맵리듀스 환경에서 블룸 필터를 사용한 적응적 조인 처리

(Adaptive Join Processing Using Bloom Filters in a MapReduce Environment)

배 혜 찬 † 이 태 휘 † 김 형 주 ††
(Hye-Chan Bae) (Taewhi Lee) (Hyoung-Joo Kim)

요 약 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 적응적 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 적응적으로 선택함으로써 안정적인 조인 성능을 보장함을 확인한다.

키워드: 맵리듀스, 블룸 필터, 양성 오류율, 조인 처리, 적응적 조인

Abstract 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 out the redundant records using Bloom filters was proposed. Nevertheless, if 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 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

· 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 20120005695)

† 비 회 원 : 서울대학교 컴퓨터공학부
hcbae@idb.snu.ac.kr
twlee@idb.snu.ac.kr
(Corresponding author임)
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjk@snu.ac.kr

논문접수 : 2013년 1월 2일
심사완료 : 2013년 3월 5일

정보과학회논문지: 데이타베이스 제40권 제4호(2013.8)

1. 서 론

데이터 분산 처리 기술의 발전으로 인하여 대용량 데이터의 처리, 분석 능력이 향상되었다. 이에 따라 기존에는 포기할 수밖에 없었던 다양한 분석을 누적된 방대한 양의 데이터에 적용함으로써 의미 있는 정보를 찾아내 높은 사업 가치를 창출할 수 있게 되었다. 이를 가능하게 하는 대표적인 분산 데이터 처리 기술로 맵리듀스(MapReduce)[1]를 꼽을 수 있다.

맵리듀스란 2004년에 구글에서 발표한 분산 프로그래밍 모델로, 자동 분산 처리를 지원하여 사용자가 복잡한 분산 처리 부분을 고려하지 않고 맵(map)과 리듀스(reduce)의 두 함수(function)로 프로그램을 쉽게 작성할 수 있도록 한다. 또한, 고가의 서버 컴퓨터 대신 저렴한 일반 컴퓨터를 클러스터로 사용 가능하도록 간결한 장애 복구 기능을 제공한다.

맵리듀스는 데이터의 추출, 집계 연산 등 데이터 분석을 위한 여러 연산을 효과적으로 처리할 수 있다. 하지만, 필수 연산 중 하나인 조인 연산을 처리하는 데에는 한계가 있다. 맵리듀스는 주로 단일 데이터셋의 처리를 목적으로 설계되었으나, 간결하고 편리한 분산 처리 능력을 이용하기 위해 그 사용 범위를 여러 데이터셋을 조인하는 데까지 확장했기 때문이다. 가장 큰 단점은 조인에 참여하는 레코드들을 찾기 위하여 조인에 참여하지 않는 레코드들까지 입력 데이터 전체를 맵퍼(Mapper)에서 리듀서(Reducer)로 전송해야 한다는 점이다. 이는 불필요한 네트워크 입출력 비용을 발생시켜 성능 저하를 일으킨다.

맵리듀스에서 조인 연산의 성능을 향상시키기 위해 [2-4] 등 많은 연구들이 활발하게 이루어지고 있는 가운데, [4]에서는 맵리듀스 환경에서 블룸 필터(Bloom Filter)[5]를 이용하여 조인 연산의 성능을 개선시키는 방법이 제안되었다. [4]는 블룸 필터를 이용하여 맵 단계에서 조인에 참여하지 않는 레코드들을 여과하여 리듀서로 전송되는 레코드들의 수를 줄임으로써 조인 연산의 성능을 개선시켰다. 이 기법은 블룸 필터에 추가되는 구분키(distinct key)의 수와 조인에 참여하는 레코드들의 수가 많지 않은 경우 큰 폭의 성능 향상을 보였다.

하지만 블룸 필터를 사용하여 조인 연산을 수행할 때 오히려 성능이 더 나쁘게 나오는 경우도 존재한다. 예를 들어, 블룸 필터에 추가되는 구분키의 수가 너무 많은 경우, 블룸 필터의 비트가 대부분 참(true)으로 설정되어 결국 대부분의 입력 레코드들이 여과되지 않고 리듀서로 전송된다. 따라서 레코드들을 여과하는 이점 없이 블룸 필터를 생성, 전송, 사용하는 과정에서 추가적인 CPU 및 네트워크 비용이 발생하여 블룸 필터를 사용하지 않은 경우보다 성능이 더 저하된다. 맵리듀스에서는 방대한 양의 데이터를 다루고, 로그 데이터 등 정제되지 않은 데이터를 처리하는 경우가 많아 입력 데이터셋에 대한 통계 정보를 미리 알 수 없어 이러한 문제를 가중시킨다[2].

이에 본 연구에서는 블룸 필터의 효율성을 주기적으로 검사하여 수행시간 중에 유연하게 블룸 필터의 사용 여부를 결정하는 적응적(adaptive) 조인 처리 기법을 제안한다. 수행시간 중에 블룸 필터가 더 이상 효율적이지 않다고 판단되면, 즉시 블룸 필터의 사용을 중단하고 그 시점 이후부터는 기존 맵리듀스에서의 방법대로 블룸 필터 없이 조인 연산을 진행한다. 본 연구의 핵심은 효율성의 검사를 위해 블룸 필터의 양성 오류(false positive)를 추정하고 이를 이용하여 필터의 사용여부를 판단하는 것이다. 유연하게 블룸 필터의 사용여부를 결정함으로써 우리는 맵리듀스에서 효율적인 조인 연산의 성능을 항상 보장할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 맵리듀스에서의 조인 연산 처리 기법, 블룸 필터와 그 활용에 관한 관련 연구들을 살펴보고, 본 연구의 목표가 되는 주요 관련 연구인 맵리듀스에서 블룸 필터를 사용한 조인 연산과 그 한계점에 대해 자세히 알아본다. 3장에서는 이를 개선하여 블룸 필터의 효율성에 따라 필터를 적용하는 적응적 조인 연산 처리 기법을 설명하고, 4장에서 실험 결과를 제시하며 우리가 제안하는 기법의 성능을 검증한다. 마지막 5장에서는 결론을 맺으며 논문을 마무리한다.

2. 관련 연구

이 장에서는 먼저 맵리듀스 환경에서 사용 가능한 조인 연산 처리 기법들에 대해 살펴본다. 그 다음, 블룸 필터에 대해 간략히 소개한 후 블룸 필터의 성질, 이를 효율적으로 사용하려고 시도한 관련 연구들을 살펴본다. 마지막으로 본 논문과 밀접한 관련이 있는 맵리듀스에서 블룸 필터를 이용한 조인 연산 처리 기법[4]에 대해 설명한다.

2.1 맵리듀스에서의 조인 연산 처리

대용량 데이터의 처리와 분석에 대한 요구가 늘어남에 따라 맵리듀스에서 조인 연산의 성능을 개선하는 연구가 활발히 이루어지고 있다. 맵리듀스의 조인 연산 처리는 어느 단계에서 조인 연산을 수행하느냐에 따라 크게 맵측 조인(map-side join)과 리듀스측 조인(reduce-side join)의 두 종류로 나눌 수 있다[3].

맵측 조인은 맵 단계에서 조인 연산을 수행하여 그 결과를 출력하므로 맵퍼에서 리듀서로 모든 레코드들을 전송할 필요가 없어 리듀스측 조인보다 효율적이다. 하

지만 맵측 조인은 특정 조건을 만족시키는 특수한 상황에서만 사용이 가능하다[6]. 맵-머지 조인(Map-Merge join)[3]은 조인하는 데이터셋들이 사전에 조인 키에 따라 동일한 조건으로 분할, 정렬되어 있어야만 사용 가능하다. 브로드캐스트 조인(Broadcast join)[2]은 한 데이터셋의 크기가 메모리에 전부 적재될 정도로 작은 경우에만 효율적이다.

리듀스측 조인은 일반적인 상황에서 사용이 가능하지만, 레코드들이 맵퍼에서 리듀서로 전송되어야 하기 때문에 성능이 떨어진다. 재분할 조인(Repartition join)[2]은 맵리듀스에서 가장 보편적으로 사용되는 리듀스측 조인이다. 하지만 재분할 조인은 같은 조인 키를 가지는 레코드들을 찾기 위해 모든 레코드를 리듀서로 전송해야 한다. 재분할 조인은 그림 1과 같이 각 레코드에 소속 데이터셋을 나타내는 고유 태그(tag)를 부여하여 리듀서로 전송하고, 리듀서는 태그를 참조하여 조인 연산을 수행하는 방식으로 이루어진다. 그림 1의 예에서 데이터셋 R과 S로부터 실제로 조인되는 레코드는 R의 <a3, b3>과 S의 <a3, c3>뿐이지만, 이를 찾기 위해 모든 레코드들이 리듀서로 전송됨을 알 수 있다. 세미-조인(Semi-Join)의 맵리듀스 버전[2]이 있으나 조인 연산 수행을 위해 3개의 맵리듀스 잡(job)을 수행하여 레코드를 여러 번 처리해야 한다는 문제점을 지닌다. [4]는 재분할 조인에서 리듀서로 보내지는 불필요한 레코드들을 줄이기 위해 블룸 필터를 사용하여 그에 따른 성능 향상을 보였다. [4]는 주요 관련 연구이므로 이에 대해서는 2.3절에서 상세하게 살펴본다.

2.2 블룸 필터

블룸 필터[5]는 작은 공간으로 데이터셋을 표현하고 어떤 원소 데이터가 그 데이터셋에 속하는지를 빠른 시간 내에 판별해주는, 확률에 기반한 자료 구조이다. 크기가 작고 효율적이어서, 분산 시스템에서 정보 처리량과 네트워크 비용의 감소를 위해 빈번하게 사용되는 기술 중 하나이다[7]. 블룸 필터는 m개의 비트 공간을 가지며, k개의 독립적인 해쉬 함수를 사용하여 원소 데이터를 삽입하거나 검색한다. 원소 데이터를 삽입할 때에는 k개의 해쉬 함수에 적용한 결과값들에 대응되는 필터의 위치 비트를 참(true)으로 설정한다. 어떤 원소 데이터가 데이터셋에 속해 있는지 검색할 때에는 같은 방법으로 k개의 해쉬 함수를 적용해 그 결과값들에 대응되는 필터의 위치 비트 값을 검사한다. 해당되는 모든 비트 값이 참이면, 그 원소 데이터는 데이터셋에 속한다고 판단하게 된다.

블룸 필터는 해쉬 함수를 사용하여 원소 데이터를 추가하기 때문에 양성 오류(false positive)가 발생할 수 있다. 양성 오류란 실제로는 데이터셋에 속하지 않은 원소 데이터를 블룸 필터를 이용해 판단한 결과, 데이터셋에 속한다고 잘못 판단되는 경우를 말한다. n개의 원소 데이터를 블룸 필터에 추가한 후의 양성 오류율(false positive rate)은 다음과 같다[5].

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

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[8]. 원소 데이터의 개수를 사전에 파악할 수 없는 경우, 블룸 필터의 크기가 충분치 않으면 오히려 성능 저하를 야기할 수 있다. 이러한 경우의 비효율성을 개선하기 위한 연구들 또한 활발히 이루어지고 있다. [9]에서는 높은 확률로 양성 오류가 발견되는 상황을 블룸 역설(Bloom paradox)이라고 정의하고, 데이터의 선험적 확률(a priori probability)을 통해 역설을 일으킬 가능성이 높은 원소 데이터는 무시하고 블룸 필터를 만드는 선택적 블룸 필터

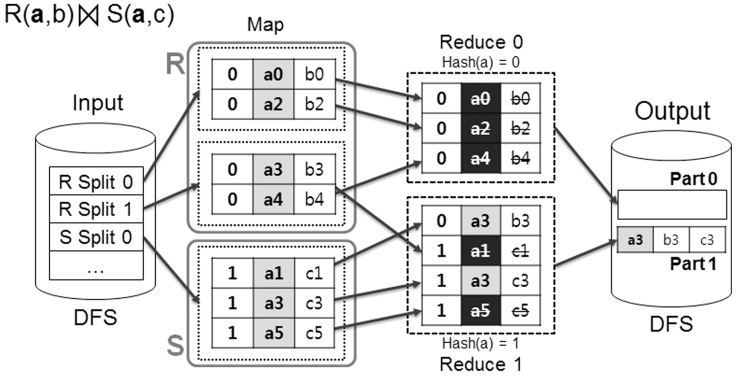


그림 1 재분할 조인

Fig. 1 Repartition join

(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. 이러한 환경에서는 데이터가 무한하게 입력될 수 있기 때문에 시간이 지날수록 양성 오류율이 1에 가까워진다. 이를 방지하기 위해 버퍼 교체 정책(buffering replacement policy)과 같이 블룸 필터에 삽입된 지 오래된 원소 데이터를 삭제하는 방법으로 양성 오류율을 특정 수준으로 유지시켰다. 하지만, 이들 연구에서는 원소 데이터를 삭제하기 때문에 음성 오류(false negative)가 발생할 수 있어, 조인 연산에 적용 시 잘못된 결과를 출력하게 되므로 사용이 불가능하다.

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, 그 충분치 않은 정보를 추정하는 다양한 방법들이 연구되었다. [12]는 블룸 필터에 추가된 구분 원소(distinct element)의 개수를 모를 때 블룸 필터에서 참으로 설정된 비트 수를 이용해 추정하는 방법을 제안하였다. [13]은 2개의 블룸 필터를 합집합(union)할 경우의 양성 오류율을 추정하는 방법을 제안하였다. 본 연구에서는 블룸 필터의 이러한 수학적 특성들을 이용하여 필터의 효율성을 검사하는 데 활용한다.

2.3 맵리듀스에서 블룸 필터를 사용한 조인 연산 처리

이 절에서는 본 연구의 주요 관련 연구인 맵리듀스에서 블룸 필터를 이용한 조인 연산[4]에 대해 설명한다. 이 연구는 맵리듀스의 오픈 소스 프레임워크인 하둡(Hadoop)[14]을 수정하여, 단일 맵리듀스 잡에서 블룸 필터를 통해 재분할 조인에서 리듀서로 보내지는 불필요한 중간 레코드들을 감소시켜 조인 성능을 개선하였다.

이 기법의 핵심은 두 데이터셋에 대해 조인 연산을 수행할 때, 처리 순서를 결정하여 첫 번째 데이터셋으로부터 블룸 필터를 동적으로 생성하고, 그 필터를 사용해 두 번째 데이터셋의 불필요한 레코드들, 즉, 조인에 참여하지 않는 레코드들을 여과하는 것이다. 이 수행작업은 전부 맵 단계에서 이루어지므로, 중간 레코드들을 리듀스로 전송하는 네트워크 비용 및 리듀스 처리 시간을 감소시켜 전체 조인 연산의 수행시간을 단축시킨다. 하둡은 데이터셋의 순서 개념이 없기 때문에 기본적으로 입력 데이터셋들을 여러 개의 스플릿(split)들로 나누어 스플릿의 크기에 따라서 태스크를 할당하는데, 이러한 상황에서는 블룸 필터를 생성하여 활용할 수 없다. 이를 해결하기 위해 이 연구에서는 데이터셋의 처리 순서를 결정하여, 그 순서대로 처리가 되도록 구현하였다.

그림 2는 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 여과되는 데이터셋인 탐색 입력(probe input)으로 정의하였다. 조인 연산의 흐름은 다음의 6단계를 거친다.

1. **잡 제출(Job submission).** 잡이 제출되면 R을 위한

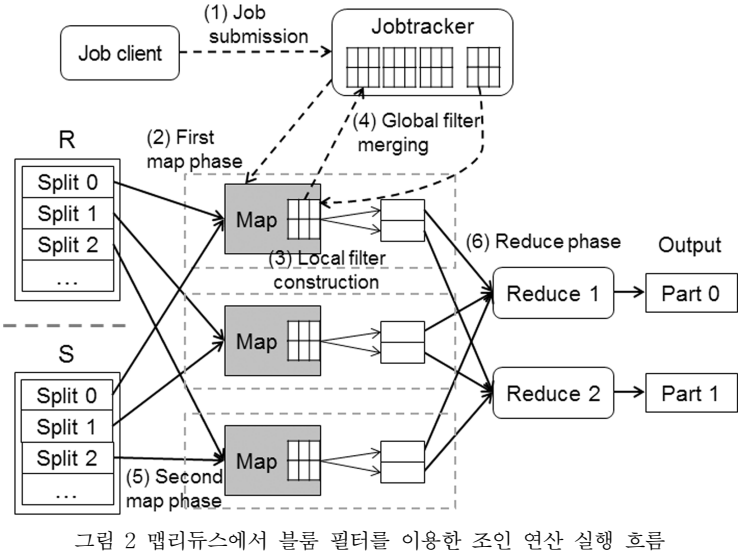


그림 2 맵리듀스에서 블룸 필터를 이용한 조인 연산 실행 흐름

Fig. 2 Execution overview of join processing using Bloom filters in MapReduce)[4]

m_1개의 맵 태스크(map task), S를 위한 m_2개의 맵 태스크 그리고 r개의 리듀스 태스크(reduce task)가 생성된다.

2. **첫 번째 맵 단계(First map phase).** 잡트래커는 m_1개의 맵 태스크 또는 리듀스 태스크를 태스크트래커들에게 할당한다. 각 맵 태스크트래커는 입력 스플릿을 읽어 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 신호(heartbeat signal)를 통해 모든 태스크트래커의 지역 필터들을 취합한다. 이를 병합하여 빌드 입력 R에 대한 전역 필터(global filter)를 생성한다. 잡트래커는 다시 모든 태스크트래커에게 완성된 전역 필터를 전송하여 그것을 공유한다. 전역 필터의 생성과 공유가 완료될 때까지, 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).** 이 단계에서는 실질적인 조인 연산이 이루어지며, 재분할 조인의 경우와 동일하다.

하지만 이 연구는 블룸 필터를 사용한 경우 오히려 성능이 저하되는 경우도 존재한다. 성능 저하가 발생하는 상황을 살펴보면, 첫 번째는 데이터셋의 대부분 레코드들이 조인에 참여하는 경우이다. 이 경우, 조인되지 않는 레코드의 수가 적기 때문에 여과할 대상이 되는 레코드 수가 애초에 적어 오히려 성능이 저하된다. 두 번째는 블룸 필터의 양성 오류율(false positive rate)이 높아지는 경우이다. 2.2절에서 언급한 것처럼, 블룸 필터의 크기가 적고 필터에 추가되는 원소 데이터의 개수가 많을수록 양성 오류율이 증가하고 조인되지 않는 레코드들이 여과되지 않아 성능이 저하된다. 맵리듀스의 특성 상 입력 데이터셋에 대한 통계 정보를 미리 알 수 없는 경우, 블룸 필터의 크기를 항상 알맞게 조정할 수 없어 문제가 발생한다.

이렇게 블룸 필터로 인한 성능 개선을 기대할 수 없는 경우, 지역 필터를 생성, 탐색하는 CPU 비용과 전역 필터를 합병, 공유하는 네트워크 비용이 추가로 발생하여 오히려 성능을 악화시킨다. 이에 본 연구에서는 블룸 필터의 효율성을 주기적으로 검사하여 효율적이라고 판단될 때에만 블룸 필터를 사용하는 적응적 조인 방법을 제안한다.

3. 블룸 필터를 사용한 적응적 조인

우리는 블룸 필터를 생성하는 과정에서 필터의 양성 오류율(false positive rate)을 주기적으로 측정하고 이것을 기준으로 필터를 적응적으로 사용한다. 우리는 크게 지역 필터를 생성하는 단계와 전역 필터를 병합하는 단계의 두 단계에서 블룸 필터의 효율성을 검사한다. 각 단계에서 추정 혹은 계산된 블룸 필터의 양성 오류율이 정해진 임계값(threshold)을 넘어 그 필터가 비효율적이라고 판단되면, 잡트래커는 모든 태스크트래커들에게 즉시 블룸 필터의 사용을 취소하도록 알린다.

블룸 필터는 각 리듀서에 전송되는 중간 결과의 파티션 별로 하나씩 생성되어, 지역 필터와 전역 필터는 리듀서의 개수만큼의 블룸 필터를 가진다. 블룸 필터의 효율성 검사는 각 파티션의 블룸 필터마다 개별적으로 수행되며, 전체 블룸 필터가 아닌 일부 파티션의 블룸 필터만을 비효율적이라고 판단할 수 있다. 이러한 경우, 전체 조인 연산에서는 블룸 필터를 사용하되, 비효율적이라고 판단된 일부 블룸 필터는 제외하고 사용하게 된다. 대부분의 블룸 필터가 비효율적이라고 판단되는 경우는 전체 블룸 필터의 사용 자체를 취소함으로써, 기존의 맵리듀스 조인 연산 방식으로 회귀한다.

3.1 지역 필터 생성 단계에서의 효율성 검사

본 단계에서 각 태스크트래커는 빌드 입력(build input)의 스플릿(split)으로부터 지역 필터를 생성한다. 정확한 블룸 필터의 효율성은 지역 필터들을 병합하여 전역 필터를 생성한 후에야 확인 가능하다. 하지만 지역 필터를 생성하는데 사용되는 빌드 입력의 크기가 매우 크거나 구분 키의 개수가 많은 경우, 전역 필터를 생성하기 이전에 미리 필터의 비효율성을 파악함으로써 조인의 성능을 향상할 수 있다. 우리는 지역 필터를 생성하는 동안 주기적으로 모든 지역 필터에 대한 정보를 취합하여 전역 필터의 양성 오류율을 추정한다.

전역 필터의 양성 오류율을 추정하는 방법으로는 전역 필터를 생성하는 과정과 동일하게 각 태스크트래커가 자신의 지역 필터를 잡트래커에게 전송한 후, 잡트래커가 이를 병합하여 양성 오류율을 계산하는 방법이 있다. 하지만, 모든 태스크트래커들의 지역 필터 전체를 주기적으로 계속 보내는 것은 네트워크 상의 병목 현상을 야기할 수 있다. 따라서 우리는 지역 필터 전체를 보

내는 대신, 지역 필터에 삽입한 원소 데이터의 개수 정보만을 잡트래커로 보내는 방법을 사용하여 네트워크 부하를 줄인다. 원소 데이터의 개수 정보는 전역 필터의 양성 오류율을 추정하기 위한 최소한의 정보이다. 잡트래커와 태스크트래커 간의 정보 전달은 하트비트(heartbeat) 신호에 정보를 함께 실어 나르는 피기백(piggyback) 방식을 이용한다.

[13]의 연구는 블룸 필터에 삽입된 각 원소 개수를 이용하여 블룸 필터들의 합집합에 대한 양성 오류율을 추정하는 공식을 제시한다. 우리는 이를 활용하여 전역 필터의 양성 오류율을 추정한다. 지역 필터 내의 한 블룸 필터, 가령 r 번째 블룸 필터의 비트 수가 m 이고 삽입된 원소의 개수가 n_r 이라면 이 필터의 어떤 비트가 참(true)일 확률 $P_{bltSetTo1}(r)$ 은 다음과 같이 나타낼 수 있으며, 이는 2.2절의 양성 오류율 공식에서 해쉬 함수의 개수에 따른 k 제곱을 제외한 것과 같다.

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

이를 토대로 두 개의 블룸 필터, 가령 $r-1$ 번째와 r 번째 블룸 필터를 병합한 필터에서 어떤 비트가 참일 확률을 추정할 수 있다. 병합한 필터의 양성 오류율을 정확히 알기 위해서는 두 블룸 필터에 삽입한 구분 원소들 중 중복되는 원소들의 개수를 알아야 하는데, 우리는 이를 알지 못한다. 그러나 [13]과 같이 각 원소들의 분포가 독립이라고 가정하면, 두 집합이 독립일 경우에 합집합의 확률을 $P(A \cup B) = P(A) + P(B) - P(A) \cdot P(B)$ 과 같이 구할 수 있다. 이에 따라, 지역 필터 내의 r 번째 블룸 필터까지 병합한 필터에서 어떤 비트가 참일 확률 $P'_{error}(r)$ 은 다음과 같이 구할 수 있다.

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

따라서 전체 지역 필터들에 대한 양성 오류율 P_{error} 는 이를 해쉬 함수의 개수 k 만큼 곱하여 구할 수 있다.

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

이렇게 지역 필터를 생성하는 단계에서 블룸 필터가 비효율적이라고 판단되어 필터의 사용을 취소하면 블룸 필터를 계속 이용하였을 때와 비교하여 다음의 비용들을 절감할 수 있다. 먼저 태스크트래커는 아직 처리를 완료하지 못한 빌드 입력에 대해 더 이상 지역 필터를 만들 필요가 없어 CPU 비용을 절감할 수 있다. 또한 전역 필터를 생성할 필요가 없어 그에 따른 CPU 및 네트워크 비용을 절감할 수 있다. 마지막으로 다음 데이터셋 즉, 탐색 입력(probe input)으로부터 조인되는 레코드를 찾기 위해 필터의 탐색을 수행하는데 발생하는 CPU 비용을 절감할 수 있다.

3.2 전역 필터 병합 단계에서의 효율성 검사

태스크트래커들이 빌드 입력에 대한 모든 맵 태스크를 할당 받아 처리를 완료하면, 각 태스크트래커는 자신이 처리한 스플릿들에 대해 완성된 지역 필터를 가진다. 이 지역 필터들을 잡트래커가 취합하고 병합함으로써 전역 필터를 생성하게 된다. 여기서 하나의 지역 필터가 전역 필터로 병합될 때마다 전역 필터의 양성 오류율을 계산하여 필터의 사용 여부를 결정할 수 있다.

전역 필터 병합 단계에서는 실제 지역 필터를 병합한 후의 정확한 비트 정보를 알 수 있다. 그러나 양성 오류율 계산에 필요한 정보인 병합한 필터에 삽입되어 있는 원소의 개수는 역시 알 수 없는 상황이다. 이 문제를 해결하기 위해 우리는 [12]의 방법을 사용하여 원소 개수를 추정한다. 블룸 필터의 비트 수가 m 이고 해쉬 함수의 개수가 k일 때, n개의 구분 원소를 삽입한 후에 참인 비트의 기대값 $\hat{S}(n)$ 은 다음과 같이 구할 수 있다.

$$\hat{S}(n) = m \times \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)$$

이 수식을 참인 비트의 수를 $\hat{S}(n)$ 대신 t 로 대입하여 변형하면, 참인 비트가 t 일 때 블룸 필터에 삽입한 구분 원소 n 의 기대값 $\hat{S}^{-1}(t)$ 을 다음과 같이 구할 수 있다.

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

우리는 이를 통해 병합한 전역 필터에 삽입된 구분 원소의 개수를 추정한 후, 2.2절의 양성 오류율 계산 공식을 적용하여 전역 필터의 양성 오류율을 계산한다.

본 단계는 전역 필터를 병합하는 단계이므로 태스크트래커가 자신의 지역 필터를 완성시키고 잡트래커로 전송하는 과정에서 발생하는 CPU 및 네트워크 비용은 절감할 수 없다. 하지만 전역 필터를 모든 태스크트래커들에게 공유하는 데에 따른 네트워크 비용과 다음 데이터셋인 탐색 입력으로부터 필터를 탐색하는 과정에서 발생하는 CPU 비용은 여전히 절감할 수 있다.

3.3 블룸 필터 사용의 취소

3.1절과 3.2절에서 설명한 바와 같이, 본 연구는 지역 필터를 생성하는 단계와 전역 필터를 병합하는 두 가지 단계에서 각각 블룸 필터의 양성 오류율을 계산 또는 추정하여 그 효율성을 검사한다. 계산한 양성 오류율이 미리 정의된 임계값을 넘는 경우, 그 필터는 비효율적이라고 판단된다. 표 1에서는 블룸 필터의 사용이 비효율적이라고 판단하여 그 사용을 취소하는 경우, 각 단계별로 절감 가능한 비용들을 정리하였다. 블룸 필터가 비효율적일 때, 지역 필터 생성 단계에서 가급적 일찍 블룸 필터의 사용을 취소하는 편이 낭비되는 비용을 더 절감할 수 있다.

이러한 블룸 필터의 효율성 검사와 취소 동작을 위해

표 1 각 케이스 별로 절감 가능한 비용들
Table 1 Cost to reduce in each case

Cost to reduce	Cost reduction	
	Local filter construction phase	Global filter merging phase
Cost of building local filters	○	×
Cost of merging and distributing global filters	○	△
Cost of probing filters during second map phase	○	○

서는, 필요한 정보를 잡트래커와 태스크트래커 간에 주고 받아야 한다. Hadoop에는 TaskTrackerAction 클래스를 통해 하트비트 신호를 통해 태스크트래커에 명령을 내리며, 우리는 두 가지 TaskTrackerAction 클래스인 ReceiveFilterStatusAction 클래스와 WithdrawFilterAction 클래스를 추가하여 이를 처리하였다. 잡트래커가 일부 블룸 필터를 비효율적이라고 판단한 경우, 모든 태스크트래커에게 전체 필터의 효율성 정보를 포함하는 ReceiveFilterStatusAction을 보낸다. 이를 수신한 태스크트래커는 비효율적이라고 판단된 특정 필터들을 파악함으로써 남은 태스크 수행에 해당 필터들을 사용하지 않는다. 잡트래커가 대부분의 블룸 필터를 비효율적이라고 판단한 경우, 모든 태스크트래커들에게 블룸 필터 사용의 취소를 알리는 WithdrawFilterAction을 보낸다. 이를 수신한 태스크트래커는 지역 필터의 생성 작업 혹은 전역 필터를 공유할 때까지 대기하는 작업을 즉시 중단하고, 기존 맵리듀스의 조인 연산 방식으로 회귀한다.

4. 성능 평가

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

4.1 데이터셋과 질의

제안한 기법의 성능평가를 위해 우리는 TPC-H 벤치마크 [15] 100GB의 데이터셋 중에서 두 개의 테이블 ORDERS와 LINEITEM의 조인 연산을 실험 질의(query)로 삼았다. 우리는 [4]의 실험에서 사용한 것과 같이 집계(aggregation) 부분이 제외된 TPC-H Q4질의를 실험에 사용하였으며 그림 3은 그 질의를 나타낸다. LINEITEM 테이블의 열(column) orderkey는 ORDERS 테이블의 열 orderkey를 참조하는 외래키(foreign key) 관계를 가지기 때문에 orderkey를 기준으로 조인이 수행된다. 그리고 조인 선택도(join selectivity)를 조절하기 위해 ORDERS 테이블의 열 orderdate의 기간을 12, 24, 48, 그리고 72개월로 변경하며 실험을 진행하였다. ORDERS 테이블을 빌드 입력(build input)으로, LINEITEM 테이블을 탐색 입력(probe input)으로 설정하였다. 표 2는 블룸 필터를 사용한 조인과 기존 맵리듀스 조인을 orderdate의 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 3 실험 질의
Fig. 3 Test query

4.2 실험 결과

우선 우리가 제안한 기법의 성능을 평가하기 위해 수

표 2 중간 결과 및 최종 결과의 레코드 개수
Table 2 The number of intermediate/output records

Interval months	Map output in join using Bloom filters		Map output in repartition join	Reduce output
	| ORDERS |	| LINEITEM |	| LINEITEM |	| ORDERS ▷◁ 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

행시간을 측정하였다. 실험은 기존 맵리듀스의 조인 연산, 블룸 필터를 사용한 조인 연산, 그리고 우리가 제안한 블룸 필터를 사용한 적응적 조인 연산의 3가지 연산 방법을 비교하였으며, 모두 동일한 조건에서 실험을 수행하였다. 이 절에서 우리는 간결한 표현을 위해 기존 맵리듀스의 기본 조인 방법인 재분할 조인을 '재분할 조인(repartition join)'으로, 블룸 필터를 사용한 조인을 '블룸 조인(Bloom join)'으로, 그리고 우리가 제안한 블룸 필터를 사용한 적응적 조인을 '적응적 조인(adaptive join)'으로 표현하겠다.

그림 4는 세 가지 조인 연산에 대한 수행 시간을 나타낸 것이다. 이 결과로부터, 적응적 조인은 재분할 조인과 블룸 조인 중 더 빠른 수행 시간을 보이는 연산과 유사한 성능을 보임을 알 수 있다. 구체적으로 살펴보면, 적응적 조인은 블룸 조인이 재분할 조인보다 더 빠른 수행 시간을 보이는 12개월과 24개월에서는 블룸 조인과 거의 유사한 성능을 보였고, 재분할 조인이 블룸 조인보다 더 빠른 수행 시간을 보이는 48개월과 72개월에서는 재분할 조인과 거의 유사한 성능을 보였다. 이를 통해, 우리가 제안한 적응적 조인이 블룸 필터의 비효율성으로 인해 생길 수 있는 성능 저하를 막고 효율적인 조인 성능을 보장함을 알 수 있다.

다음 실험으로 블룸 필터가 비효율적이라고 판단되어

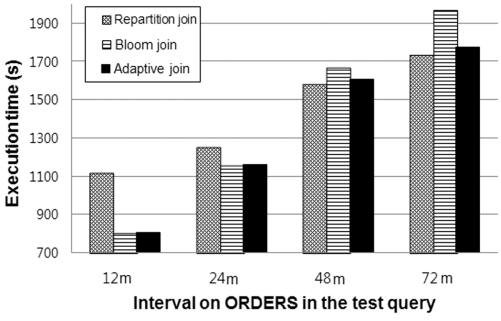
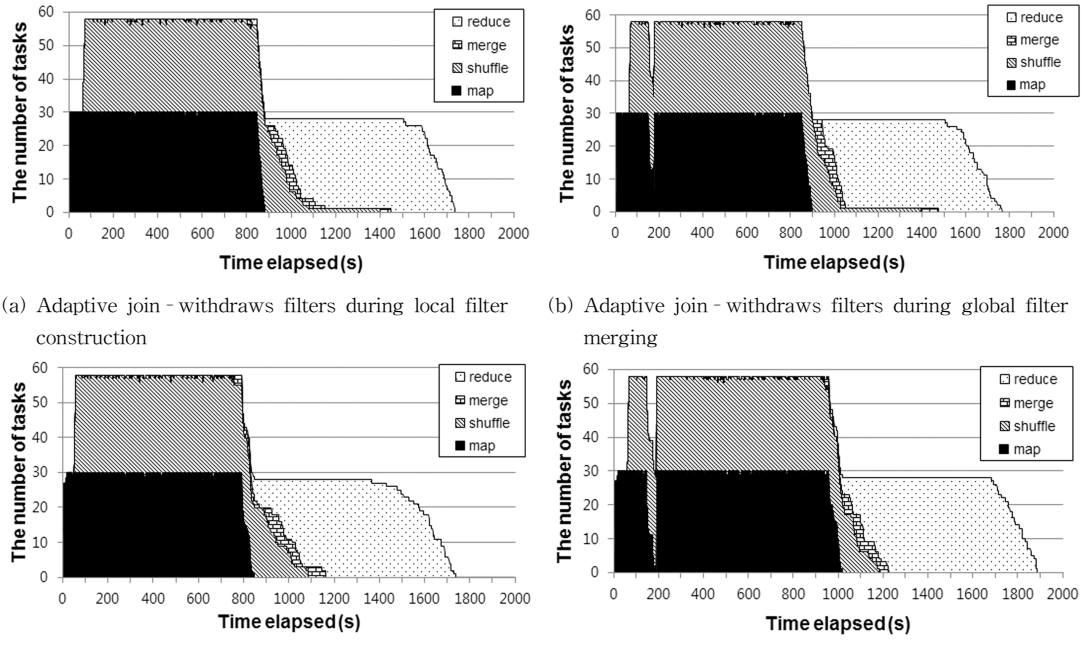


그림 4 조인 성능 - 수행 시간

Fig. 4 Join performance - execution time

필터의 사용을 취소하는 경우, 절감할 수 있는 비용을 확인하기 위해 잡이 수행되는 동안의 각 단계별 태스크 수의 변화를 분석하였다. 이때의 ORDERS 데이터셋은 그림 4에서 블룸 필터가 가장 비효율적인 성능을 보인 72개월로 설정하였다. 그림 5(a)와 (b)는 적응적 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 재분할 조인의 타임라인을 나타내며, (d)는



(a) Adaptive join - withdraws filters during local filter construction

(b) Adaptive join - withdraws filters during global filter merging

(c) Repartition join

(d) Bloom join

그림 5 조인 처리 시 태스크 타임라인

Fig. 5 Task timelines during join processing

블룸 조인의 타임라인을 나타내고 있다.

그림 5(a)는 지역 필터 생성 단계에서 블룸 필터의 사용을 취소하는 경우의 타임라인이다. 이 타임라인은 블룸 필터를 사용하지 않는 그림 5(c)인 재분할 조인과 그 모양이 유사함을 알 수 있다. 이는 지역 필터를 생성하는 단계에서 블룸 필터의 사용을 취소하는 경우에는 재분할 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 5(b)는 전역 필터를 합병하는 과정에서 블룸 필터의 사용을 취소하는 경우의 타임라인이다. 이 경우의 타임라인은 그림 5(d)인 블룸 조인과 그 모양이 유사하다. 그런데 그림 5(d)인 블룸 조인에서는 약 150초에서 200초 사이의 구간에서 맵 태스크의 개수가 0개까지 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. 반면, 적응적 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 5(b)에서 맵 태스크의 개수가 어느 정도까지만 낮아지다가 다시 높아지는 것으로부터 (d)에서 발생하는 네트워크 비용이 절감됨을 확인할 수 있다.

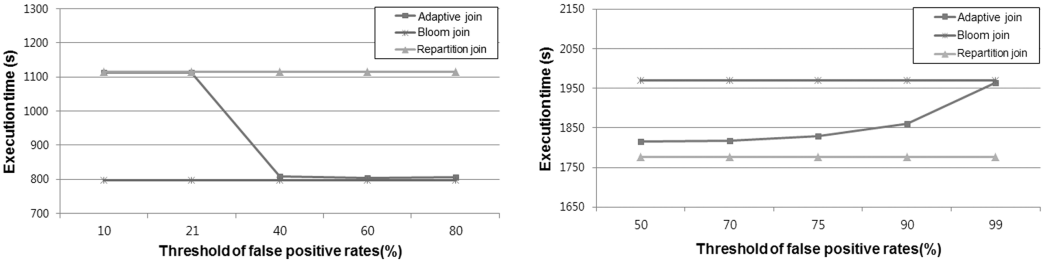
마지막으로 블룸 필터의 사용여부를 결정하는 양성 오류율의 임계값을 변경해 가며 적응적 조인의 실험을 진행하였다. 그림 6(a)는 빌드 입력인 ORDERS기간을 12개월로 설정하여 실험을 수행한 것이다. 이 경우, 양성 오류율의 임계값이 약 21%가 되는 지점이 바로 블룸 필터의 사용 여부를 결정하는 기준점이 된다. 이는 빌드 입력으로부터 생성된 전역 필터의 양성 오류율이 21%를 조금 초과함을 의미한다. 이 경우는 양성 오류율이 높지 않으며, 양성 오류율의 임계값을 이보다 높게 설정함으로써 블룸 필터의 성능 효과를 기대할 수 있다. 이에 비해, 기간을 72개월로 설정하여 실험을 수행한 그림 6(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.

클러스터의 구성, 입력 데이터셋의 크기와 분포 등 조인 연산을 수행하는 환경에 따라 필터의 성능이 달라지기 때문에 임계값은 연산을 수행하는 사용자가 설정한다. 하지만 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.

5. 결론 및 향후 연구

본 연구에서는 맵리듀스 환경에서 조인 연산의 성능 향상을 위해 블룸 필터를 사용하는 기존 연구를 알아보았고, 블룸 필터가 효율적이지 못한 상황에서 블룸 필터를 사용함으로 인해 발생하는 역효과를 방지하기 위해 그 사용여부를 적응적으로 결정하는 적응적 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 방법이 기존의 맵리듀스 조인 연산과 블룸 필터를 사용한 조인 연산 중 성능이 좋은 연산 방법을 적응적으로 선택함으로써 효율적인 성능을 보장함을 보였다.

본 연구는 데이터셋에 대한 통계적인 정보가 없는 상황을 가정하였는데, 그러한 통계 정보가 존재하는 상황



(a) ORDERS 12m (b) ORDERS 72m

그림 6 임계값의 변화에 따른 수행시간

Fig. 6 Execution time varying threshold of false positive rates

이라면 블룸 필터의 크기나 양성 오류율의 임계값 등을 자동으로 최적의 값을 계산하여 설정할 수 있을 것이다. 또한 블룸 필터뿐 아니라 데이터셋의 특성을 잘 반영할 수 있는 다른 종류의 필터를 적용함과 동시에 그것을 적응적으로 적용한다면 조인 연산의 성능을 더욱 향상시킬 수 있을 것이다.

References

[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] 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.

[3] 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.

[4] Taewhi Lee, Kisung Kim, and Hyoung-Joo Kim, "Join Processing Using Bloom Filter in MapReduce," in *Proc. of the 2012 ACM Research in Applied Computation Symposium (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] Jason Venner, "Pro Hadoop," Apress, 1st edition, 2009.

[7] 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.

[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," *Distributed and Parallel Databases*, vol.28, no.2-3, 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] http://hadoop.apache.org/.

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

배 혜 찬

2011년 2월 아주대학교 정보및컴퓨터공학부 학사. 2013년 2월 서울대학교 컴퓨터공학부 석사. 2013년 1월~현재 삼성전자 소프트웨어센터 사원. 관심분야는 데이터베이스, 빅데이터 처리, 시맨틱 웹

이 태 휘

2004년 2월 서울대학교 컴퓨터공학부 학사. 2004년 3월~현재 서울대학교 컴퓨터공학부 석박사 통합과정 재학중. 2007년 6월~2010년 4월 티맥스소프트 선임연구원. 관심분야는 텍스트/그래프 데이터 검색, 대규모 데이터 처리, 시맨틱 웹

김 형 주

정보과학회논문지 : 데이타베이스
제 40 권 제 1 호 참조