EE477 Database and Big Data Systems HW4

20170616 정희진

1. Hardware

I used "8Gb C-die DDR4 SDRAM" datasheet for memory page size(1KB) and "BarraCuda SSD" for disk IOPS(9000). Samsung DDR4 16GB PC4-25600 is 84380 won. Seagate Barracuda SSD 500GB is 91150 won. Therefore, P, M, I, D are shown below.

P	1024
M	5.150
I	9000
D	0.178027

So, X is 0.003933 seconds. Therefore, the 5 minute rule doesn't holds today.

2. B+tree

(a) The time to search for a given record is

$$(log_m \mathrm{N})(90 + 0.07\mathrm{m} + \mathrm{a} + \mathrm{b}log_2 \mathrm{m}) \approx \frac{90+0.07m}{\ln m} \times \ln N + b \times lnN$$

We need to minimize $\frac{90+0.07m}{\ln(m)}$

$$(\frac{90 + 0.07m}{\ln m})' = \frac{0.07 \times lnm - \frac{90}{m} - 0.07}{(\ln m)^2} = 0$$

$$0.07 \times lnm - \frac{90}{m} - 0.07 = 0$$

$$\mathrm{m} \approx 277.864$$

However m is an integer, so we need to compare m=277 and m=278.

$$\frac{90 + 0.07 \times 278}{\ln(278)} - \frac{90 + 0.07 \times 277}{\ln(277)} < 0$$

Therefore, the value of m minimizes the time to search for a given record is 278.

(b) The time to search for a given record is

$$(log_m \mathrm{N})(45 + 0.07\mathrm{m} + \mathrm{a} + \mathrm{b}log_2 \mathrm{m}) \approx \frac{45+0.07m}{\ln m} \times \ln N + b \times lnN$$

We need to minimize $\frac{45+0.07m}{\ln(m)}$

$$(\frac{45 + 0.07m}{\ln m})' = \frac{0.07 \times \ln m - \frac{45}{m} - 0.07}{(\ln m)^2} = 0$$

$$0.07 \times \ln m - \frac{45}{m} - 0.07 = 0$$

$$\mathrm{m} \approx 158.191$$

However m is an integer, so we need to compare m=158 and m=159.

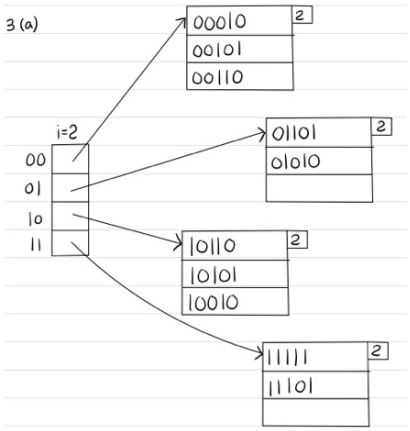
$$\frac{45 + 0.07 \times 159}{\ln(159)} - \frac{45 + 0.07 \times 158}{\ln(158)} > 0$$

Therefore, the value of m minimizes the time to search for a given record is 158.

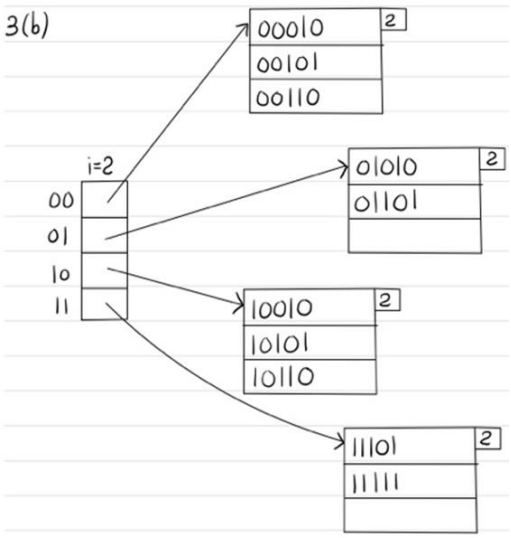
So when the seek and latency constant decreases, m becomes small.

3. Extensible Hashing

(a)

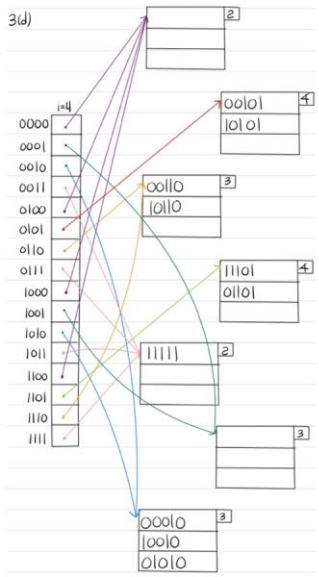


(b)



(c) If only the order of inserted tuple is different and all other things are the same, the number of buckets, global depth, and local depth of data blocks are all the same. However, the order of tuples within a data block can be different.

(d)



(e) As shown in the picture above, there are a lot of buckets that are not needed (with empty data blocks). If tuples are added and the hash function value of the tuples are evenly divided based on the low order bits, the data is evenly distributed and unnecessary buckets are reduced, so it is not a serious problem. However, if the hash function values of tuples are not evenly divided based on the low order bits, unnecessary buckets will increase, which is a serious problem.

4. Query Processing

(a) The number of I/Os : 33

Let's assume the number of blocks selecting tuples of Price < 10 over Book is B(Book)/3 = 33 blocks. We have to read 33 blocks one by one, so we need 33 I/Os.

(b) The number of I/Os : 8000

We have to read 1000 blocks of Cust first, so we need 1000 I/Os. Then, we have to read 7000 blocks of Order one by one, so we need more 7000 I/Os. Thus, the sum of I/Os is 8000 I/Os.

(c) The number of I/Os : 7001000

Each block of Cust is joined with 7000 blocks of Order, so we need 7001 I/Os. All of blocks of Cust need to be joined, so we need to multiply by 1000. Thus, the number of I/Os is 7001000 I/Os.

(d) The number of I/Os : 71000

We have to read 100 blocks of Cust first, so we need 100 I/Os. Then, we have to read 7000 blocks of Order one by one, so we need more 7000 I/Os. The sum of I/Os is 7100 I/Os and the number of loops is 1000/100 = 10 loops. So, we need to multiply the two. Thus, the number of I/Os is 71000 I/Os.

(e) The number of I/Os : 71000

We have to read 100 blocks of Order first, so we need 100 I/Os. Then, we have to read 1000 blocks of Cust one by one, so we need more 1000 I/Os. The sum of I/Os is 1100 I/Os and the number of loops is 7000/100 = 70 loops. So, we need to multiply the two. Thus, the number of I/Os is 77000 I/Os.

(f) The number of I/Os : 70100

We have to read 10 blocks of Book first, so we need 10 I/Os. Then, we have to read 7000 blocks of Order one by one, so we need more 7000 I/Os. The sum of I/Os is 7010 I/Os and the number of loops is 100/10 = 10 loops. So, we need to multiply the two. Thus, the number of I/Os is 70100 I/Os.

(g) The number of I/Os : 21300

First, we need to read Order and hash to buckets on Order.BookID and write them into disk. So, we need 7000*2 = 14000 I/Os. Also, we need to read Book and hash to buckets on Book.BookID and write them into disk. So, we need more 100*2 = 200 I/Os. Then, we need to read one Book bucket (100/10 = 10 I/Os) and read the blocks in corresponding Order bucket one by one (7000/10 = 700 I/Os) and join. We need to repeat 11–1 = 10 times for other buckets. Thus, the number of I/Os is 14000+200+(10+700)*10 = 21300 I/Os.

5. Crash Recovery

(a)

A	80
B	140
C	10
D	10
E	10
F	10
G	10

(b)

A	80
B	140
C	10
D	10
E	10
F	10
G	10

(c)

A	80
B	140
C	70
D	50
E	10
F	10
G	10

(d)

A	80
B	140
C	70
D	50
E	70
F	10
G	10

(e)

A	80
B	140
C	70
D	50
E	70
F	10
G	10

(f)

A	80
B	140
C	70
D	50
E	70
F	130
G	100

6. Concurrency Control

(a)

T1	T2	A	B
		a	b
READ(A, t)			
t := t+10			
WRITE(A, t)		a+10	
READ(B, t)			
t := t*3			
WRITE(B, t)			b*3
	READ(B, s)		
	s := s*2		
	WRITE(B, s)		b*6
	READ(A, s)		
	s := s+7		
	WRITE(A, s)	a+17	

The final result is A = a+17, B = b*6

T1	T2	A	B
		a	b
	READ(B, s)		
	s := s*2		
	WRITE(B, s)		b*2
	READ(A, s)		
	s := s+7		
	WRITE(A, s)	a+7	
READ(A, t)			
t := t+10			
WRITE(A, t)		a+17	
READ(B, t)			
t := t*3			
WRITE(B, t)			b*6

The final result is A = a+17, B = b*6

Therefore, (T1, T2) and (T2, T1) are equivalent.

(b) Serializable schedule:

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

(c) We can divide them into 4 cases.

Let's assume that we represent M; N; when the action M has to precede the action N.

(Case 1) $r_1(A); w_1(A); r_2(A); w_2(A);$ and $r_1(B); w_1(B); r_2(B); w_2(B);$

There is only one case.

$$r_1(A); w_1(A); r_1(B); w_1(B); r_2(B); w_2(B); r_2(A); w_2(A);$$

(Case 2) $r_2(A); w_2(A); r_1(A); w_1(A);$ and $r_1(B); w_1(B); r_2(B); w_2(B);$

There isn't any case.

Because $r_1(B); w_1(B); can\ not\ precede\ r_1(A); w_1(A);$ and $r_2(A); w_2(A)\ can\ not\ precede\ r_2(B); w_2(B);$

(Case 3) $r_2(A); w_2(A); r_1(A); w_1(A);$ and $r_2(B); w_2(B);\ r_1(B); w_1(B);$

There is only one case.

$$r_2(B); w_2(B);\ r_2(A); w_2(A); r_1(A); w_1(A); r_1(B); w_1(B);$$

(Case 4) $r_1(A); w_1(A); r_2(A); w_2(A);$ and $r_2(B); w_2(B);\ r_1(B); w_1(B);$

It is only necessary that $r_1(A); w_1(A);$ and $r_2(B); w_2(B);$ have to precede $r_2(A); w_2(A);$ and $r_1(B); w_1(B);$

So, $\frac{6!}{3!3!} \times \frac{6!}{3!3!} = 400$ (※ There is an action like 't:=t+10' between $r_{1\ or\ 2}(A\ or\ B)\ and\ w_{1\ or\ 2}(A\ or\ B)$)

Therefore, there are 402 serializable schedules.

References

https://www.samsung.com/semiconductor/global.semi/file/resource/2017/12/x16%20only_8G_C_DDR4_Samsung_Spec_Rev1.5_Apr.17.pdf

https://www.seagate.com/www-content/datasheets/pdfs/barracuda-ssd-DS1984-3-1806-WW-en_US.pdf