MongoDB에서 B-트리 인덱스와 Fractal 트리 인덱스를 이용한 성능 비교

장성호* · 김수희*

*호서대학교

Performance Comparisons on MongoDB with B-Tree Indexes and Fractal Tree Indexes

Seongho Jang* · Suhee Kim*

*Department of Computer Engineering, Hoseo University

E-mail : j2000moom@nate.com, shkim@hoseo.edu

요 약

빅데이터가 다양한 가치를 만들어내기 시작하면서, 더 다양하면서도 막대한 량의 데이터를 수용할 수 있는 데이터베이스가 필요하게 되었다. 그래서 기존 RDBMS의 복잡도와 용량 한계를 극복하기 위한 목적으로 NoSQL 데이터베이스가 등장하게 되었고, 그 중 대표적으로 MongoDB가 많이 사용되며, 오픈 소스로 제공되고 있다.

MongoDB에서 사용되는 B-트리 인덱스는 데이터양이 증가함에 따라 그 성능이 현저히 떨어진다. Fractal 트리 인덱스는 B-트리의 삽입 알고리즘을 개선하여 상당한 성능향상을 가능하게 한다.

이 논문에서는 MongoDB에서 B-트리 인덱스를 사용하는 경우와 Fractal 트리 인덱스를 사용하는 경우를 구별하여 그 성능을 비교해 본다.

ABSTRACT

As Big data began to produce a variety of values, a database that allows for huge amount of data with varieties became to be needed. Therefore, for the purpose of overcoming the limitations of the complexity and capacity of the existing RDBMS, NoSQL databases were introduced. Among the different types of NoSQL databases, MongoDB is most commonly used and is offered as open sources.

The B-Tree index, used in MongoDB, experiences a significant decrease in performance as the amount of data increases. The fractal tree index enables to enhance the performance of B-Tree substantially by improving B-Tree’s insertion algorithm. In this paper, the performances of MongoDB when using B-Tree Index and when using Fractal Tree Index are compared..

키워드

MongoDB, Index, B-Tree, Fractal Tree, Performance

Ⅰ. 서 론

사람들은 매일 250경 바이트의 데이터를 생성하고 있으며, 지난 2년 동안에만 현재 전 세계에 존재하는 데이터의 90%에 달하는 방대한 데이터들이 생성되었다. 이렇게 기존 데이터베이스 관리 도구로 데이터를 수집, 저장, 관리, 분석할 수 있는 역량을 넘어서는 대량의 정형 또는 비정형 데이터들을 관리하기 위한 기술로 NoSQL이 등장했다. NoSQL 중에서도 대표적으로 MongoDB가 많이 사용되는데, 사용과 운영이 다른 시스템에 비해서 매우 쉽고, JSON 기반의 Document를 제공하기 때문에, 데이터 구조에 대한 이해 및 사용이 쉽다. MongoDB에서는 B-트리 인덱스를 사용한다. B-트리 인덱스 구조는 데이터양이 증가할수록 연산 속도가 현저히 떨어지게 되는데, MongoDB의

B-트리 인덱스를 Fractal 트리 인덱스로 대치하여 그 성능을 현저하게 개선한 사례를 발표하였다[1]. 이 논문에서는 MongoDB에서 B-트리 인덱스를 사용하는 경우와 Fractal 트리 인덱스를 사용하는 경우를 구별하여 그 성능을 실험을 통하여 다시 비교 분석해 보고자 한다.

Ⅱ. B-트리 인덱스와 Fractal 트리 인덱스

1. B-트리 인덱스

B-트리는 Bayer & McCreight가 제안하였으며, 인덱스를 조직하는 방법으로 가장 널리 사용되고 있다[2]. Douglas Comer는 1979년에 발표한 그의 논문에서 "B-트리는 사실상 데이터베이스 시스템의 인덱스를 위한 표준 구조다" 라고 했다[3].

차수 m인 B-트리는 m-원 탐색 트리로서 다음과 같은 특성을 가지고 있다.

① 루트와 리프를 제외한 내부 노드는 최소 m/2, 최대 m개의 서브트리를 갖는다.
② 루트는 리프가 아니면 적어도 두 개의 서브트리를 가진다.
③ 모든 리프는 같은 레벨에 있다.
④ 리프가 아닌 노드의 키 값을 수는 그 노드의 서브트리 수보다 하나 적으며, 각 리프 노드는 최소 m/2-1개, 최대 m-1개의 키 값을 가진다.

B-트리의 장점으로는 데이터의 삽입과 삭제 후에 항상 균형을 유지하며 각 노드의 절반 이상이 키 값을 저장하므로 저장 장치의 효율성이 있다. 데이터의 크기 N, 블록의 크기 B, 하나의 노드에 대한 자식들의 수 $\theta(B)$일 때, B-트리는 탐색, 삽입, 수정, 삭제에서 $O(\log_B N)$ 블록의 접근으로 수행할 수 있으므로 모든 연산에서 상당히 좋은 성능을 가지고 있다. 그러나 데이터의 크기가 100TB 급 이상으로 많아지며, 데이터가 초기 상태가 아니고 시간이 상당히 지나 빈번한 삽입, 수정, 삭제가 일어난 후에 B-트리 인덱스는 삽입이나 범위검색에서 반드시 좋은 성능을 보장할 수가 없다. 많은 연속적인 데이터들이 물리적으로 인접해 있는 것이 아니라 비연속적으로 여러 블록에 걸쳐 산재하기 때문이다.

2. Fractal 트리 인덱스

Fractal-트리 인덱스는 B-트리처럼 동일한 연산들을 구현한다[4]. Fractal-트리 인덱스는 효율적으로 작은 크기의 빈번한 write 연산들을 더 크지만, 더 적은 횟수의 write 연산들로 대치한다. 이 점이 더 좋은 압축과 더 좋은 삽입 성능을 가능하게 한다. Fractal-트리는 Cache Oblivious Algorithms에 기반하고 있다[5]. Cache Oblivious Algorithms 모델에서는 알고리듬의 성능은 디스크로부터 캐시로 블록 전송수로 측정된다.

이제 Fractal-트리 인덱스를 설명하기 위해 데이터 크기로 N개의 행들이 있다고 가정하자. 이 경우 Fractal-트리 인덱스는 $\log_2 N$개의 배열을 준비한다. 각 배열의 크기는 2의 지수승이다. 그리고 각 배열은 데이터로 완전히 차 있거나 완전히 비어 있다. 그리고 각 배열은 정렬된다.

만약 N=4 이면, Fractal-트리의 구조는 다음과 같은 형태이다. A1={ }, A2={ }, A3={12,17,23,30} 만약 N=10이면 Fractal-트리의 구조는 A1={ }, A2={5,10}, A3={ }, A4={3,6,8,12,17,23,26,30}와 같은 형태이다.

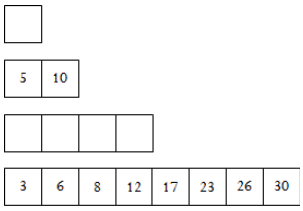

그림 1. Fractal 트리의 예

- Fractal 트리의 삽입

임시 저장소로 원래의 Fractal-트리에 있는 각 크기의 배열을 추가한다. 이 임시 배열들은 초기에 비어있다.

• 15 삽입
① 원래의 Fractal-트리에 15를 삽입할 장소가 1-배열에 있으므로 넣는다.

• 7 삽입
① 원래의 Fractal-트리에 7를 삽입할 수가 없어 임시 1-배열에 넣는다.
② 다음으로 2개의 1-배열을 합병 정렬하여 2-배열에 넣는다.
③ 다음으로 2개의 2-배열을 합병 정렬하여 4-배열에 넣는다.
④ 그러면 최종적인 형태는 <그림 2>와 같다.

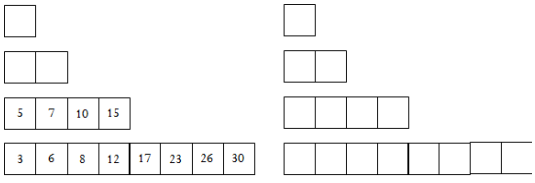

그림 2. Fractal 트리의 삽입

- 전방향 포인터 지정

각 엘리먼트는 그 엘리먼트가 fractional 캐스케이딩을 이용하여 그 다음 배열의 어디에 위치하는 가를 명시하는 전방향 포인터를 갖게 한다. 이

렇게 함으로써 검색 속도가 $\log^2 N$ 에서 $O(\log N)$ 수준으로 향상된다.

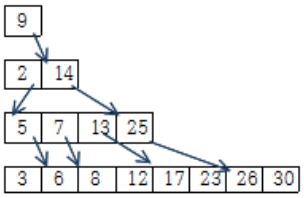

그림 3. 전방향 포인터 지정

- Fractal 트리의 성능

- 삽입 성능
  - 크기 X의 2 배열을 병합하는 데 필요한 디스크 블록 접근 수는 $O(X/B)$
  - $O(X)$개의 엘리먼트들이 합병되므로 단위 엘리먼트 당 I/O는 $O(1/B)$
  - 각 엘리먼트가 병합되는 최대 수는 $O(\log N)$
  - 그러므로 평균 삽입 비용은 $O(\frac{\log N}{B})$
- 검색 성능
  - $O(\log N)$개의 각 배열에 이진 검색을 수행
  - N 개 보다 작은 엘리먼트들 ($X < N$) 로 이루어진 배열을 대상으로 이진 검색을 하므로 각 배열에 대한 검색 성능은 $O(\log N)$
  - 그러한 배열들이 $O(\log N)$개 있으므로 전체 검색에 대한 성능은 $\log^2 N$
  - 이를 개선하기 위해 각 엘리먼트에 대해 전방향 포인터를 지정하면 각 레벨에서의 검색은 N이 크기에 의존하지 않는 횟수의 검색
  - 그러므로 전방향 포인터를 유지하면 검색성능이 $O(\log N)$로 개선

Ⅲ. 실험 방법

MongoDB에서 B-트리 인덱스를 사용하였을 때와 Fractal 인덱스를 사용하였을 때의 삽입 속도, 삽입과 검색을 병행했을 때의 처리 속도에 대한 성능을 비교해 보기 위해 실험을 수행한다.

1. 컬렉션 구조

실험에 사용된 컬렉션 구조는 <표 1>과 같다. 이 컬렉션에서 3개의 키 속성을 대상으로 각각 보조 인덱스를 생성한다.

표 1 컬렉션 구조

Key	Data Type	데이터 생성
NO	Integer	랜덤
Name	String	랜덤
Creation Time	Date	현재 시간

2. 서버 환경

실험에 이용된 서버의 사양은 다음과 같다.
- HP Micro Server, RAM 8G, 1TB Hard Disk
- OS : Ubuntu 12.04 LTS 64bit
- MongoDB 2.0.4 64bit
- TokuMX 1.3.1 64bit

실험을 수행하기 위해 필요한 코드는 자바 스크립트로 작성하였다.

3. 실험1: 문서 삽입 성능 비교

실험1에서는 삽입 연산 속도를 측정한다. B-트리 인덱스와 Fractal 인덱스를 적용하였을 때의 속도를 비교하여 본다. 하나의 스레드를 생성하여 삽입 연산을 수행한다.
컬렉션에 필드별로 랜덤 숫자와 랜덤 문자열, 삽입되는 시점의 시간이 삽입되며, 1억 건의 문서를 삽입하고 종료한다.

4. 실험2: 문서 삽입과 검색을 병행한 성능 비교

실험2에서는 하나의 쓰레드로 문서를 삽입한다. 또 하나의 스레드로 범위 검색을 수행한다. 범위검색은 매 60초 마다 수행하는데, 랜덤 정수(N)를 생성하여, 이 값과 같거나 큰 (NO >= N) 문서를 1,000개 검색하는 범위 검색을 수행하고, 걸린 시간을 측정한다. 문서 삽입 쓰레드는 5,000만 건의 문서를 삽입하고 종료한다. 범위 검색 쓰레드는 삽입이 종료될 때까지 수행한다.

Ⅳ. 실험 결과

(그림 4)는 실험1에서의 삽입 연산 성능을 비교한 그래프이다. 그래프를 보면 삽입 성능이 확연하게 차이가 난다는 것을 알 수 있다. 실험1에서, B-트리 인덱스는 초당 평균 536건의 문서를 삽입하였고, Fractal 트리 인덱스는 초당 평균 11,196건의 문서를 삽입하였다. 결과적으로 Fractal 트리 인덱스를 적용하였을 때, 약 1,989%의 향상된 성능을 보였다.
실험2의 삽입 실험에서, B-트리 인덱스는 초당 평균 1,496건의 문서를 삽입하였고, Fractal 트리 인덱스는 8,045건의 문서를 삽입하였다(그림 5).

삽입 연산과 검색 연산이 동시 병행되어 이루어지기 때문에 초당 문서 삽입 속도가 실험1보다 떨어지는 것을 알 수 있다. 결과적으로 Fractal 트리를 이용하였을 때, 약 438%의 향상된 성능을 보였다.

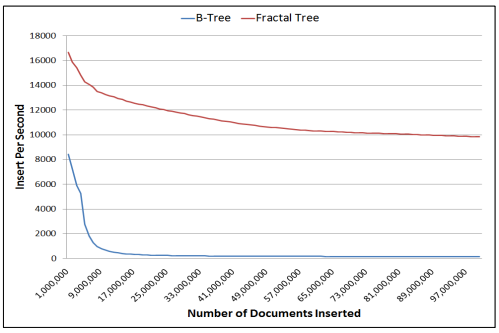

그림 4. 실험1 : 삽입 성능 비교

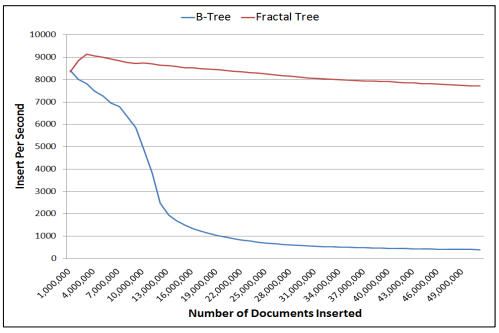

그림 5. 실험2 : 삽입 성능 비교

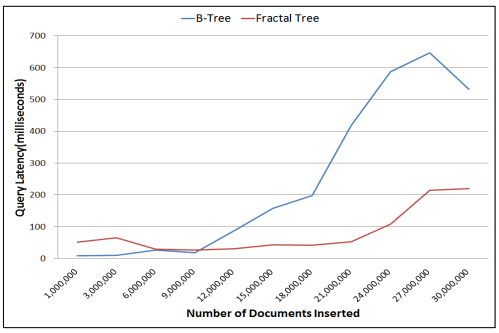

그림 6. 실험2 : 범위 검색 성능 비교

실험2의 범위 검색 실험에서는 랜덤 정수(N)를 생성하여, 이 값과 같거나 큰 (NO >= N) 문서를 1,000개 검색하는 범위 검색을 수행하고, 걸린 시간을 측정한다. B-트리 인덱스는 1,000 개의 문서를 검색하는데 평균 32,212ms가 걸렸고, Fractal 트리 인덱스는 평균 222ms가 걸렸다(그림 6). 결과적으로 Fractal 트리를 이용하였을 때, 약 14,409%의 향상된 성능을 보였다.

Ⅴ. 결 론

본 논문에서는 대표적인 NoSQL인 MongoDB를 이용하여, B-트리 인덱스를 사용하였을 때와 Fractal 트리 인덱스를 사용하였을 때의 성능을 비교해 보기 위해 두 종류의 실험을 수행하였다.

실험1에서는 문서 삽입 성능만을 비교하고, 실험2에서는 문서 삽입과 검색을 병행한 성능을 비교하였다. 실험1에서는 Fractal 트리 인덱스를 적용하였을 때 약 1,989%의 향상된 성능을 보였다. 실험2에서도 역시 Fractal 트리 인덱스에서 삽입 연산에서 약438%, 범위 검색에서 약 14,409%의 성능 향상을 보였다.

B-트리 인덱스는 데이터양이 증가할수록 연산 속도가 현저히 떨어지게 되는데, Fractal 트리 인덱스를 사용하면 데이터양이 매우 증가하더라도 그 성능이 조금 떨어지지만 B-트리 인덱스의 성능과 비교하면 매우 우수함을 확인하였다. 이전에 발표한 성능 측정 비교 결과와 이 연구의 결과로, Fractal 트리 인덱스를 빅데이터용 데이터베이스에서 도입할 가치가 있는 인덱스 구조라는 것을 확인할 수 있다.

빅데이터를 잘 활용하기 위해서는 이를 처리하는 데이터베이스의 성능이 매우 중요하다. 빅데이터를 위한 데이터베이스가 매우 많이 출시되고 있는 시점에 기존의 인덱스 성능을 향상한 새로운 인덱스의 출현이 빅데이터 처리를 더 용이하게 할 것이라 기대한다.

참고문헌

[1] (September. 2013)“Improve MongoDB with TokuMX and Fractal Tree Indexing.” Tokutek.<http://www.tokutek.com/resources/product-docs/> (1 May. 2014)

[2] Bayer, R. and E. McCreight, “Organization and Maintenance of Large Ordered Indexes,” Acta Information Vol. 1, No. 3, 1972, 30 ~ 39.

[3] Comer, D., “The Ubiquitous B-tree,” ACM Computing Surveys, Vol. 11, No. 2, June 1979, 121 ~ 137.

[4] (June 22. 2013).“Fractal Tree Indexing,”Madalgorithmist’s blog. <http://madalgorithmist.wordpress.com/2013/06/22/fractal-tree-indexing/> (1.May. 2014)

[5] Harald Prokop, Cache-Oblivious Algorithms, Master Thesis, Dept. of Electrical Engineering and Computer Csience, MIT, 1999.