사)
- 1986년 2월 : 카이스트 컴퓨터공학과(공학 석사)
- 1991년 2월 : 카이스트 컴퓨터공학(공학 박사)
- 1996년 ~ 1997년 : Univ. of Minnesota, Visiting Scholar
- 2002년 ~ 2004년 : Penn State Univ., Visiting Scholar
- 2004년 ~ 현재 : 전북대학교 IT정보공학과 교수

<관심분야> : 공간데이터베이스, 클라우드 컴퓨팅, 데이터베이스 정보보호