공학석사 학위논문

맵리듀스 환경에서 블룸 필터를 사용한 탄력적 조인 연산 처리

Adaptive Join Processing Using Bloom Filter
in a MapReduce Environment

2013년 2월

서울대학교 대학원
전기·컴퓨터 공학부
배 혜 찬

맵리듀스 환경에서 블룸 필터를 사용한 탄련적 조인 연산 처리

지도교수 김 형 주

이 논문을 공학석사 학위논문으로 제출함
2012 년 12 월

서울대학교 대학원
전기·컴퓨터 공학부
배 혜 찬

배혜찬의 공학석사 학위논문을 인준함
2013 년 1 월

위 원 장	이 상 구	(인)
부위원장	김 형 주	(인)
위 원	김 선	(인)

초 록

대용량 데이터의 처리, 분석을 위해 분산 프로그래밍 모델인 맵리듀스가 여러 분야에서 활용되고 있다. 그러나 맵리듀스는 조인 연산을 처리할 때 조인되지 않는 레코드들까지 맵퍼에서 리듀서로 전송하는데, 이는 불필요한 네트워크 비용을 발생시켜 조인 성능을 저하시킨다. 이러한 문제를 개선하기 위해 맵리듀스에서 블룸 필터를 사용하여 리듀서로 전송되는 레코드를 미리 여과하는 조인 방법이 제안되었다. 하지만 블룸 필터에 삽입되는 원소 데이터의 개수가 너무 많아지는 경우, 필터의 이점을 기대할 수 없으며 필터를 사용하기 위한 추가적인 비용으로 인하여 블룸 필터를 사용하지 않고 처리하는 것보다 오히려 성능이 더 저하될 수 있다. 이에 본 논문은 주기적으로 블룸 필터의 효율성을 검사하여 필터의 사용여부를 동적으로 결정하는 탄력적 조인 연산 기법을 제안한다. 이를 위해, 우리는 필터에 삽입된 키의 개수를 활용하여 블룸 필터의 양성 오류율을 추정하고, 필터가 비효율적이라고 판단된 경우, 그 시점 이후로는 필터를 사용하지 않고 조인 연산을 처리하도록 한다. 실험을 통하여, 제안한 기법이 기본 맵리듀스 조인과 블룸 필터를 사용한 조인 중 보다 나은 성능을 보이는 연산 방법을 탄력적으로 선택함으로써 안정적인 조인 성능을 보장함을 확인한다.

주요어 : 맵리듀스, 블룸 필터, 양성 오류율, 조인, 탄력적 조인

학 번 : 2011-20854

목 차

표 목차

그림 목차

제 1 장
서론

정보 기술의 발전에 힘입어 생성되는 데이터의 양은 과거에 비해 가히 폭발적이다. 이런 방대한 양의 데이터를 분석하여 의미 있는 데이터를 찾아내는 것이야말로 가치 창출의 첫걸음이라 할 수 있다. 이전에는 쌓여만 가는 대용량 데이터를 처리할 방법이 없어 가치 창출의 수단으로 사용되지 못하고 버려지는 경우가 대부분이었지만, 오늘날에는 맵리듀스(MapReduce)[1]가 이를 처리할 수 있게 됨에 따라 여러 분야에서 새로운 가치를 창출하는데 활용되고 있다. 맵리듀스란 분산 환경에서 대용량 데이터셋을 처리할 수 있는 대표적인 프로그래밍 모델이다. 맵리듀스가 분산, 병렬 및 오류 처리에 대한 쉽고 편리한 기능들을 제공함으로써, 사용자는 내부적인 처리에 대한 걱정 없이 단일 컴퓨터에서처럼 처리하고자 하는 함수를 작성하는 것만으로 분산 처리를 구현할 수 있게 되었다.

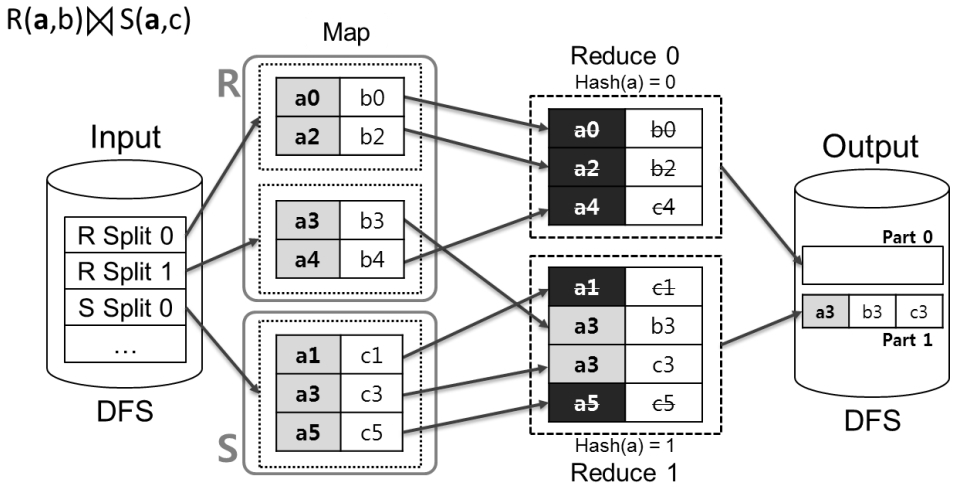


그림 1 맵리듀스 조인 연산의 한계점

대용량 데이터의 효과적인 분석을 위해 필수 불가결한 데이터셋들 사이의 조인 연산 또한 맵리듀스를 통해 수행할 수 있다. 하지만 맵리듀스를 이용한 조인 연산에는 한계점이 있다. 맵리듀스는 주로 단일 데이터셋을 처리하도록 설계되었기 때문에 여러 데이터셋들을 조인하는 데에는 적합하지 않은 부분이 존재한다는 것이다. 가장 큰 단점은 조인에 참여하는 같은 키를 가지는 레코드들을 찾기 위해서는 모든 입력 레코드들이 맵퍼(Mapper)에서 리듀서(Reducer)로 보내져야만 조인 연산을 수행할 수 있다는 점이다. 반대로 생각해보면, 조인에 참여하지 않는 레코드들 또한 리듀서로 전송된다는 것으로 이는 불필요한 네트워크 입출력(I/O) 비용을 발생시킨다. 그림 1은 그 한계점을 보여주는 대표적 예이다. 그림 1에서는 데이터셋 R과 S로부터 실제적으로 조인되는 레코드는 R의 〈a3, b3〉과 S의 〈a3, c3〉뿐이지만, 조인 연산을 수행하기 위해 모든 레코드들이 리듀서로 전송됨을 보여주고 있다. 이러한 불필요한 레코드들을 얼마나 줄일 수 있는지 여부에 따라 맵리듀스에서 조인 연산의 성능 향상 정도가 좌우된다.

맵리듀스에서 조인 연산의 성능을 개선하기 위해 [2, 3, 4] 등 많은 연구들이 활발하게 이루어지고 있는 가운데, [4]에서는 맵리듀스 환경에서 블룸 필터(Bloom

Filter)[5]를 이용하여 조인 연산의 성능을 개선시키는 방법이 제안되었다. 즉, 블룸 필터를 이용하여 사전에 조인에 참여하지 않는 레코드들을 걸러내어 리듀서로 전송되는 레코드들의 수를 줄임으로써 조인 연산의 성능을 개선시켰다. 이 방법은 블룸 필터에 추가되는 구분키(distinct key)의 수와 조인에 참여하는 레코드들의 수가 많지 않은 경우 큰 폭의 성능 향상을 보였다.

하지만 문제는 블룸 필터를 사용하여 조인 연산을 하는 것이 블룸 필터를 사용하지 않는 경우보다 성능이 더 나쁘게 나오는 경우가 존재한다는 것이다. 대표적으로 블룸 필터에 추가되는 구분키의 수가 너무 많아서 블룸 필터의 대부분 비트가 참(true)으로 설정되는 경우를 들 수 있다. 이 경우, 결국 대부분의 입력 레코드들이 리듀서로 전송되기 때문에 블룸 필터로 인한 조인 성능 향상을 기대할 수 없다. 그 뿐만 아니라, 블룸 필터를 생성, 공유 그리고 사용하는 과정에서 추가적인 CPU 및 네트워크 비용이 발생함으로 인해 블룸 필터를 사용하지 않은 경우보다 성능이 더 저하될 수 있다. 기존의 관계형 데이터베이스와는 달리 입력 데이터셋에 대한 통계적 정보를 미리 알지 못하는 맵리듀스에서 이러한 문제는 미리 예측하기 힘드나, 성능 개선에 상당한 영향을 미치는 문제이므로 간과할 수 없다.

이에 본 연구에서는 블룸 필터의 효율성을 주기적으로 검사하여 수행시간 중에 유연하게 블룸 필터의 사용여부를 결정하는 탄력적(Adaptive) 조인 연산 처리 기법을 제안한다. 만약 수행시간 중에 블룸 필터가 더 이상 효율적이지 않다고 판단되면, 그 시점에서 즉시 블룸 필터의 사용을 중단하고 기존 맵리듀스에서의 조인 연산 방법으로 회귀하게 된다. 효율성의 검사를 위해 블룸 필터의 긍정 오류(false positive)를 계산하고 그 값을 이용하는 것이 본 연구의 핵심이다. 하지만 맵리듀스 환경에서 입력 데이터셋에 대한 통계적 정보는 한정되어 있기 때문에, 수행시간 중에 얻을 수 있는 정보를 바탕으로 긍정 오류를 추정하는 기법을 사용한다. 유연하게 블룸 필터의 사용여부를 결정함으로써 우리는 맵리듀스에서 조인 연산의 최소한의 성능을 항상 보장할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 맵리듀스에서 조인 연산을 처리하는 기법들, 블룸 필터의 비효율성을 극복하기 위해 제안된 연구들 및 블룸 필터의 수학적 특성을 분석한 연구들을 살펴본다. 3장에서는 본 연구의 타겟인 맵리듀스에서

블룸 필터를 사용한 조인 연산과 그 한계점에 대해 살펴본다. 이를 개선한 탄력적 조인 연산 처리 기법을 4장에서 설명하고, 5장에서 그 실험 결과를 분석하여 우리가 제안하는 기법의 성능을 검증한다. 마지막 6장에서는 결론을 맺으며 논문을 마무리한다.

제 2 장
관련 연구

이 장에서 우리는 먼저 맵리듀스 환경에서 사용된 조인 연산 기법들에 대해 살펴본다. 이후 블룸 필터에 대해 간략히 소개한 후, 블룸 필터에서 발생할 수 있는 비효율성을 개선하기 위한 연구들과 블룸 필터의 수학적 특성에 관한 연구들을 살펴본다.

2.1 맵리듀스에서의 조인 연산 처리

최근 대용량 데이터의 처리와 분석에 대한 관심이 점차 높아져감에 따라 맵리듀스에서 조인 연산의 성능을 개선하는 연구가 활발히 이루어지고 있다. 제안된 다양한 기법들은 크게 두 가지 부류로 나눌 수 있는데, 맵-사이드 조인(map-side join)과 리듀스-사이드 조인(reduce-side join)이 그것이다. 각 부류는 맵리듀스의 어느 단계에서 실제적인 조인 연산이 일어나는지를 나타낸다.

맵-사이드 조인은 리듀스(reduce) 단계 이전에 조인 연산을 수행하므로 맵퍼(mapper)에서 리듀서(reducer)로 모든 레코드들을 전송할 필요가 없다. 불필요한 네트워크 입출력(I/O) 비용을 발생시키지 않으므로 가장 좋은 조인 연산 성능을 보장할 수 있다. 하지만 맵(map) 단계의 조인 연산은 많은 제약을 받기 때문에 특수한 상황에서만 사용이 가능하다 [7]. 맵-머지 조인(Map-Merge join) [2]은 맵

단계에서 입력 레코드들을 읽어 합치는 단순한 처리 기법이지만 이를 수행하기 위해서는 조인하는 2개의 데이터셋들이 사전에 조인 키(join key)에 의해 분할되고 정렬되어야 한다. 브로드캐스트 조인(Broadcast join) [3]은 하나의 데이터셋의 크기가 메모리에 전부 적재될 정도로 작은 경우에 사용할 수 있는 맵-사이드 조인이다. 이 경우 작은 사이즈의 데이터셋은 클러스터 내의 모든 컴퓨터로 즉시 복제되고 각 노드의 메모리에 적재되어 조인 연산이 수행된다.

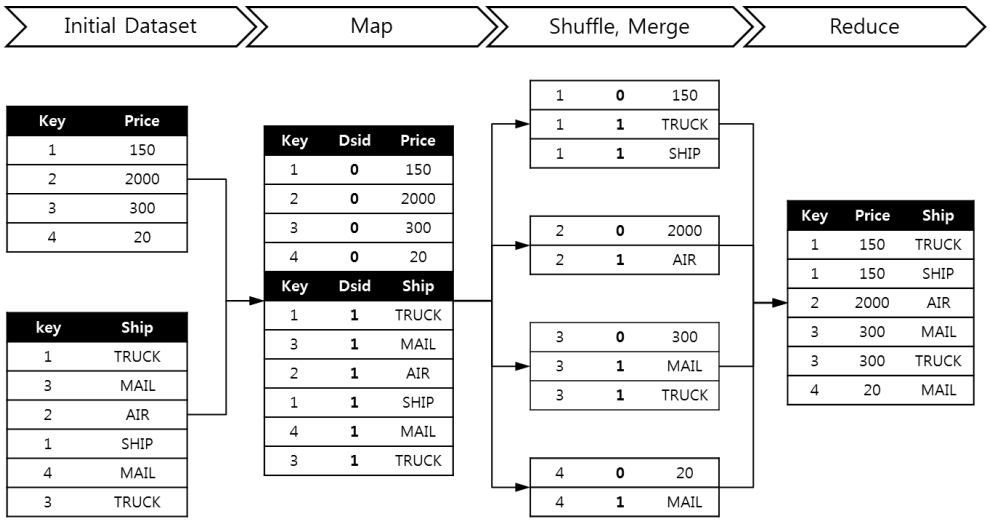


그림 2 재분할 조인(Repartition join)

반면 리듀스-사이드 조인은 레코드들이 맵퍼에서 리듀서로 전송되어야 하기 때문에 맵-사이드 조인에 비해서 성능은 떨어지지만, 제약 없이 모든 상황에서 조인 연산을 수행할 수 있다. 따라서 리듀스-사이드 조인이 일반적인 조인 연산이라 할 수 있다. 재분할 조인(Repartition join) [3]은 맵리듀스 가장 일반적으로 사용되는 리듀스-사이드 조인이다. 이 조인은 각각의 데이터셋에 고유 태그(tag)를 부여하고 같은 조인 키를 가지는 레코드들을 같은 리듀서로 전송한다. 리듀서는 데이터셋의 태그를 참조하여 조인 연산을 수행한다. 그림 2는 재분할 조인의 처리 과정을

설명한다. 그림 2의 예에서는 각각의 데이터셋에 0과 1의 고유 태그를 붙인 후, 같은 조인 키의 값을 가지는 레코드들끼리 묶는다. 최종적으로 같은 키의 값을 가지면서 서로 다른 태그값을 가지는 레코드들끼리 조인시키는 것으로 리파티션 조인을 수행한다. 세미-조인(Semi-Join) [3]은 리듀서로 전송되는 네트워크 비용을 줄이기 위해 필터의 개념을 도입했지만 조인 연산 수행을 위해 3개의 잡(job)이 필요하다는 문제점을 가지고 있다. [4]는 재분할 조인에서 리듀서로 보내지는 불필요한 레코드들을 줄이기 위해 블룸 필터를 사용하여 그에 따른 성능 향상을 보였다. [4]에 대해서는 다음 장에서 상세하게 살펴본다.

2.2 블룸 필터

본 연구의 기반이 되는 블룸 필터(Bloom filter) [5]에 대해서 설명한다. 최근 분산 시스템에서 정보 처리량과 네트워크 비용의 감소를 위해 확률에 기반한 기술들이 이용되고 있다. 그러한 기술들 중 가장 빈번하게 사용되는 기술 중 하나가 블룸 필터이다 [6]. 블룸 필터는 작은 공간으로 데이터셋을 표현하고 어떤 원소 데이터가 그 데이터셋에 속하는지를 빠른 시간 내에 판별해주는, 확률에 기반한 자료 구조이다. 블룸 필터는 고정된 m개의 비트를 가지고 k개의 독립적인 해쉬 함수를 사용하여 데이터셋에 속한 n개의 원소 데이터를 추가하는 것으로 그 데이터셋을 표현할 수 있다. 즉, 원소 데이터를 k개의 해쉬 함수에 적용한 결과값들에 대응되는 필터의 위치 비트를 참(true)으로 설정하여 추가한다. 이후에 어떤 원소 데이터가 데이터셋에 속하는지 판단할 때에는, 같은 방법으로 k개의 해쉬 함수를 적용해 그 결과값들에 대응되는 필터의 위치 비트 값을 비교한다. 대응되는 모든 비트 값들이 참이라면 그 원소 데이터는 데이터셋에 속해있다고 판단한다.

블룸 필터는 해쉬 함수를 사용하여 원소 데이터를 추가하기 때문에 긍정 오류(false positive)가 발생할 수 있다. 긍정 오류란, 실제로는 데이터셋에 속하지 않은 원소 데이터가 블룸 필터에 의한 판단 결과 데이터셋에 속한다고 결론을 내리는 상황을 말한다. n개의 원소 데이터를 블룸 필터에 추가한 후의 긍정 오류율(false positive rate)은 다음과 같다.

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-\frac{kn}{m}}\right)^{k}$$

[8]에 따르면, 블룸 필터에 추가된 원소 데이터의 개수 n에 대한 블룸 필터의 비트 수 m 즉, $\frac{m}{n}$ 과 양성 오류율 사이에는 트레이드오프(trade-off) 관계가 있다.

2.3 블룸 필터의 비효율성 개선

2.2절에서 언급한 바와 같이 블룸 필터는 고정된 크기를 가진다. 그렇기 때문에, 블룸 필터에 추가되는 원소 데이터의 개수를 사전에 파악할 수 없는 경우, 블룸 필터의 비효율성으로 인해 오히려 성능 저하를 야기할 수 있다. 블룸 필터에서 발생할 수 있는 이와 같은 비효율성을 개선하기 위한 연구들이 활발히 이루어지고 있다.

[9]에서는 높은 확률로 긍정 오류가 발견되는 상황을 블룸 역설(Bloom paradox)이라고 정의하고, 데이터의 선험적 확률(a priori probability)을 통해 역설을 일으킬 가능성이 높은 원소 데이터를 무시하고 블룸 필터를 만드는 선택적 블룸 필터(Selective Bloom filter)를 제안하였다. 하지만 이것은 원소 데이터들의 분포확률을 파악하고 있는 경우에만 적용할 수 있다. [10, 11]에서는 스트리밍이나 프록시와 같은 네트워크 어플리케이션을 위한 개량된 블룸 필터를 제안하였다. 여기에 사용되는 데이터들은 무한하게 입력될 수 있는 유형들이기 때문에 시간이 지날수록 양성 오류율이 1에 가까워지는 것은 자명한 일이다. 이를 방지하기 위해 버퍼 교체 정책(buffering replacement policy)과 같이 블룸 필터에 추가된 지 오래된 원소 데이터를 삭제하는 방법으로 양성 오류율을 어느 정도 수준으로 유지시켰다. 다만, 원소 데이터를 삭제하기 때문에 음성 오류(false negative)가 발생할 수 있으며 음성 오류가 허용되지 않는 조인 연산에는 사용할 수 없다.

2.4 블룸 필터의 수학적 특성

블룸 필터의 활용도가 높아짐에 따라 블룸 필터의 수학적 특성에 관한 연구 또한

활발히 진행되고 있다. 특히 블룸 필터를 사용하는 데에 필요한 정보가 충분치 않은 네트워크나 분산 데이터베이스와 같은 분야에서는, 충분치 않은 정보를 추정하는 다양한 방법들이 연구되었다.

[12]은 블룸 필터에 추가된 구분 원소(distinct element)의 개수를 모를 때 블룸 필터에서 참으로 설정된 비트 수를 이용해 추정하는 방법을 제안하였으며, 그 식은 다음과 같다.

$$\hat{S}^{-1}(t) = \frac{\ln\left(1-\frac{t}{m}\right)}{k \times \ln\left(1-\frac{1}{m}\right)}$$

추정된 구분 원소의 개수는 $1-\mathrm{e}^{\mathrm{t}-1-\hat{S}(n_l)} \times \left(\frac{\hat{S}(n_l)}{t-1}\right)^{t-1} - e^{-\frac{\left(t+1-\hat{S}(n_r)\right)^2}{2\hat{S}(n_r)}}$ 이상의 확률로 $n_l \le \hat{S}^{-1}(t-1)$ 과 $\hat{S}^{-1}(t+1) \le n_r$ 을 만족하는 범위 (n_l, n_r)에 속하는 값을 가진다. [13]는 2개의 블룸 필터를 합집합(union)할 경우의 양성 오류율을 다음과 같은 식으로 추정하는 방법을 제안하였다.

$$\mathrm{P}_{\mathrm{error}}\left(BF_{OR(BF_1,BF_2)}\right) = \left(\left(1-\left(1-\frac{1}{m}\right)^{kn_1}\right)+\left(1-\left(1-\frac{1}{m}\right)^{kn_2}\right)-\left(1-\left(1-\frac{1}{m}\right)^{kn_1}\right) \times \left(1-\left(1-\frac{1}{m}\right)^{kn_2}\right)\right)^{\mathrm{k}}$$

본 연구에서는 블룸 필터의 이러한 수학적 특성들을 이용하여 필터의 효율성을 검사할 수 있다.

제 3 장
맵리듀스에서 블룸 필터를 사용한 조인 연산

이 장에서는 본 연구의 타겟인 맵리듀스에서 블룸 필터를 이용한 조인 연산 [4]에 대해 설명한다. 이 연구는 블룸 필터를 이용하여 조인 연산을 수행하는 블룸 조인(Bloomjoin) [14]을 맵리듀스의 오픈 소스 프레임워크인 하둡(Hadoop) [15]에 적용하여 조인 성능을 개선하였다.

1장에서 언급한 그림 1과 같이 조인에 참여하지 않음에도 불구하고 리듀서로 보내지는 중간 레코드들로 인해 발생하는 네트워크 비용을 줄이기 위하여 블룸 필터가 사용되었다. 이 기법의 핵심은 두 개의 데이터셋들에 대해 조인 연산을 수행할 때, 첫 번째 데이터셋으로부터 블룸 필터를 생성하고 그 필터를 사용해 두 번째 데이터셋의 불필요한 레코드들 즉, 조인에 참여하지 않는 레코드들을 걸러내는 것이다. 이 수행작업은 전부 맵 단계에서 이루어지며, 걸러진 레코드들에 의한 네트워크 비용의 감소가 전체 조인 연산의 수행시간을 단축시킨다. 하지만 하둡은 기본적으로 입력 데이터셋들을 여러 개의 스플릿(split)들로 나눈 후 데이터셋의 순서나 데이터셋 내의 스플릿 위치 순서에 상관없이 태스크를 할당하기 때문에, 특정 데이터셋으로부터 블룸 필터를 생성할 수 없다는 문제점이 있다. 이를 해결하기 위해 이 연구에서는 여러 데이터셋들이 입력으로 들어오는 경우, 입력 데이터셋의 순서대로 처리가 되도록 구현하였다.

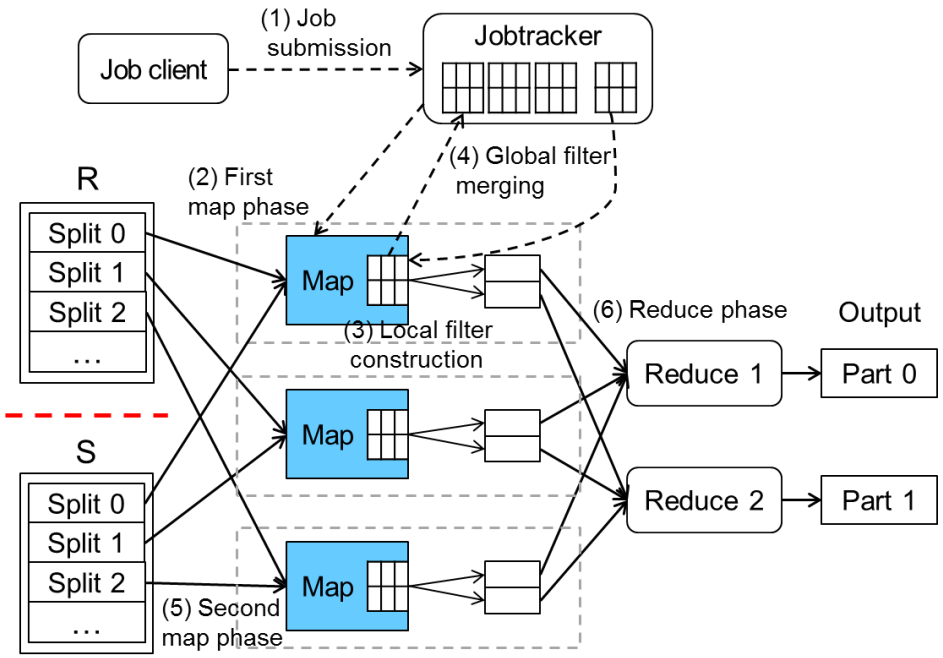


그림 3 맵리듀스에서 블룸 필터를 이용한 조인 연산 실행 흐름[1]

그림 3은 이 연구에서 구현한 데이터셋 R과 S의 조인 연산에 대한 실행 흐름을 보이고 있다. 여기서 R을 첫 번째 데이터셋 즉, 블룸 필터를 생성하는 데이터셋인 생성 입력(build input)으로 정의하고, S를 두 번째 데이터셋 즉, 블룸 필터로부터 걸러내는 데이터셋인 탐색 입력(probe input)으로 정의하였다. 조인 연산의 흐름은 다음의 6단계를 거친다.

1. **잡 제출(Job submission).** 잡이 제출되면 R을 위한 m_1개의 맵 태스크(map task), S를 위한 m_2개의 맵 태스크 그리고 r개의 리듀스 태스크(reduce task)가 생성된다.

[1] Figure 3 of Join Processing Using Bloom Filter in MapReduce, In RASC '12

2. **첫 번째 맵 단계(First map phase).** 잡트래커는 m_1개의 맵 태스크 또는 리듀스 태스크를 태스크트래커들에게 할당한다. 한 개의 맵 태스크트래커는 입력 스플릿을 읽어 키/값 쌍으로 변환한 후 맵 함수(map function)를 수행한다.

3. **지역 필터 생성(Local filter construction).** 첫 번째 맵 단계의 결과로 생성된 중간 레코드들은 r개의 파티션으로 나누어져 r개의 태스크트래커로 각각 보내진다. 또한 각 태스크트래커는 파티션의 개수만큼인 r개의 블룸 필터들을 생성한다. 이 필터들은 하나의 태스크트래커에서만 사용되므로 지역 필터(local filter)라고 정의한다.

4. **전역 필터 병합(Global filter merging).** m_1개의 맵 태스크의 실행이 모두 완료되면, 잡트래커는 하트비트(heartbeat) 신호를 통해 모든 태스크트래커의 지역 필터들을 취합한다. 이를 병합하여 빌드 입력 R에 대한 전역 필터(global filter)를 생성한다. 잡트래커는 다시 모든 태스크래커에게 완성된 전역 필터를 전송하여 그것을 공유한다. 전역 필터의 생성과 공유가 완료될때까지, 잡트래커는 탐색 입력 S에 대한 맵 태스크를 할당하지 않는다.

5. **두 번째 맵 단계(Second map phase).** 잡트래커는 m_2개의 맵 태스크 또는 리듀스 태스크를 태스크트래커들에게 할당한다. 이때 태스크트래커는 공유한 전역 필터를 사용하여 필터에 속하지 않는 탐색 입력 S의 레코드들을 걸러낸다.

6. **리듀스 단계(Reduce phase).** 하둡의 기본 조인 연산 방식과 같으며, 이 단계에서 실질적인 조인 연산이 이루어진다.

맵리듀스에서 블룸 필터를 이용한 조인 연산은 조인되는 레코드들의 수가 적거나 빌드 입력의 구분 키(distinct key)의 개수가 적은 경우, 블룸 필터를 사용하지 않을 때 대비 최고 2배 빠른 수행 시간을 보임으로써 조인 연산 성능의 개선을 입증하였다. 하지만 모든 경우에 대해서 블룸 필터가 성능 향상을 보인 것은 아니다. 오히려 블룸 필터를 사용하지 않을 때보다 더 좋지 않은 성능을 보이는 경우도

존재했다.

이러한 성능 저하가 발생할 수 있는 상황을 살펴보면 첫 번째는 데이터셋의 대부분 레코드들이 조인에 참여하는 경우(high join cardinality)이다. 이 경우, 대부분 레코드들이 조인되므로 필터에 의해 걸러진 레코드들의 수가 적고 따라서 성능 향상 또한 미미하다. 두 번째는 블룸 필터의 양성 오류율(false positive rate)이 높아지는 경우이다. 2.2절에서 언급한 것처럼, 블룸 필터의 비트 수가 적고 필터에 추가되는 원소 데이터의 개수가 많을수록 양성 오류율은 증가한다. 양성 오류율이 높아진다는 것은 조인되지 않는 많은 레코드들 또한 블룸 필터에 의해 걸러지지 않음을 의미하며 이는 성능 저하로 연결된다. 데이터셋들에 대한 통계적 정보를 사전에 계산해놓고 질의 시에 이용할 수 있는 관계형 데이터베이스와는 달리, 맵리듀스는 수행하기 전에 그 정보를 미리 파악할 수 없다. 따라서 블룸 필터의 크기는 입력 데이터셋의 크기에 따라 동적으로 정해질 수 없고, 이것은 블룸 필터에 추가되는 원소 데이터의 개수에 의해 양성 오류율이 결정된다는 것과 일맥상통한다.

블룸 필터로 인한 성능 개선을 기대할 수 없는 경우, 지역 필터를 생성, 탐색하는 CPU 비용과 전역 필터를 합병, 공유하는 네트워크 비용이 추가로 발생하여 블룸 필터를 사용하지 않는 기본적인 맵리듀스 조인 연산보다 성능이 더 저하될 수 있다. 이에 본 연구에서는 블룸 필터의 효율성을 주기적으로 검사하여 효율적이라고 판단될 때에만 블룸 필터를 사용하는 탄력적 조인 기법을 제안한다.

제 4 장
블룸 필터의 탄력적 조인 연산

이 장에서는 우리가 제안하는 블룸 필터의 탄력적 조인 연산 기법을 설명한다. 입력 데이터셋에 대한 통계적 정보가 없어 블룸 필터의 크기 m 을 임의로 고정하여 사용한다면, 결국 블룸 필터에 추가되는 원소 데이터들의 개수 n 과 양성 오류율(false positive rate)이 밀접한 관계에 있다고 할 수 있다. 따라서 필터에 추가된 원소 데이터들의 개수를 중심으로 블룸 필터의 효율성을 검사하여 블룸 필터의 사용 여부를 결정하였다. 효율성의 검사와 필터의 사용여부에 관한 정보들은 하트비트(heartbeat) 신호에 의해 태스크트래커에서 잡트래커로, 그리고 잡트래커에서 태스크트래커로 전달된다.

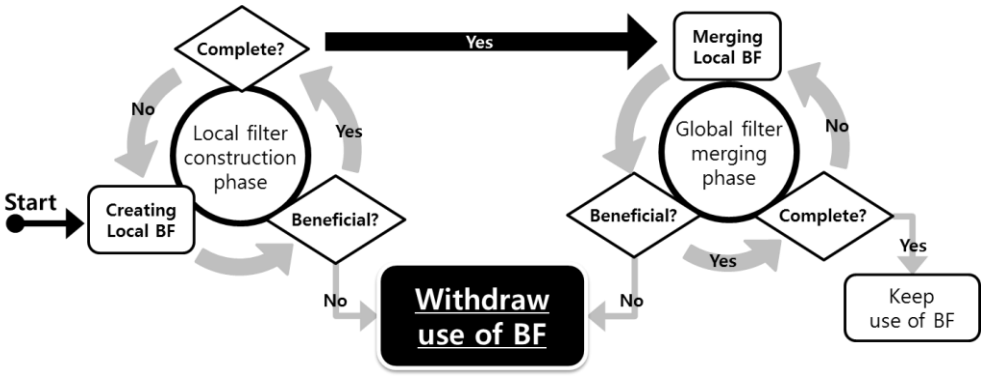


그림 4 전체 시스템 구성도

그림 4 는 우리가 제안하는 기법의 전체 시스템 구성도를 나타낸다. 시스템은 크게 두 단계로 블룸 필터의 효율성을 검사한다. 첫 번째는 지역 필터를 생성하는 단계(local filter construction)에서 효율성을 검사하고, 두 번째는 전역 필터를 병합하는 단계(global filter merging) 에서 효율성을 검사한다. 어느 단계든 계산 또는 추정된 블룸 필터의 양성 오류율이 정해진 임계값을 넘어 그 필터가 비효율적이라고 판단되면, 그 시점에서 잡트래커는 블룸 필터 사용의 취소를 모든 태스크트래커들에게 알림으로써 기존의 맵리듀스 조인 연산 방식으로 회귀하게 된다.

4.1 하트비트 신호와 엄빌리컬 규약

하둡(Hadoop)은 내부적으로 하나의 잡트래커와 여러 개의 태스크트래커들 사이에서 정보를 송수신하는 방법으로 하트비트 신호(heartbeat signal)를 사용한다. 그림 5 에서 보는 바와 같이, 이 신호는 각 태스크트래커에서 잡트래커로 주기적으로 발생되며, 여기에는 태스크트래커의 현재 상태 및 태스크의 진행 정보를 포함한다. 또한 하트비트 신호의 응답(response)으로 잡트래커는

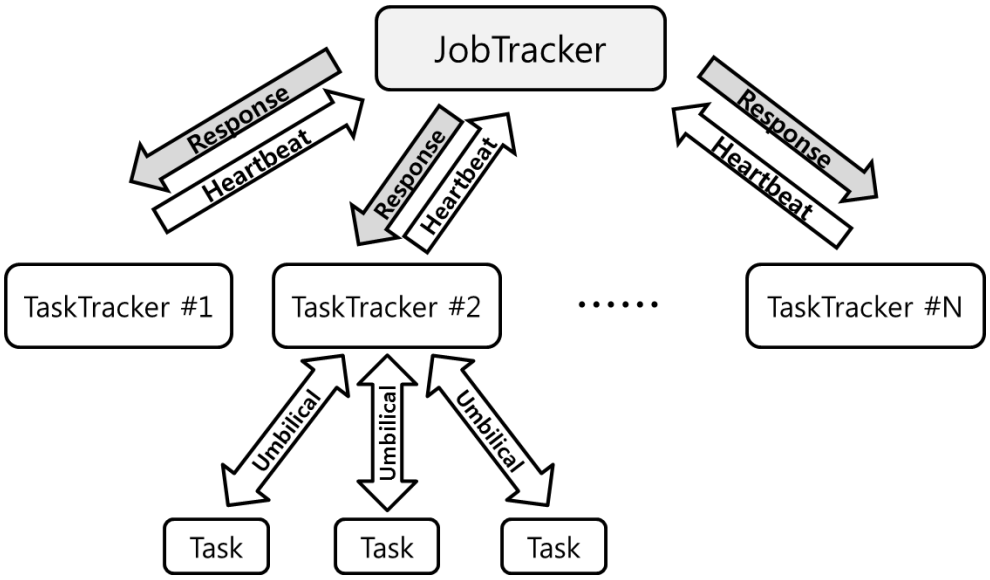


그림 5 하트비트 신호와 엄빌리컬 규약

하트비트 신호를 보내온 태스크트래커에게 새로운 태스크를 할당하는 등의 관리를 할 수 있다. 이 신호는 잡트래커와 태스크트래커 사이의 유일한 통신 채널로서 여기에 효율성의 검사를 위한 통계 정보를 피기배킹(piggybacking) 할 수 있다. 다만, 맵리듀스를 구성하는 클러스터의 크기가 커질수록 1 개의 노드로만 구성되는 잡트래커의 오버헤드는 커질 수 밖에 없다. 따라서 네트워크 I/O 비용을 최소로 할 수 있는 한정된 정보를 피기배킹하는 것이 중요하다.

하나의 태스크트래커는 여러 개의 태스크를 할당해 동시에 수행할 수 있다. 때문에, 각각의 태스크가 서로에게 그리고 태스크트래커에게 영향을 미치지 않아야 한다. 이를 위해 태스크트래커는 새로운 자바 가상 머신(JVM)을 실행하여 그곳에서 태스크를 수행하도록 한다. 이렇게 독립적으로 구성되는 태스크트래커와 태스크 사이의 통신을 위하여 엄빌리컬 규약(umbilical protocol)을 정의하여 사용한다. 이 규약을 통해 우리는 태스크트래커의 지역 필터를 태스크들과 공유할 수 있고, 각 태스크의 수행 결과 생기는 필터를 태스크트래커의 지역 필터로 병합함으로써 하나의 지역 필터를 유지할 수 있다.

4.2 지역 필터를 생성하는 단계에서의 효율성 검사

지역 필터를 생성하는 단계에서 효율성을 검사하는 것은 전역 필터를 병합하는 단계에 도달하기 전에 이미 너무 많은 원소 데이터들이 블룸 필터에 추가된 경우를 고려하기 위함이다. 예를 들어, 빌드 입력(build input)으로부터 블룸 필터를 생성하는 과정에서 대략 절반 정도의 진행률에 도달했음에도 불구하고 이미 필터의 대부분 비트가 참(true)으로 설정되어 양성 오류율이 지나치게 높아진다면, 지역 필터들을 병합하여 최종 완성된 전역 필터의 양성 오류율을 계산할 때까지 블룸 필터 생성 작업을 계속하는 것은 불필요한 CPU 와 네트워크 I/O 비용을 낭비하는 것이다. 이 단계에서 블룸 필터의 사용을 취소하는 경우를 케이스 1 로 표현하며, 표 1 은 케이스 1 에서 절감할 수 있는 비용들을 나타낸다.

블룸 필터의 취소 케이스 1. 여러 지역 필터들로부터 추정된 양성 오류율이 임계값을 넘어 블룸 필터의 사용을 취소하는 경우

절감 가능한 비용	절감 여부
지역 필터를 완성하는데 낭비되는 비용	○
전역 필터를 합병 & 공유하는데 낭비되는 비용	○
탐색 입력으로부터 필터를 탐색하는데 낭비되는 비용	○

표 1 케이스 1에서 절감 가능한 비용들

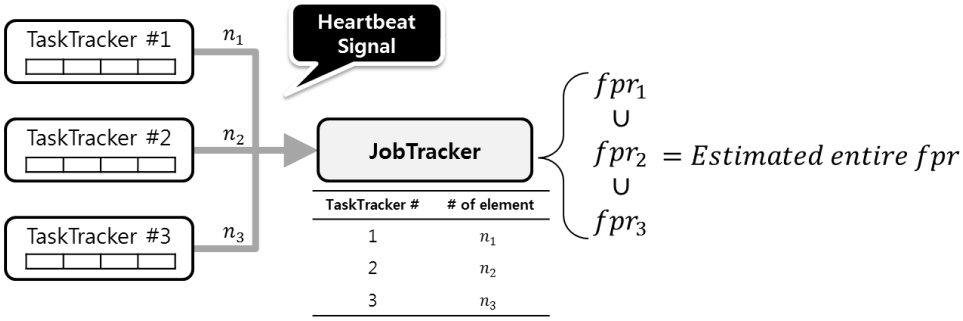


그림 6 지역 필터를 생성하는 단계에서의 양성 오류율 추정

그림 6 은 지역 필터를 생성하는 단계에서 양성 오류율을 추정하는 흐름을 보여준다. 이 단계에서 각 태스크트래커는 지역 필터를 생성하는 중이므로, 전역 필터를 병합하는 단계에 도달하기 전까지는 지속적으로 전체 태스크트래커들의 양성 오류율을 추정하여 블룸 필터의 효율성을 검사할 수 있어야 한다. 그와 동시에, 잡트래커로 병목 현상이 생기지 않도록 하트비트 신호에 피기배킹하는 통계 정보는 최소가 되어야 한다. 이 두 가지 조건을 모두 만족시키기 위해, 각 태스크트래커는 자신의 블룸 필터 자체가 아닌, 블룸 필터에서 현재까지 추가된 빌드 입력의 원소 데이터 개수를 하트비트 신호에 실어 보낸다. 잡트래커는 내부적으로 모든 태스크트래커로부터 수신한 원소 데이터 개수들을 배열에 담아 관리하고 있으며, 이 값들을 사용해 태스크트래커들이 관리하는 지역 필터들 전체의 양성 오류율을 추정한다. [13]에서 소개한 2 개 블룸 필터들의 합집합에 대한 양성 오류율 추정 공식을 확장하여 전체 블룸 필터들의 합집합에 대한 양성 오류율을 추정한다.

$$\mathrm{P_{error}} = {P'_{error}}^{k}(r)$$

$$\mathrm{P}'_{\mathrm{error}}(r) = \begin{cases} P'_{error}(r-1) + P_{bitSetTo1}(r) - P'_{error}(r-1) \times P_{bitSetTo1}(r), & r > 0 \\ 0, & r = 0 \end{cases}$$

$$\mathrm{P}_{\mathrm{bitSetTo1}}(r) = 1 - \left(1 - \frac{1}{m}\right)^{kn_r}$$

$\mathrm{P}_{\mathrm{bitSetTo1}}(r)$은 r 번째 블룸 필터에서 주어진 한 개의 비트가 참(true)일 확률을 의미한다. [13]의 공식을 활용하면 $\mathrm{P}'_{\mathrm{error}}(r)$을 통해 전체 지역 필터들을 합집합 했을 때, 한 개의 비트가 참일 확률을 추정할 수 있다. 따라서 전체 지역 필터들에 대한 양성 오류율은 $\mathrm{P}_{\mathrm{error}}$를 통해 구할 수 있다.

일반적으로 태스크트래커들에 존재하는 각 지역 필터의 양성 오류율이 모두 일정하게 증가하는 것과는 달리, 빌드 입력의 데이터 불균형(data skew)이 심한 경우, 다른 태스크트래커의 지역 필터들에 비해 특정 태스크트래커에 존재하는 지역 필터의 양성 오류율이 유난히 높을 수 있다. 이 경우, 전체 필터의 양성 오류율을 추정할 필요없이 하나의 태스크트래커에 대한 지역 필터에 대한 정보만으로도 비효율성을 판단할 수 있다. 이 경우를 케이스 1 확장으로 표현한다.

블룸 필터의 취소 케이스 1 확장. 하나의 지역 필터로부터 계산된 양성 오류율이 임계값을 넘어 블룸 필터의 사용을 취소하는 경우

4.1 절에서 언급한 것과 같이, 맵리듀스의 잡트래커는 클러스터의 규모가 커지더라도 1 개의 노드로 고정되어 있기 때문에, 확장성을 고려하여 조금이라도 CPU 비용을 줄이려는 노력은 필요하다. 따라서 잡트래커는 블룸 필터에 추가된 원소 데이터의 개수를 보낸 특정 태스크트래커의 지역 필터에 대한 양성 오류율을 먼저 O(1) 시간 내에 계산하여 비효율성을 검사함으로써 전체 필터들의 양성 오류율을 O(n) 시간 내에 추정하는 작업을 생략할 수 있어 CPU 비용을 절감할 수 있다.

4.3 전역 필터를 병합하는 단계에서의 효율성 검사

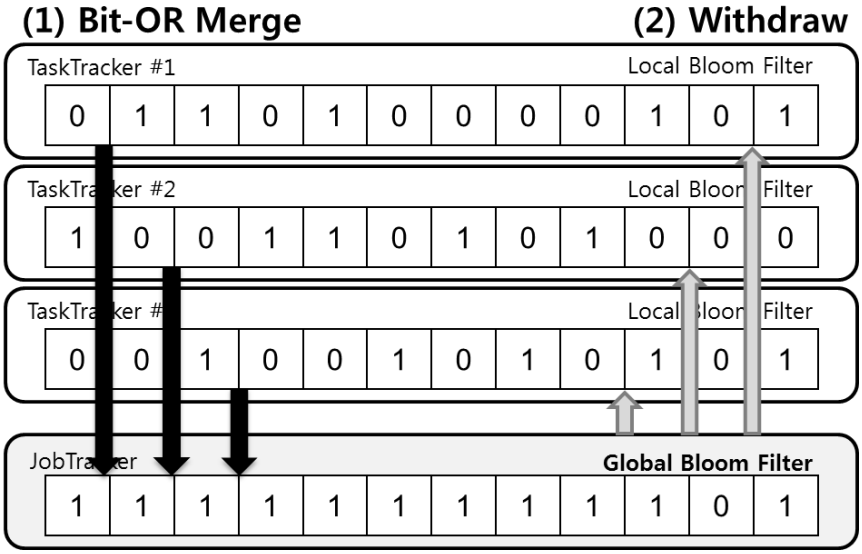


그림 7 전역 필터를 병합하는 단계에서의 효율성 검사

전역 필터를 병합하는 단계에서의 효율성 검사는 그림 7 과 같은 흐름으로 이루어진다. 잡트래커가 빌드 입력에 해당하는 모든 태스크를 태스크트래커들에게 할당하면, 잡트래커는 이후에 들어오는 모든 태스크트래커들에게 자신의 지역 필터를 보내라는 명령을 전달한다. 이 명령을 수신한 태스크트래커는 다음 하트비트 신호에 자신의 지역 필터를 첨부하여 잡트래커에게 전송하고 모든 지역 필터들을 병합하고 전역 필터가 완성되어 공유될 때까지 대기하게 된다. 잡트래커는 태스크트래커로부터 지역 필터를 수신할 때마다 전역 필터에 병합 작업을 수행한 후, 생성 중인 전역 필터의 효율성을 검사한다. 만약 생성 중인 전역 필터가 비효율적이라고 판단되면, 즉시 전역 필터의 생성 작업을 중단함과 동시에 블룸 필터의 사용을 취소한다. 이 단계에서 블룸 필터의 사용을 취소하는 경우를 케이스 2 로 표현하며, 표 2 는 케이스 2 에서 절감할 수 있는 비용들을 나타낸다

블룸 필터의 취소 케이스 2. 병합된 전역 필터로부터 계산된 양성 오류율이 임계값을 넘어 블룸 필터의 사용을 취소하는 경우

절감 가능한 비용	절감 여부
지역 필터를 완성하는데 낭비되는 비용	X
전역 필터를 합병 & 공유하는데 낭비되는 비용	△
탐색 입력으로부터 필터를 탐색하는데 낭비되는 비용	○

표 2 케이스 2에서 절감 가능한 비용들

이 단계에서는 지역 필터들로부터 전역 필터를 생성하는 단계이므로, 각 태스크트래커는 자신의 지역 필터를 완성시킬 수 밖에 없다. 따라서 케이스 2 에 의해 블룸 필터의 사용이 취소되는 경우, 지역 필터를 완성을 위한 비용은 절감할 수 없다. 또한 잡트래커는 지역 필터를 수신할 때마다 전역 필터를 병합하므로 생성 중인 전역 필터의 효율성 검사는 최대 리듀서의 개수인 r 번만큼 이루어질 수 있다. 이것은 일반적으로는 전역 필터를 합병하는데 낭비되는 비용을 케이스 1 만큼은 아니더라도 어느 정도는 절감할 수 있지만, 최악의 경우에 그 비용은 절감할 수 없다는 것을 의미한다. 이를 표 2 에서는 △로 표시하였다. 하지만 최종적으로는 블룸 필터의 사용을 취소하는 경우이기 때문에, 탐색 입력으로부터 필터를 탐색하는데 낭비되는 비용은 여전히 절감할 수 있다.

지역 필터들을 병합하는 과정에서 생기는 문제는 병합된 블룸 필터에서 추가된 원소 데이터의 개수를 알 수 없다는 것이다. 왜냐하면 병합은 두 개의 블룸 필터들을 비트-OR 연산한 결과이기 때문이다. 빌드 입력의 원소 데이터가 중복을 허용하지 않는 경우라면, 병합하려는 두 개의 블룸 필터에서 추가된 각각의 원소 데이터의 개수들의 더함으로써 병합된 블룸 필터에서 추가된 원소 데이터의 개수를 얻을 수 있다. 하지만 빌드 입력의 원소 데이터가 중복을 허용하는 경우라면, 그 방법을 사용할 수 없다. 이 문제를 해결하기 위해, 우리는 [12]의 방법을 사용하였다. 이것은 블룸 필터에서 참(true)으로 설정된 비트의 개수로부터 원소 데이터의 개수를

추정하는 방법이다. 이 방법을 통해 생성 중인 전역 필터에 추가된 원소 데이터의 개수를 추정하고, 이 값을 사용해 필터의 양성 오류율을 계산하여 그 효율성을 검사한다.

4.4 블룸 필터 사용의 취소

4.2 절과 4.3 절에서 설명한 바와 같이, 본 연구에서 제안한 조인 기법은 두 가지 케이스에서 각각 블룸 필터의 양성 오류율을 계산 또는 추정하여 그 효율성을 검사한다. 계산한 양성 오류율이 미리 정의된 임계값 이상이 되는 경우, 그 필터는 비효율적이라고 판단된다. 표 3 에서는 비효율적이라고 판단하여 블룸 필터의 사용을 취소하는 각 케이스 별로 절감 가능한 비용들을 정리하였다. 이를 통해 블룸 필터가 비효율적일 때, 케이스 1 에서 블룸 필터의 사용을 취소하는 것이 낭비되는 비용을 더 절감할 수 있다는 사실을 알 수 있다.

절감 가능한 비용	절감 여부	
	케이스 1	케이스 2
지역 필터를 완성하는데 낭비되는 비용	○	X
전역 필터를 합병 & 공유하는데 낭비되는 비용	○	△
탐색 입력으로부터 필터를 탐색하는데 낭비되는 비용	○	○

표 3 각 케이스 별로 절감 가능한 비용들

리듀서의 개수가 r 개만큼 존재하면 모든 지역 필터와 전역 필터는 내부적으로 r 개만큼의 블룸 필터를 가지기 때문에, 전부가 아닌 일부 블룸 필터들만이 비효율적이라고 판단될 수 있다. 이 경우 전체적인 조인 연산의 흐름 상에서는 블룸 필터의 사용을 계속 유지하면서, 비효율적이라고 판단된 일부 블룸 필터들만을

사용하지 않을 수 있다. 그렇게 사용이 취소된 일부 블룸 필터에 대해서는 탐색 입력이 블룸 필터를 탐색하는데 낭비되는 CPU 비용을 절감할 수 있다. 하지만 대부분의 블룸 필터가 비효율적이라고 판단되는 경우는 전체 블룸 필터의 사용 자체를 취소함으로써, 기존의 맵리듀스 조인 연산 방식으로 회귀할 수 있다.

우리는 이러한 블룸 필터들의 비효율 정도를 고려하기 위해 구현 과정에서 두 가지 TaskTrackerAction 클래스인 ReceiveFilterBeneficialStateAction 클래스와 WithdrawalFilterAction 클래스를 추가했다. 잡트래커가 일부 블룸 필터들을 비효율적이라고 판단한 경우, 모든 태스크 트래커들에게 전체 필터들의 효율성 정보를 포함하는 ReceiveFilterBeneficialStateAction 을 보낸다. 이를 수신한 태스크트래커는 비효율적이라고 판단된 특정 필터들을 파악함으로써 남은 태스크 수행에 해당 필터들을 사용하지 않는다. 잡트래커가 일부가 아닌 대부분의 블룸 필터들을 비효율적이라고 판단한 경우, 모든 태스크트래커들에게 블룸 필터 사용의 취소를 알리는 WithdrawalFilterAction 을 보낸다. 이를 수신한 태스크트래커는 지역 필터의 생성 작업 혹은 전역 필터를 공유하기 위해 대기하는 작업을 즉시 중단하고, 기존 맵리듀스의 조인 연산 방식으로 회귀하게 된다.

4.5 양성 오류율의 임계값의 설정

양성 오류율의 임계값은 블룸 필터의 사용여부를 결정하는 기준이 되는 값을 의미한다. 맵리듀스를 수행하기 위한 클러스터의 크기, 그 클러스터를 구성하는 컴퓨터의 사양, 네트워크 속도, 조인할 데이터셋들의 종류 및 크기, 그리고 조인을 수행하는 사용자의 의도 등은 그때마다 다르기 때문에, 양성 오류율의 임계값은 이를 수행하려는 사용자가 정의한다.

다만 우리는 세 번째 실험을 통하여, 양성 오류율의 임계값을 결정하는 최소한의 가이드라인을 제시하였고 이를 통해 일반적으로 블룸 필터를 사용한 조인 연산의 성능을 기대할 수 있다.

제 5 장
성능평가

이 장에서는 우리가 제안한 기법에 대한 실험 결과를 설명한다. 실험에 사용된 클러스터는 1 개의 잡트래커와 10 개의 태스크트래커로 구성되어 있다. 각 노드에 해당하는 컴퓨터는 3.1GHZ 쿼드코어 CPU, 4GB RAM, 2TB 하드디스크, 운영체제 32bit Ubuntu 10.10 의 사양을 가지며, 제안한 기법은 하둡(Hadoop) 0.20.2 버전에서 구현하였다. HDFS 블록 크기는 128MB 로 설정하였고, 각 태스크트래커는 동시에 3 개의 맵 태스크와 3 개의 리듀스 태스크를 할당 받을 수 있도록 설정하였다. 하지만 전체 태스크트래커는 동시에 최대 30 개의 맵 태스크와 28 개의 리듀스 태스크까지 수행할 수 있다. 실험에 사용한 블룸 필터의 크기(m)는 2Mbits (≈ 2048 * 1024)로 설정하였고, 해쉬 함수(k)는 2 개를 사용하였다. 마지막으로 양성 오류율의 임계값(threshold)은 70%로 설정하였다.

5.1 데이터셋과 질의

제안한 기법의 성능평가를 위해 우리는 TPC-H 벤치마크 [16] 100GB 의 데이터셋 중에서 두 개의 테이블 ORDERS 와 LINEITEM 의 조인 연산을 실험 질의(query)로 삼았다. 우리는 [4]의 실험에서 사용한 것과 같이 집계(aggregation) 부분이 제외된 TPC-H Q4 질의를 실험에 사용하였으며 그림 8 은 그 질의를 나타낸다. LINEITEM 테이블의 열(column) orderkey 는 ORDERS 테이블의 열 orderkey 를 참조하는 외래키(foreign key) 관계를 가지기 때문에 orderkey 를 기준으로 조인이 수행된다. 그리고 조인 선택도(join selectivity)를 조절하기 위해 ORDERS 테이블의 열 orderdate 의 기간을 12, 24, 48, 그리고 72 개월로 변경하며 실험을 진행하였다. ORDERS 테이블을 빌드 입력(build input)으로, LINEITEM 테이블을 탐색 입력(probe input)으로 설정하였다. 표 4 는 블룸 필터를 사용한 조인과 기존 맵리듀스 조인을 orderdate 의 기간별로 수행했을 때, 그에 따른 맵 결과와 리듀스 결과 레코드의 개수를 나타내고 있다.

```
SELECT *
FROM lineitem l, orders o
WHERE l.orderkey = o.orderkey
      AND l.commitdate < l.receiptdate
      AND o.orderdate >= 1992-01-01
      AND o.orderdate < 1992-01-01 + '?' months
```

그림 8 집계를 제외한 TPC-H Q4 질의

interval months	Bloom Filter join map output		Hadoop map output	reduce output |ORDERS LINEITEM|
	|ORDERS|	|LINEITEM|	|LINEITEM|	
12	22.8M	151.5M	379M	57.7M

24	45.6M	279.3M	115.2M
48	91.1M	366.3M	230.3M
72	136.7M	378.7M	345.6M

표 4 맵의 중간 결과 및 리듀스의 결과 레코들의 개수

5.2 실험 결과

우선 우리가 제안한 기법의 성능을 평가하기 위해 수행시간을 측정하였다. 실험은 기존 맵리듀스의 조인 연산(Hadoop), 블룸 필터를 사용한 조인 연산(Join w/ BF), 그리고 우리가 제안한 블룸 필터를 사용한 탄력적 조인 연산(Adaptive join w/ BF)의 3 가지 연산 방법을 비교하였으며, 모두 동일한 조건에서 실험을 수행하였다. 이 절에서 우리는 간결한 표현을 위해 기존 맵리듀스의 기본 조인 방법인 재분할 조인을 '하둡 조인'으로, 블룸 필터를 사용한 조인을 '블룸 조인'으로, 그리고 우리가 제안한 블룸 필터를 사용한 탄력적 조인을 '탄력적 조인'으로 표현하겠다.

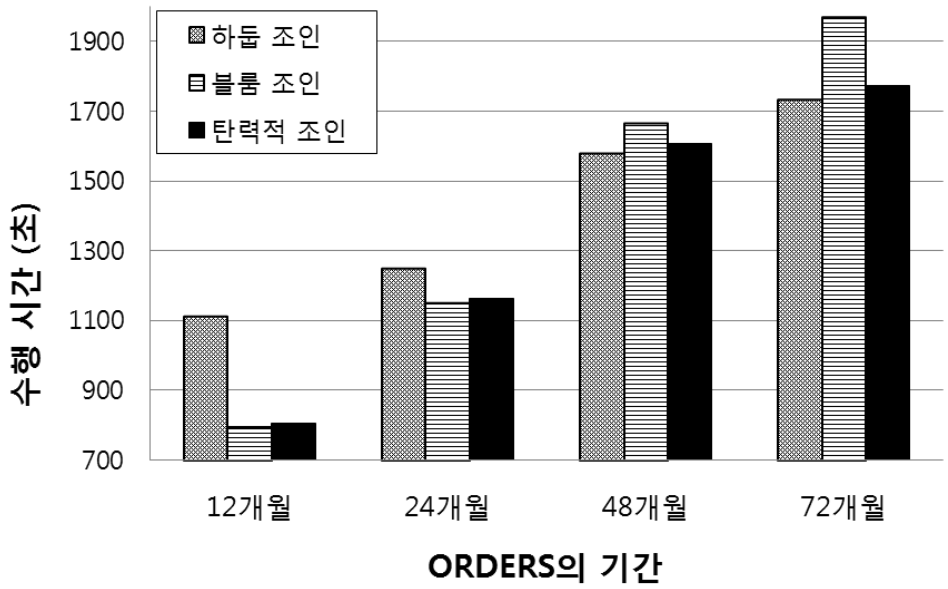
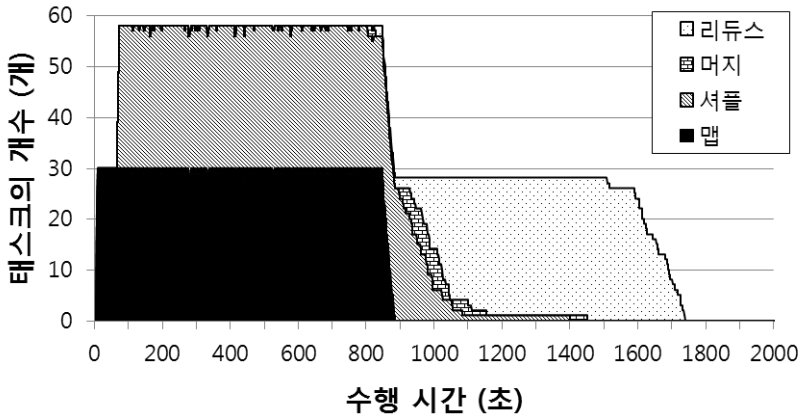


그림 9 조인 연산의 수행시간

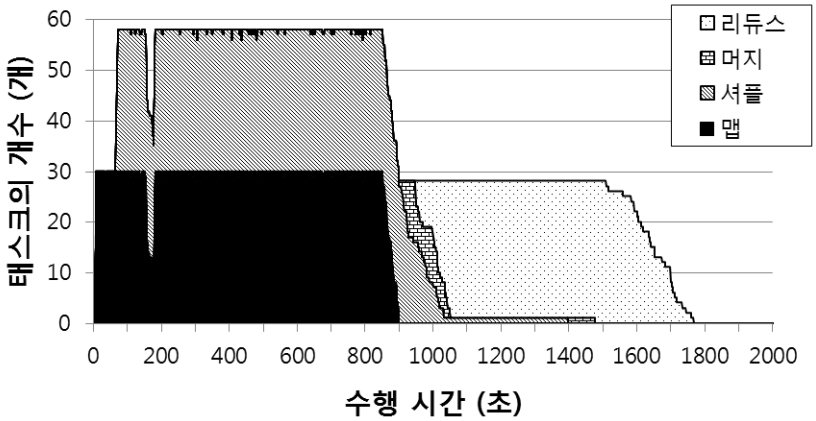
그림 9 는 세 가지 조인 연산에 대한 수행 시간을 나타낸 것이다. 이 결과로부터, 탄력적 조인은 하둡 조인과 블룸 조인 중 더 빠른 수행 시간을 보이는 연산과 유사한 성능을 보임을 알 수 있다. 구체적으로 살펴보면, 탄력적 조인은 블룸 조인이 하둡 조인보다 더 빠른 수행 시간을 보이는 12 개월과 24 개월에서는 블룸 조인과 거의 유사한 성능을 보였고, 하둡 조인이 블룸 조인보다 더 빠른 수행 시간을 보이는 48 개월과 72 개월에서는 하둡 조인과 거의 유사한 성능을 보였다. 이를 통해, 우리가 제안한 탄력적 조인이 블룸 필터의 비효율성으로 인해 생길 수 있는 성능 저하를 막고 효율적인 조인 성능을 보장함을 알 수 있다.

다음 실험으로 블룸 필터가 비효율적이라고 판단되어 필터의 사용을 취소하는 경우, 절감할 수 있는 비용을 확인하기 위해 잡이 수행되는 동안의 각 단계별 태스크 수의 변화를 분석하였다. 이때의 ORDERS 데이터셋은 그림 9 에서 블룸 필터가 가장 비효율적인 성능을 보인 72 개월로 설정하였다. 그림 10 의 (a)와 (b)는 탄력적

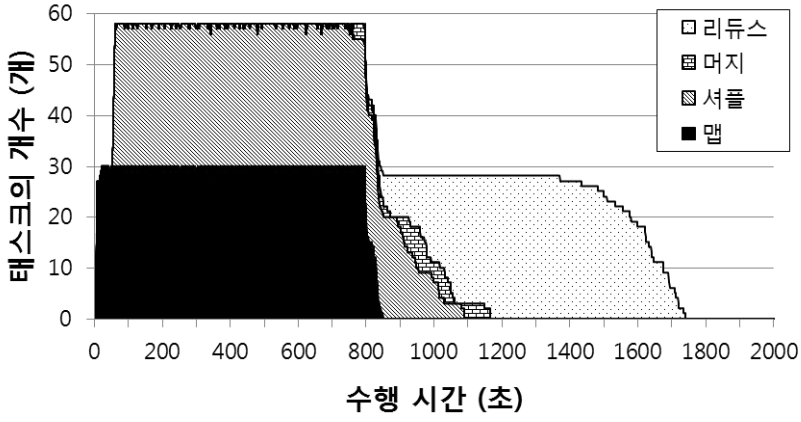
조인에서 블룸 필터의 사용을 취소하는 두 가지 경우에 대한 타임라인을 나타내는데, 각각의 경우에 해당하는 상황을 만들기 위해 양성 오류율의 임계값을 기존의 70%로 설정한 (a)와 달리 (b)는 임계값을 97%로 설정하였다. 그리고 (c)는 하둡 조인의 타임라인을 나타내며, (d)는 블룸 조인의 타임라인을 나타내고 있다.



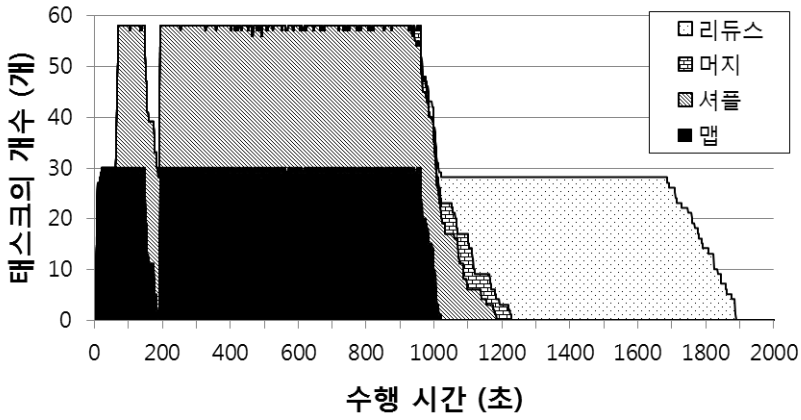
(a) 탄력적 조인 (지역 필터 생성 단계에서 필터 사용 취소)



(b) 탄력적 조인 (전역 필터 합병 단계에서 필터 사용 취소)



(c) 하둡 조인



(d) 블룸 조인

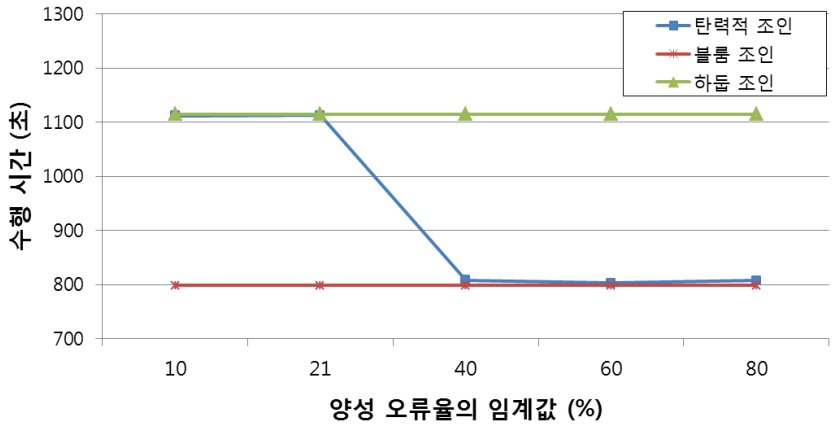
그림 10 조인 연산에 대한 태스크 타임라인

그림 10 의 (a)는 지역 필터 생성 단계에서 블룸 필터의 사용을 취소하는 경우의 타임라인이다. 이 타임라인은 블룸 필터를 사용하지 않는 그림 10 의 (c)인 하둡 조인과 그 모양이 유사함을 알 수 있다. 이는 지역 필터를 생성하는 단계에서 블룸 필터의 사용을 취소하는 경우에는 하둡 조인과 유사한 성능을 보임을 의미한다. 다만 (a)의 맵 수행시간이 (c)에 비하여 조금 긴 것을 볼 수 있는데 이것은 블룸 필터의 사용을 취소하기 전까지는 지역 필터를 생성하기 위한 비용이 발생하기 때문이다.

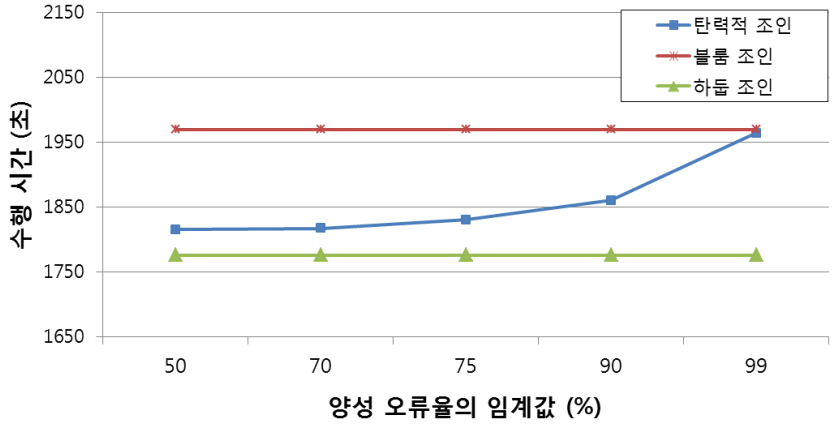
그림 10 의 (b)는 전역 필터를 합병하는 과정에서 블룸 필터의 사용을 취소하는 경우의 타임라인이다. 이 경우의 타임라인은 그림 10 의 (d)인 블룸 조인과 그 모양이 유사하다. 그런데 그림 10 의 (d)인 블룸 조인에서는 약 150 초에서 200 초 사이의 구간에서 맵 태스크의 개수가 0 개까지 줄어들었다가 다시 증가하는 것을 볼 수 있다. 이는 전역 필터를 합병하고 공유하는 작업이 완료될 때까지 태스크트래커들은 다음 맵 태스크들을 할당 받을 수 없으며 따라서 대기하고 있기 때문이다. 이것이 블룸 필터를 사용하기 위해 소비되는 가장 큰 비용인 네트워크 비용이다. 하지만 탄력적 조인은, 전역 필터를 생성하는 중에 필터가 비효율적이라는 것을 파악하여 필터의 사용을 취소할 경우 대기 중이던 태스크트래커들이 즉시 블룸 필터 사용을 취소하고 다음 맵 태스크를 할당받는다. 그림 10 의 (b)에서 맵 태스크의 개수가 어느 정도까지만 낮아지다가 다시 높아지는 것으로부터 (d)에서 발생하는 네트워크 비용이 절감됨을 확인할 수 있다. 그런데 만약 그림 10 의 (a)와 같이 전역 필터를 합병하기도 전에 필터의 사용을 취소하게 되면 이 네트워크 비용 자체를 절감할 수 있으므로, 블룸 필터를 사용하지 않는 (c)의 하둡 조인과 거의 유사한 타임라인 형태를 가지게 된다. 즉 블룸 필터가 비효율적인 경우, 지역 필터를 생성하는 단계에서 그 사용을 취소하는 것이 전역 필터를 합병하는 단계에서 취소하는 것보다 비용을 더욱 절감할 수 있음을 의미한다.

이처럼 그림 10 의 태스크 타임라인 분석을 통해, 우리가 제안한 탄력적 조인이 블룸 필터에 의해 발생되는 추가적인 비용을 절감할 수 있을 뿐 아니라 전역 필터 단계보다는 지역 필터 생성 단계에서 필터의 사용을 취소하는 경우에 절감되는 비용이 더욱 극대화됨을 확인하였다.

마지막으로 블룸 필터의 사용 취소를 결정하는 양성 오류율의 임계값을 결정하기 위한 실험을 진행하였다. 앞서 4.5 절에서 언급한 것처럼, 조인 연산을 수행할 때마다 클러스터의 구성 및 사이즈, 입력 데이터셋의 종류, 그리고 사용자의 의도 등이 달라지기 때문에 임계값은 그것을 수행하려는 사용자가 설정해야 한다. 하지만 그림 11 을 통해 대략적인 가이드라인을 제시할 수 있다.



(a) 12 개월의 ORDERS



(b) 72 개월의 ORDERS

그림 11 임계값의 변화에 따른 수행시간

그림 11 의 (a)는 빌드 입력인 ORDERS 기간을 12 개월로 설정하여 실험을 수행한 것이다. 이 경우, 양성 오류율의 임계값이 약 21%가 되는 지점이 바로 블룸 필터의 사용 여부를 결정하는 기준점이 된다. 이는 빌드 입력으로부터 생성된 전역 필터의 양성 오류율이 21%를 조금 초과함을 의미한다. 이 경우는 양성 오류율이 높지 않으며, 양성 오류율의 임계값을 이보다 높게 설정함으로써 블룸 필터의 성능 효과를 기대할 수 있다. 이에 비해, 기간을 72 개월로 설정하여 실험을 수행한 그림 11 의

(b)에서는 약 99%의 양성 오류율 임계값이 블룸 필터의 사용 여부를 결정하는 기준점이 된다. 하지만 99% 이상의 지나치게 높은 값을 임계값으로 설정한다면, 블룸 필터를 사용하지 않는 것이 유리한 상황임에도 불구하고 필터를 계속 사용함으로써 오히려 조인 연산의 성능이 저하됨을 볼 수 있다. 본 실험을 통해 임계값 설정에 대한 최소한의 지침을 세울 수 있다. 그것은 보수적으로 블룸 필터의 성능 효과를 기대하기 위해 높은 임계값을 설정하되, 지나치게 높은 값은 피함으로써 블룸 필터의 역효과를 방지하는 것이다.

지금까지의 실험 결과를 통해 우리는 제안한 탄력적 조인 알고리즘이 하둡 조인과 블룸 조인 중 성능이 좋은 쪽을 선택하여 최소한의 성능을 보장함을 보였으며 적절한 양성오류율 설정에 대한 최소의 가이드라인을 제시하였다.

제 6 장
결론 및 향후 연구

본 연구에서는 맵리듀스 환경에서 조인 연산의 성능 향상을 위해 블룸 필터를 이용하는 기존 방법을 알아보았고, 효율적이지 못한 상황에서 블룸 필터를 사용함으로 인해 발생하는 역효과를 방지하기 위해 그 사용여부를 탄력적으로 결정하는 조인 기법을 제안하였다. 우리는 수행 시간 중에 블룸 필터의 사용을 취소하는 두 가지 단계를 정의하고 각각의 단계에서 블룸 필터의 양성 오류율을 계산 및 추정하는 방법을 사용하여 필터의 효율성을 검사하였다. 그리고 실험 결과를 통해 우리가 제안한 알고리즘이 기존의 맵리듀스 조인 연산과 블룸 필터를 사용한 조인 연산 중 성능이 좋은 연산 방법을 탄력적으로 선택함으로써 효율적인 성능을 보장함을 보였다.

본 연구에서는 2-웨이 조인 연산을 기준으로 연구가 이루어졌으며 이것을 확장하여 향후 연구에서는 멀티-웨이(multi-way) 조인 연산에 대해서도 적용 가능해야 할 것이다. 또한 데이터셋에 대한 추가적인 정보를 사전에 얻을 수 있다면, 블룸 필터뿐 아니라 그 데이터셋의 특성을 잘 반영할 수 있는 다른 종류의 필터 또한 탄력적으로 적용함으로써 조인 연산의 성능을 더욱 향상시킬 수 있을 것이다.

참고문헌

[1] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Proc. of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp.137-150, 2004.

[2] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon, "Parallel Data Processing with MapReduce: A Survey," in Proc. of the ACM SIGMOD Record, vol.40, no.4, pp.11-20, 2011.

[3] Spyros Blanas, Jignesh M. Patel, Vuk Ercegovac, Jun Rao, Eugene J. Shekita, and Yuanyuan Tian, "A Comparison of Join Algorithms for Log Processing in MapReduce," in Proc. of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10), pp.975-986, 2010.

[4] Taewhi Lee, Kisung Kim, and Hyoung-Joo Kim, "Join Processing Using Bloom Filter in MapReduce," in Proc. of the 2012 ACM Symposium on Research in Applied Computation (RACS '12), pp.100-105, 2012.

[5] Burton H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," Communications of the ACM (CACM), vol.13, no.7, pp.422-426, 1970.

[6] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz, "Theory and Practice of Bloom Filters for Distributed Systems," IEEE Communications Surveys and Tutorials, vol.14, no.1, pp.131-155, 2012.

[7] Jason Venner, "Pro Hadoop," Apress, 1 edition, 2009.

[8] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," IEEE/ACM Transactions on Networking, vol.8, no.3, pp.281-293, 2000.

[9] Ori Rottenstreich and Issac Keslassy, "The Bloom Paradox: When not to use a Bloom filter?," in Proc. of the 2012 IEEE INFOCOM, pp.1638-1646, 2012.

[10] Fan Deng and Davood Rafiei, "Approximately Detecting Duplicates for Streaming Data using Stable Bloom Filters," in Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD '06), pp.25-36, 2006.

[11] MyungKeun Yoon, "Aging Bloom Filter with Two Active Buffers for Dynamic Sets," IEEE Transactions on Knowledge and Data Engineering, vol.22, no.1, pp.134-138, 2010.

[12] Odysseas Papapetrou, Wolf Siberski, and Wolfgang Nejdl, "Cardinality Estimation and Dynamic Length Adaptation for Bloom Filters," Distrib Parallel Databases, vol.28, pp-119-156, 2010.

[13] Loizos Michael, Wolfgang Nejdl, Odysseas Papapetrou, and Wolf Siberski, "Improving Distributed Join Efficiency with Extended Bloom Filter Operations," in Proc. of the 21st International Conference on Advanced Information Networking and Applications (AINA), pp.187-194, 2007.

[14] Lothar F. Mackert and Guy M. Lohman, "R* Optimizer Validation and Performance Evaluation for Distributed Queries," in Proc. of the 12th International Conference on Very Large Data Bases (VLDB), pp.149-159, 1986.

[15] http://hadoop.apache.org/.

[16] http://www.tpc.org/tpch/.

Abstract

Adaptive Join Processing Using Bloom Filter in a MapReduce Environment

Hye-Chan Bae

Department of Computer Science and Engineering

The Graduate School

Seoul National University

MapReduce, a distributed programming model, has been used in many fields to process and analyze large volumes of data. However, MapReduce has a limitation to process join operations in that it transmits all the records, including ones that are not joined, from mappers to reducers. This causes unnecessary network costs and degrades the join performance. To handle this problem, the join technique that filters the redundant records out using Bloom filters was proposed. Nevertheless, in cases that the number of data elements inserted into Bloom filter is too large, the performance of the join processing with Bloom filters can be worse than that without Bloom filters, because of additional costs to use them. This paper proposes an adaptive join processing technique that dynamically determines whether to use of Bloom filters by checking the efficiency of them periodically. For this purpose, we estimate the false positive rate of Bloom filters with the numbers of the elements inserted into them. If it is judged that the filters are inefficient, the join operation is processed without them. The experiments show that the proposed technique ensures

the stable performance of the join processing by choosing the better technique adaptively between the basic MapReduce join and the join using Bloom filter.

Keywords: MapReduce, Bloom filter, False positive rate, Join processing, Adaptive join

Student Number: 2011-20854