# 분산 환경에서의 클러스터화된 밀집 인덱스 기반 효율적인 불균등 분포 데이터의 조인 기법

김재형, 박상현
연세대학교 컴퓨터과학과
e-mail:jaehyungkim@cs.yonsei.ac.kr
sanghyun@cs.yonsei.ac.kr

# Dense Clustering Index Based Efficient Join Method to Handle Skewed Data in Distributed Environment

Jae Hyung Kim, Sanghyun Park
Department of Computer Science, Yonsei University, Seoul, South Korea

요 약

오픈소스로부터 촉발된 분산 시스템의 보편화로 기존 상용 시스템으로는 제공하지 못한 다양한 종류의 서비스가 각광받고 있다. 특히, 테라바이트 단위를 넘어 페타바이트 단위의 데이터를 다루는 서비스의 등장으로 드러난 오픈소스 분산 시스템의 문제를 개선하기 위한 시도가 학계 및 업계에서 다각적으로 이뤄지고 있다. 이러한 시도는 새로운 방법론을 제시하는 것에서부터 기존 분산 데이터베이스 관리 시스템(Distributed DBMS)에서 사용된 방법론들을 적용하는 것까지 다양하게 이뤄지고 있다. 본 논문에서는 특정 키 값(Key Value)에 불균등 분포된 데이터에 대한 조인 연산의 탐색 공간을 밀집 인덱스를 통해 줄여 비교적 높은 시간 복잡도를 완화하는 방법론을 제시하고자 한다.

## 1. 서론 및 연구 배경

최근 수년간 분산 환경 기반의 오픈소스 분산 시스템이 쏟아져 나오면서 빅데이터 플랫폼으로 대표되는 분산 시스템에 대한 관심을 촉발시켰다. 가장 대표적인 분산 시스템인 하둡(Hadoop)[1]은 구글 파일 시스템[2]과 맵리듀스[3] 논문을 통해 소개된 개념을 구현한 오픈소스 분산 시스템이다. 하둡 분산 파일 시스템(Hadoop Distributed File System)과 맵리듀스 프레임워크(MapReduce Framework)의 조합으로 구성된 하둡은 단일 노드에서 처리하지 못하는 막대한 양의 데이터와 수행할 연산을 여러 개의 노드에 분산하여 처리함으로써 높은 확장성을 제공하고, 각 노드에 대한 고장감내(Fault Tolerance) 기능을 지원하여 고가용성을 보장한다. 뿐만 아니라, 이를 기반으로 하는 다양한 종류의 분산 시스템들도 함께 등장했는데, 맵리듀스(MapReduce)의 사용성 문제를 극복하기 위해 SQL 기반 분산 시스템인 Hive[4]와 구글의 빅테이블[5]과 NoSQL 개념을 도입한 컬럼기반 데이터베이스인 HBase[6]가 대표적이다. 뿐만 아니라, SQL 처리 엔진을 직접 하둡 파일 시스템과 연결하여 맵리듀스 프로그래밍 모델을 사용하지 않고 분산 처리를 수행하는 오픈소스 분산 시스템들도 등장하기 시작했다[7, 8, 9].

※ 이 논문은 2012년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (2012R1A2A1A01010775).

이러한 오픈소스 분산 시스템을 통합적으로 이용해 수테라 단위에서 페타 단위까지 막대한 양의 데이터를 다루는 서비스가 등장하면서 이전에는 예상하지 못한 성능관점에서의 한계점이 지적되기 시작했다. 그 중에서도 분산 분산 시스템이 적용되는 데이터의 유형이 특정 키 값에 편중되는 경우가 빈번하게 발생하였는데, 이럴 경우 특정 노드에 데이터를 처리하기 위한 부하가 집중되어 시스템의 전체적인 응답속도가 느려지는 경향을 보였다[12].

본 논문에서는 이러한 문제를 해결하기 위한 노력으로 일반적인 불균등 분포의 조인 연산을 수행하기 위한 방법론에 전통적인 데이터베이스에서 사용되는 외부 합병 정렬[10]과 밀집 인덱스를 활용한다. 이를 통해, 특정 키 값에 편중된 데이터에 대한 조인 연산을 모든 데이터를 탐색하는 대신에 클러스터화된 밀집 인덱스를 탐색하여 장시간 수행되는 연산의 효율성을 극대화하여, 분산 시스템의 전체적인 응답속도를 향상시킬 수 있을 것으로 기대된다. 클러스터화된 밀집 인덱스란 모든 튜플에 대한 인덱스를 가지는 대신에 중복되는 것들 중 가장 앞선 튜플의 포인터만 유지하는 것을 의미한다. 또한 특정 구현에 의존성을 없애고 일반적인 방법론을 제시함으로써 하둡 분산 파일 시스템 기반의 환경에 적용될 수 있도록 한다.

본 논문의 나머지 부분의 구성은 다음과 같다. 2장에서는 하둡 분산 파일 시스템 기반의 분산 시스템에서 데이터의 불균등 분포를 다루기 위한 노력들을 다루고, 3장에서는 본 논문에서 제시하는 조인 기법을 소개하고, 4장에

서는 해당 기법의 비용을 이론적으로 분석한다. 마지막으로 5장에서는 향후 연구 방향과 결론을 제시한다.

## 2. 관련 연구

본 논문에서 제시한 기법의 기반이 되는 하둡은 앞서 언급한 바와 같이 맵리듀스와 하둡 분산 파일 시스템의 조합으로 이뤄져있다.

### 2.1 맵리듀스와 하둡 분산 파일 시스템

맵리듀스는 분산 환경에서 데이터를 다루기 위한 프로그래밍 모델을 의미하며, 간단하게 매퍼(Mapper)와 리듀서(Reducer)로 구성된다. 사용자가 매퍼와 리듀서 함수를 직접 구현하면 하둡 시스템이 해당 작업들을 분산 노드에서 각각 수행함으로써 막대한 양의 데이터를 병렬적으로 처리할 수 있다. 분산 파일 시스템은 분산된 노드 각각에 데이터를 분할 저장하기 위한 논리적인 가상의 파일 시스템을 나타낸다. 이러한 프로그래밍 모델과 분산 파일 시스템의 구현이 바로 하둡이다. 하둡을 활용하여 데이터를 처리하는 연구가 진행됨에 따라 기존의 방법론들을 맵리듀스 모델에 맞게 전환하려는 시도가 주류를 이뤘다. 특히, 기존의 데이터베이스에서 데이터를 다루는 방식을 그대로 적용하여 사용자가 굳이 맵리듀스 프로그래밍 모델을 학습하지 않고, 기존 SQL과 유사한 문법을 가진 언어로 데이터를 처리할 수 있는 방법론이 활발하게 제시되었다.

### 2.2 Hive

Hive는 하둡 플랫폼에 SQL을 처리할 수 있는 별도의 계층을 만들어 사용자로부터 입력받은 SQL 구문을 맵리듀스 모델로 자동으로 변경해주는 오픈소스 분산 시스템이다. 실제로 데이터를 다루는 사용자는 프로그래밍 언어에 익숙하지 않은 경우가 대부분이므로, 저수준(Low Level) 언어인 맵리듀스의 사용성 문제를 해결하기 위한 방안이 될 수 있다. 하지만 Hive를 이용한 서비스에서 실제 맵리듀스를 통해 얻는 이익보다 SQL과 맵리듀스 사이의 변환에 대부분의 시스템 자원이 소모된다는 주장이 제기되고 있으며, 실제로 Hive에서 질의 수행 시 응답속도는 맵리듀스로 변환을 거치기 때문에 다음에 소개될 계열의 시스템에 비해 비교적 느린 편인 것으로 되어있다.

### 2.3 SQL on Hadoop

맵리듀스의 사용성 문제를 극복하기 위해 제시된 오픈소스 분산 시스템들의 접근 방법 중 맵리듀스를 기반으로 하는 분산 시스템들은 태생적 한계를 극복하지 못하고, 성능상 문제점을 드러냈다. 자연스럽게 이를 극복하기 위해 맵리듀스 계층을 덜어내고 직접적으로 하둡 분산 파일 시스템을 활용하는 분산 시스템들이 등장했다. 대표적으로 Cloudera의 Impala[7]를 예로 들 수 있는데, 이는 구글의 Dremel[11]로부터 영감을 얻어 하둡 분산 파일 시스템 상에서 Batch Processing 대신에 Ad-Hoc Query를 처리할 수 있는 기능을 탑재한 오픈소스 기반 분산 시스템이다. 이와 달리, Data Warehouse를 위한 Batch Processing을 지원하는 Tajo[8]와 같은 분산 시스템도 존재한다. 이들은 맵리듀스 대신 SQL 계층(SQL Stack)을 하둡 분산 파일 시스템 위에 두고, 직접 구현한 분산 질의 엔진을 통해 데이터를 처리한다. 눈에 띄는 또 다른 차이점은 조인 연산 등의 중간 결과(Intermediate Relation)를 하드디스크(HDD)에 저장(Materialization)하지 않고, 메인 메모리(RAM) 상에 저장하여 최대한 Pipelining 기법을 사용할 수 있도록 최적화되어있다. 이러한 방법으로 분산 처리의 이점을 살리고, 병렬로 수행되는 작업을 통해 성능을 극대화하는 것을 목적으로 한다.

## 3. 불균등 분포 데이터 조인 기법

앞서 언급한 분산 시스템들의 공통된 문제는 분산된 데이터 간에 조인 연산을 수행할 때 데이터를 특정 노드로 전송해야하는 태생적인 문제가 있다[12]. 또한 특정 노드에 데이터가 밀집되는 경우가 많다. 예를 들어, 공간 질의를 다루는 경우 특정 지역에 공간 데이터가 밀집되는 경우를 생각할 수 있다. 이는 수 테라바이트 혹은 그 이상의 데이터를 다루는 환경에서는 네트워크 간 통신비용으로 인해 병목현상을 초래하게 되며, 전송된 데이터에 대한 연산도 특정 노드에서만 집중적으로 수행되는 현상으로 이어진다. 이러한 문제를 해결하기 위해 본 논문에서는 하둡 분산 파일 시스템에 기초한 SQL 계층을 가진 분산 시스템에 보편적으로 적용될 수 있는 기법을 제시한다.

### 3.1 기존 불균등 분포 데이터의 조인 방법

기존 불균등 분포를 위한 조인 기법 중 가장 대표적인 인덱스 기반 조인 기법을 소개한다. 그 중에서도 Skew Join 기법[13]은 Hive에서 사용되는 것으로 다음과 같은 가정을 바탕으로 한다.

(1) 릴레이션 Ra는 기본키를 가지고 있다.
(2) 릴레이션 Rb는 Ra의 키를 참조한다.
(3) count(Ra) << count(Rb)
단, count(R)은 릴레이션 R의 튜플(Tuple)의 개수를 의미한다.

두 릴레이션에 대한 조인 연산에서 특정 키 값에 몰린 데이터를 특정 노드로 전송한 후에 해당 데이터를 하둡 분산 파일 시스템에 저장한다. 그 후에 해당 데이터가 위치한 노드에서 조인 연산이 수행된다. 만약, 맵리듀스 기반의 분산 시스템이라면 매퍼가 실행될 것이고, SQL on Hadoop 계열의 분산 시스템이라면 분산 질의 엔진이 실행될 것이다. 조인 연산의 절차는 다음과 같다.

(1) 기본키를 가진 릴레이션 Ra에 대한 해쉬 테이블(Hash Table)을 구축한다.

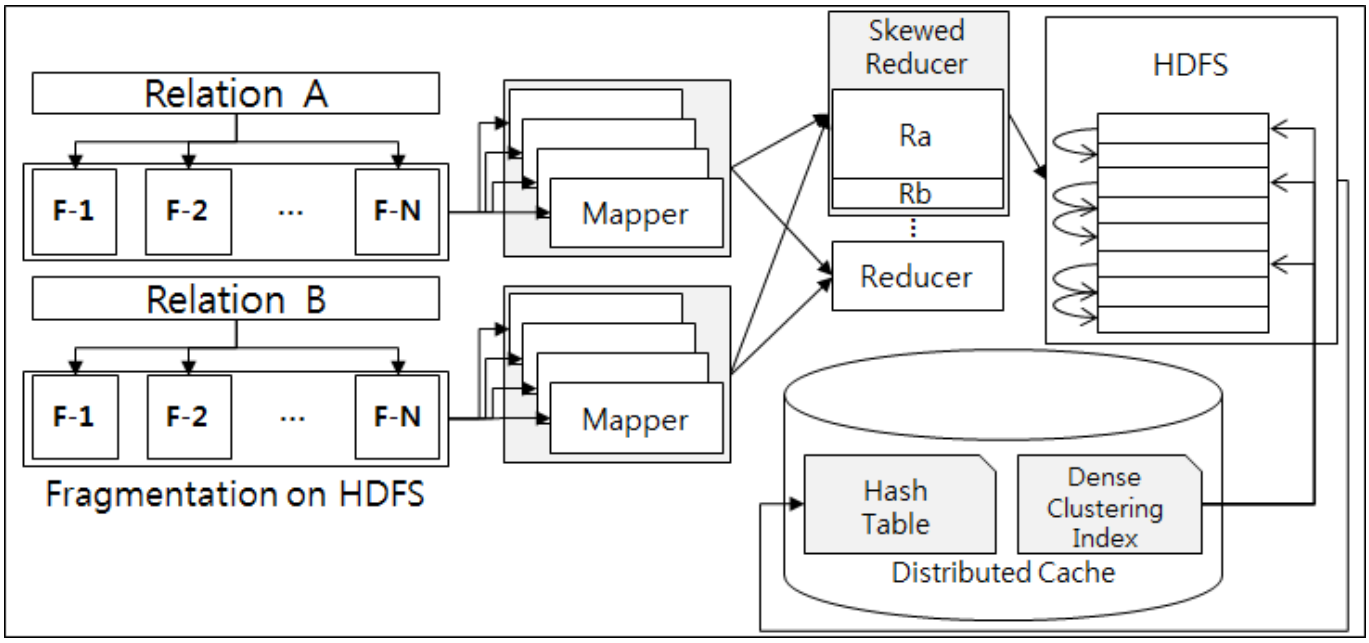


그림 1 클러스터화된 밀집 인덱스 구축

(2) 해당 해쉬 테이블을 분산 캐쉬에 저장한다.

(3) 각 노드에서 릴레이션 Rb의 튜플을 순차적으로 읽어 분산 캐쉬에 저장된 해쉬 테이블을 이용해 조인 연산을 수행한다.

이러한 조인 방법은 데이터의 불균등 분포로 인해 같은 키 값을 갖는 데이터의 비율이 높음에도 불구하고 모든 튜플에 대한 탐색을 수행하여 그 효율이 떨어진다.

### 3.2 제안 기법

본 논문에서 제안하는 불균등 분포 조인 기법은 인덱스 기반 조인 기법에 속한다. 전체적인 흐름은 그림 1에서 확인할 수 있다. 제안 기법 역시 앞서 3장 1절에서 가정한 전제를 그대로 따른다.

먼저 각각의 분산 질의 엔진 사이에 정해진 해쉬값에 따라 데이터를 전송한다. 각 엔진에서 데이터의 분포가 불균등일 경우, 기본키를 가진 릴레이션 Ra에 대한 해쉬 테이블을 구성하고, 외래키를 가진 릴레이션 Rb에 대해 외부 합병 정렬을 수행하여 해당 데이터를 하둡 분산 파일 시스템에 기록한다. 이 때, 릴레이션 Rb에 대한 밀집 인덱스를 모든 유일한(Unique) 키 값에 대해 생성한다. Rb의 데이터는 불균등 분포를 따르므로 밀집 인덱스의 크기는 분산 캐쉬에 저장하기에 충분한 것으로 가정한다. 이 과정은 아래 표 1에서 확인할 수 있다.

(표 1) 제안 기법의 알고리즘

**1. 밀집 인덱스 구축 단계**

```
for each node
  if cardinality(Rb) / num_of_rows(Rb) * 100%
      < Threshold
  then
    createHashTable(Ra);
    mergeSortAndCreateDenseIndex(Rb);
```

**2. 조인 단계**

```
for each tuple group s in dense index
  v = searchHashTable(value of s);
  if s exist in hash table
  then
    join tuple group s with v;
  else
    continue;
```

이후 단계에서는 릴레이션 Ra에 대한 해쉬 테이블과 릴레이션 Rb에 대한 밀집 인덱스를 활용해 조인연산을 수행한다. 해당 절차는 표 1의 조인 단계에서 나타난 바와 같다. 밀집 인덱스는 모든 유일한 튜플에 대해 생성되어 있으므로 인덱스만 선형 탐색하여 릴레이션 Ra의 해쉬 테이블과 비교하면 조인을 위한 탐색을 마칠 수 있다.

## 4. 비용 분석

제안된 기법의 연산 비용을 추정하기 위해 다음과 같은 비용 모델을 사용한다. 제안 기법은 기존의 방법과 비교해 밀집 인덱스를 위해 외부 합병 정렬을 수행할 때 메모리상에서 각 런(Run)을 정렬하여 해당 노드의 로컬(Local) 하드디스크에 임시 저장하는 단계와 임시 저장된 런의 블록을 하드디스크로부터 읽어 메모리상에서 전체 블록에 대한 정렬을 수행하는 단계가 추가된다. 런이란 외부 합병 정렬에서 전체 튜플을 일정 크기로 분할한 청크

(Chunk)를 의미하는데 일정 개수의 블록으로 구성된다. 또한 전체 런을 정렬 중에 밀집 인덱스를 구축하는 비용이 추가된다.

### 4.1 비용 기준(Cost Criteria)

먼저 외부 합병 정렬의 대상이 되는 전체 튜플의 개수를 n이라 하고, 알고리즘 수행 시 발생하는 블록 단위의 하드디스크 I/O의 횟수를 성능측정의 척도로 사용한다. 또한 조인 연산 수행 시 성능 측정을 위해 비교 횟수를 척도로 사용한다.

### 4.2 비용 추정(Cost Estimation)

아래의 단계에 따라 비용을 추정할 수 있다. 기존 알고리즘과 비교해서 가감된 부분만 계산한다. 릴레이션 R의 튜플 개수는 B(R)으로 나타내고, 메모리에 M개의 블록을 위한 공간이 있다고 가정할 때, 전체 런의 개수는 B(R)/(M-1)이다.

(1) 합병 정렬 시, 런 단위 정렬 및 I/O 비용
- 런 단위 인메모리 정렬 비용 : $O(B(R)\log B(R))$
- 블록 단위 I/O 비용 : 2B(Rb)

(2) 감소하는 탐색 비용
- 기존 : B(Rb) * H(Ra)
- 제안 기법 : C(Rb) * H(Ra)

(3) 전체 비용
- 기존 : B(Rb) * H(Ra)
- 제안 기법 : C(Rb) * H(Ra) + 2B(Rb)

단, H(R)은 해쉬 테이블의 탐색 비용이며, C(R)은 릴레이션 R의 키 값에 대한 밀집 인덱스의 카디널리티(Cardinality)를 의미한다.

결과적으로 특정 버킷에 값이 몰릴 경우, 중복된 키 값이 증가할수록 기존 방법의 효율이 떨어지는 경향을 보인다.

## 5. 결론 및 향후 연구 계획

데이터베이스의 병렬 처리(Parallel Processing)와 분산 처리(Distributed Processing)는 오랜 기간 동안 연구된 분야다. 연산과 데이터를 분산시켜 병렬 처리를 통해 부하를 분산하여 빠른 응답속도를 확보하고자 하는 노력은 계속되어왔지만, 불균등 분포 데이터의 조인 연산에서 발생하는 병목 현상은 지금까지도 해결되어야할 문제로 인식되고 있다[14].

본 논문에서는 불균등 분포 데이터의 조인 연산을 효율적으로 처리하기 위해 기존의 외부 합병 정렬 알고리즘과 밀집 인덱스를 활용하여 탐색 시간을 줄여 조인 연산의 수행시간을 줄이는 방법을 제시하였다.

향후 연구방향은 해당 방법론을 실제 오픈소스 분산 시스템에 적용하여 실험적 결과를 확보하고자 한다. 또한, 밀집 인덱스를 위해 생성한 정렬된 데이터를 일회성이 아닌 영구적으로 활용하여 해당 릴레이션에 대해 같은 해쉬 함수를 사용할 경우 유용하게 활용할 수 있는 방안을 찾고자 한다.

## 참고문헌

[1] White, Tom. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012.

[2] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.

[3] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[4] Thusoo, Ashish, et al. "Hive: a warehousing solution over a map-reduce framework." Proceedings of the VLDB Endowment 2.2 (2009): 1626-1629.

[5] Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." ACM Transactions on Computer Systems (TOCS) 26.2 (2008): 4.

[6] George, Lars. HBase: the definitive guide. O'Reilly Media, Inc., 2011.

[7] https://github.com/cloudera/impala

[8] Choi, Hyunsik, et al. "Tajo: A distributed data warehouse system on large clusters." Data Engineering (ICDE), 2013 IEEE 29th International Conference on. IEEE, 2013.

[9] Xin, Reynold S., et al. "Shark: SQL and rich analytics at scale." Proceedings of the 2013 international conference on Management of data. ACM, 2013.

[10] S. Dasgupta, C. Papadimitriou, and U. Vazirani, "Algorithms", 2008

[11] Melnik, Sergey, et al. "Dremel: interactive analysis of web-scale datasets." Proceedings of the VLDB Endowment 3.1-2 (2010): 330-339.

[12] Märtens, Holger. "A Classification of Skew Effects in Parallel Database Systems." Euro-Par 2001 Parallel Processing. Springer Berlin Heidelberg, 2001. 291-300.

[13] Mofidpoor, Mahsa, Nematollaah Shiri, and T. Radhakrishnan. "Index-based join operations in Hive." Big Data, 2013 IEEE International Conference on. IEEE, 2013.

[14] Lakshmi, M. Seetha, and Philip S. Yu. "Effect of skew on join performance in parallel architectures." Proceedings of the first international symposium on Databases in parallel and distributed systems. IEEE Computer Society Press, 2000.