대규모 데이터베이스 시스템에서 인덱스를 이용한 범위 질의 방법

김치연*

A Range Query Method using Index in Large-scale Database Systems

Chi-Yeon Kim*

요 약

최근 데이터 양이 폭발적으로 증가함에 따라, 데이터를 저장하고 검색하고 다루기 위한 대규모 데이터베이스 시스템이 등장하였다. 이 환경에서는 일관성과 가용성, 결함 허용 등 다양한 이슈가 존재한다. 이 논문에서는 데이터 관리와 트랜잭션 관리가 분리된 구조를 갖는 대규모 데이터베이스 시스템에서, 효율적인 범위 질의 방법에 대하여 다룬다. 동일한 구조에서 두 모듈의 독립성을 보장하고, 팬텀 문제를 해결하기 위하여 파티션을 이용한 범위 질의 방법에 대한 연구가 있었지만, 범위 질의가 키 값으로 명세되는 경우에만 효율적이었다. 이에 이 논문에서는 키 값이 아닌 다른 속성으로 범위 질의가 주어질 때 효율을 개선할 수 있는 방법을 제안하고자 한다. 제안하는 방법에서는 분리된 두 모듈의 독립성은 보장하며, 부분 인덱스를 사용함으로써 범위 질의를 위한 오버헤드를 줄일 수 있다.

ABSTRACT

As the amount of data increases explosively, a large scale database system is emerged to store, retrieve and manipulate it. There are several issues in this environments such as, consistency, availability and fault tolerance. In this paper, we address a efficient range-query method where data management services are separated from transaction management services in large-scale database systems. A study had been proposed using partitions to protect independence of two modules and to resolve the phantom problem, but this method was efficient only when range-query is specified by a key. So, we present a new method that can improve the efficiency when range-query is specified by a key attribute as well as other attributes. The presented method can guarantee the independence of separated modules and alleviate overheads for range-query using partial index.

키워드

Large-scale Database Systems, Range query, Phantom problem, Partition
대규모 데이터베이스 시스템, 범위 질의, 팬텀 문제, 파티션

Ⅰ. 서 론

전통적인 데이터베이스 시스템은 새로운 응용의 등장에 따라 함께 변화하고 있다. 최근의 응용에서 두드러진 특징은 데이터 양의 폭발적인 증가이다. 전통적인 분산 데이터베이스 시스템에서 부족한 처리 속도와 확장성을 지원하기 위해 병렬 데이터베이스 시스템이 출현하였는데, 데이터 양이 증가하면서 하나의

* 목표해양대학교 해양컴퓨터공학과(gegujang2@mmu.ac.kr)
접수일자 : 2012. 08. 22 심사(수정)일자 : 2012. 09. 10 게재확정일자 : 2012. 10. 05

데이터베이스는 하나의 물리적인 시스템에 저장되기 어렵고, 확장성과 성능, 유연성 등의 새로운 특징을 수용할 필요가 발생하게 되었다[1]. 병렬 데이터베이스 시스템이 대용량의 데이터를 처리하기 위한 기반은 만들었지만 엄청난 데이터 용량과 글로벌한 사용자 분포, 그리고 네트워크를 통한 데이터 공유 등의 요구는 해결할 수 없었다. 결국 가용성과 확장성, 그리고 견고함을 보장할 수 있는 새로운 패러다임을 필요로 하게 되었는데, 그것이 바로 대규모 데이터베이스 시스템이다[1]. 이렇게 등장한 대규모 데이터베이스 시스템에서는 이전의 환경과 비교할 수 없을 정도의 초대용량의 데이터를 저장하고 다루며, 이전의 기술과 다른 관련 기술들을 필요로 하고 있다.

대규모 데이터베이스 시스템은 전 세계적인 사용자를 가진 Amazon, Google, 그리고 Facebook과 같은 다양한 응용에서 사용되고 있으며, 업체들은 각 응용의 목적에 맞는 고유한 스토리지 시스템을 개발하였다. 그 결과로, Amazon의 Dynamo[2], Google의 Bigtable[3], Megastore[4], Facebook의 Cassandra[5] 등 다양한 스토리지 시스템이 개발되었다.

대규모 데이터베이스 시스템에서 다루어야 할 다양한 이슈 중, 이 논문에서는 범위 질의 방법에 대하여 다루고자 한다. 범위 질의는 어떤 속성의 하위 값과 상위 값 사이에 있는 모든 레코드들을 검색하는 연산으로[6], 관계형 데이터베이스에서는 술어(predicate)에 기반을 두고 여러 개의 튜플을 접근하는 것을 허용한다[7]. 술어는 하나의 키나 범위를 갖는 여러 개의 키로 명세될 수 있고, 키 값에 의한 범위 질의는 일반적으로 레코드들이 기본키나 이차키에 의해 B-트리 구조로 클러스터링 되어 있을 때 효율적이다[7]. 범위 질의의 예로, 특정 아이디 범위에 있는 직원들의 월급을 갱신하거나 최근 며칠 사이에 갱신된 이미지를 검색하는 예를 찾을 수 있다.

대규모 데이터베이스 시스템의 하나인 클라우드 환경에서 범위 질의를 다루기 위한 몇몇 연구들이 제안되었다[7][8]. 그 중 잠금(locking)에 기반을 둔 범위 질의 알고리즘을 제안한 연구에서는 트랜잭션 관리와 데이터 관리가 분리된 구조를 사용하고 있다[7]. 이 연구는 분리된 구조로 인하여 유연성을 제공하고, 전통적인 DBMS 기능을 클라우드 환경에 쉽게 적용할 수 있다는 장점이 있다. 하지만 이 방법에서는 분리된 두 모듈의 독립성을 보장하기 위하여 기존에 사용되던 B-트리 기반의 방법을 사용할 수 없고, 키가 아닌 다른 속성 값에 의해 주어진 범위 질의에 대해서는 테이블 전체를 검색해야만 결과를 얻을 수 있다는 문제점이 있다. 범위 질의는 늘 키 값에 의해서만 주어지는 것은 아니다. 예를 들어, Facebook의 이미지 저장 스토리지 시스템인 Haystack[9]의 경우, 키는 64비트 이미지 아이디이다. 사용자가 이미지를 검색하거나 갱신할 때 늘 이미지 아이디를 사용하지는 않는다. 검색 횟수나 갱신 날짜 등으로 검색하는 일은 특별하지 않기 때문에, 키 값이 아닌 속성에 의한 범위 질의를 다룰 필요가 있다.

이 논문에서는 분리된 구조가 갖는 장점을 최대한 활용하기 위하여 그림 1과 같은 분리된 구조를 가정하고, 키가 아닌 속성에 대해서도 범위 질의를 효율적으로 수행할 수 있는 방법에 대하여 제안하고자 한다. 제안한 범위 질의 방법은 파티션 단위의 잠금을 사용하며, 각 파티션에 대해 부분(partial) 인덱스를 구축함으로써 키가 아닌 속성에 대한 범위 질의를 효율적으로 수행한다. 팬텀 방지를 위해 인텐션 잠금 모드를 사용하고, 분리된 모듈 사이의 추가적인 메시지 교환이 없어서 범위 질의에 관련된 오버헤드를 최소화할 수 있다.

이 논문의 구성은 다음과 같다. 2장에서는 범위 질의에 관련된 개념과 제안한 연구들을 살펴보고, 3장에서는 인덱스를 이용한 새로운 범위 질의 기법을 제안한다. 4장에서는 제안하는 방법의 분석을 다루며, 5장에서는 결론을 기술한다.

II. 관련 연구

이 장에서는 범위 질의를 다룰 때 발생하는 팬텀 문제와 잠금을 이용하여 범위 질의를 수행할 때 사용되는 잠금 모드, 그리고 파티션에 대하여 기술한다.

2.1 팬텀의 발생

범위 질의는 범위에 포함된 여러 개의 튜플을 접근하기 때문에, 선택된 튜플의 집합이 트랜잭션이 완료될 때까지 변경되지 않을 것을 요구한다. 선택된 튜플 집합에 대한 처리는 잠금과 같은 동시성 제어 메커니

즘에 의해 구현되며, 트랜잭션 수행 도중 다른 충돌 관계의 트랜잭션이 데이터를 접근하지 못하도록 막는다[10]. 범위 질의를 수행할 때 선택된 키의 범위는 트랜잭션이 끝날 때까지 변경되지 않아야 하며, 범위 안에 새로운 레코드가 삽입되는 것도 막아야 한다[7]. 트랜잭션 수행 도중 충돌을 피해 삽입된 레코드를 "팬텀"이라 한다. 예를 들어, 아래의 질의에서 트랜잭션 T는 [10, 40] 범위에 해당하는 id를 가진 회사원들의 봉급을 갱신하길 원한다. 이 때 T는 이 범위에 해당하는 모든 레코드에 배타적 잠금을 설정해야 한다. 이 경우에, 트랜잭션 T′이 T가 종료되기 전에 [10, 40] 사이에 새로운 레코드를 삽입한다면 그것은 팬텀이 된다.

```
UPDATE Employee SET salary = salary + 2000
WHERE id >= 10 AND id <= 40;
```

범위 질의 수행시에 발생하는 팬텀 문제를 해결하기 위해서는 범위 안에 포함된 레코드를 접근할 때 충돌 관계의 연산을 허용하지 않아야 한다.

2.2 잠금 모드

일반적으로 잠금을 이용한 동시성 제어 방법에서 사용하는 잠금 모드는 공유(**S**hared) 잠금과 배타적(e**X**clusive) 잠금이다. 공유 잠금은 데이터를 읽기 위한 잠금으로 다른 공유 잠금과 양립가능하다. 배타적 잠금은 데이터를 갱신하기 위한 잠금으로 어떤 잠금과도 양립되지 않는다. 범위 질의를 수행할 때는 두 가지 잠금 모드 외에 인텐션(intention) 잠금 모드를 사용한다. 잠금이 설정되는 리소스의 계층구조를 높은 레벨부터 데이터베이스, 테이블, 페이지, 그리고 레코드로 분류할 때, 하위 레벨의 리소스에 공유 잠금이나 배타적 잠금을 설정하기 위해 그보다 상위 리소스에 인텐션 잠금을 설정한다. 예를 들어, 테이블의 한 레코드에 공유 잠금을 설정하기 위해서는, 먼저 테이블 전체에 공유 인텐션 잠금(IS)을 설정한다. 인텐션 잠금의 목적은 인텐션 잠금이 설정된 리소스(예를 들어, 테이블)보다 낮은 레벨의 리소스(예를 들어, 레코드)에서 더 세부적인(finer) 잠금이 설정될 수 있음을 알리기 위해서이다. 이렇게 함으로써 충돌 관계에 있는 잠금이 동시에 설정되는 것을 막는다[10]. 범위 잠금을 위해 MGL(Multi-Granularity Locking)을 사용할 수 있다[10]. 표 1은 MGL에서 사용되는 잠금 양립성 테이블이다. 여기서 X 표시는 충돌이 없어 잠금이 수여될 수 있음을 의미한다. IS나 IX 모드는 하위 리소스에서 S나 X 잠금을 설정할 때 사용한다. SIX는 스캔을 위한 모드로, 범위에 포함된 전체 레코드에 S 잠금이 설정되고, 개별 레코드에는 X 잠금을 설정하기 위한 모드이다.

표 1. 잠금 양립성 테이블
Table 1. Lock compatibility table

Lock Mode	IS	IX	S	SIX	X
IS	x	x	x	x	
IX	x	x			
S	x		x		
SIX	x				
X					

2.3 파티션

파티션은 데이터베이스 분야에서 오랫동안 사용되어온 기술이다. 중앙 집중 시스템에서는 하나의 파일이 하나의 디스크에 저장될 수 없을 정도로 크거나, 파일 접근 횟수가 하나의 디스크에 의해 지원될 수 없을 정도로 많을 때 파티션을 사용했다[11]. 이후, 병렬 데이터베이스 시스템에서는 병렬성을 높이기 위한 방법으로 파티션을 이용했고, 파티셔닝 전략으로 범위 파티셔닝(range partitioning)과 라운드-로빈 파티셔닝, 그리고 해쉬 함수를 이용한 방법이 있다[11]. 이 중 범위 파티셔닝은 연관된 데이터들이 하나의 물리적 스토리지에 저장되도록 클러스터링 함으로써 좋은 성능을 보일 수 있는 반면, 편향(skew)의 문제가 발생할 수 있다. 파티션을 구축하기 위한 파티셔닝 방법으로는 고정 파티션과 동적 파티션 방법이 있다[7]. 고정 파티션은 일정한 레코드의 범위로 파티션을 할당하는 방법이고, 동적 파티션은 공간과 데이터를 기준으로 레코드의 밀도를 맞추는 방법이다. 고정 파티션은 한번 결정되면 바뀌지 않기 때문에 단순하지만 레코드의 삽입, 삭제가 진행되다보면 파티션내의 레코드 밀도가 매우 불균형하게 된다. 동적 파티션은 파티션 사이의 균형은 좋지만 균형을 위한 잦은 병합과 분리가 발생한다는 단점이 있다.

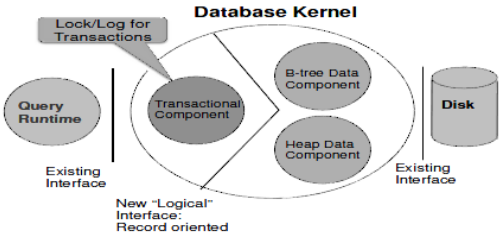

그림 1. 분리된 구조를 가진
Fig. 1 Unbundled architecture

클라우드 환경에서 범위 질의를 수행하기 위해 파티션을 이용하기 위해 그림 1과 같이 트랜잭션 관리(TC)와 데이터 관리(DC)를 위한 모듈이 분리된 환경에서 범위 질의 방법을 제안한 방법이 있다[7]. 이 연구에서는 범위에 포함되는 다수의 레코드에 잠금을 설정할 때 팬텀 레코드를 방지하고, 범위를 인식하기 위하여 speculative visit과 table locking, 그리고 partition locking을 이용한 방법을 제안하였다. 여기에서 파티션은 레코드보다는 크고 테이블보다는 작은 범위의 잠금 리소스로, 동시성은 향상시키고 잠금 오버헤드는 감소시킬 수 있는 논리적인 단위이다. 파티션 잠금을 이용하여 두 모듈의 독립성을 해치지 않으면서도 범위 질의를 효율적으로 해결할 수 있음을 보였다. 하지만 이 연구에서 제안한 범위 질의 방법은 범위가 키 값에 의해 주어진 경우에만 효율적이어서, 키 값이 아닌 다른 속성의 범위로 주어진 질의는 모든 리소스를 접근해야만 결과를 얻을 수 있다. 일반적으로 범위 질의는 반드시 키 값에 의해서만 주어지는 것이 아니므로, 키 값이 아닌 다른 속성에 이용한 범위 질의에 대해서도 고려되어야 하고, 이 논문에서는 그 문제를 집중적으로 다루고자 한다.

III. 인덱스를 이용한 범위 질의

이 장에서는 그림 1과 같이 트랜잭션 관리와 데이터 관리가 분리된 구조를 기반으로 범위 질의를 효율적으로 수행할 수 있는 방법에 대하여 기술한다.

3.1 문제 정의

그림 1의 TC 모듈에서 유지되는 Employee 테이블과 파티션이 아래 그림 2와 같은 경우를 가정한다. Employee 테이블에는 아이디 1~30번의 레코드가 저장되어 있고, 그 중 10개씩의 레코드가 고정 파티션 방법으로 파티션 p1, p2, p3에 각각 할당되어 있다. 이 상태에서 아래의 범위 질의가 발생하였다. (테이블의 키는 아이디이고, 아이디 5에서 15번 사이에 있는 직원들의 월급을 10% 인상한다.)

UPDATE Employee SET salary=salary+salary*0.1
WHERE id >=5 AND id <=15;

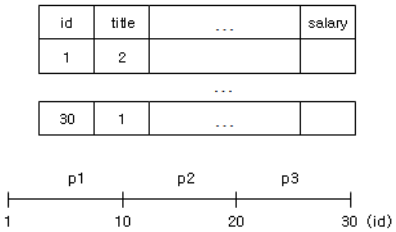

그림 2. Employee 테이블과 파티션의 예
Fig. 2 Example of employee table and its partition

이를 위해 범위가 포함된 파티션 p1, p2에 범위 질의를 수행하기 위한 잠금이 설정될 것이고, 갱신을 수행한 다음, 결과가 DC에 전달되어 최종 데이터는 갱신된다. 하지만 다음과 같은 질의를 가정해보자. (직급이 2번(대리)부터 4번(과장)인 직원의 월급을 10% 인상한다.)

UPDATE Employee SET salary=salary+salary*0.1
WHERE title >= 2 AND title <=4;

이 질의는 모든 아이디에 대하여 직급 번호가 2번에서 4번 사이이면 월급을 10% 인상하는 질의이다. 따라서 모든 테이블의 레코드가 질의의 범위가 되고, 전체 테이블에 범위 잠금을 설정해야 수행가능하다. 테이블 단위의 잠금은 간단하지만 잠금 단위가 커서 동시성이 심각하게 저하된다. 특히, 배타적 잠금의 경우에는 다른 잠금을 허용할 수 없게 된다. 이러한 문제는 클라우드 환경에서 자주 발생될 것으로 예상된다. 페이스 북의 사진 이미지를 저장하는 Haystack의 경우, 키는 64비트의 photo id이다[9]. 하지만 사용자들은 사진의 id가 아니라 제목이나 검색 횟수, 갱신 날짜 등으로 질의를 요청하는 경우가 많다. 때문에 키 값으로만 수행할 수 있는 범위 질의는 효율이 매우

낮아질 수 있다.

3.2 부분 인덱스의 적용

파티션을 이용하여 범위 질의를 수행한 방법의 문제는 분리된 두 모듈의 독립성으로 인하여 잦은 메시지 교환이 어렵다는 점이다. 즉, TC는 실제 저장된 데이터의 논리적 정보만 가질 뿐, 물리적 정보에 대해서는 모르기 때문에 B-tree 인덱스를 구성할 수 없다. 그래서 테이블의 논리적인 정보로만 파티션을 구성하여 범위 잠금에 활용하였는데, 파티션을 구성하는 기준이 키 값이기 때문에 키가 아닌 범위 질의에 대해서는 전체 레코드를 접근해야 하는 문제가 발생한다. 이 문제를 해결하기 위하여 TC가 물리적인 인덱스를 구축하게 되면 두 모듈 사이의 많은 메시지 교환이 필요하고, 두 모듈을 분리한 장점을 잃게 된다. 따라서 두 모듈의 독립성을 보장하면서 범위 질의의 효율을 높이는 방법이 필요하다.

이 논문에서는 이를 위해, 파티션과 인덱스를 조합한 방법을 도입하고자 한다. 먼저, TC는 테이블의 논리적인 정보를 이용하여 파티션을 구축한다. 파티션 구축 전략으로 고정 파티션, 동적 파티션 어느 쪽을 사용하는지는 상관없다. 두 방법 모두 키 값을 기준으로 파티션이 구축되기 때문이다. 그리고 구축된 각 파티션에 인덱스를 구축한다. 여기서 인덱스는 파티션에 대한 인덱스이기 때문에 “부분(partial) 인덱스”로 정의한다. 부분 인덱스는 파티션이 구성될 때 구성되거나, 추가적으로 필요하다고 판단될 때 TC가 추가할 수 있다. 부분 인덱스의 구성은 파티션과 독립적이다. 어떤 속성으로 인덱스를 구축할 것인지는 질의 최적기를 통해 수집된 정보를 이용할 수 있다. 예를 들어, 테이블의 키가 직원의 아이디라면 파티션은 아이디를 기준으로 구성되고, 직급에 따른 범위 질의가 많다면 직급을 기준으로 부분 인덱스를 구축할 수 있다. 원래 레코드에 쉽게 접근하기 위해서 각 인덱스는 파티션에 있는 레코드를 가리키는 포인터 필드를 갖는다. 따라서 이 방법에서는 부분 인덱스를 위한 추가적인 메모리가 필요하다.

3.3 예제 시나리오

3.1절의 Employee 테이블에 구축된 파티션 p1과 직급 번호로 구축된 부분 인덱스(inx_title)가 그림 3과 같을 때, 다음 질의를 고려한다.

UPDATE Employee SET salary=salary+salary*0.1
WHERE title >= 2 AND title <=4;

부분 인덱스가 없다면 범위 질의를 수행하기 위하여 파티션 안의 모든 레코드를 검색해야 한다. 직급 번호 조건을 만족하는 레코드에 정해진 순서가 없기 때문이다. 예에서는 파티션 p1만 보여주고 있지만, 부분 인덱스가 없다면 레코드가 범위를 만족하는지 검사하기 위해 전체 레코드를 접근해야 한다. 최악의 경우 파티션 안에 범위를 만족하는 레코드가 하나도 없을 수도 있다. 이런 파티션을 empty 파티션이라 하는데[7], 부분 인덱스가 없다면 empty 파티션에 대해서도 모든 레코드 검색이 필요하다. 하지만 인덱스가 있다면 불필요한 레코드 검색은 발생하지 않는다.

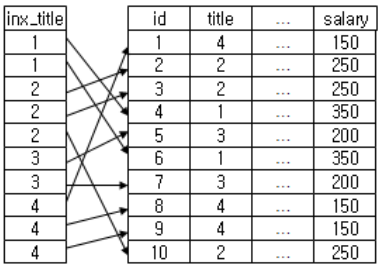

그림 3. 부분 인덱스의 예
Fig. 3 Example of partial index

부분 인덱스를 사용함으로써 범위 질의에서 얻을 수 있는 장점은 범위 잠금을 설정하기 위한 불필요한 레코드 검색을 없앰으로써 트랜잭션의 응답 시간을 향상시킬 수 있다는 점이다. 키 값이 아닌 속성으로 범위 질의가 주어지면 인덱스를 통해 범위를 찾고 해당 레코드만 접근함으로써 불필요한 접근을 줄일 수 있다. 키 값 속성으로 주어진 범위 질의는 원래의 방법인 파티션 잠금을 이용하여 해결할 수 있다. 트랜잭션의 응답 시간의 향상은 그만큼 잠금이 설정된 시간을 줄일 수 있어서 결국 동시성 향상으로 이어질 수 있다.

Ⅳ. 분 석

이 장에서는 제안한 방법의 성능 분석을 위해 매개

변수를 이용한 분석적 방법을 사용한다. 분석에 사용된 매개변수는 표 2와 같다.

4.1 트랜잭션 응답 시간

트랜잭션 응답 시간의 대부분은 레코드를 읽는 시간과 잠금 설정 시간이 차지한다. 트랜잭션의 응답 시간이 짧을수록 잠금을 빨리 해제할 수 있기 때문에 동시성이 향상될 수 있다. 범위 질의를 위하여 파티션만을 이용한 방법과 제안하는 방법을 비교하며, 하나의 파티션을 기준으로 한다. 비교 대상은 키 값이 아닌 범위 질의의 경우이다.

범위 질의를 위하여 파티션만 이용한 방법에서 키 값이 아닌 속성으로 주어진 범위 질의를 위해서는 파티션 전체의 레코드를 접근하여 해당 속성 값을 비교한 후, 범위에 해당하는 레코드에 잠금을 설정하고 읽는 시간이 필요하다. 따라서 다음 (식 1)과 같이 잠금 설정 시간이 구해진다.

$$(N_R - N_Q)t_r + N_Q(t_r + t_l) \quad \text{(식 1)}$$

하지만 부분 인덱스를 사용한다면 인덱스 접근 후 인덱스 범위에 해당하는 레코드들에 대해서만 잠금을 설정하고 읽는 시간이 필요하다. 따라서 다음 (식 2)와 같이 잠금 설정 시간이 구해진다.

$$t_i + N_Q(t_r + t_l) \quad \text{(식 2)}$$

일반적으로 레코드를 접근하는 시간보다 인덱스를 접근하는 시간이 짧고, 상수 시간으로 가정할 수 있기 때문에 (식 2)의 시간이 훨씬 더 짧다. 두 식은 하나의 파티션에 필요한 시간이기 때문에, 테이블 내의 전체 파티션에 적용한다면 차이는 더 커질 수 있다. 특히, N_Q가 0인 empty 파티션의 경우, (식 1)은 파티션의 모든 레코드를 다 검색해야 하지만, (식 2)는 인덱스를 접근하는 시간만 필요하게 되어 차이를 명확히 알 수 있다.

4.2 메모리 오버헤드

제안하는 방법에서는 파티션에 대한 부분 인덱스를 추가로 유지해야 하기 때문에 오버헤드가 될 수 있다. 20 페타 바이트 이상의 이미지를 저장하고 있는 Haystack의 경우, 이미지를 저장할 때 필요한 필드는 헤더, 쿠키, 키, 대체키, 플래그, 크기, 데이터, 풋터, 체크섬 등이다[9]. 이 중 범위 질의에 사용가능한 필드로는 크기와 이미지에 대한 설명인 데이터 정도이다. 다른 응용이라 할지라도 부분 인덱스를 구성하는 타입은 문자열, 날짜, 숫자 타입 중 하나가 될 것으로 예상가능하다. 논문에서 사용한 부분 인덱스는 두 개의 필드로 구성되는데 하나는 질의가 가능한 타입의 인덱스 필드와 파티션 내(2~2048)에서 레코드 위치를 가리키는 포인터이다. 인덱스 전체가 차지하는 용량을 정확히 계산하기는 어렵지만 테이블 전체에 대한 인덱스가 아니라 파티션에 대한 인덱스이므로 관계형 데이터베이스에서 필요로 하는 인덱스보다는 작은 용량을 차지할 것으로 예상할 수 있다.

표 2. 성능 분석을 위한 매개변수들
Table 2. Parameters for performance analysis

약어	내 용
R	$[k_l, k_u]$ 사이의 요구된 질의 범위
N_R	범위 R 내의 레코드 수
N_Q	범위 질의를 만족하는 레코드 수
t_r	DC로부터 하나의 레코드를 읽는 시간
t_l	하나의 레코드에 잠금 설정하는 시간
t_i	부분 인덱스 접근 시간

Ⅴ. 결 론

이 논문에서는 대규모 데이터베이스 시스템에서 발생할 수 있는 범위 질의에 대하여 다루었다. Facebook과 같은 응용과 더불어 대규모 데이터베이스는 갈수록 그 응용 범위가 늘어나고 있다.

범위 질의를 다룬 기존의 연구의 문제점은 키 값에 의한 범위 질의만 효율적으로 수행될 뿐, 키가 아닌 속성에 의한 범위 질의는 전체 테이블을 대상으로 잠금을 설정해야 원하는 레코드를 찾을 수 있기 때문에 처리 속도와 동시성이 저하되는 문제가 있었다. 이를 해결하기 위해 트랜잭션 관리 모듈에서 만들어진 파티션에 부분 인덱스를 추가함으로써 잠금이 설정되는 시간을 최소화하고 불필요한 레코드를 접근하지 않도록 함으로써 잠금 오버헤드를 감소시켰다.

제안하는 방법은 추가로 부분 인덱스를 유지하기 위한 공간을 필요로 하므로 오버헤드가 발생한다. 제

안한 사용한 부분 인덱스는 질의가 가능한 타입을 가리키는 인덱스 필드와 파티션 내에서 레코드 위치를 가리키는 포인터로 구성된다. 인덱스는 파티션 내의 튜플들을 구별하기 위해 사용되므로 시스템에 큰 부담은 되지 않을 것으로 판단된다.

이 연구 결과에 추가하여, 범위 질의의 성능이 파티션 구축 방법에 많은 영향을 받는 만큼, 대규모 데이터베이스 환경에 효율적으로 적용할 수 있는 파티셔닝 방법에 대한 연구를 진행하고자 한다.

참고 문헌

[1]So Youn Lee, Esteban Meneses, "Survey of Large-scale Database Systems," [Online] https://wiki.engr.illinois.edu/download/attachments/197298696/survey.pdf

[2] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. “Dynamo: amazon’s highly available key-value store,“ SIGOPS Oper. Syst. Rev., Vol. 41, pp. 205 – 220, Oct. 2007.

[3] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data,” In Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementatio, Vol. 7, pp. 205-218, 2006.

[4] Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. "Megastore: Providing scalable, highly available storage for interactive services," In Proc. CIDR, pp. 223-234, 2011.

[5] A. Lakshman and P. Malik, “"Cassandra: a decentralized structured storage system,” Operating Systems Review, Vol. 44, No. 2, pp. 35 – 40, 2010.

[6] http://en.wikipedia.org/wiki/Range_query

[7] D. Lomet, M. F. Mokbel, "Locking Key Ranges with Unbundled Transaction Services," Proceedings of the VLDB Vol. 2, pp. 265-276, August 2009.

[8] S. Wu, D. Jiang, B. C. Ooi, and K. L. W. "Towards elastic transactional cloud storage with range query support," In Int’l Conference on Very Large Data Bases (VLDB), Vol. 3, pp. 506-514, 2010.

[9] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, “"Finding a needle in haystack: Facebook’s photo storage,” in OSDI, pp. 47 – 60, 2010.

[10] David B. Lomet, "Key range locking strategies for improved concurrency," In VLDB ’93: Proceedings of the 19th international conference on Very Large Data Bases, pp. 655 – 664, 1993.

[11] D. DeWitt and J. Gray, “"Parallel database systems: the future of high performance database systems,”" Commun. ACM, Vol. 35, pp. 85-98, June 1992.

저자 소개

김치연(Chi-Yeon Kim)

1992년 전남대학교 전산통계학과 졸업 (이학사)

1994년 전남대학교 대학원 전산통계학과 졸업(이학석사)

1999년 전남대학교 대학원 전산통계학과 졸업(이학박사)

2002년~현재 목포해양대학교 해양컴퓨터공학과 교수

※ 관심분야 : 트랜잭션 관리, 클라우드 컴퓨팅