DFTL Simulator Report

2016311821 한승하

이번 DFTL에선 지난번 sector-based simulator코드의 많은 부분을 그대로 사용하였습니다. 따라서 지난 코드에서 수정부분 위주로 서술하였습니다.

<Global Variables>

```
typedef struct l2p_table{
        char dirty; //D for dirty C for clean N for Not used yet
        u32 *ppn;
        u32 *lpn;
        u32 access_time;
}cache;

typedef struct caches{
        cache* l2p_cache;
}l2p;

l2p *CMT;

typedef struct memory_map{
        char used; // U for used N for Not used yet
        u32 mapped_page;
}map_table;

typedef struct g2m{
        map_table* gtd_cache;
}table;

table *GTD;

typedef struct ppn_table{
        char type;  // U for user M for map
        char vaild;   //I for invaild, V for Vaild, F for free
}ppn;

ppn *ppn_inf;

u32 *free_page_ptr_user;
u32 *free_map_page;
u32 *free_user_page;
u32 *free_page_ptr_map;
```

이번과제에서 사용한 Global variables입니다. Bank별 CMT와 CMT의 매 map page (l2p_cache 로 표현됨) 에는 N_MAP_ENTRIES_PER_PAGE만큼의 ppn, lpn쌍과 map page의 access time 그리고 dirty bit으로 구성되어 있습니다.

GTD는 N_MAP_PAGES_PB만큼의 GTD table (map table로 표현됨)와 각가의 entry에 used bit과 mapped_page로 구성되어 있습니다.

PPN의 valid 그리고 page type (user, map)을 체크하기 위한 ppn table과 , free page 갯수 그리고 다음 write을 수행할 page를 가르키는 ptr을 저장하는 변수를 각각 user와 map page에 대해 선언해 주었습니다.

<Ftl Open>

```
void ftl_open()
{
        int i,j,k;
        CMT = (l2p*)malloc(sizeof(l2p)*N_BANKS);
        for(i=0;i<N_BANKS;i++) CMT[i].l2p_cache = (cache*)malloc(sizeof(cache)* N_CACHED_MAP_PAGE_PB);
        for(i=0;i<N_BANKS;i++)
        {
                for(j=0;j<N_CACHED_MAP_PAGE_PB;j++)
                {
                        CMT[i].l2p_cache[j].ppn = (u32*)malloc(sizeof(u32)*N_MAP_ENTRIES_PER_PAGE);
                        CMT[i].l2p_cache[j].lpn = (u32*)malloc(sizeof(u32)*N_MAP_ENTRIES_PER_PAGE);
                        for(k=0;k<8;k++) CMT[i].l2p_cache[j].ppn[k] = MAX;
                        CMT[i].l2p_cache[j].dirty = 'N';
                        CMT[i].l2p_cache[j].access_time = 0;
                }
        }
        GTD = (table*)malloc(sizeof(table)*N_BANKS);
        for(i=0;i<N_BANKS;i++) GTD[i].gtd_cache = (map_table*)malloc(sizeof(map_table)* N_MAP_PAGES_PB);
        for(i=0;i<N_BANKS;i++)
        {
                for(j=0;j<N_MAP_PAGES_PB;j++)
                {
                        for(k=0;k<8;k++) GTD[i].gtd_cache[j].used = 'N';
                        GTD[i].gtd_cache[j].mapped_page = MAX;
                }
        }
        ppn_inf = (ppn*)malloc(sizeof(ppn)*N_BANKS*BLKS_PER_BANK*PAGES_PER_BLK);
        for(i=0;i<N_BANKS*BLKS_PER_BANK*PAGES_PER_BLK;i++)
        {
                ppn_inf[i].vaild = 'F';
                ppn_inf[i].type = 'N';
        }
        free_map_page = (u32*)malloc(sizeof(u32)*N_BANKS);
        free_user_page = (u32*)malloc(sizeof(u32)*N_BANKS);
        free_page_ptr_user = (u32*)malloc(sizeof(u32)*N_BANKS);
        free_page_ptr_map = (u32*)malloc(sizeof(u32)*N_BANKS);
        for(i=0;i<N_BANKS;i++)
        {
                free_map_page[i] = (N_MAP_BLOCKS_PB + N_OP_MAP_BLOCKS_PB) * PAGES_PER_BLK;
                free_user_page[i] = (N_USER_BLOCKS_PB+N_OP_USER_BLOCKS_PB) * PAGES_PER_BLK;
                free_page_ptr_user[i] = 0;
                free_page_ptr_map[i] = PAGES_PER_BLK;
        }
        nand_init(N_BANKS,BLKS_PER_BANK,PAGES_PER_BLK);
        return;
}
```

Ftl_open입니다. 각 변수에 대해 정의된 만큼 선언해 주었으며 초기값을 설정해 주었습니다.

<Ptr inc>

```
void inc_user_page(u32 bank)
{
        free_page_ptr_user[bank]++;
        if(free_page_ptr_user[bank]%PAGES_PER_BLK == 0 )
        {
                if(free_page_ptr_user[bank] == BLKS_PER_BANK*PAGES_PER_BLK) free_page_ptr_user[bank] = 0;
                while(ppn_inf[bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_user[bank]].vaild != 'F'||free_page_ptr_user[bank] == free_page_ptr_map[
bank])
                {
                        free_page_ptr_user[bank] += PAGES_PER_BLK;
                        if(free_page_ptr_user[bank] >= BLKS_PER_BANK*PAGES_PER_BLK) free_page_ptr_user[bank] = 0;
                }
        }
}
void inc_map_page(u32 bank)
{
        free_page_ptr_map[bank]++;
        if(free_page_ptr_map[bank]%PAGES_PER_BLK == 0 )
        {
                if(free_page_ptr_map[bank] == BLKS_PER_BANK*PAGES_PER_BLK) free_page_ptr_map[bank] = 0;
                while(ppn_inf[bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_map[bank]].vaild != 'F' ||free_page_ptr_user[bank] == free_page_ptr_map[
bank]  )
                {
                        free_page_ptr_map[bank] += PAGES_PER_BLK;
                        if(free_page_ptr_map[bank] >= BLKS_PER_BANK*PAGES_PER_BLK) free_page_ptr_map[bank] = 0;
                }
        }
}
```

Page_ptr을 증가 시켜주기 위해 inc함수를 작성하였습니다. Page가 blk의 시작인 경우, 해당 blk가 사용된 blk인지 확인 후, 사용 중인 blk이라면 pages_per_blk만큼 증가시켜주며 free blk을 찾아서 return해 줍니다.

<CMT search>

```
int CMT_search(int* cmt_index, int* ppn_index, u32 lpn , u32 bank)
{
        int i,j;
        for(i=0;i<N_CACHED_MAP_PAGE_PB;i++)
        {
                for(j=0;j<N_MAP_ENTRIES_PER_PAGE;j++)
                {
                        if(CMT[bank].l2p_cache[i].lpn[j] == lpn)
                        {
                                *cmt_index = i;
                                *ppn_index = j;
                                s.cache_hit++;
                                return 0;
                        }
                }
        }
        s.cache_miss++;
        return 1;
}
```

Cmt search함수입니다. Cmt를 뒤지며, 해당 lpn에 해당하는 entry가 있다면, cmt index와 ppn index를 반환하고, cache hit을 증가시켜줍니다. 모든 cmt에 해당 lpn이 없다면 miss를 증가시켜주고 return 합니다.

<CMT Miss>

```
void CMT_miss(int* cmt_index, int* ppn_index, u32 lpn , u32 bank)
{
        u32 i,idx,j = 0;
        u32 ppn = 0;
        u32 t = MAX;
        int victim = 0;
        idx = lpn/(8*N_BANKS);
        for(i=0;i<N_CACHED_MAP_PAGE_PB;i++)
        {
                if(CMT[bank].l2p_cache[i].access_time < t)
                {
                        t = CMT[bank].l2p_cache[i].access_time;
                        victim = i;
                }
        }
        u32 index = CMT[bank].l2p_cache[victim].lpn[0] / (8*N_BANKS);
        if(CMT[bank].l2p_cache[victim].dirty == 'D')
        {
                while(free_map_page[bank] == PAGES_PER_BLK*N_MAP_GC_THRESHOLD) map_garbage_collection(bank);
                if(GTD[bank].gtd_cache[index].used != 'N') ppn_inf[GTD[bank].gtd_cache[index].mapped_page].vaild = 'I';
                else GTD[bank].gtd_cache[index].used = 'U';
                map_write(bank,bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_map[bank],victim);
                GTD[bank].gtd_cache[index].mapped_page = bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_map[bank];
                ppn_inf[bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_map[bank]].vaild = 'V';
                ppn_inf[bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_map[bank]].type = 'M';
                free_map_page[bank]--;
                inc_map_page(bank);
        }
        if(GTD[bank].gtd_cache[idx].used == 'N')
        {
                for(i=0;i<N_MAP_ENTRIES_PER_PAGE;i++)
                {
                        if(i * N_BANKS + idx*N_BANKS*8 == lpn)
                        {
                                CMT[bank].l2p_cache[victim].ppn[i] = ppn;
                        }
                        CMT[bank].l2p_cache[victim].lpn[i] = i * N_BANKS + idx*N_BANKS*8 + bank;
                        CMT[bank].l2p_cache[victim].ppn[i] = MAX;
                }
                CMT[bank].l2p_cache[victim].access_time = now();
                CMT[bank].l2p_cache[victim].dirty = 'D';
        }
```

```
        else
        {
                ppn = GTD[bank].gtd_cache[idx].mapped_page;
                map_read(bank,ppn,victim);
        }
        for(i=0;i<N_CACHED_MAP_PAGE_PB;i++)
        {
                for(j=0;j<N_MAP_ENTRIES_PER_PAGE;j++)
                {
                        if(CMT[bank].l2p_cache[i].lpn[j] == lpn)
                        {
                                *cmt_index = i;
                                *ppn_index = j;
                                break;
                        }
                }
        }
        return ;
}
```

Cmt miss상황이 발생하면 LRU 정책에 의해 cmt victim을 추방하고 필요한 page를 읽어와야 합니다. 따라서 cmt page를 돌며 access_time이 가장 오래된 page를 찾습니다. 이후 Dirty bit이 설정되어 있다면 GTD에 mapped_page를 update해주고 이전 mapped_page에 해당하는 ppn을 invalid 설정해준 후, map_write를 이용해 기록해줍니다. 만일 남은 map blk가 trashhold와 같다면 map_gc를 수행해 줍니다. 이후 새로운 map page에 대해 ppn table을 update해 주고 map_ptr을 증가시켜 줍니다.

GTD를 확인해 해당 map page가 적힌 적이 없다면, 빈 CMT자리에 새로운 lpn에 대한 mapping 정보를 적어줍니다. 그리고 실제로 쓰일 lpn에 대해 ppn을 기록해주고 dirty bit과 access time을 설정해줍니다.

만일 쓰인 적이 있는 page라면 GTD에서 ppn을 가져와 map_read로 읽어들여 줍니다.

이후 cmt_index와 ppn_index를 찾아 return 해줍니다.

<Map Read>

```
void map_read(u32 bank, u32 map_page, u32 cache_slot)
{
        u32 *read_buffer = (u32 *)calloc(SECTORS_PER_PAGE, sizeof(u32));
        u32 spare;
        int i;
        int blk = (map_page - bank*BLKS_PER_BANK*PAGES_PER_BLK)/PAGES_PER_BLK;
        int page = (map_page - bank*BLKS_PER_BANK*PAGES_PER_BLK - blk*PAGES_PER_BLK);
        nand_read(bank,blk,page,read_buffer,&spare);
        for(i=0;i<N_MAP_ENTRIES_PER_PAGE;i++)
        {
                CMT[bank].l2p_cache[cache_slot].lpn[i] = spare;
                spare += N_BANKS;
                CMT[bank].l2p_cache[cache_slot].ppn[i] = read_buffer[i];
        }
        CMT[bank].l2p_cache[cache_slot].access_time = now();
        CMT[bank].l2p_cache[cache_slot].dirty = 'C';
}
```

Map_read함수입니다. Map_page에 해당하는 data를 읽어와 해당 cmt_slot에 기록해 줍니다.

<Map Write>

```
void map_write(u32 bank, u32 map_page, u32 cache_slot)
{
        u32 *write_buffer = (u32 *)calloc(SECTORS_PER_PAGE, sizeof(u32));
        u32 spare = CMT[bank].l2p_cache[cache_slot].lpn[0];
        int i;
        int blk = (map_page - bank*BLKS_PER_BANK*PAGES_PER_BLK)/PAGES_PER_BLK;
        int page = (map_page - bank*BLKS_PER_BANK*PAGES_PER_BLK - blk*PAGES_PER_BLK);
        for(i=0;i<N_MAP_ENTRIES_PER_PAGE;i++)   write_buffer[i] = CMT[bank].l2p_cache[cache_slot].ppn[i];
        nand_write(bank,blk,page,write_buffer,spare);
        s.map_write += SECTORS_PER_PAGE;
}
```

Map_write 함수입니다. Map_page에 해당하는 ppn에 cache_slot에 해당하는 CMT entry를 기록해 줍니다.

<Map Garbage_collection>

```
void map_garbage_collection(u32 bank)
{
        s.map_gc++;
        int i,j,invaild_count,store_value = 0;
        int vaild_count,victim = 0;
        u32 spare;
        int free_count = 0;
        u32 r_buffer[SECTORS_PER_PAGE];
        memset(r_buffer, 0, DATA_SIZE);
        u32 s_point = bank*BLKS_PER_BANK*PAGES_PER_BLK;
        for(i=0;i<BLKS_PER_BANK;i++)
        {
                invaild_count = 0;
                free_count = 0;
                for(j=0;j<PAGES_PER_BLK;j++)
                {
                        if(ppn_inf[s_point+i*PAGES_PER_BLK].type == 'M' && ppn_inf[s_point+i*PAGES_PER_BLK+j].vaild == 'I') invaild_count++;
                        if(ppn_inf[s_point+i*PAGES_PER_BLK+j].vaild == 'F') free_count++;
                }
                if(invaild_count>store_value)
                {
                        victim = i;
                        store_value = invaild_count;
                }
                if(free_count == PAGES_PER_BLK && free_page_ptr_user[bank] != i*PAGES_PER_BLK) free_page_ptr_map[bank] = i*PAGES_PER_BLK;
        }
        vaild_count = 0;
        for(j=0;j<PAGES_PER_BLK;j++)
        {
                if(ppn_inf[s_point+victim*PAGES_PER_BLK+j].vaild == 'V')
                {
                        vaild_count++;
                        nand_read(bank,victim,j,r_buffer,&spare);
                        nand_write(bank,free_page_ptr_map[bank]/PAGES_PER_BLK,free_page_ptr_map[bank]%PAGES_PER_BLK,r_buffer,spare);
                        GTD[bank].gtd_cache[spare/(8*N_BANKS)].mapped_page = bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_map[bank];
                        ppn_inf[bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_map[bank]].vaild = 'V';
                        ppn_inf[bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_map[bank]].type = 'M';
                        inc_map_page(bank);
                        s.map_gc_write += SECTORS_PER_PAGE;
                }
        }
        for(j=0;j<PAGES_PER_BLK;j++)
        {
                ppn_inf[s_point+victim*PAGES_PER_BLK+j].vaild = 'F';
                ppn_inf[s_point+victim*PAGES_PER_BLK+j].type = 'N';
        }
        nand_erase(bank,victim);
        if(store_value != 0)
        {
                free_map_page[bank] += PAGES_PER_BLK - vaild_count;
        }
}
```

Map gc입니다. 기본적으로 Greedy policy를 취하기에 이전 과제들과 victim을 선정하는 코드는 동일합니다. 이후 victim이 선정된 이후에 read해온 spare (lpn) 값에 대한 GTD를 수정해줍니다. 또한 해당 ppn_table도 update해 줍니다.

이후 victim page의 ppn_table을 free로 설정해주고 nand_erase를 수행합니다.

<Ftl Read>

```
void ftl_read(u32 lba, u32 num_sectors, u32 *read_buffer)
{
        u32 *buffer = (u32 *)calloc(SECTORS_PER_PAGE, sizeof(u32));
        int i;
        u32 read_buffer_ptr = 0;
        u32 num_sectors_to_read;
        u32 lpn = lba / SECTORS_PER_PAGE;
        u32 sect_offset = lba % SECTORS_PER_PAGE;
        u32 sectors_remain = num_sectors;
        while(sectors_remain != 0)
        {
                if(sect_offset + sectors_remain < SECTORS_PER_PAGE) num_sectors_to_read = sectors_remain;
                else num_sectors_to_read = SECTORS_PER_PAGE - sect_offset;
                ftl_read_page(lpn,buffer);
                for(i=0;i<num_sectors_to_read;i++)
                {
                        read_buffer[read_buffer_ptr + i] = buffer[sect_offset + i];
                }
                read_buffer_ptr += num_sectors_to_read;
                sect_offset = 0;
                sectors_remain -=num_sectors_to_read;
                lpn++;
                if(lpn == MAX_LPN) lpn = 0;
        }
        return;
}
```

Ftl_Read입니다. Lba를 기준으로 하는 ftl read는 Sector-based ftl과 동일합니다.

Lpn이 최댓값을 넘어가는 경우를 고려해 if(lpn == MAX_LPN) lpn = 0; 부분을 수정해 주었습니다.

<Ftl Read by page>

```
void ftl_read_page(u32 lpn, u32 *read_buffer)
{
        int cmt_index = 0;
        int ppn_index = 0;
        int bank = lpn%N_BANKS;
        if(CMT_search(&cmt_index,&ppn_index,lpn,bank)) CMT_miss(&cmt_index,&ppn_index,lpn,bank);
        if(CMT[bank].l2p_cache[cmt_index].ppn[ppn_index] == MAX) return;
        int blk = (CMT[bank].l2p_cache[cmt_index].ppn[ppn_index] - bank*BLKS_PER_BANK*PAGES_PER_BLK)/PAGES_PER_BLK;
        int page = (CMT[bank].l2p_cache[cmt_index].ppn[ppn_index] - bank*BLKS_PER_BANK*PAGES_PER_BLK)%PAGES_PER_BLK;
        u32 spare;
        nand_read(bank,blk,page,read_buffer,&spare);
        return;
}
```

Page 단위로 read 하는 ftl_read_page입니다. CMT_search 함수를 이용해 Cmt를 뒤진 후, 존재하지 않는다면 CMT_miss 함수를 호출해 줍니다.

해당 ppn이 MAX (초기값)이라면 쓰여지지 않은 page이므로 return합니다. (read_modify wirte를 위함)

아니라면 blk과 page를 계산 한 후 read해줍니다.

<Ftl Write>

```
void ftl_write(u32 lba, u32 num_sectors, u32 *write_buffer)
{
        u32 num_sectors_to_write;
        u32 *buffer = (u32 *)calloc(SECTORS_PER_PAGE, sizeof(u32));
        int i;
        u32 write_buffer_ptr = 0;
        u32 lpn = lba / SECTORS_PER_PAGE;
        u32 sect_offset = lba % SECTORS_PER_PAGE;
        u32 remain_sectors = num_sectors;
        while(remain_sectors != 0)
        {
                for(i=0;i<SECTORS_PER_PAGE;i++) buffer[i] = 0;
                if(sect_offset + remain_sectors >= SECTORS_PER_PAGE) num_sectors_to_write = SECTORS_PER_PAGE - sect_offset;
                else num_sectors_to_write = remain_sectors;
                if(sect_offset != 0)
                {
                        ftl_read_page(lpn,buffer);
                }
                for(i=0;i<num_sectors_to_write;i++)
                {
                        buffer[sect_offset + i] = write_buffer[write_buffer_ptr + i];
                }
                ftl_write_page(lpn,buffer);
                s.ftl_write += SECTORS_PER_PAGE;
                write_buffer_ptr += num_sectors_to_write;
                sect_offset = 0;
                remain_sectors -= num_sectors_to_write;
                lpn++;
                if(lpn == MAX_LPN) lpn = 0;
        }
}
```

Ftl_write함수입니다. Ftl_read와 마찬가지로 lba단위로 write 하는 코드는 sector-based ftl과 동일합니다.

<Ftl Write by page>

```
void ftl_write_page(u32 lpn, u32 *write_buffer)
{
        int bank = lpn%N_BANKS;
        while(free_user_page[bank] == PAGES_PER_BLK*N_GC_THRESHOLD) garbage_collection(bank);
        int cmt_index,ppn_index;
        if(CMT_search(&cmt_index,&ppn_index,lpn,bank)) CMT_miss(&cmt_index,&ppn_index,lpn,bank);
        u32 ppn = bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_user[bank];
        if(CMT[bank].l2p_cache[cmt_index].ppn[ppn_index] != MAX) ppn_inf[CMT[bank].l2p_cache[cmt_index].ppn[ppn_index]].vaild = 'I';
        CMT[bank].l2p_cache[cmt_index].ppn[ppn_index] = ppn;
        CMT[bank].l2p_cache[cmt_index].dirty = 'D';
        CMT[bank].l2p_cache[cmt_index].access_time = now();
        int blk = (ppn - bank*BLKS_PER_BANK*PAGES_PER_BLK)/PAGES_PER_BLK;
        int page = (ppn - bank*BLKS_PER_BANK*PAGES_PER_BLK - blk*PAGES_PER_BLK);
        nand_write(bank,blk,page,write_buffer,lpn);
        ppn_inf[bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_user[bank]].vaild = 'V';
        ppn_inf[bank*BLKS_PER_BANK*PAGES_PER_BLK + free_page_ptr_user[bank]].type = 'U';
        free_user_page[bank]--;
        inc_user_page(bank);
        return;
}
```

Ftl write by page입니다. 남은 blk수가 Threshold와 같다면 gallbage_collection을 실행해줍니다.

Read와 동일하게 cmt_search를 수행 후 miss라면 cmt_miss를 수행해줍니다.

이후 두 함수를 이용해 얻은 index를 이용해 cmt의 ppn가 dirty bit, access time을 update해줍니다. 또한 ppn table에도 invaild를 표시해 줍니다. 이후 새로운 ppn에 write을 해주고 valid 와 type을 기록해 준 후, free page 개수를 줄이고 ptr을 증가시켜 줍니다.

<Garbage Collection>

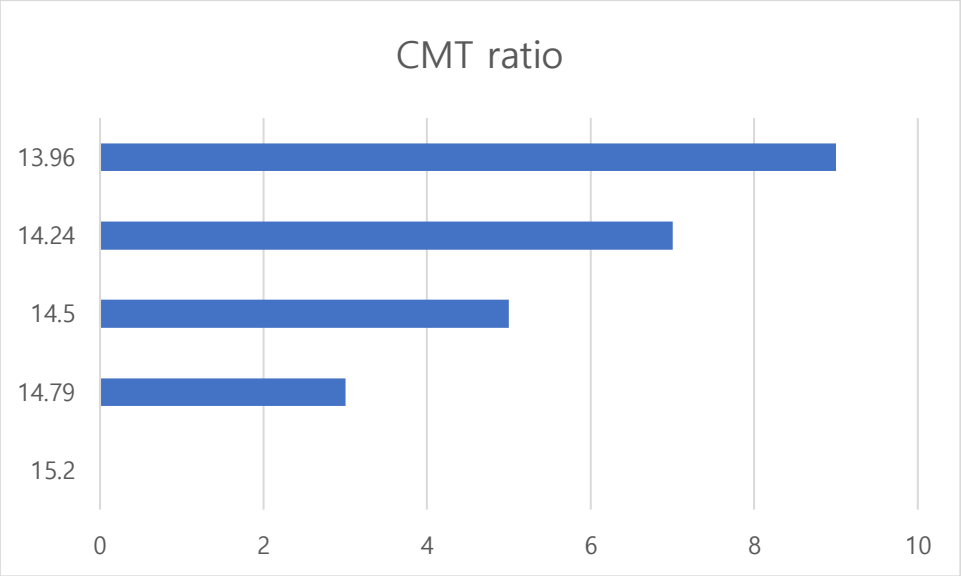
```
void garbage_collection(u32 bank)
{
        s.gc++;
        int i,j,invaild_count,vaild_count,store_value = 0;
        int victim = 0;
        int free_count = 0;
        u32 spare;
        u32 r_buffer[SECTORS_PER_PAGE];
        memset(r_buffer, 0, DATA_SIZE);
        u32 s_point = bank*BLKS_PER_BANK*PAGES_PER_BLK;
        for(i=BLKS_PER_BANK-1;i>=0;i--)
        {
                invaild_count = 0;
                free_count = 0;
                for(j=0;j<PAGES_PER_BLK;j++)
                {
                        if(ppn_inf[s_point+i*PAGES_PER_BLK].type == 'U' && ppn_inf[s_point+i*PAGES_PER_BLK+j].vaild == 'I') invaild_count++;
                        if(ppn_inf[s_point+i*PAGES_PER_BLK+j].vaild == 'F') free_count++;
                }
                if(invaild_count>store_value)
                {
                        victim = i;
                        store_value = invaild_count;
                }
                if(free_count == PAGES_PER_BLK && free_page_ptr_map[bank] != i*PAGES_PER_BLK) free_page_ptr_user[bank] = i*PAGES_PER_BLK;
        }
        vaild_count = 0;
        for(j=0;j<PAGES_PER_BLK;j++)
        {
                if(ppn_inf[s_point+victim*PAGES_PER_BLK+j].vaild == 'V')
                {
                        vaild_count++;
                        nand_read(bank,victim,j,r_buffer,&spare);
                        free_user_page[bank]++;
                        ftl_write_page(spare,r_buffer);
                        s.gc_write += SECTORS_PER_PAGE;
                }
        }
                                                                                315,1
        nand_erase(bank,victim);
        if(store_value != 0)
        {
                free_user_page[bank] += PAGES_PER_BLK - vaild_count;
        }
        for(j=0;j<PAGES_PER_BLK;j++)
        {
                ppn_inf[s_point+victim*PAGES_PER_BLK+j].vaild = 'F';
                ppn_inf[s_point+victim*PAGES_PER_BLK+j].type = 'N';
        }
}
```

Garbage collection 함수입니다. 이 또한 이전과제들과 victim을 선정하는 코드는 동일합니다.

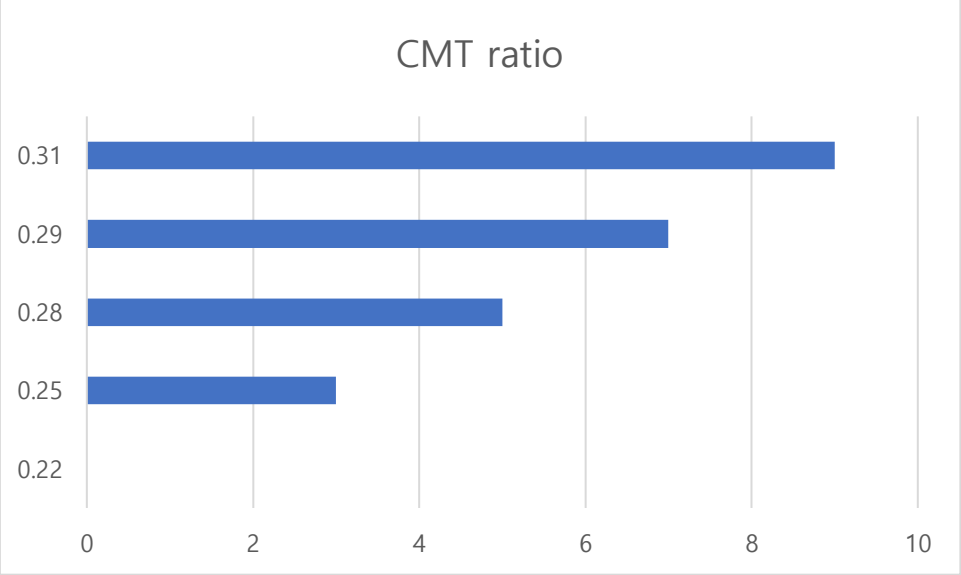
Victim이 선정되었다면 해당 blk에서 vaild한 page들을 읽어와 ftl_write_page를 통해 새로운 page에 기록해줍니다. 이후 nand erase를 수행해주고 free_user_page를 새로 얻은 free page만큼 증가시켜 주고 ppn_table또한 갱신해 줍니다.

<Result Analize>

No Cache 부터 CMT ratio를 9까지 증가시켜 실험해본 결과는 아래와 같았습니다.



WAF



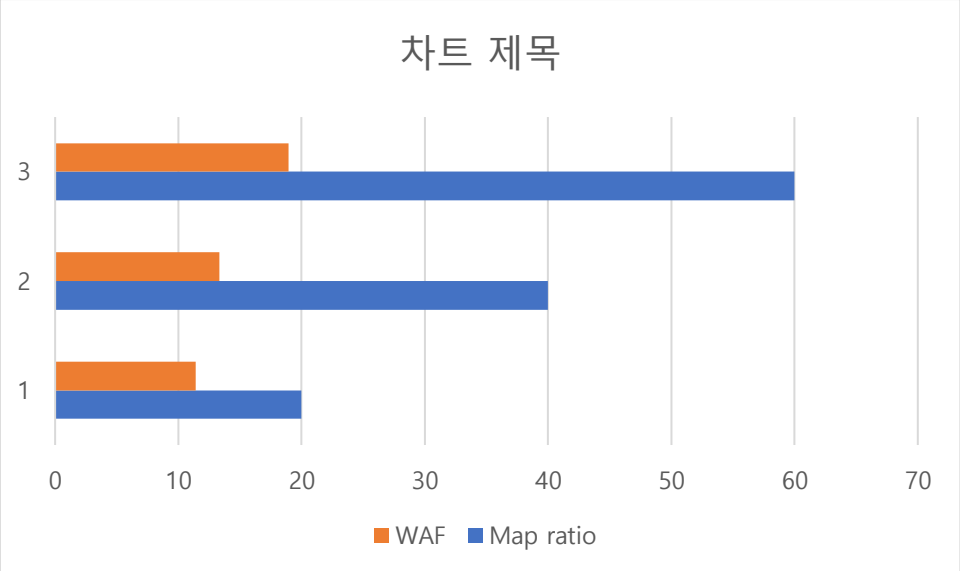
Hit ratio

위 결과에 따르면 cmt ratio가 커질수록 waf값은 줄어들고 hit ratio 값은 늘어나는 것을 확인할 수 있습니다.

WAF	CMT ratio
15.2	No Cache
14.79	3
14.5	5
14.24	7
13.96	9

hit ratio	CMT ratio
0.22	No Cache
0.25	3
0.28	5
0.29	7
0.31	9

다음은 Map ratio를 각각 20,40,60으로 변경하였을 때의 결과입니다. 시간이 지나치게 오래 걸려 부득이하게 5run을 기준으로 작성하였습니다.



hit ratio	Map ratio
0.33	20
0.28	40
0.22	60

Hit ratio는 그래프가 이상이 생겨 값으로 첨부합니다.

위 결과에 따르면 map op영역이 늘어날수록 hit ratio는 줄어들고 waf값은 비약적으로 늘어나는 것을 볼 수 있습니다.