E477 Database and Big Data Systems, Spring 2021
HW4

Name: 김준범

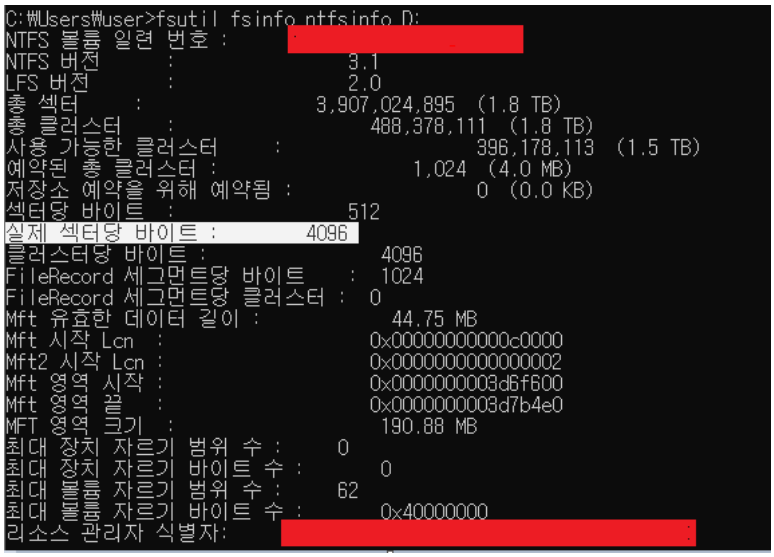
Student ID: 20180155

Discussion Group (People with whom you discussed ideas used in your answers): None

On-line or hardcopy documents used as part of your answers: None

1. Hardware

https://www.lunavi.com/blog/know-your-storage-constraints-iops-and-throughput에서 일반적인 disk의 IOPS(IO per second / block size)를 HDD의 경우 55-180, SSD의 경우를 3000-40000를 갖는다는 것을 찾았고 이 범위 중간 값을 활용하였다.
현재 내가 사용하고 있는 컴퓨터는 Seagate BarraCuda 7200/64M (ST2000DM006, 2TB)의 HHD를 갖으며 가격은 66000원이다. 내 컴퓨터의 sector의 크기는 4096Byte이기 때문에 D(unit 당 가격은) 0.000135168원이다.



현재 내가 사용하고 있는 memory는 삼성전자 DDR4-2400 램 8GB이고 가격은 47870원이기 때문에 1MB당 5.1원이다. Window X64의 page size는 4KB이므로 1MB안에 page 개수는 약 250 따라서 내 컴퓨터 사양을 기준으로 $\frac{\$D\ P}{I\ \$M} = \frac{0.000135168 * 250}{5.1 * 117.5} = 5.6 \times 10^{-5}$으로 계산되었다.

2. B+ tree

(a) What value of m minimizes the time to search for a given record? An approximate answer is OK. The value you obtain should be independent of b. (HINT: If you come up with an equation which is hard to solve algebraically, try plugging in values to locate the root of the equation.)

각 block은 m개의 pointer를 가지기 때문에 height가 h인 B+ tree는 최대 m^h개의 record를 가질 수 있다. B+ tree의 index는 full이라는 조건에 의해, $m^h = N$으로 $\mathrm{h} = \log_m N$이 되어 search를 하는데 검사해야 하는 block의 개수는 $\log_m N$개가 된다. Block 하나를 memory로 가져오는 것이 $(90 + 0.07\mathrm{m})$ ms이고 memory로 가져온 block에 대해 원하는 pointer를 찾는 것 $(\mathrm{a} + \mathrm{b}\log_2 M)ms$ 이기 때문에 search에 필요한 전체 시간은 (검사할 block 수) x (disk I/O시간+binary search 시간)이다. 따라서 search time $\mathrm{s(m)} = \log_m N(90 + 0.07\mathrm{m} + \mathrm{a} + \mathrm{b}\log_2 M)$ ms가 된다.

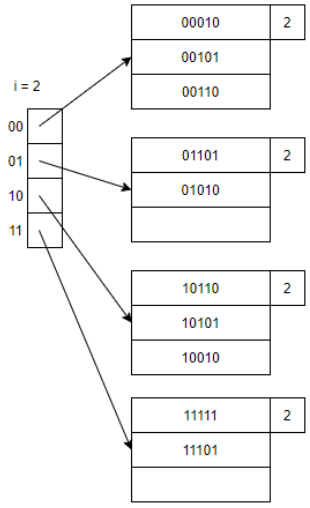
Search time이 최소가 되는 m을 찾기 위해 $\mathrm{s'(m)} = 0$이 되게 하는 m을 구하면 된다. 우선 s(m)에 밑 변환 공식을 사용하면 $\mathrm{s(m)} = \frac{lnN}{lnm}(90 + 0.07m + a + b\frac{lnm}{ln2}) = \frac{lnN}{lnm}(90 + a + 0.07m) + b\frac{lnN}{ln2}$가 되어 $\mathrm{s'(m)} = \mathrm{lnN}\frac{0.07lnm - \frac{1}{m}(90+a+0.07m)}{lnm^2}$로 계산된다. 따라서 $\mathrm{s'(m)} = 0$이 되기 위해선 분자가 0이 되어야 하며 정리하면 $\mathrm{m(lnm} - 1) = \frac{90+a}{0.07}$가 된다. 문제의 조건에서 a는 90ms에 비해 매우 작은 숫자라고 제시되었기 때문에 90+a를 90으로 근사하여 m을 찾으면 m= 278을 구할 수 있다.

(b) What happens as the seek and latency constant (90ms) decreases? For instance, if this constant is cut in half, how does the optimum m value change?

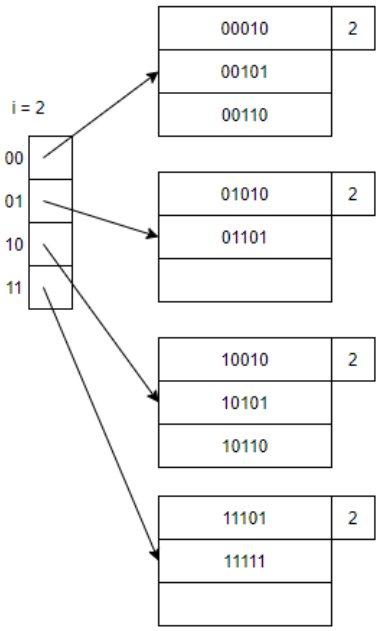
Seek and latency constant가 줄어들면 m이 감소한다. 만약 90ms가 절반인 45ms로 감소하면 $\mathrm{m(lnm} - 1) = \frac{45+a}{0.07} \approx \frac{4500}{7}$을 만족하는 m의 값은 158.xx가 되어 크게 감소하는 것을 확인 할 수 있다.

3. Extensible Hashing

(a) We insert records in the following order: Ashley, Karen, Brian, Jeff, Chris, Harold, Ethel, George, Frank, Daniel Show the structure after all these records have been inserted.



(b) Suppose, instead we insert records in the following order: Ashley, Brian, Chris, Daniel, Ethel, Frank, George, Harold, Jeff, Karen Show the structure after all these records have been inserted.



(c) Does the final structure depend on the order in which records are inserted? Explain.

(a)와 (b)를 비교해보면 bucket안에 있는 record의 순서만 다를 뿐, 같은 record들이 들어있는 것을 볼 수 있다. Input으로 넣어줄 hashed key value들을 관찰해보면 0으로 시작하는 것이 5개, 1로 시작하는 것이 5개가 있는 것을 볼 수 있다. 한 bucket은 최대 3개의 record만 가질 수 있기 때문에 무조건 split이 일어나야 한다. 00으로 시작하는 것은 3개, 01로 시작하는 것은 2개, 10으로 시작하는 것은 3개, 11로 시작하는 것은 2개이기 때문에 i=2로 split한 이후 모든 record를 bucket에 넣을 수 있게 되어 더 이상 split을 할 필요가 없어진다. 따라서 각 bucket에서 record의 순서를 고려하지 않는다면 어떻게 넣어도 최종 structure은 변하지 않게 된다.

4. Query Processing

(a) Selection of Price < 10 over Book. Memory = 10 blocks.

Book의 block 1개를 memory로 읽고 조건에 맞는지 확인하는 과정을 100번 반복한다. **100 I/Os**

(b) One pass join of Order and Cust. Memory =1001 blocks.

Order의 block을 모두 memory에 올리고(1000 I/O) Cust의 block을 하나씩 memory에 올려서 join한다(7000 I/O). **8000 I/Os**

(c) Nested loop join of Order and Cust. Cust is the outer relation. Memory=2 blocks.

Cust의 block 1개를 memory로 올리고(1 I/O) Order의 block을 하나씩 memory에 올려서 join한다(7000 I/O). 이 과정을 1000번 반복한다. 1000x(1+7000) = **7001000 I/Os**

(d) Nested loop join of Order and Cust. Cust is the outer relation. Memory=101 blocks.
Cust의 block 100개를 memory로 올리고(100 I/O) Order의 block을 하나씩 memory에 올려서 join한다(7000 I/O). 이 과정을 10번 반복한다. 10x(100+7000) = **71000 I/Os**

(e) Nested loop join of Order and Cust. Order is the outer relation. Memory=101 blocks.
Order의 block 100개를 memory로 올리고(100 I/O) Cust의 block을 하나씩 memory에 올려서 join한다(1000 I/O). 이 과정을 70번 반복한다. 70x(100+1000) = **77000 I/Os**

(f) Nested loop join of Order and Book. Book is the outer relation. Memory=11 blocks.
Book의 block 10개를 memory로 올리고(10 I/O) Order의 block을 하나씩 memory에 올려서 join 한다(7000 I/O). 이 과정을 10번 반복한다. 10x(10+7000) = **70100 I/Os**

(g) Hash join of Order and Book. Memory = 11 blocks. (You may assume that you have a hash function over Order.BookID and Book.BookID that distributes the tuples evenly into equally sized buckets.)
Min(B(Order),B(Book)) = 100이고 $\sqrt{100} = 10 \leq M = 11$ 이므로 hash join이 가능하고 I/O 수는 3(100+7000) = **21300 I/Os**이다.

5. Crash Recovery

(a) if the system crashes just before line 9 is written to disk?
Commit된 T1만 redo, T2는 undo한다. **A=80, B=140, C=10, D=10, E=10, F=10, G=10**

(b) if the system crashes just before line 13 is written to disk?
Commit된 T1만 redo, T2와 T3는 undo한다. **A=80, B=140, C=10, D=10, E=10, F=10, G=10**

(c) if the system crashes just before line 18 is written to disk?
Commit된 T1,T2만 redo, T3,T4는 undo한다. **A=80, B=140, C=70, D=50, E=10, F=10, G=10**

(d) if the system crashes just after line 19 is written to disk?
Commit된 T1,T2,T3만 redo, T4는 undo한다. **A=80, B=140, C=70, D=50, E=70, F=10, G=10**

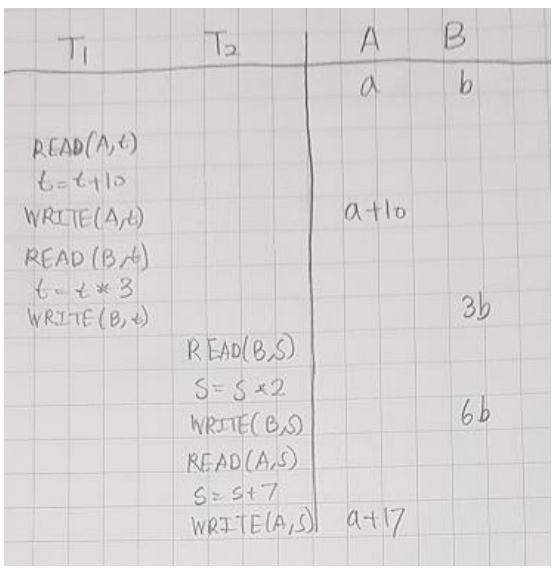
(e) if the system crashes just before line 22 is written to disk?
END CKPT를 먼저 봤으므로 commit된 T2,T3는 redo, T4는 undo한다.
A=80, B=140, C=70, D=50, E=70, F=10, G=10

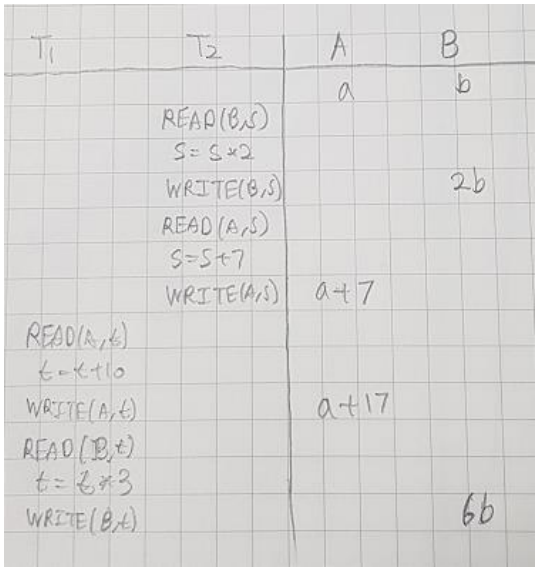
(f) if the system crashes just after line 22 is written to disk?
END CKPT를 먼저 봤으므로 commit된 T2,T3,T4는 redo한다.
A=80, B=140, C=70, D=50, E=70, F=130, G=100

6. Concurrency Control

(a) It turns out that both serial orders have the same effect on the database; that is, (T1, T2) and (T2, T1) are equivalent. Demonstrate this fact by showing the effect of the two transactions on an arbitrary initial database state.

아래 사진은 initial database state가 A = a, B = b일 때 (T1,T2)와 (T2,T1) serial schedule을 나타낸 것이다.





(T1,T2)의 경우 최종 A, B가 a+17, 6b이고 (T2,T1)의 경우도 최종 A, B가 a+17, 6b로 같은 결과를 갖는다. 임의의 initial database state에 두 serial schedule이 동일한 결과를 갖기 때문에 (T1,T2)과 (T2,T1)은 equivalent하다.

(b) Give one example each of a serializable schedule and a nonserializable schedule of the 12 actions above.

- Serializable schedule

READ(A, t); t := t+10; WRITE(A, t); READ(B, s); s := s*2; WRITE(B, s); READ(B, t); t := t*3; WRITE(B, t); READ(A, s); s := s+7; WRITE(A, s);

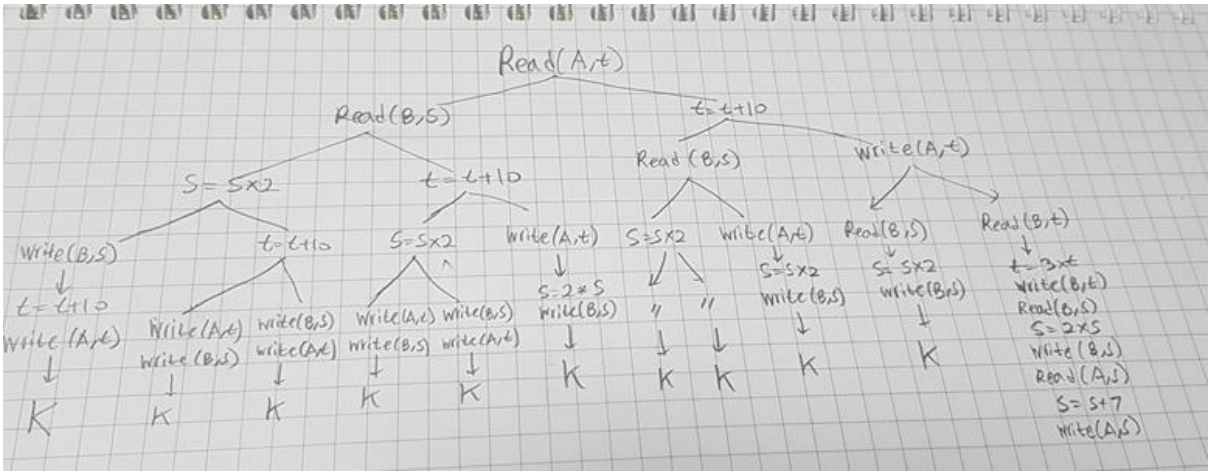
- Nonserializable schedule

READ(A, t); t := t+10; READ(B, s); s := s*2; WRITE(B, s); READ(A, s); s := s+7; WRITE(A, s); WRITE(A, t); READ(B, t); t := t*3; WRITE(B, t);

(c) How many serializable schedules of the 12 actions are there?

T1: READ(A, t); t := t+10; WRITE(A, t); READ(B, t); t := t*3; WRITE(B, t);
T2: READ(B, s); s := s*2; WRITE(B, s); READ(A, s); s := s+7; WRITE(A, s);

위 12가지 action에 대해 Serializable schedule을 만들 때는 아래의 조건들을 만족해야 한다.
1. A 또는 B를 읽고 다시 값을 쓰기 전에 A 또는 B를 읽으면 안된다.
2. 같은 transaction 내에서는 순서가 바뀔 수 없다.
3. t 또는 s에 A의 값을 읽고 연산한 다음 A에 쓰기 전까지 B의 값을 t 또는 s로 읽으면 안된다. B의 경우도 마찬가지이다.



1,2,3 조건을 만족하며 Read(A, t)에서 시작한 schedule을 그려보면 위와 같이 그려진다. 예를 들어 가장 왼쪽 경로는 Read(A, t); Read(B, s); s = s*2; Write(B,s) 인데 A에 아직 바뀐 값이 써지지 않았기 때문에 Read(A, s)가 나올 수 없어 t = t +10; write(A, t)로 이어지게 된다. K라고 써져 있는 경로는 READ(B, t); t := t*3; WRITE(B, t); 와 READ(A, s); s := s+7; WRITE(A, s); 6개의 action만을 남겨둔 상태에서의 가능한 모든 경로를 의미한다. K에서는 2번 조건을 위배하지 않는 상황에서 자유롭게 배열이 가능하기 때문에 aaabbb를 배열하는 경우와 같아(자리만 정해지면 순서는 정해져 있기 때문) $\frac{6!}{3!3!}$ = 20의 경우의 수를 가진다. 따라서 Read(A, t)에서 시작한 schedule은 k가 총 10개 + 경로 1개 이므로 20*10 + 1 = 201의 경우가 가능하고 Read(B, s)에서 시작한 schedule도 이와 유사한 방법으로 201이 된다. 따라서 **402**개의 serializable schedule이 가능하다.