creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

아웃소싱 데이터베이스를 위한 효율적인 집계 질의 인증 기법

Efficient Authentication of Aggregation Queries for Outsourced Databases

2016년 12월

서울대학교 대학원

전기 · 정보공학부

신 종 민

아웃소싱 데이터베이스를 위한 효율적인 집계 질의 인증 기법

지도 교수 심 규 석

이 논문을 공학석사 학위논문으로 제출함
2016년 12월

서울대학교 대학원
전기 · 정보공학부
신 종 민

신종민의 공학석사 학위논문을 인준함
2016년 12월

위 원 장 _____ 김태환 _____ (인)

부위원장 _____ 심규석 _____ (인)

위 원 _____ 홍성수 _____ (인)

초 록

데이터베이스 아웃소싱은 기업 및 기관에서 빅데이터 분석을 위해 사용하는 보편적인 방법으로 자리잡고 있다. 데이터 소유자는 데이터베이스 아웃소싱을 통해 질의 처리와 같은 계산량이 많은 작업과 저장 및 관리 등의 업무를 제 3자 서버에 위탁할 수 있다. 이러한 패러다임은 데이터 소유자가 비싼 인프라를 구축하지 않고도 대용량의 데이터를 관리할 수 있게 한다. 하지만 보안상 위협이 항상 존재하는 네트워크의 특성상 제 3자 서버는 데이터를 변조하고 질의 결과를 조작하는 공격자들의 공격에 노출되어있다. 따라서 서버는 사용자에게 질의 결과가 정확하다고 증명할 수 있어야 한다. 이러한 문제를 해결하기 위해 구간 질의, kNN 질의, 함수 질의 등 다양한 질의에 대한 인증 기법들이 연구되었다. 하지만 빅데이터 분석에 있어 집계 질의의 활용도가 높음에도 불구하고 집계 질의에 대한 인증 기법 연구는 부족하며 기존 연구는 고차원 데이터나 서로 다른 값의 수가 많은 데이터에 대해 비효율적이다.

본 연구에서는 집계 질의 인증을 위한 자료구조 AMR-tree를 제안한다. 그리고 독립 노드와 독립 서브트리 개념을 도입하고 AMR-tree를 활용한 효율적인 증거 생성 방법과 증명 방법을 제안한다. 그 후 제안한 기법의 성능을 검증하기 위해 데이터의 상이 값 수, 레코드 개수, 그리고 차원 크기를 각각 변경시켜가며 실험을 진행하였으며 실험 결과를 통해 제안한 기법의 성능이 우수함을 보였다.

주요어 : 아웃소싱 데이터베이스, 질의 인증, 집계 질의, 인증 자료구조
학　번 : 2015-20939

목 차

초록 i

목차 ii

제 1 장 서론 1
제 1 절 연구의 배경 1
제 2 절 연구의 내용 2

제 2 장 관련 연구 5
제 1 절 1차원 구간 질의 인증 기법 5
제 2 절 다차원 구간 질의 인증 기법 7
제 3 절 다차원 구간 집계 질의 인증 기법 10

제 3 장 집계 질의 인증 기법 12
제 1 절 문제 정의 12
제 2 절 AMR-tree 14
제 3 절 인증 자료구조 생성 단계 19
제 4 절 증거 생성 단계 23
제 5 절 증명 단계 25

제 4 장 실험 및 결과 27
제 1 절 실험 환경 및 데이터 27
제 2 절 상이 값 수 변화 실험 결과 28
제 3 절 레코드 개수 변화 실험 결과 30
제 4 절 차원 크기 변화 실험 결과 32

제 5 장 결론 33

참고문헌 35

Abstract 37

제 1 장 서론

제 1 절 연구의 배경

최근 IT 분야의 발전으로 데이터의 양이 폭발적으로 증가하고 있으며 다양한 분야에서 이러한 데이터를 분석하고 활용하려는 시도가 늘고 있다. 대용량의 데이터를 관리하고 분석하기 위해서는 비싼 인프라 구축이 필수적이기 때문에 이러한 비용문제로 개인이 빅데이터를 분석하는 것은 어렵다. 인프라 구축으로 인한 비용 문제를 해결하기 위한 대체 방안으로 데이터베이스 아웃소싱(database outsourcing)이 주목을 받고 있다. 데이터베이스 아웃소싱이란 데이터 소유자(data owner)가 제 3자 서버(third party server, 이하 '서버'라 한다)에 데이터 관리와 질의 처리 업무를 위탁하는 것이다. 즉, 데이터 소유자를 대신하여 서버가 데이터를 관리하고 사용자들로부터 질의를 받아 처리하는 것으로 비싼 인프라 구축이 필요하지 않다는 장점이 있다. 하지만 네트워크 특성상 보안 위협이 항상 존재하기 때문에 서버는 공격자들의 공격에 노출되어 있다. 따라서 서버에 아웃소싱된 데이터베이스는 변조될 수 있으며 또한 서버가 사용자로부터 받은 질의를 처리한 결과도 악의적으로 조작될 수 있어 서버가 처리한 질의결과를 신뢰하기 어렵다. 이로 인해 질의 결과가 정확한지 증명하는 질의 인증(query authentication) 기법의 중요도가 높아지고 있으며 최근까지 1차원 구간 질의[2], 다차원 구간 질의[3, 4], 함수 질의[10], kNN 질의[11] 등 여러 종류의 질의에 대한 인증 기법들이 연구되어왔다. 하지만 빅데이터 분석에 있어 활용도가 높은 집계 질의에 대한 질의 인증 연구는 아직 부족하다. 특히 기존 집계 질의 연구 [7]은 구간 조건을 가지는 집계 질의에 대한 인증 기법을 제안하였지만 데이터가 고차원이거나 속성별로 서로 다른 값이 많을 때 비효율적이다. 실생활 데이터는 일반적으로 차원 크기가 크며 속성별 서로 다른 값의 수가 많기 때문에 차원 크기에 영향이 적고 속성별 서로 다른 값의 수에 영향을 받지 않는 집계 질의 인증 기법이 필요하다.

제 2 절 연구의 내용

질의 인증은 데이터 소유자가 데이터베이스를 아웃소싱하였을 때 사용자가 신뢰할 수 없는 서버로부터 전달받은 질의 결과가 정확한지 확인하는 것이다. 질의 인증은 일반적으로 데이터 소유자가 처리하는 첫 번째 단계, 서버가 처리하는 두 번째 단계, 그리고 사용자가 처리하는 세 번째 단계로 진행되며 그 과정은 그림 1과 같으며 각 과정에 대해 자세히 설명한다.

첫 번째 단계는 데이터 소유자가 질의 인증을 위한 자료구조를 생성하는 단계이다. 데이터 소유자는 먼저 공개 키 알고리즘(e.g., RSA [8])을 이용하여 공개 키(public key)와 비밀 키(private key)를 생성한다. 이때 비밀 키는 데이터 소유자만 가지도록 하며, 공개 키는 누구나 접근 가능하도록 한다. 이후 데이터의 요약정보가 저장되는 *인증 자료구조*(authenticated data structure)를 생성하고 비밀 키로 인증 자료구조의 요약정보를 암호화하고 이를 *서명*(signature)이라 한다. 인증 자료구조는 데이터에 대한 요약 정보를 가지는 자료구조로 데이터가 변경되었을 때 데이터의 요약정보도 변경되므로 인증 자료구조 또한 변경되어야 한다. 만약 공격자가 데이터를 변경하였을 때, 변경된 데이터의 요약정보와 인증 자료구조가 일치하지 않으므로 데이터가 변경되었다는 것을 쉽게 알 수 있다. 서명은 인증 자료구조의 요약정보에 대해 암호화한 값으로 인증 자료구조가 원본과 일치하는지, 변경되었는지 확인하기 위함이

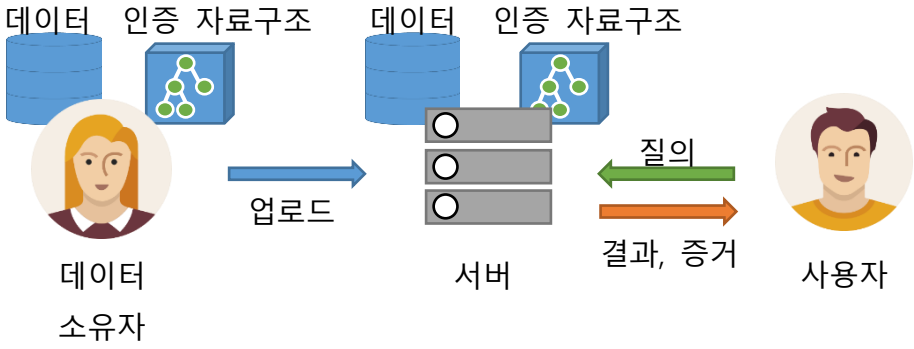


그림 1 질의 인증 과정

다. 만약 공격자가 데이터와 인증 자료구조를 변경하였을 때, 인증 자료구조의 요약정보는 원본 인증 자료구조의 요약정보와 일치할 수 없다. 따라서 서명을 데이터 소유자의 공개 키로 복호화한 값과 인증 자료구조의 요약정보를 비교하여 인증 자료구조가 원본인지 아닌지 쉽게 확인 가능하다. 만약 공격자가 공격자의 비밀 키를 사용하여 서명을 생성한다면, 데이터 소유자의 공개 키로 서명을 복호화하였을 때 비정상적인 값이 나오므로 인증 자료구조의 요약정보와 일치하지 않아 인증 자료구조가 조작되었다는 것을 알 수 있다. 데이터 소유자는 인증 자료구조와 서명을 생성하여 데이터와 함께 서버에 업로드 한다. 이 단계는 데이터를 업로드 할 때 단 한번 일어나는 단계이다.

두 번째 단계는 증거 생성 단계로 서버가 사용자로부터 질의를 받았을 때 질의에 대한 결과를 구하고, 질의 결과가 정확하다는 *증거*(proof)를 생성하는 단계이다. 증거는 질의 결과가 정확한지, 그리고 데이터가 변조되지 않았는지 확인하는 데 사용된다. 증거에는 인증 자료구조의 요약정보를 구하는 데 필요한 값들과 데이터 소유자의 서명, 그리고 질의 결과가 정확한지 확인할 수 있는 값들이 포함되어 있다. 이 단계는 질의를 처리할 때마다 수행되는 단계이다.

세 번째 단계는 증명 단계로 사용자가 서버로부터 질의 결과와 증거를 전달 받았을 때 질의 결과가 정확한지 증명하는 단계이다. 사용자는 전달받은 증거와 질의 결과를 통해 인증 자료구조의 요약 정보를 재생성하고, 데이터 소유자의 공개 키로 서명을 복호화하여 재생성한 요약 정보와 일치하는지 확인한다. 만약 서버의 데이터가 조작된 경우 서버의 인증 자료구조가 변경되었을 것이기 때문에 복호화한 서명, 즉 원본 인증 자료구조의 요약정보가 증거를 통해 재생성한 요약정보와 일치할 수 없어 조작된 데이터임을 사용자가 쉽게 알 수 있다. 이 단계는 두 번째 단계와 같이 질의를 처리할 때마다 수행되는 단계이다.

본 논문에서는 집계 질의를 위한 인증 기법을 제안한다. 집계 질의는 집계 함수와 구간 조건을 가지는 질의로 구간 조건을 만족하는 여러 레코드에 대해 집계 함수로 구한 값을 결과로 한다. 집계 함수는 다수의

레코드에 대해 하나의 값을 계산하는 함수로 SUM, AVERAGE, COUNT, MIN, MAX 등이 있다. 집계 질의는 실생활에서 자주 사용되는 질의 중 하나로 그 예로는 매출 데이터에서 특정 분기에 해당하는 영업 이익의 총합을 구하거나, 거래 내역 데이터에서 특정 제품이 거래된 건수를 찾아보는 등이 있다. 이렇듯 집계 질의는 활용도가 매우 높으며 빅데이터 분석을 위한 필수적인 질의 중 하나이다. 하지만 집계 질의에 대한 질의 인증 연구는 부족하며 집계 질의 인증을 위한 기존 연구는 고차원이거나 속성별로 서로 다른 값이 많을 때 매우 비효율적이다.

본 논문에서는 데이터의 차원 크기와 속성별 서로 다른 값의 수에 영향이 적은 효율적인 인증 자료구조 AMR-tree를 제안한다. 그리고 독립 노드와 독립 서브트리 개념을 도입하고 이를 활용하여 효율적인 증거 생성 방법을 제안한다. 또한, 이를 통해 생성된 증거를 바탕으로 집계 질의 결과가 정확한지 증명하는 방법을 제안한다. 먼저 2장에서 기존의 질의 인증 연구들과 집계 질의 인증 연구에 대해 간략한 소개를 한다. 그 후 3장에서 다차원 구간 집계 질의에 대한 문제 정의와 함께 다차원 구간 집계 질의 인증을 위한 효율적인 인증 자료구조를 소개하고 이를 생성하는 방법과 이를 활용한 증거 생성 방법, 그리고 증명 방법을 제안한다. 또한, 4장에서 제안한 인증 자료구조와 기존의 인증 자료구조를 다양한 실험을 통해 비교하고 그 결과를 분석한다. 마지막으로 5장에서는 내용을 정리하여 결론을 맺는다.

제 2 장 관련 연구

제 1 절 1차원 구간 질의 인증 기법

1차원 구간 질의는 한 개의 속성에 대한 구간 조건을 가지며 그 조건을 만족시키는 레코드들을 찾는 질의이다. [1]에서는 MH–tree [2]를 이용하여 1차원 구간 질의에 대한 인증 기법을 제안하였으며 이후 여러 연구들에서 MH–tree를 활용한 인증 기법이 제안되었다. 이 절에서는 MH–tree를 이용한 1차원 구간 질의 인증 기법에 대해 설명한다.

MH–tree를 생성하는 과정은 다음과 같다. 레코드를 구간 조건의 대상이 되는 속성 값으로 정렬하고 이진 트리를 생성한다. 리프 노드는 레코드의 해쉬 값을 가지며 내부 노드는 두 자식 노드의 해쉬 값을 이어 붙여(concatenation) 구한 해쉬 값을 가진다. MH–tree를 생성한 후 루트의 해쉬 값에 대해 데이터 소유자의 비밀 키로 서명을 생성한다. 예를 들어 그림 2는 8개의 정렬된 레코드($p_1 \leq p_2 \leq \cdots \leq p_8$)들에 대해 생성한 MH–Tree이다. 리프 노드는 레코드의 해쉬 값을 가지며 예제에서 H_1은 p_1에 대한 해쉬 값이다. 내부 노드는 두 자식 노드의 해쉬 값을 이어 붙여 구한 해쉬 값을 가지며 예제에서 $\mathrm{H}_{1:4}$는 두 자식 노드의 해쉬 값 $\mathrm{H}_{1:2}$와 $\mathrm{H}_{3:4}$를 이어 붙여 구한 해쉬 값 $\mathrm{H}_{1:4} = h(\mathrm{H}_{1:2}|\mathrm{H}_{3:4})$이다. 이때 h는 해쉬 함수이며 $|$는 두 값을 이어 붙이는 연산이다. MH–tree를 생성한 후 루트 노드의 해쉬 값 $\mathrm{H}_{1:8}$에 대해 서명 $Sig(\mathrm{H}_{1:8})$을 생성한다.

MH–tree를 사용하여 증거를 생성하는 방법은 다음과 같다. $\{p_a, \ldots, p_b\}$가 질의 결과일 때 p_{a-1}과 p_{b+1}를 경계 레코드라 하자. 증거를 생성하기 위해 먼저 경계 레코드를 찾는다. 그 후 루트 노드에서 왼쪽 경계 레코드로 가는 경로를 따라 각 노드를 방문하면서 왼쪽 형제 노드의 해쉬 값을 증거에 추가한다. 리프 노드를 방문하면 왼쪽 경계 레코드를 증거에 추가한다. 그리고 루트 노드에서 오른쪽 경계 레코드로 가는

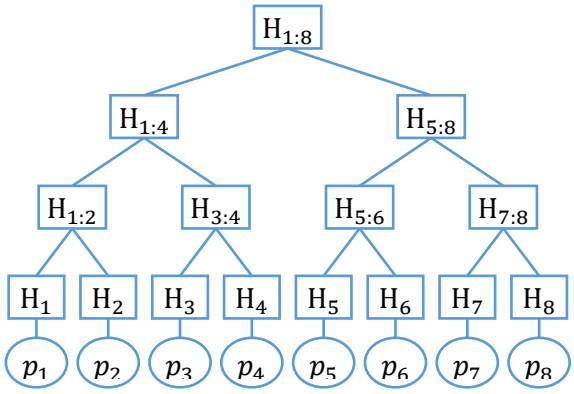


그림 2 MH-tree 구조

경로를 따라 각 노드를 방문하면서 오른쪽 형제 노드의 해쉬 값을 증거에 추가한다. 리프 노드를 방문하면 오른쪽 경계 레코드를 증거에 추가한다. 예를 들어 그림 2에서 질의 결과가 $\{p_4, p_5\}$라고 하자. 이때 경계 레코드는 p_3와 p_6이다. 먼저 루트 노드에서 왼쪽 경계 레코드인 p_3로 가는 경로를 따라 각 노드를 방문한다. 이때 왼쪽 형제 노드인 $H_{1:2}$을 증거에 추가한다. 그리고 리프 노드를 방문하였을 때 왼쪽 경계 레코드 p_3를 증거에 추가한다. 이 과정을 오른쪽 경계 레코드 p_6에 대해 수행하면 증거는 $[H_{1:2}, [p_3]]$, $[[p_6], H_{7:8}]$와 같다. 서버는 이와 같이 생성한 증거와 데이터 소유자가 생성한 서명을 함께 사용자에게 보낸다.

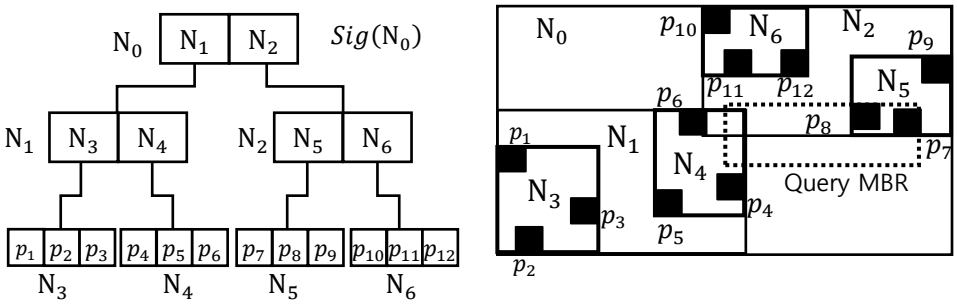
MH-tree를 통해 생성한 증거로 질의 결과가 정확한지 증명하는 방법은 다음과 같다. 먼저 질의 결과에 포함된 레코드들이 질의 결과를 만족하는지 확인한다. 그 후 증거를 통해 질의 결과 외 질의를 만족하는 레코드는 없는지, 증거에 포함된 데이터는 변조되지 않았는지 확인한다. 이는 경계 레코드가 질의를 만족하지 않는지 확인한 뒤, 경계 레코드와 결과 레코드, 그리고 증거에 포함된 노드들의 해쉬 값을 통해 루트 노드의 해쉬 값을 구하고 이를 서명과 비교하여 확인할 수 있다. 만약 결과 레코드나 경계 레코드가 변조되었다면 이들을 통해 구한 루트의 해쉬 값은 데이터 소유자가 생성한 MH-tree의 루트 해쉬 값과 다를 것이기 때문에 서명과 일치할 수 없다. 따라서 위의 과정을 통해 질의 결과가 정확한지 증명할 수 있다.

제 2 절 다차원 구간 질의 인증 기법

다차원 구간 질의는 여러 속성에 대한 구간 조건을 가지며 구간 조건을 만족하는 모든 레코드를 찾는 질의이다. 다차원 구간 질의 인증을 위해 [3]에서는 속성마다 MH–tree를 생성하는 프레임워크를 제안하였고 [4]에서는 다차원 데이터를 위한 자료구조인 R*–tree [5]를 기반으로 하는 MR–tree (Merkle R*–tree)를 제안하였다. 본 연구는 MR–tree를 확장하여 사용하므로 이 절에서 MR–tree에 대해 자세히 설명한다.

MR–tree는 MH–tree와 R*–tree를 확장한 인증 자료구조이며 트리의 구조는 R*–tree와 동일하다. 노드의 엔트리 최대 개수를 f라 할 때, 각 노드는 MBR(minimum bounding rectangle)과 엔트리 집합 $\mathrm{E} = \{e_1, e_2, \dots e_f\}$, 해쉬 값 H로 구성된다. 리프 노드의 경우 엔트리 e_i는 레코드이고, 내부 노드의 경우 자식 노드를 가리키며 자식 노드 e_i의 MBR을 MBR_i, 해쉬 값을 H_i로 나타낸다. 리프 노드의 해쉬 값 H는 레코드들을 이어 붙여 구한 해쉬 값 $\mathrm{H} = h(e_1|e_2|\dots|e_f)$이다. 이때 h는 해쉬 함수이며 | 는 두 값을 이어 붙이는 연산이다. 내부 노드의 해쉬 값 H는 자식 노드들의 MBR과 해쉬 값을 이어 붙여서 해쉬 함수 h를 통해 구한 해쉬 값 $\mathrm{H} = h(\mathrm{MBR}_1|\mathrm{H}_1|\mathrm{MBR}_2|\mathrm{H}_2|\dots|\mathrm{MBR}_f|\mathrm{H}_f)$이다. 각 노드의 해쉬 값은 엔트리들의 요약정보로 볼 수 있으며 루트의 해쉬 값은 전체 레코드들의 요약정보로 볼 수 있다. 따라서 데이터 소유자는 비밀 키로 루트의 해쉬 값에 대해 암호화하여 서명을 생성한다.

MR–tree를 이용하여 증거를 생성하는 과정은 다음과 같다. 서버는 사용자로부터 질의를 받았을 때, 루트부터 깊이 우선 순회를 수행하며 질의 MBR과 겹치는 자식 노드는 방문하고, 질의 MBR과 겹치지 않는 자식 노드는 증거에 노드를 추가한다. 리프 노드를 방문하였을 때는 모든 엔트리를 증거에 추가한다. 리프 노드의 엔트리들을 통해 리프 노드의 MBR과 해쉬 값을 재생성할 수 있으며 형제 노드들을 바탕으로 부모 노드의 MBR과 해쉬 값을 재생성할 수 있다. 증거에 추가된 엔트리들을



(a) 트리 구조 (b) 데이터

그림 3 MR-tree 구조

바탕으로 이 과정을 반복하면 루트의 MBR과 해쉬 값을 구할 수 있다. 그림 3은 MR-tree의 예제이다. 그림 3 (a)는 MR-tree의 구조를 나타내며 그림 3 (b)는 이를 2차원으로 표현한 것이다. 그림 3 (b)의 실선은 노드의 MBR을 나타내고 검은색 사각형은 레코드를 나타낸다. 그리고 점선으로 된 사각형은 질의 MBR을 나타내며 그 질의를 만족하는 레코드는 p_7과 p_8이다. 이때 증거를 생성하는 과정은 다음과 같다. 루트 노드를 방문하였을 때 N_1은 질의 MBR과 겹치므로 방문한다. N_1의 경우 N_3는 질의 MBR과 겹치지 않아 증거에 추가한다. N_4의 경우 질의 MBR과 겹치므로 방문하는데, N_4는 리프 노드이므로 모든 엔트리 p_4, p_5, p_6을 증거에 추가한다. 이후 루트 노드로 돌아가 N_2가 질의 MBR과 겹치므로 N_2를 방문한다. N_2의 엔트리 중 N_6은 질의 MBR과 겹치지 않으므로 증거에 추가하고, N_5는 질의 MBR과 겹쳐 방문한다. N_5는 리프 노드이므로 모든 엔트리 p_7, p_8, p_9을 증거에 추가한다. 이를 통해 생성된 증거는 $[[N_3, [p_4, p_5, p_6]], [[p_7, p_8, p_9], N_6]]]$와 같다. 이후 증거에 데이터 소유자의 서명 $Sig(N_0)$을 추가하여 사용자에게 결과와 함께 보낸다.

MR-tree를 통해 생성한 증거를 이용하여 결과가 정확한지 확인하는 방법은 다음과 같다. 먼저 증거가 조작되지 않았다는 것을 증명하고, 증거를 바탕으로 질의 결과가 정확한지 증명한다. 증거가 조작되지 않았다는 것을 증명하기 위해

증거를 바탕으로 MR-tree의 루트 해쉬 값을 구하고 서명과 일치하는지 확인하여 원본인지 확인한다. 증거에는 트리의 노드를 재생성하는

데 필요한 엔트리들이 포함되어있기 때문에 엔트리들을 이용하여 부모 노드를 재생성하고, 이를 반복하여 루트의 해쉬 값을 구할 수 있다. 루트의 해쉬 값은 모든 레코드에 대한 요약정보이므로 데이터가 조작되면 루트의 해쉬 값도 변경되어 원본과 달라지게 된다. 따라서 이를 서명을 복호화한 값 즉, 루트의 원본 해쉬 값과 일치하는지 확인한다. 그 후 결과 레코드들이 구간 조건을 만족하는지 확인하고, 증거에서 결과 레코드들을 제외한 레코드들 중 구간 조건을 만족하는 레코드가 없는지 확인하여 질의 결과가 정확한지 증명할 수 있다. 그림 3을 통해 증명 과정의 예제를 보인다. 그림 3 (b)에서 질의의 구간 조건을 만족하는 레코드는 p_7과 p_8이며 이에 대한 증거는 $[[\mathrm{N}_3, [p_4, p_5, p_6]], [[p_7, p_8, p_9], \mathrm{N}_6]]]$이다. 먼저 루트의 해쉬 값을 구한다. 먼저 p_4, p_5, p_6를 이용하여 N_4의 MBR과 해쉬 값을 구하고, 구한 N_4와 증거에 있는 N_3를 이용하여 N_1의 MBR과 해쉬 값을 구한다. 마찬가지로 p_7, p_8, p_9를 통해 N_5의 MBR과 해쉬 값을 구하고 이와 증거에 있는 N_6을 이용하여 N_2의 MBR과 해쉬 값을 구한다. 그 후 N_1과 N_2를 이용하여 루트 N_0의 해쉬 값을 구할 수 있다. N_0의 해쉬 값이 데이터 소유자의 서명 $Sig(\mathrm{N}_0)$을 복호화한 값과 일치하는지 확인하여 데이터가 조작되지 않았다는 것을 증명한다. 그 후 질의 결과 p_7과 p_8이 실제로 질의를 만족하는지 검사하고, 증거에서 질의 결과를 제외한 레코드들 p_4, p_5, p_6, p_9 중 구간 조건을 만족하는 레코드가 없는지 확인하여 결과가 정확하다는 것을 증명한다.

다차원 구간 질의는 구간 조건을 만족하는 레코드 자체를 찾는 것이 목적이다. 하지만 집계 질의는 구간 조건을 만족하는 레코드들의 집계 값을 찾는 것이 목적이다. 따라서 집계 질의 인증을 위해 MR-tree를 사용하면 구간 조건을 만족하는 레코드에 대한 증거를 생성하고 사용자가 직접 집계 값을 구해야 하기 때문에 증거를 생성하는 데 많은 시간이 걸리고 그 크기도 매우 크다. 또한 질의 결과를 증명하는 데 많은 시간이 걸리기 때문에 비효율적이다. 따라서 MR-tree는 집계 질의 인증을 위한 인증 자료구조로 사용하기에 부적합하다.

제 3 절 집계 질의 인증 기법

집계 질의는 다수의 속성에 대한 구간 조건을 가지며 구간 조건을 모두 만족하는 레코드들의 집계 값을 구하는 질의이며 집계 질의 인증 기법은 구간 조건을 만족하는 레코드들의 집계 값이 정확한지 확인하는 방법이다. [6]은 집계 질의 인증을 위해 인증 자료구조인 APS-tree를 제안하고 이를 활용한 증거 생성 및 증명 방법을 제안하였다. APS-tree는 전위 합 배열(prefix-sum array)[7]과 MH-tree를 확장한 인증 자료구조이다.

전위 합 배열은 데이터 큐브가 있을 때, 데이터 큐브의 각 원소에 대한 전위 합을 미리 구해놓은 배열이며 전위 합 배열을 사용하면 효율적으로 집계 값을 구할 수 있다. 데이터 큐브 A가 있을 때 전위 합 배열 P의 각 원소를 구하는 식은 다음과 같다.

$$P[x_1, x_2, \dots, x_m] = Sum(0{:}\,x_1, 0{:}\,x_2, \dots, 0{:}\,x_m)$$

$$= \sum_{0}^{x_1} \dots \sum_{0}^{x_m} A[i_1, \dots i_m]$$

전위 합 배열 P와 구간 조건$(\ell_1{:}\,h_1, \ell_2{:}\,h_2, \dots, \ell_m{:}\,h_m)$이 주어졌을 때, 다음 식과 같이 2^m개의 꼭지 값(corner point)들을 찾고 이들을 이용하여 레코드들의 합을 구할 수 있다.

$$\mathrm{Sum}(\ell_1{:}\,h_1, \ell_2{:}\,h_2, \dots, \ell_m{:}\,h_m) = \sum_{\forall x_j \in \{\ell_j - 1, h_j\}} \left\{ \left(\prod_{i=1}^{d} s(i) \right) \times P[x_1, x_2, \dots, x_m] \right\}$$

$$s(j) = \begin{cases} 1, & x_j = h_j \\ -1, & x_j = \ell_j \end{cases}, \quad 1 \le j \le m$$

전위 합 배열의 크기는 데이터 큐브의 크기와 동일하다. 각 속성별로 서로 다른 값을 상이 값 수(number of distinct values)라 하고 i 번째 상이 값 수를 $|V_i|$로 나타낼 때, 속성 수가 m인 데이터에 대해 데이터 큐브의 크기는 $O(\prod_{i=1}^{m} |V_i|)$이다. 이때 하나의 전위 합 배열은 특정 속

GPA	Age	Score
3.1	21	40
3.3	20	30
3.1	21	60
3.2	21	20

(a) 데이터

Age / GPA	20	21
3.1	0	100
3.2	0	60
3.3	30	0

(b) 데이터 큐브

Age / GPA	20	21
3.1	0	100
3.2	0	160
3.3	30	190

(c) 전위 합 배열

그림 4 전위 합 배열 예제

성의 집계 값을 구할 수 있다. 따라서 임의의 속성에 대한 집계 값을 구하기 위해서는 모든 속성에 대해 전위 합 배열을 생성하여야 하므로 전체 공간 복잡도는 $O(m\prod_{i=1}^{m}|V_i|)$이다.

그림 4는 3차원 데이터에 대한 예제로 그림 4 (a)는 예제 데이터이며 그림 4 (b)는 데이터에 대한 데이터 큐브이고 그림 4 (c)는 데이터 큐브에 대한 전위 합 배열이다. 데이터는 GPA, Age, Score 속성으로 이루어져 있으며, 질의는 Score에 대한 집계 질의만 있다고 가정한다. 이때, 데이터 큐브와 전위 합 배열의 크기는 2×3이다. Age가 21이고, GPA가 3.2 이상인 레코드들의 Score 집계 값을 구하는 질의가 있을 때, 집계 값은 $P[3,2]-P[1,2]-P[3,1]-P[1,1]=60$이다.

APS-tree는 전위 합 배열을 1차원 배열로 만들고 이에 대해 MH-tree를 생성한 구조이다. 증거를 생성하는 방법은 2^m개의 꼭지 값을 찾고, 각 꼭지 값에 대하여 MH-tree를 사용하여 증거를 생성한다. 증명 방법은 증거를 통해 각 꼭지 값이 정확한지 증명한 후 꼭지 값들을 이용하여 집계 값을 구하고 질의 결과와 일치하는지 확인한다. APS-tree는 전위 합 배열에 대한 MH-tree 구조를 가지므로 전체 크기는 전위 합 배열과 같아 $O(m\prod_{i=1}^{m}|V_i|)$이다. APS-tree를 이용하여 증거를 생성할 때는 2^m개의 꼭지 값에 대해 MH-tree를 사용하여 증거를 생성하여야 하므로 각 꼭지 값마다 $O(M\log M)$이다. 이때 M은 전위 합 배열의 크기 $O(m\prod_{i=1}^{m}|V_i|)$를 의미한다. 일반적으로 실생활 데이터는 속성 수와 상이 값 수 모두 크기 때문에 APS-tree는 비효율적이다.

제 3 장 집계 질의 인증 기법

제 1 절 문제 정의

집계 질의는 다차원 구간 조건을 만족하는 레코드들에 대해 집계 함수로 계산한 값을 찾는 질의이다. 다차원 구간 조건은 테이블에 존재하는 속성들에 대한 구간 조건이며 집계 함수는 여러 레코드를 하나의 값으로 계산하는 함수이다. 집계 함수의 대상이 되는 속성은 테이블에 존재하는 속성이어야 하며 집계 함수의 예로는 SUM, COUNT, AVERAGE, MIN, MAX 등이 있다. 예를 들어 그림 5의 학생 테이블에 대해 사용자가 다음과 같은 집계 질의를 전송한다고 하자. "SELECT SUM(점수) FROM 학생 WHERE 20 ≤ 나이 ≤ 21 AND 3.9≤ GPA". 질의는 학생 테이블에서 나이와 GPA 속성에 대한 구간 조건을 가지며 집계함수의 대상이 되는 속성은 점수 속성이다. 질의는 구간 조건을 모두 만족하는 레코드들 $\{r_2, r_3, r_7\}$의 점수 속성 값을 집계한 값으로 그 결과는 250이다.

학생 테이블

rID	나이	GPA	점수
r_1	20	3.5	80
r_2	21	4.0	90
r_3	21	3.9	70
r_4	19	3.6	85
r_5	24	3.8	65
r_6	20	3.2	50
r_7	20	3.9	90

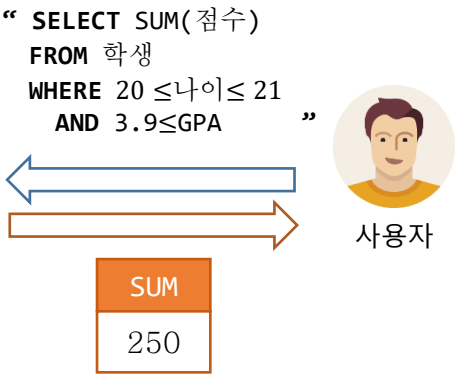


그림 5 집계 질의 예제

집계 질의 인증은 신뢰할 수 없는 서버로 데이터베이스가 아웃소싱되었을 때, 사용자가 서버로부터 받은 집계 질의의 결과가 정확한지 확인하는 것이다. 집계 질의 인증 문제는 데이터 소유자, 서버, 사용자 입장에서 각각 달리 정의된다.

데이터 소유자의 경우, 데이터에 대해 인증 자료구조를 생성하고 자료구조에 대한 서명을 생성하는 문제로 정의된다. 인증 자료구조는 데이터에 대한 요약 정보를 가지고 있으며 데이터가 변조되었을 경우 인증 자료구조도 그에 맞게 변경되어야 한다. 서명은 인증 자료구조의 요약정보에 대해 데이터 소유자의 비밀 키를 이용하여 생성된다.

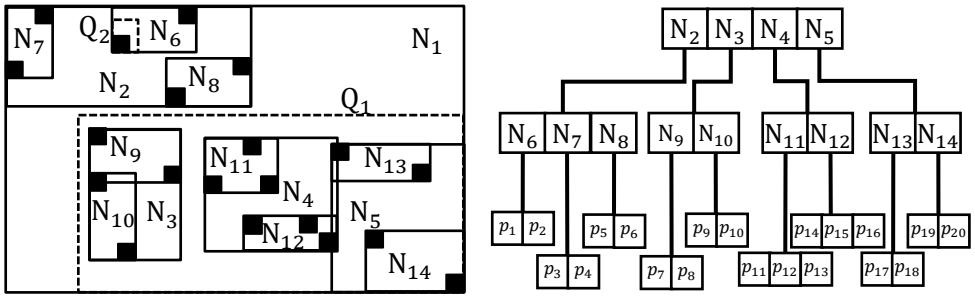
서버의 경우, 아웃소싱된 데이터베이스와 인증 자료구조가 주어지고 사용자로부터 집계 질의를 받았을 때, 질의에 대한 결과를 구하고 그 결과가 정확한지 증명할 수 있는 증거를 생성하는 문제로 정의한다. 증거는 인증 자료구조를 재생성 하는 데 필요한 값들과 서명으로 구성된다.

사용자의 경우, 서버로부터 질의 결과와 증거를 받았을 때 증거를 이용하여 질의 결과가 정확한지 확인하는 문제로 정의한다. 사용자는 증거를 바탕으로 결과가 구간 조건을 만족하는 모든 레코드들의 집계 값과 일치하는지 확인하고, 증거에 있는 값들이 변조된 것은 아닌지 서명을 통해 확인하여야 한다.

제 2 절 AMR-tree

본 논문에서는 집계 질의 인증을 위해 MR-tree를 확장한 인증 자료구조인 AMR-tree (Aggregation Merkle R*-tree)를 제안하고 이를 활용한 증거 생성 방법과 증명 방법을 제안한다. 집계 질의 인증을 위한 단순한 방법은 다차원 구간 질의 인증 기법을 사용하는 것이다. 집계 질의는 구간 조건을 만족하는 레코드들의 집계 값을 찾는 질의이고 다차원 구간 질의는 구간 조건을 만족하는 레코드 자체를 찾는 질의로 두 질의의 목적은 다르다. 하지만 다차원 구간 질의를 처리하여 구한 레코드들을 통해 집계 값을 구할 수 있기 때문에 다차원 구간 질의를 위한 인증 기법을 사용할 수 있다. 다차원 구간 조건 질의를 위한 인증 기법인 MR-tree를 사용하여 구간 조건을 만족하는 모든 레코드에 대한 증명을 생성하고 사용자가 직접 모든 레코드에 대한 집계 값을 구하여 집계 질의 결과가 정확한지 증명할 수 있다. 하지만 집계 질의의 결과는 단 하나의 집계 값이므로 MR-tree를 사용한 집계 질의 인증 방법은 증거의 크기가 크고 증명하는 데 많은 시간이 걸린다. 본 논문에서 제안하는 AMR-tree는 두 가지 방법을 통해 MR-tree를 확장시키고 효율적인 집계 질의 인증을 가능하게 한다.

첫 번째 방법은 각 노드의 자손 노드에 포함되어 있는 레코드들의 집계 값을 미리 구하는 방법이다. MR-tree의 경우 다차원 구간 질의 인증을 위한 인증 자료구조로 구간 조건을 만족하는 모든 레코드를 증거에 추가한다. 하지만 집계 질의의 목적은 레코드 자체가 아닌 집계 값을 구하는 것이다. 만약 어떤 노드의 MBR이 질의 MBR에 포함되면 해당 노드의 자손 노드에 포함된 모든 레코드는 구간 조건을 만족한다. 따라서 인증 자료구조를 생성할 때 각 노드의 자손 노드에 포함된 레코드들의 집계 값을 구하여 저장하고, 이후 증거를 생성할 때 자손 노드들을 방문하여 레코드들을 증거에 추가하기보다 구해놓은 집계 값만을 증거에 추가하는 것이 효율적이다. 이를 통해 증거를 생성하는 데 걸리는 시간을 줄일 수 있으며 증거의 크기를 줄여 네트워크 오버헤드를 줄일 수 있다. 또한 집계 값을 구하기 위한 계산량을 줄여 사용자가 결과를 증명하는



(a) 데이터 (b) 트리 구조

그림 6. MR-tree 예제

데 걸리는 시간을 줄일 수 있다. 그림 6은 MR-tree의 방법과 AMR-tree의 방법을 사용하였을 때 나타나는 차이를 보이기 위한 예제다. 그림 6 (a)는 데이터를 2차원으로 나타낸 것으로 실선으로 된 사각형은 노드의 MBR을 나타내며, 검은 사각형은 레코드를 나타내고, 점선으로 된 사각형들은 질의 MBR을 나타낸다. 그림 6 (b)는 그림 6 (a)를 트리 구조로 나타낸 것이다. Q_1 질의에 대해 MR-tree의 증거 생성 방법으로 증거를 생성하였을 때, N_2의 경우 질의 MBR과 겹치지 않으므로 방문하지 않고 증거에 추가된다. 반면에 N_3, N_4, N_5의 경우 질의 MBR과 겹치기 때문에 모든 자손 노드를 방문하게 되고 리프 노드를 방문하면 해당 노드에 있는 모든 레코드들을 추가하게 된다. 이를 통해 구해진 증거는 $[N_2, [[p_7, p_8], [p_9, p_{10}]], [[p_{11}, p_{12}, p_{13}], [p_{14}, p_{15}, p_{16}]], [[p_{17}, p_{18}], [p_{19}, p_{20}]]]$와 같다. 만약 각 노드마다 자손 리프 노드에 있는 레코드의 집계 값을 미리 구해놓았다면, 질의 MBR에 MBR이 포함되는 N_3, N_4, N_5의 경우 자손 노드들을 방문하지 않고 미리 구해놓은 집계 값만을 증거에 추가하면 된다. 따라서 증거는 $[N_2, N_3, N_4, N_5]$와 같으며 이는 MR-tree의 증거 생성 방법을 사용하였을 때보다 증거 생성 시간도 짧고 증거의 크기가 작으며 증거의 엔트리 수가 적기 때문에 집계 값을 구하여 증명하는 데 걸리는 시간도 짧다.

두 번째 방법은 증거를 생성하는 데 전체 트리를 이용하는 것이 아닌 질의 MBR을 포함하는 서브트리만 이용하는 방법이다. MR-tree의 경우, 질의 MBR과 겹치지 않아 자손 노드에 구간 조건을 만족하는 레

코드가 없는 것이 명확한 노드도 증거를 생성할 때 루트부터 깊이 우선 순회를 수행하여 증거에 추가되는 경우가 빈번히 발생한다. 이렇듯 자손 노드에 구간 조건을 만족하는 레코드가 없는 것이 명확한 노드는 증거에 불필요하며, 이로 인해 트리의 높이가 크거나 노드의 최대 엔트리 개수가 많을 때 비효율적이다. 본 논문에서는 독립 서브 트리 개념을 도입하여 증거의 크기와 생성 시간을 줄이는 방법을 제안한다. 어떤 노드가 조상 노드와 자손 노드를 제외한 다른 노드와 MBR이 겹치지 않을 때 이를 독립 노드라 하고, 독립 노드를 루트로 하는 서브트리를 독립 서브트리라고 하자. 만약 어떤 독립 서브트리가 질의 MBR을 포함하면 구간 조건을 만족하는 레코드는 해당 독립 서브트리의 리프 노드에만 존재할 수 있으며 해당 독립 서브트리의 리프 노드들을 제외한 다른 리프 노드에는 구간 조건을 만족하는 레코드가 있을 수 없다. 따라서 독립 서브트리의 루트부터 증거 생성을 시작해도 사용자는 독립 서브트리를 제외한 다른 노드에 구간 조건을 만족하는 레코드가 없다는 것을 알 수 있어 결과가 정확한지 증명할 수 있다. 이러한 속성을 이용하여 AMR-tree의 경우, 질의 MBR을 포함하는 독립 서브트리를 찾고 해당 독립 서브트리만을 이용하여 증거를 생성한다. 이를 통해 증거를 생성하는 데 걸리는 시간과 증명하는 데 걸리는 시간을 줄일 수 있으며 증거의 크기를 줄일 수 있다. 그림 6을 예로 들면 Q_2 질의에 대해 N_2와 N_6가 질의 MBR을 포함하는 독립 노드이다. 이때 MR-tree를 통해 증거를 생성하면 [[[p_1, p_2], N_7, N_8], N_3, N_4, N_5] 와 같다. N_6은 질의 MBR을 포함하는 독립 노드로 자손 리프 노드를 제외한 다른 리프 노드인 N_7, N_8, N_9, N_{10}, N_{11}, N_{12}, N_{13}, N_{14}에는 구간 조건을 만족하는 레코드가 있을 수 없고 그러므로 MR-tree로 생성한 증거에서 N_7, N_8, N_3(N_9, N_{10}의 조상 노드), N_4(N_{11}, N_{12}의 조상 노드), N_5(N_{13}, N_{14}의 조상 노드)는 불필요하다. 그럼에도 증거에 추가되는 이유는 두 가지가 있다. 첫 번째는 N_7, N_8에 구간 조건을 만족하는 레코드가 없는지, N_3, N_4, N_5의 MBR이 질의 MBR과 겹치지 않아 자손 리프 노드에 구간 조건을 만족하는 레코드가 없는지 확인하기 위함이다. 두 번째는 루트의 해쉬 값을 구하기 위해

필요하기 때문이다. N_2의 해쉬 값을 구하기 위해 N_7, N_8이 필요하고 루트의 해쉬 값을 구하기 위해 N_3, N_4, N_5가 필요하다. AMR-tree에서는 이러한 불필요한 노드가 증거에 추가되는 것을 줄이기 위해 독립 노드와 독립 서브트리 개념을 도입하고 이를 활용한다. 노드 중 자손과 조상 노드를 제외한 다른 노드와 MBR이 겹치지 않는 노드들을 독립적이라고 하며 독립 노드를 루트로 하는 서브트리를 독립 서브트리라고 한다. AMR-tree는 서명을 생성할 때 루트 노드 뿐만 아니라 모든 독립 노드에 대해 서명을 생성한다. 그리고 증거를 생성할 때, 질의의 MBR을 포함하는 독립 서브트리를 찾는다. 이러한 서브트리 중 그 높이가 가장 낮은 서브트리를 최소 독립 서브트리라고 하자. 질의 MBR을 포함하는 최소 독립 서브트리를 찾아 해당 서브트리의 루트부터 깊이 우선 순회를 시작하여 증거를 생성한다. 그림 6을 예로 들었을 때, 질의 MBR을 포함하는 독립 노드는 N_2와 N_6이 있다. N_6을 루트로 하는 서브트리가 N_2를 루트로 하는 서브트리보다 높이가 낮으므로 최소 독립 서브트리가 되며 이에 대해 증거 생성을 시작한다. N_6은 루트이며 리프 노드이기 때문에 결과적으로 증거는 $[p_1, p_2]$가 된다. MR-tree를 통해 생성된 증거에서 불필요하였던 N_7, N_8, N_3, N_4, N_5가 추가되지 않았음을 확인할 수 있다.

AMR-tree의 구조는 다음과 같다. 노드는 엔트리 집합 $E = \{e_1, e_2, e_3, \dots, e_f\}$와 집계 배열 $S[1:m]$, 해쉬 배열 $H[1:m]$로 이루어진다. 이때 f는 노드의 최대 엔트리 개수이며 m은 데이터의 속성 수이다. 리프 노드의 i 번째 엔트리 e_i는 레코드를 가지고 내부 노드의 i 번째 엔트리 e_i는 자식 노드를 가리키는 포인터 p_i를 가지며 해당 자식 노드의 최소 경계 사각형을 MBR_i, 집계 배열을 S_i, 해쉬 배열을 H_i로 표현한다. 집계 배열은 자손 리프 노드에 있는 레코드들의 각 속성별 집계 값이며 해쉬 배열은 자손 노드들을 요약한 정보를 가진다. 리프 노드 집계 배열의 d 번째 원소 $S[d]$는 레코드들의 d 번째 속성 값 $e_i[d]$을 집계한 값 $S[d] = e_1[d] + e_2[d] + \cdots + e_f[d]$이며 리프 노드 해쉬 배열의 d 번째 원소 $H[d]$는 d번째 집계 값 $S[d]$, 그리고 레코드들을 이어 붙여 구한 해쉬 값 $H[d] = h(S[d]|e_1|e_2|\dots|e_f)$이다. 내부 노드 집계 배열의 d 번째 원소 $S[d]$

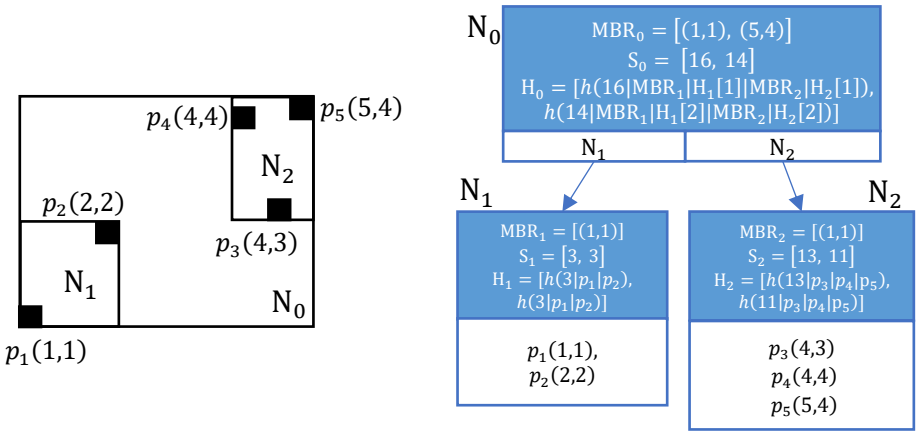
는 자식 노드 집계 배열의 d 번째 원소를 집계 한 값 $\mathrm{S[d]} = \mathrm{S_1[d]} + \mathrm{S_2[d]} + \mathrm{S_3[d]} + \cdots + \mathrm{S}_f\mathrm{[d]}$ 이며 해쉬 배열의 d 번째 원소 H[d]는 해당 노드 집계 배열의 d 번째 원소 S[d]와 자식 노드마다 MBR과 d번째 해쉬 배열 원소를 이어 붙여 구하고 $\mathrm{H[d]} = h\left(\mathrm{S[d]|MBR_1|H_1[d]}\middle| \ldots \middle|\mathrm{MBR}_f\middle|\mathrm{H}_f\mathrm{[d]}\right)$와 같다. 서명은 모든 독립 노드마다 해당 노드의 해쉬 배열을 이어 붙여 $\mathrm{Sig(H[1]|H[2]| \ldots |H[d])}$와 같이 생성한다.

제 3 절 인증 자료구조 생성

AMR-tree를 생성하는 과정은 다음과 같다. R*-tree를 생성한 후 각 노드의 집계 배열과 해쉬 배열을 구하고 모든 독립 노드에 서명을 생성한다. 그림 8은 AMR-tree를 생성하는 알고리즘의 의사 코드이다. BuildAMR-tree 함수는 AMR-tree를 생성하는 함수로 BuildR*-tree 함수를 통해 R*-tree를 생성하고(행 1), Aggregation 함수를 호출하여 각 노드의 집계 배열과 해쉬 배열을 구한다(행 2). 그 후 Sign 함수를 통해 모든 독립 노드를 찾아 서명을 생성한다(행 3). BuildR*-tree 함수는 R*-tree를 생성하는 함수로 생성된 R*-tree의 루트를 반환한다. Aggregation 함수는 노드 N을 입력으로 받아 노드 N이 리프 노드가 아닐 때, 자식 노드들마다 재귀 호출하여 자식 노드들의 집계 값과 해쉬 값을 구하게 하고(행 2-3) 자식 노드들의 집계 값을 통해 노드 N의 집계 값을 구한다(행 4-6). 노드 N이 리프 노드인 경우에는 레코드들을 이용하여 노드 N의 집계 값을 구한다(행 4-5). 집계 값을 구한 후 각 속성별 해쉬 값 H[d]를 구한다. 노드 N이 리프 노드일 때 집계 값과 각 레코드를 이어 붙인 후 해쉬 값을 구하고(행 7-9), 리프 노드가 아닐 때 집계 값과 자식노드들의 MBR과 해쉬 값을 이어 붙인 후 해쉬 값을 구한다(행 7-11). Sign 함수는 노드 N을 입력으로 받는다. 만약 노드 N이 루트라면 서명을 생성할 때 루트 노드인 것을 명시하기 위해 root를 앞에 붙여 생성한다(행 1-2). 독립 노드일 경우 독립 노드임을 명시하기 위해 independent를 붙여 생성하고(행 3-4), 루트나 독립 노드가 아닐 경우 서명을 생성하지 않는다(행 5). 그 후 자손 노드에 대해 재귀 호출하여 독립 노드를 찾아 서명을 생성한다(행 6-7).

그림 7은 AMR-tree의 생성 과정을 보이기 위한 예제로 그림 7 (a)는 예제 데이터에 대하여 R*-tree를 생성하고 이를 2차원으로 나타낸 것이다. 사각형은 레코드를 나타내고 실선의 사각형은 노드의 MBR을 나타낸다. 그림 7 (b)는 그림 7 (a)의 데이터에 대해 생성한 AMR-tree를 트리 구조로 나타낸 것이다. AMR-tree는 먼저 Aggregation 함수를

통해 각 노드의 집계 배열과 해쉬 배열을 구한다. N_1의 경우 집계 배열 S_1은 p_1과 p_2를 더한 값들 [3,3]이며 해쉬 배열 H_1의 1 번째 원소 $H_1[1]$은 $S_1[1]$와 레코드들을 이어 붙여 구한 해쉬 값 $h(3|p_1|p_2)$이며 2 번째 원소 $H_1[2]$은 $S_1[2]$와 레코드들을 이어 붙여 구한 값 $h(3|p_1|p_2)$이다. 노드 N_2도 마찬가지로 구할 수 있다. 노드 N_0의 두 자식 노드에 대한 집계 배열과 해쉬 배열을 구하였으므로 루트 노드의 집계 배열과 해쉬 배열을 구할 수 있다. 루트 노드 N_0의 집계 배열은 각 차원마다 자식 노드들의 집계 값의 합으로 $[S_1[1]+S_2[1],\ S_1[2]+S_2[2]]$와 같고 이는 $[16, 14]$이다. 해쉬 배열의 1 번째 원소 $H_0[1]$는 1 번째 집계 배열 원소 $S_0[1]$와 자식 노드마다 MBR과 1 번째 해쉬 배열 원소를 이어 붙여 구한 해쉬 값 $h(S_0[1]|MBR_1|H_1[1]|MBR_2|H_2[1])$이며 2 번째 원소 $H_0[2]$ 또한 이와 동일하게 구한 값 $h(S_0[2]|MBR_1|H_1[2]|MBR_2|H_2[2])$이다. Aggregation 함수가 종료되면 Sign 함수를 통해 모든 독립 노드에 서명을 생성한다. 루트 노드인 N_0에 대해 먼저 서명을 생성하고 그 후 모든 독립 노드에 대해 서명을 생성한다. N_0는 루트 노드로 서명은 $Sig(\text{root}|H_1[1]|H_0[2])$과 같다. N_1과 N_2는 독립 노드이므로 각각 $Sig(\text{independet}|H_1[1]|H_1[2])$, $Sig(\text{independent}|H_2[1]|H_2[2])$와 같이 서명을 생성한다. Sign 함수가 종료되면 AMR-tree의 생성이 완료된다.



(a) 예제 데이터 (b) AMR-tree의 구조

그림 7. AMR-tree 예제

```
Function BuildAMR-tree
Input:
Output:
      Root of AMR-tree
Begin
1.  root = BuildR*-tree()
2.  Aggregation(root)
3.  Sign(root)
4.  Return root
End

Function Aggregation
Input:
      N: Node
Output:
Begin
1.  For each entry e ∈ N.E
2.     If N is not a leaf
3.        Aggregation(e.p)
4.     For each d = 1 to m
5.        If N is a leaf, N.S[d] += e[d]
6.        Else N.S[d] += e.S[d]
7.  For each d = 1 to m
8.     If N is a leaf
9.        N.H[d] = h(N.S[d]|e_1|e_2| ... |e_f)
10.    Else
11.       N.H[d] = h(N.S[d]|MBR_1|H_1[d]| ... |MBR_f|H_f[d])
End

Function Sign
Input:
      N: Node
Output:
Begin
1.  If N is a root
2.     N.sign = Sig(root|N.H[1]|N.H[2]| ... |N.H[m])
3.  Else If N is independent
4.     N.sign = Sig(independent|N.H[1]|N.H[2]| ... |N.H[m])
5.  Else N.sign = nil
6.  For each entry e in N.E
7.     Sign(e.p)
End
```

그림 8. AMR-tree 생성 알고리즘의 의사 코드

제 4 절 증거 생성 단계

사용자는 증거를 이용하여 질의 결과를 구하는 데 사용된 데이터가 변조되지 않았는지, 질의 결과가 정확한지 확인하여야 한다. 서명은 원본 인증 자료구조의 요약정보에 대한 암호화된 값이므로 이를 이용하여 데이터가 변조되지 않았는지 확인한다. AMR-tree에서 서명은 독립 노드마다 생성되므로 증거를 생성하기 위해 AMR-tree 전체가 아닌 질의 MBR을 포함하는 독립 서브트리 중 높이가 최소인 독립 서브트리만을 이용한다. 증거에는 루트의 해쉬 값을 구하는 데 필요한 엔트리들이 추가되어야 하므로 루트에서부터 깊이 우선 순회를 수행한다. 이 때 자식 노드 중 질의 MBR과 겹치지만 포함되지 않는 자식 노드들만 방문하며, 질의 MBR에 포함되거나 겹치지 않는 노드의 경우, 해당 노드의 정보(MBR, 집계 값, 해쉬 값)를 증거에 추가한다. 질의 MBR에 포함되는 자식 노드의 경우 자손 노드에 존재하는 모든 레코드가 질의의 구간 조건을 만족하므로 자식 노드를 방문하지 않고 해당 노드의 정보만을 증거에 추가하며, 질의 MBR과 겹치지 않는 자식 노드의 경우 자손 노드에 존재하는 모든 레코드가 질의의 구간 조건을 만족하지 않으므로 방문하지 않고 노드의 정보만 증거에 추가한다. 질의 MBR과 겹치지만 포함되지 않는 노드의 경우, 자손 노드에 존재하는 레코드 중 어떤 레코드가 질의를 만족할지 알 수 없기 때문에 방문한다. 그리고 만약 리프 노드에 방문하였을 때는 모든 엔트리를 증거에 추가한다. 정리하면 루트부터 깊이 우선 순회를 하며 다음과 같은 세 종류의 엔트리를 증거에 추가한다: 1) 방문한 리프 노드의 모든 엔트리. 2) 방문한 내부 노드의 엔트리 중 질의 MBR에 포함되는 엔트리. 3) 방문한 내부 노드의 엔트리 중 질의 MBR과 겹치지 않는 엔트리.

그림 9은 AMR-tree를 사용하여 증거를 생성하는 알고리즘의 의사 코드이다. ProofConstruction 함수는 질의 Q가 주어졌을 때 증거 π를 찾는 함수이다. 질의 Q는 구간 조건 MBR과 집계할 속성 attr를 가진다. 수행 과정은 다음과 같다. FindSubtree 함수를 통해 질의 MBR을 포함하는 최소 독립 서브트리의 루트 sroot를 찾고 sroot의 서명을 증

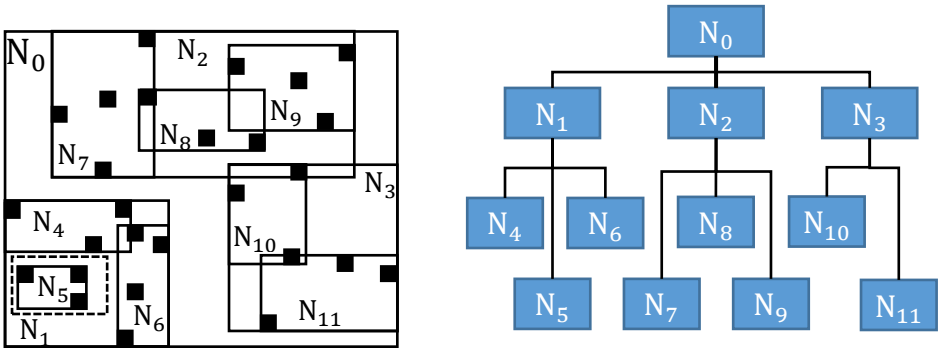
```
Function ProofConstruction
Input:
      Q: Query
Output:
      Proof
Begin
1.  sroot = FindSubtree (root, Q);
2.  Append sroot.sign To π
3.  Append Hash values except for sroot.H[Q.attr] To π
4.  Append RangeQuery (sroot, Q) To π
5.  Return π
End

Function FindSubtree
Input:
      Q: Query
      N: Node
Output:
      Root of independent subtree
Begin
1.  If there exists e_i ∈ N.E such that
    e_i is an independent node and contains Q.MBR
2.     Return FindSubtree (e_i, Q)
3.  Return N
End

Function RangeQuery
Input:
      Q: Query
      N: Node
Output:
      Proof
Begin
1.  For each entry e ∈ N.E
2.     If N is leaf Then Append e To π
3.     Else
4.        If Q.MBR contains e.MBR
5.           Append e To π
6.        Else If e.MBR overlaps with Q.MBR
7.           Append RangeQuery (e.p, Q) To π
8.        Else Append e To π
9.  Return p
End
```

그림 9. 증거 생성 알고리즘의 의사 코드

거에 추가한다(행 1-2). 그리고 집계 속성을 제외한 다른 속성들의 해쉬 값을 증거 π에 추가한다(행 3). 그리고 RangeQuery 함수를 통해 루트 노드의 해쉬 값 H[Q.attr]을 구하는 데 필요한 엔트리들을 찾아 증거 π에 추가한 후 π를 반환한다(행 4-5). FindSubtree 함수는 최소

독립 서브트리를 찾기 위해 루트 노드부터 시작하여 질의를 포함하는 독립 노드를 찾고 재귀 호출하여 최소 독립 서브트리를 찾는다(행 1-3). RangeQuery 함수는 자식 노드가 질의 MBR에 포함되거나 겹치지 않을 때 엔트리를 추가하고 그 외의 경우에 재귀 호출하여 반환된 증거를 증거 p에 추가한다(행 4-8).

그림 10는 AMR-tree를 통해 증거를 생성하는 과정을 보이기 위한 예제이다. 그림 10 (a)는 예제 데이터와 이에 대해 생성한 AMR-tree를 2차원으로 표현한 것이다. 실선은 노드의 MBR을 나타내고 검은 사각형은 레코드를 나타내며 점선은 질의 MBR을 나타낸다. 그림 10 (b)는 AMR-tree를 트리 구조로 나타낸 것이다. 서명은 루트 노드인 N_0과 독립 노드인 N_1, N_5에 대해 생성되어 있다. 먼저 질의 MBR을 포함하는 높이가 최소인 독립 서브트리를 찾는다. N_1을 루트 노드로 하는 서브트리가 높이가 최소인 독립 서브트리가 된다. 따라서 N_1의 서명을 증거에 추가한다. 그 후 N_1에서 깊이 우선 순회를 수행하며 해쉬 값을 구하는데 필요한 엔트리들을 증거에 추가한다. N_4와 N_6은 질의 MBR과 겹치지 않으므로 방문하지 않으며 N_5는 질의 MBR에 포함되므로 방문하지 않고 이들의 노드 정보만 증거에 추가한다. 이를 통해 생성한 증거는 $[Sig(N_1), [N_4, N_5, N_6]]$이며 이때 i는 집계 속성 이외의 속성을 의미한다.



(a) 예제 데이터 (b) 트리 구조

그림 10. 증거 생성 예제

제 5 절 증명 단계

AMR-tree를 사용하여 생성한 증거는 서명과 집계 속성을 제외한 다른 속성에 대한 해쉬 값, 그리고 집계 속성 해쉬 값을 구하는 데 필요한 엔트리들을 포함한다. 사용자는 증거를 이용하여 집계 속성 해쉬 값을 구하고 서명을 데이터 소유자의 공개 키로 복호화화여 값이 일치하는지 확인한다. 이를 통해 사용자는 증거에 포함되어 있는 엔트리들이 변조되지 않았음을 알 수 있다. 그 후 엔트리 중 질의 MBR에 포함되는 엔트리들에 대한 집계 값을 구하고 이를 질의 결과와 비교하여 질의 결과가 정확한지 확인한다.

그림 11은 증명 알고리즘의 의사 코드이다. Verification 함수는 먼저 RootHash 함수를 통해 루트의 노드 정보(MBR_r, H_r, S_r)과 증거에 포함된 엔트리 중 구간 조건을 만족하는 엔트리들을 집계한 값 R_r을 구한다(행 1). 그리고 증거에 포함된 엔트리들이 원본인지 확인하기 위해 루트의 집계 속성 해쉬 값 H_r과 증거에 포함된 집계 속성을 제외한 속성의 해쉬 값들을 이어 붙여 서명을 데이터 소유자의 공개 키로 복호화한 값과 비교한다(행 2). 그 후 질의 결과 R이 정확한지 확인하기 위해 RootHash 함수로 구한 R_r과 비교한다(행 3). RootHash 함수는 증거의 엔트리가 중첩된 증거인지 트리의 엔트리인지 구분하고(행 3) 중첩된 증거일 때 재귀 호출하여 그 반환 값으로 MBR, 해쉬 값, 집계 값을 갱신하며, 트리의 엔트리일 때 이를 사용하여 MBR, 해쉬 값, 집계 값을 갱신한다. 이때 엔트리가 질의를 만족하면 결과 값을 갱신한다.

```
Function Verification
Input:
      Q: Query
      R: Result
      Π: Proof
Output:
      Accept or Reject
Begin
1.  (MBR_r, H_r, S_r, R_r) = RootHash(Q, π)
2.  If π.sign^{-1} ≠ (π.H[1]| ... |H_r| ... |π.H[m]) Then Return Reject
3.  If R ≠ R_r Then Return Reject
4.  Return Accept
End

Function RootHash
Input:
      Q: Query
      Π: Proof
Output:
      MBR, Hash value, Sum, Result computed with given π
Begin
1.  Set str=∅, mbr=NULL, agg = 0, res = 0
2.  For each entry e in π
3.     If e is an entry in the tree
4.        If e satisfies Q
5.           Update res using e.S[Q.attr]
6.        Update mbr, agg, and str
            using e.MBR, e.S[Q.attr], and e.H[Q.attr]
7.     Else If e is a nested proof
8.        (MBR_c, H_c, S_c, R_c) = RootHash (Q, e)
9.        Update mbr, agg, res, and str
            using MBR_c, S_c, R_c, and H_c
10. Return (mbr, h(mbr|agg|str), agg, res)
End
```

그림 11. 증명 알고리즘의 의사 코드

제 4 장 실험 및 결과

제 1 절 실험 환경 및 데이터

실험에 사용한 머신은 Intel® Core(TM) i3-4150 CPU @ 3.50GHz에 메인 메모리 용량이 12GB이다. 기존의 연구 APS-tree[7]의 성능은 레코드의 개수가 아닌 데이터의 상이 값 수에 영향을 받는다. 이에 비해 AMR-tree는 상이 값 수보다 레코드의 개수에 영향을 많이 받는다. 또한 두 자료구조 모두 차원 크기에 영향을 받는다. 따라서 본 논문에서는 레코드 개수, 상이 값 수, 그리고 차원의 크기의 변화에 따른 성능 차이를 보이기 위해 다음 세 가지의 데이터를 생성하여 실험하였다. (i) 차원의 크기와 레코드 개수를 고정하고 상이 값 수를 변화시킨 데이터. (ii) 차원의 크기와 상이 값 수를 고정하고 레코드 개수를 변화시킨 데이터. (iii) 상이 값 수와 레코드 개수를 고정하고 차원의 크기를 변화시킨 데이터.

구현을 위해 사용한 언어는 C++이며 사용한 공개 키 알고리즘은 RSA[8]을 사용하였으며 해쉬 함수는 SHA256을 사용하였다. 실험 결과는 인증 자료구조 생성 시간, 질의 처리 시간, 증거 크기에 대해 측정하였다. 이때 질의 처리 시간은 서버가 증거를 생성하는 시간과 사용자가 그 증거를 통해 증명하는 데 걸린 시간을 합한 것이다. 실험 결과는 1000개의 질의 처리 결과에 대한 평균 값으로 하였으며 인증 자료구조의 생성시간이 500초가 넘을 경우 실험 결과에서 제외하였다.

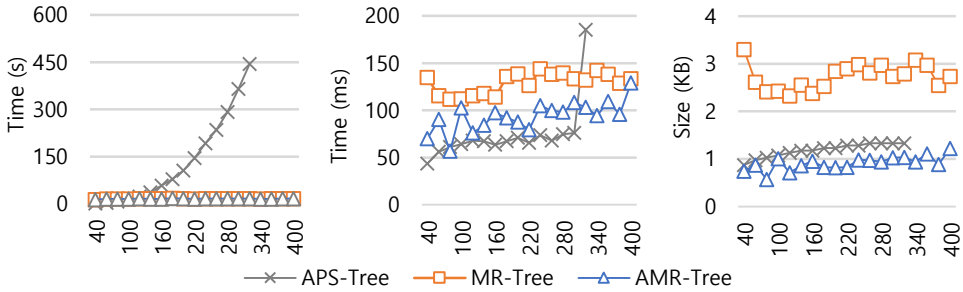
질의 인증 과정은 크게 인증 자료구조 생성 단계, 증거 생성 단계, 증명 단계로 나눠진다. 질의 인증에 있어서 질의 처리는 매우 빈번히 발생되는 반면 인증 자료구조 생성은 데이터를 업로드할 때 단 한번 수행된다. 따라서 질의 처리 시간이 매우 중요하며 네트워크 오버헤드를 줄이기 위해 증거 크기 또한 매우 중요하다. 반면에 인증 자료구조는 데이터를 업로드할 때 단 한번 수행되므로 상대적으로 중요성이 낮다. 그러므로 실험 결과를 분석할 때 질의 처리 시간과 증거 크기에 중점을 두고 성능을 검증한다.

제 2 절 상이 값 수 변화 실험 결과

데이터의 속성 수가 m이고 i 번째 속성의 상이 값 수가 $|V_i|$일 때, 공간 복잡도가 $O(m\prod_{i=1}^{m}|V_i|)$인 APS-tree는 상이 값 수에 영향을 많이 받는 반면 MR-tree와 AMR-tree는 영향을 받지 않는다. 상이 값 수가 증가함에 따른 성능 변화를 보이기 위해 3차원의 1백만개 레코드를 가지는 데이터에 대해 속성별 상이 값 수를 40에서 400까지 20씩 증가시키며 실험을 진행하였다.

그림 12는 위의 데이터를 통해 실험한 결과이며 결과 그래프의 가로 축은 속성별 상이 값 수를 나타낸다. 그림 12 (a)는 자료구조 생성 시간을 나타낸 그래프이다. APS-tree는 공간 복잡도가 $O(m\prod_{i=1}^{m}|V_i|)$이므로 APS-tree의 생성 속도가 상이 값 수에 영향을 많이 받을 것으로 예상되었으며 실험한 결과, 예상한 바와 같이 상이 값 수가 증가함에 따라 생성 시간이 급격하게 증가하는 것을 볼 수 있었다. 특히 상이 값 수가 340 이상에서 생성 시간이 500초를 넘으며 실험 결과에서 제외된 것을 확인할 수 있었다. 그림 12 (b)는 질의 처리 시간을 나타낸 그래프이다. APS-tree의 경우 상이 값 수가 300이하일 때 질의 처리 시간이 서서히 증가하지만 320이상에서 급증하였고 그 이후는 실험결과에서 제외되었다. 반면에 MR-tree와 AMR-tree는 상이 값 수에 대해 일정한 관계를 보이지 않았다. 이는 MR-tree와 AMR-tree의 경우 상이 값 수보다 질의의 선택도에 영향을 많이 받기 때문으로 예상된다. 그림 12 (c)는 증거의 크기를 나타낸 그래프이다. APS-tree의 경우 로그 함수 꼴로 증가하였는데 이는 APS-tree를 생성할 때 전위 합 배열에 대해 MH-tree를 생성하므로 증거의 크기는 MH-tree의 높이에 비례하기 때문이다. 이에 비해 MR-tree와 AMR-tree는 질의 처리 시간의 경우와 마찬가지로 상이 값 수에 대해 일정한 패턴을 보이지 않았으며 질의 선택도에 영향을 받기 때문으로 예상된다.

이 실험에서 사용한 데이터는 상이 값 수가 최대 400개이다. 하지만 실생활에서 수집되는 데이터의 상이 값 수는 이 데이터의 상이 값 수



(a) 인증 자료구조 생성 시간 (b) 질의 처리 시간 (c) 증거 크기

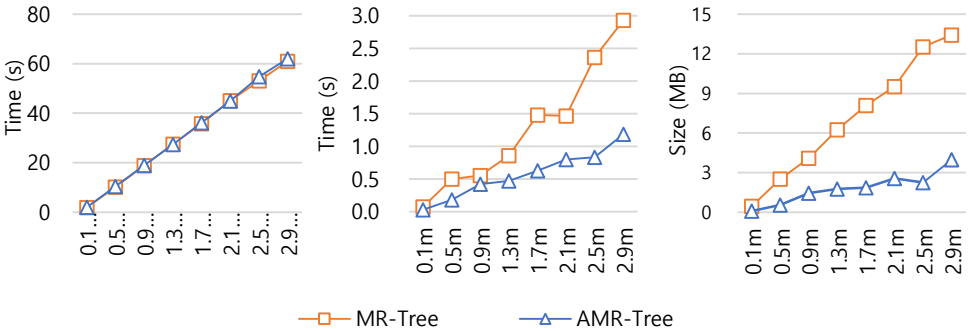
그림 12. 상이 값 수 증가 실험 결과

에 비해 매우 크다. 대형 마트에서 판매하는 개별 물품 수는 1만개에서 10만개이며 유명 온라인 쇼핑몰에서 판매하는 개별 물품 수는 4억개에 이른다. 이렇듯 이 실험에서 사용한 데이터는 실생활 데이터와 거리가 있다. 하지만 상이 값 수가 1만개가 넘는 실생활 데이터에 가까운 데이터에 대해 APS-tree는 매우 비효율적이다. 이로 인해 모든 경우에 APS-tree의 실험 결과는 제외되어 비교가 불가능하였다. 따라서 상이 값 수가 굉장히 작은 데이터에 대해 실험하였으며 상이 값 수가 작은 데이터에 대해서 AMR-tree는 APS-tree와 비슷한 성능을 보였다.

제 3 절 레코드 개수 변화 실험 결과

AMR-tree는 R*-tree의 구조를 확장하므로 레코드의 개수에 영향을 받지만 이에 반해 APS-tree는 데이터 큐브에 대한 전위 합 배열을 사용하므로 레코드 개수에 영향을 받지 않는다. 이 실험에서는 레코드 개수의 변화에 따른 성능 변화를 보이기 위해 5차원의 상이 값 수가 10,000인 데이터에 대해 레코드의 개수를 1십만개부터 2백 9십만개까지 4십만개씩 증가시키며 진행하였다.

그림 13은 레코드를 증가시키며 진행한 실험의 결과를 나타내며 가로 축은 레코드 개수를 나타낸다. APS-tree는 모든 경우에 생성 시간이 500초가 넘어 실험 결과에서 제외되었다. 그림 13 (a)는 인증 자료구조의 생성 시간을 나타내며 AMR-tree의 생성 시간이 MR-tree에 비해 최대 1.1배 느린 것을 확인할 수 있다. 이는 각 속성에 대한 집계 값을 구하고 모든 독립 노드에 서명을 생성하기 때문으로 예상된다. 그림 13 (b)는 질의 시간을 나타내며 레코드 개수가 2백 9십만개일 때 MR-tree가 AMR-tree보다 약 2.5배 느렸으며 그 차이는 레코드 개수가 증가할수록 더욱 커질 것으로 예상된다. 그림 13 (c)는 증거 크기에 대한 실험 결과로 MR-tree가 AMR-tree에 비해 증거 크기가 최대 3.1배만큼 컸으며 레코드 개수가 증가할수록 그 차이가 더 커질 것으로 예상된다. 1절에서 언급한 바와 같이 AMR-tree의 생성 속도가 MR-tree에 비해 느렸지만 AMR-tree를 통한 질의 처리 속도가 MR-tree의 경우보다 매우 빨랐으며 증거 크기 또한 매우 작았기에 집계 질의 인증에 있어 AMR-tree의 성능이 MR-tree보다 우수하였다.



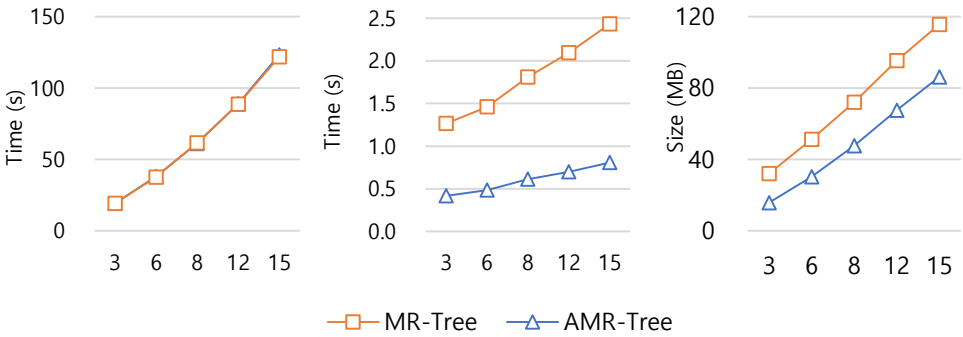
(a) 인증 자료구조 생성 시간 (b) 질의 처리 시간 (c) 증거 크기

그림 13. 레코드 개수 증가 실험

제 4 절 차원 크기 변화 실험 결과

AMR-tree는 각 노드마다 MBR을 가지며 자손 노들의 각 속성별 집계 값을 미리 구해놓기 때문에 차원의 영향을 받을 것으로 예상된다. 따라서 이 실험에서는 1백만개의 레코드를 가지며 상이 값 수가 1만인 데이터에 대해 차원을 3에서 15까지 3씩 증가시키며 진행하였다. 각 차원마다 질의 선택도를 일정하게 하기 위해 질의의 구간 조건을 만족하는 레코드 개수를 9천개에서 1만 1천개가 되도록 한정 지었다.

그림 14은 차원 크기를 증가시키며 실험한 결과를 나타내며 가로축은 차원 크기를 나타낸다. 그림 14 (a)는 차원 크기를 증가시켰을 때 인증 자료구조를 생성하는 데 걸린 시간을 나타낸다. 차원 크기가 증가하면서 AMR-tree와 MR-tree의 생성시간은 일정하게 증가하는 것을 보였으며 둘 사이의 차이는 크지 않았다. 그림 14 (b)는 질의 처리 시간을 나타낸다. 질의 처리 시간 또한 차원이 증가함에 따라 일정하게 증가하였지만 MR-tree를 사용한 질의 처리 시간보다 AMR-tree를 사용하였을 때 질의 처리 시간이 더 적게 걸렸으며 AMR-tree가 최대 3배 빠른 것을 알 수 있다. 그림 14 (c)는 증거의 크기를 차원별로 나타낸다. AMR-tree와 MR-tree 모두 차원이 증가함에 따라 일정하게 증가하였지만 AMR-tree가 MR-tree에 비해 1.3배에서 최대 2배 작은 것을 알 수 있었다. 이를 통해 차원이 증가함에 따라 인증 자료구조 생성 시간, 질의 처리 시간, 그리고 증거 크기가 일정하게 증가하면서 성능이 저하되는 것을 알 수 있었다.



(a) 인증 자료구조 생성 시간

(b) 질의 처리 시간

(c) 증거 크기

그림 14. 차원 크기 증가 실험 결과

제 5 장 결론

본 논문에서는 아웃소싱 데이터베이스 환경에서 신뢰할 수 없는 서버가 처리한 집계 질의 결과가 정확한지 증명하는 인증기법을 제안하였다. 질의 인증 과정에서 증거의 크기를 줄이기 위해 자손 노드의 집계 값을 미리 구하고 이를 활용하는 인증 자료구조인 AMR-tree를 제안하였다. 또한 증거를 생성할 때 불필요한 엔트리가 증거에 포함되는 것을 줄이기 위해 독립 노드와 독립 서브트리 개념을 도입하였으며 전체 트리가 아닌 독립 서브트리에 대해서 증거를 생성하여 증거 크기를 줄이고 증거 생성 속도를 향상시켰다. 그리고 증거를 바탕으로 집계 질의 결과가 정확한지 증명하는 방법을 제안하였다. 실험을 통해 상이 값 수가 작을 때 기존 연구인 APS-tree와 성능이 비슷한 것을 확인하였고, 상이 값 수가 클 때 레코드 개수에 따른 성능 변화와 차원 크기가 증가할 때의 성능 변화를 보였다. 상이 값 수가 클 때 APS-tree는 매우 비효율적이었으며 AMR-tree는 MR-tree에 비해 질의 처리 시간이 적게 걸렸으며 증거 크기가 작았다. 이를 통해 AMR-tree의 성능을 검증하였다.

AMR-tree는 R*-tree의 트리 구조와 동일하기 때문에 노드들의 MBR 크기에 영향을 많이 받기 때문에 R*-tree의 구조가 비효율적으로 생성되었을 때 성능 저하가 발생할 수 있다. 또한 상이 값 수가 적을 때는 APS-tree에 비해 성능이 뛰어나지는 못하였다. 따라서 향후에는 일정한 성능을 보장하며 AMR-tree에 적용 가능한 자료구조를 찾고 이를 확장할 수 있을 것이다. 또한 상이 값 수가 적을 때도 효율적인 인증 기법이 연구되어야 할 것이다.

참고 문헌

[1] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, "Authentic data publication over the internet," Journal of Computer Security 11.3: 291-314, 2003.

[2] RC. Merkle, "A digital signature based on a conventional encryption function," Conference on the Theory and Application of Cryptographic Techniques. Springer Berlin Heidelberg, 1987.

[3] D. Papadopoulos, S. Papadopoulos, and N. Triandopoulos, "Taking authenticated range queries to arbitrary dimensions," Proceedings of the 2014 ACM SIGSAC, 2014.

[4] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Authenticated indexing for outsourced spatial databases," The VLDB Journal: 631-648, 2009.

[5] N. Beckmann, HP. Kriegel, R. Schneider, and B. Seeger, "The R*–tree: an efficient and robust access method for points and rectangles," ACM SIGMOD Record. Vol. 19. No. 2, 1990.

[6] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Authenticated index structures for aggregation queries," ACM TISSEC 13.4: 32, 2010.

[7] CT. Ho, R. Agrawal, N. Megiddo, and R. Srikant, “Range queries in OLAP data cubes,” Vol. 26. No. 2. ACM, 1997.

[8] Rivest, Ronald L., Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM 21.2 (1978): 120-126.

[9] http://www.cryptopp.com

[10] G. Yang, Y. Cai, and Z. Hu, “Authentication of function queries,” IEEE International Conference on Data Engineering, 2016.

[11] M.−L. Yiu, E. Lo, and D. Yung, “Authentication of moving kNN queries,” IEEE International Conference on Data Engineering, 2011.

Abstract

Efficient Authentication of Aggregation Queries for Outsourced Databases

Jongmin Shin
Electrical and Computer Engineering
The Graduate School
Seoul National University

Outsourcing databases is to offload storage and computationally intensive tasks to the third party server. Therefore, data owners are able to manage big data and handle queries from clients without building an infrastructure with high cost. However, due to the insecurity of network systems, the third party server may be untrusted, thus the query results from the server may be tampered. This problem has motivated significant research efforts on authenticating various queries such as range query, kNN query, function query, etc. Even though aggregation queries play an important role in analyzing big data, authenticating aggregation queries has not been extensively studied, and the previous works are not effective for data with high dimension or a large number of distinct values.

In this paper, we propose the AMR-tree that is a data structure to authenticate aggregation queries. We also propose an efficient proof construction method and a verification method with the AMR-tree. Furthermore, we validate the performance of the proposed

algorithm by conducting various experiments through changing parameters such as the number of distinct values, the number of records, and the dimension of data.

Keywords : outsourced databases, query authentication, aggregation query, authenticated data structure
Student Number : 2015-20939