Bitmap Index를 이용한 RDBMS 성능향상 기법에 관한 연구

전상화*, 이언배
한국방송통신대학교 평생대학원 정보과학과
e-mail : kalinet@empal.com* lub@mail.knou.ac.kr

A Study for Performance Improvement of RDBMS on Using Bitmap Index

Sang Hwa Jeon*, Eun-Bae Lee
Dept. of Computer Science, Korea National Open University
Graduate School

요 약

데이터베이스 성능이 저하되면, 가장 먼저 SQL 튜닝을 고려한다. SQL 튜닝에서 가장 주의 깊게 사용해야하는 부분이 바로 Index의 설정과 관련된 부분이다. 본 논문에서는 OLAP 환경에서 다양하고 복잡한 질의처리 요구와 관련하여, B-Tree Index의 문제점을 개선하고 질의 성능을 향상시키기 위해서 Bitmap Index를 사용하였다. 또한, Bitmap Index 사용의 최적 임계점을 추적하기 위하여, 데이터 분포도와 조건절의 복잡도를 조사하였으며, 샘플링된 질의문을 기준으로 B-Tree Index를 사용하였을 때와 Bitmap Index를 사용하였을 때의 비교 실험을 통하여 Bitmap Index의 사용으로 RDBMS의 성능향상이 있음을 증명하였다.

1. 서론

데이터베이스에서 인덱스의 주된 역할은 주어진 값에 대한 정보를 데이터집합에서 빠르게 추출하고자 하는 것이다.[1] 전통적으로 RDBMS에서는 B-Tree라는 구조를 가지고 이러한 목적을 충족시켜왔다. B-Tree Index는 해당정보를 트리구조의 인덱스를 가지고 목적한 데이터를 빠르게 추출하기 위해서 사용되고 있다.[2]

그러나 비즈니스 환경 변화는 수년간 기업의 서로 다른 운영체제 시스템에서 축적된 데이터에 대해서 다양한 각도로 분석을 가능케 하는 이른바, 다차원 분석 통합시스템의 필요성이 강하게 강조되고 있다. 이러한 통합시스템은 대용량의 데이터 처리와 관리 기능을 요구하게 되는데, 이러한 요구는 기존의 B-Tree Index로는 해결할 수 없는 여러 가지 문제가 발생하게 되었다.

본 논문은 그동안 OLAP환경에서 RDBMS의 다량 데이터 추출시 B-Tree Index를 이용하였으나 예상과는 다르게 성능향상을 하지 못한 원인을 분석하고, Bitmap Index를 사용하기 위한 최적의 환경을 연구하였다. 그리고, 데이터 분포도와 조건절의 복잡도에 따른 최적의 Bitmap Index 사용시점을 파악하고, 예상 SQL 질의문에 Bitmap Index를 사용함으로써 Bitmap Index의 사용이 RDBMS의 성능향상에 많은 도움이 됨을 증명하고자 하였다.

2. 관련연구

대부분의 RDBMS에서 사용하는 인덱스 기법은 B-Tree Index이다.[3]

B-Tree Index는 데이터베이스에서 Key 라는 개념보다는 옵티마이저(Optimizer)가 최적의 경로를 결정하기 위해 사용하는 요소라고 볼 수 있다. 테이블은 인덱스를 전혀 가지지 않아도 되며 인덱스를 추가, 삭제하더라도 실행결과는 전혀 영향이 없다. 다만, 옵티마이저가 찾아주는 최적의 경로가 서로 다를 뿐이다. B-Tree Index를 사용하면 검색성능이 향상된다.[4] 그러나 사전지식 없이 무턱대고 사용하면 오히려 부하가 증가하여 성능에 악영향을 미치게 된다. 일반적으로 인덱스를 생성하는 컬럼(Column)의 분포도는 10~15%을 넘지 않아야 한다.[1]

컬럼의 분포도가 10~15%를 넘는다면, 옵티마이저는 B-Tree Index를 사용하지 않고 Full Scan을 수행하여 레코드를 추출하게 된다. 이러한 처리는 RDBMS의 성능을 저하시키는 원인이 되고 있다. 또한, Null 비교, AND, OR의 사용이 잦은 다양한 질의에 능동적으로 대처할 수 없으며 COUNT, SUM과 같은 집계함수 사용시 성능이 저하되는 것이 B-Tree Index이다. 이러한 환경에 유연하게 대처하고, 보다 향상

된 성능을 이끌어 내는 하나의 방법으로 Bitmap Index를 사용한다.

3. Bitmap Index를 이용한 성능향상 기법

3.1 B-Tree Index의 비효율적인 측면

B-Tree Index 구조는 많은 장점을 가지고 있지만, 몇몇의 특징적인 경우 옵티마이저가 인덱스를 사용하여 Seek 하지 않고 Full Scan 또는 Range Scan을 통해서 데이터를 추출하게 되므로 RDBMS의 성능을 저하시키는 요인이 되며 다음과 같은 단점을 가지고 있다.[5][6][7]

(1) 저장공간의 낭비

기존인덱스가 조건 비교에 사용되는 컬럼값에 대한 테이블의 원시 값을 인덱스에도 보관하고 있어야 함으로 인해 인덱스를 위한 실제 데이터의 중복 저장 때문에 저장 공간의 낭비를 초래한다.

(2) 유연성(Flexibility)의 부족

조건절의 다양한 질의 즉, AND, OR, NOT 등의 사용과 SUM과 같은 집계함수를 사용할 때 B-Tree Index는 성능향상에 도움이 되지 않는다.

(3) 분포도가 넓은 컬럼에서의 성능저하

성별과 같은 넓은 분포도를 가진 컬럼의 조건 검색 시에는 비용기반 옵티마이저가 인덱스 Seek를 사용하지 않는다.

(4) 복잡한 조건 질의의 사용 시 성능저하

End-User-Computing 환경에서 흔히 발생하는 복잡한 질의 조건으로 인해 옵티마이저가 최적 실행계획에서 인덱스를 포기하게 되며 Null과 관련하여 단일 컬럼 인덱스나 결합 인덱스의 경우에 마지막 컬럼이 Null인 경우 Null값은 인덱스 블록 내에 저장되지 않는다.

3.2 Bitmap Index의 효율적인 사용

3.2.1 성능향상 원리

Bitmap Index는 데이터 추출시 사용되는 조건을 논리연산과 비트(bit)처리 방식을 이용하기 때문에 몇 종류 안 되는 값들과 별도의 조그만 연결표(Bitmap)와 몇 비트(bit)만의 저장 공간만을 가지고 인덱스 구성을 가능하게 한다. 이러한 원리는 OLAP 환경에서 데이터웨어하우징(Data Warehousing), 의사결정지원시스템(Decision Support System) 등의 대용량 데이터를 다루는 경우 효율적인 처리속도를 보장한다. 아래의 [표 1]은 분포도가 낮은 가상의 데이터를 샘플링 한 것이다.

[표 1] 분포도가 낮은 테이블 예제

CUST#	MARI-TAL	REGION	GENDER	INCOME_LVL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2

위의 [표 1] REGION 에 Bitmap Index를 적용한 후 내부 구조를 표현하면 다음의 [표 2]과 같다.

[표 2] REGION 의 Bitmap Index 구조

CUST#	REGION='east'	REGION='central'	REGION='west'
101	1	0	0
102	0	1	0
103	0	0	1

이러한 구조의 Bitmap Index는 다음의 쿼리문 실행을 비트연산에 의해서 결과를 도출한다.

```
SELECT COUNT(*)
FROM CUSTOMER
WHERE MARITAL_STATUS='married'
AND REGION IN('central','west');
```

status='marrie'		region='central'		region='west'						결과
0		0		0		0		0	0	101
1	∧	1	∨	0	=	1	∧	1	= 1	102
1		0		1		1		1	1	103

[그림 1] SQL 질의문의 Bitmap Index 연산과정

[그림 1]은 SQL 질의문을 Bitmap Index를 이용하여 연산하는 과정을 보여주고 있다. [표 1]의 테이블을 질의하면 [그림 1]과 같은 Bit 연산에 의해서 처리되므로 Bitmap Index 사용시 저장 공간의 절약은 물론 RDBMS의 성능향상을 이끌어 낼 수 있다.

3.2.2 Bitmap Index 사용의 최적화 알고리즘

Bitmap Index를 이용하여 SQL 문장의 성능향상을 이끌어 내기 위해서는 해당 테이블의 컬럼 내부에 포함된 특정 데이터의 분포정도를 먼저 알아야 한다.

```
Hp : 컬럼의 데이터 분포도
Er[k] : 컬럼내, 특정 데이터의 분포개수
Er.count : 테이블의 컬럼 총 개수
Tr : 테이블 총 로우 수
Function_Bitmap_Index() : Bitmap Index를 수행후 Elapsed Time 리턴
Function_BTree_Index() : B-Tree Index를 수행후 Elapsed Time 리턴

 for k=1 upto Er.count
   Hp = ( Er[k] / Tr)*100;
   Ta = call Function_Bitmap_Index(Hp);
   Tb = call Function_BTree_Index(Hp);
   if(Ta<Tb) then
     bFlag=true;
   else
     bFlag=false;
loop

return bFlag;
```

[그림 2] Bitmap Index를 수행을 위한 최적분포도 추적 알고리즘

위의 [그림 2]의 알고리즘을 이용하여 Bitmap Index를 이용하기 위한 최적의 분포도를 찾아내는 방법을 제시하고 있다.
즉, 테이블에 구성된 컬럼를 돌면서 각 컬럼의 분포도를 조사한다. 그리고 추출된 분포도의 정도에 따라서 Bitmap Index를 사용하였을 때와 B-Tree인덱스를 사용하였을 때의 Elapsed Time을 조사하여 Bitmap Index를 사용하였을 때의 비용이 B-Tree인덱스를 사용하였을 때보다 작게 되는 시점을 추적하여 각 컬럼마다 최적화된 Bitmap Index를 구현할 시점을 발견한다.

```
SEARCH(QRY) : 쿼리문의 조건절 검색
Cand : 조건절 내부의 AND 개수
Cor : 조건절 내부의 OR 개수
Tc : 조건절 내부의 AND, OR의 개수 합
Ta : Bitmap Index의 Elapsed Time
Tb : B-Tree Index의 Elapsed Time

Function_Bitmap_Index() : Bitmap Index를 수행후 Elapsed Time 리턴
Function_BTree_Index() : B-Tree Index를 수행후 Elapsed Time 리턴
while(Line.End)
  SEARCH(QRY)
    if("AND") Cand++;
    if("OR") Cor++;
  Tc = Cand + Cor ;
  Ta = call Function_Bitmap_Index(Tc);
  Tb = call Function_BTree_Index(Tc);
  if(Ta<Tb) then
    bFlag=true;
  else
    bFlag=false;
loop
return bFlag;
```

[그림 3] Bitmap Index를 수행을 위한 조건절 복잡도 추적 알고리즘

위의 [그림 3]는 조건절(Where)의 질의 시 복잡도에 비례하여 B-Tree Index를 사용하였을 때보다 Bitmap Index를 사용하였을 때의 성능향상 정도를 알아보기 위해서 작성된 알고리즘이며, 조건절의 AND, OR 혼합개수가 2 부터 10 까지임을 가정하고 여러번 수행을 하여 결과를 추출한다. 즉, AND, OR의 혼합개수가 2개인 경우부터 10개인 경우까지를 중심으로 Bitmap Index를 사용하였을 때와 B-Tree Index를 사용하였을 때를 비교하여 어느 쪽이 더 성능향상에 유리한가를 알아내어 최적의 환경에서 Bitmap Index를 구현하고자하는 알고리즘이다.

3.3 구현 및 평가

전술한 알고리즘을 기반으로 Bitmap Index 구현의 최적 환경 추출하여 B-Tree Index와 Bitmap Index를 구현한 성능평가 실험을 하였다. 실험의 결과는 시스템의 사양과 운영체제, 데이터베이스의 종류, 테이블의 구조, 데이터의 량과 네트워크 환경에 따라서 다를 수 있다. 본 논문에서는 42896 건의 회원정보 데이터가 들어 있는 테이블을 대상으로 진행하였다. 실험결과의 정확성을 위해서 단일 테이블 구조 하에서 실시하였으며, 네트워크는 차단하였다.

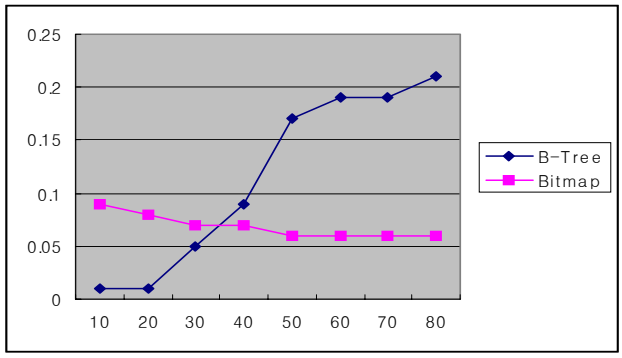
3.3.1 분포도에 따른 질의 성능평가

아래의 [표 4]의 결과는 데이터 분포도에 따른 B-Tree Index와 Bitmap Index의 Elapsed Time을 비교한 것이다.

[표 4] 분포도에 따른 성능비교

분포도	10	20	30	40	50	60	70	80
B-Tree	0.01	0.01	0.05	0.09	0.17	0.19	0.19	0.21
Bitmap	0.09	0.08	0.07	0.07	0.06	0.06	0.06	0.06

B-Tree Index는 10%~20% 정도에서 좋은 성능을 내고 있으며 약 30%가 되는 시점부터 성능저하가 두드러지고 있다.



[그림 4] 분포도에 따른 성능 비교 도표

이에 반해 Bitmap Index는 분포도에 상관없이 거의 일정한 성능을 내고 있으며, [그림 4]에서 보듯이 본 실험에서 분포도가 대략 35%이상이면 Bitmap Index를 사용하는 것이 성능향상에 도움이 됨을 알 수 있다.

3.3.2 조건절 복잡도에 따른 질의 성능평가

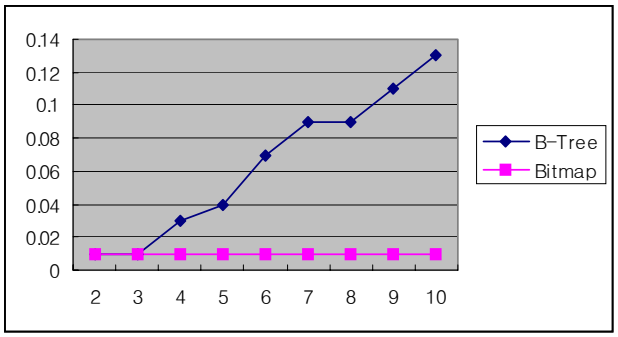
아래의 [표 5]은 조건절의 OR, AND의 복잡도 비교에서의 실험이다. 복잡도 개수와는 무관하게 Bitmap Index가 우수한 성능을 보이고 있는 반면 B-Tree Index는 복잡도가 증가할수록 Elapsed Time 이 증가하고 있음을 알 수 있다.

[표 5] 조건절 복잡도에 따른 성능비교

복잡도	2	3	4	5	6	7	8	9	10
B-Tree	0.01	0.01	0.03	0.04	0.07	0.09	0.09	0.11	0.13
Bitmap	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

아래의 [그림 5]의 도표는 [표 5]를 도식화 한 것으로, 복잡도가 3 까지는 B-Tree Index와 Bitmap

Index가 거의 동일한 성능을 보이고 있으나 복잡도 4 이상부터는 확연하게 Bitmap Index의 성능이 우수함을 알 수 있다.



[그림 5] 조건절 복잡도에 따른 성능비교 도표

3.3.3 SQL 질의문 성능향상 평가

[그림 6]과 같은 질의문을 기준으로 기존의 B-Tree Index 구조 하에서의 성능과 Bitmap Index로 전환한 후의 성능을 비교하면 [표 6]과 같다.

```
① SELECT memID,memZip,memSi,memSex
FROM Members
WHERE memSex='남' OR memLevel='중';
② SELECT memID,memSi,memLevel
FROM Members
WHERE memSex='남' AND memLevel IN('상','중') AND
memSi IN('서울','부산','광주');
③ SELECT COUNT(*) as '전체수', SUM(bonus) as '마일
리지합'
FROM Members
WHERE memSex='여' OR memLevel='상';
```

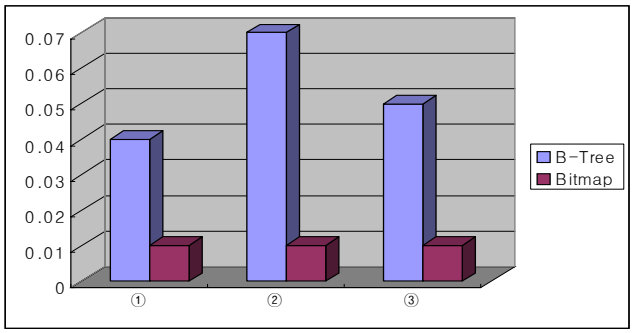
[그림 6] 평가를 위한 SQL 질의문

위의 [그림 6]의 SQL 질의문은 분포도가 나쁜 컬럼을 중심으로 ① 단순히 분포도가 나쁜 경우, ② 질의문의 조건절이 복잡한 경우, ③ 집계함수를 사용하는 경우를 가정한 질의문 이다. 이러한 질의문을 사용하여 실행한 후의 Elapsed Time을 비교하면 아래의 [표 6]과 같은 결과를 보였다.

[표 6] SQL 질의문 수행평가 결과

질의문	①	②	③
B-Tree	0.04	0.07	0.05
Bitmap	0.01	0.01	0.01

위의 [표 6]을 통해서 살펴보면, 세가지 예상 질의문 모두 B-Tree Index를 사용하였을 때보다, Bitmap Index를 사용하였을 때 Elapsed Time의 비용이 작게 소모되어서 성능향상이 있음을 알 수 있으며, [그림 6]은 [표 6]을 도식화한 그림으로써, Bitmap Index는 고른 성능분포를 보이고 있어서, 다양한 질의에 안정적인 성능향상을 보여주고 있음을 알 수 있다.



[그림 7] 질의문 수행결과 도표

4. 결론 및 향후 연구과제

그동안 Index 기법을 이용한 RDBMS의 성능향상에 대해서 많은 연구가 진행되어 왔다. 본 논문에서는 OLAP 환경에서, B-Tree Index가 가지고 있는 부적절한 사용 측면을 조사하고, 그로 인해 발생하는 성능저하를 Bitmap Index를 사용함으로써 성능향상이 있음을 증명하였다. 데이터 분포도의 경우 [그림 4]에서 보인바와 같이 약 35% 이상일 경우는 Bitmap Index를 사용하는 것이 성능향상에 도움이 되며, 조건절의 복잡도 실험 결과인 [그림 5]를 보면, Bitmap Index는 조건절의 복잡도와 관계없이 일정한 성능을 보이고 있으며, 복잡도가 4 이상인 경우는 Bitmap Index가 뛰어난 성능을 보임을 알 수 있다. SQL 질의문을 실험한 실험결과 [그림 7]에서는 Elapsed Time을 기준으로 단순하게 분포도가 나쁜 경우는 대략 평균 4 배 정도의 성능향상이 있었으며, 조건절의 복잡도에서는 약 7배 정도의 성능향상이 있었다. 또한, COUNT, SUM와 같은 집계함수의 사용 시에도 5배 정도의 성능향상이 있었다.

다소 제한적인 실험이긴 하지만 이러한 향상 정도는 대용량 데이터베이스로의 전이 시 더욱더 커질 것으로 보인다. 따라서, 향후 연구과제는 대용량 데이터베이스의 다중테이블 환경, 다중 조인 질의를 이용한 원격지 조인처리에서 성능향상을 위한 실험적 연구가 더욱 필요하다.

참고문헌

[1] 이화식, 대용량 데이터베이스 솔루션 1,2 엔코아컨설팅, 2004

[2] 주경호, 오라클 실무튜닝과 SQL 패턴학습, 사이텍미디어, 2004

[3] 김순덕, 다단계 튜닝기법을 이용한 데이터베이스 성능향상, 홍익대학교 석사논문, 2000

[4]Oracle Corp., Query Optimization in Oracle9i, Technical White Paper, 2003

[5] 김성배, 데이터베이스 처리속도 향상을 위한 SQL 튜닝, 전북대 석사논문, 2004

[6] 김교중, 대용량 데이터베이스 성능개선을 위한 튜닝방법에 관한 연구, 한남대학교 석사논문, 2004

[7] 최송희, SQL 튜닝을 이용한 데이터베이스 시스템 성능개선에 관한 연구, 경희대학교 석사논문, 2003