Improving DFTL(LRU to 2Q)

2024.08.28

Presentation by Seonju Koo

pigeon99@dankook.ac.kr

Contents

1. Motivation
2. Structure
3. Code
4. Plan

Motivation

bb.C코드에서 cmt히트 출력하는 부분 발견하여, femu 상에서 사용할 방법 찾아 봄
→ femu 내부에서 opcode, cmd 보내서 출력

```
nvme admin-passthru /dev/nvme0 --opcode=239 --cdw10=9
```

```
/*tpftl*/
if (st->access_cnt == 0) {
    st->cmt_hit_ratio = 0;
} else {
    st->cmt_hit_ratio = (double)st->cmt_hit_cnt / st->access_cnt;
}
// st->joule = st->read_joule + st->write_joule + st->erase_joule;

printf("CMT hit count: %lu\n", st->cmt_hit_cnt);
printf("CMT miss count: %lu\n", st->cmt_miss_cnt);
printf("CMT access count: %lu\n", st->access_cnt);
printf("CMT hit ratio: %lf\n", st->cmt_hit_ratio);
```

Motivation

- **기존의 DFTL cmt히트율**

실험환경: FIO 워크로드 4KB에서 1GB 60초 실행

```
CMT hit count: 40631
CMT miss count: 1422956
CMT access count: 1463587
CMT hit ratio: 0.027761
[FEMU] Log: vSSD0,Statistics print!
```

Fio random r/w 결과

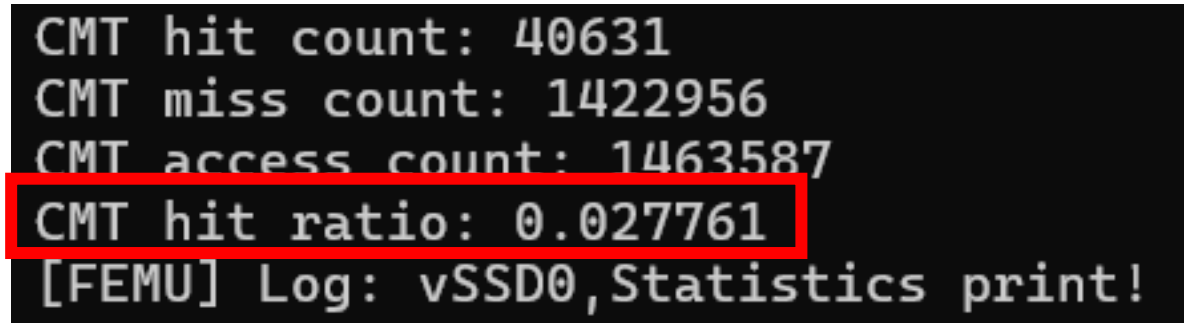
```
CMT hit count: 1265988
CMT miss count: 536411
CMT access count: 1802399
CMT hit ratio: 0.702391
[FEMU] Log: vSSD0,Statistics print!
```

Fio sequential r/w 결과

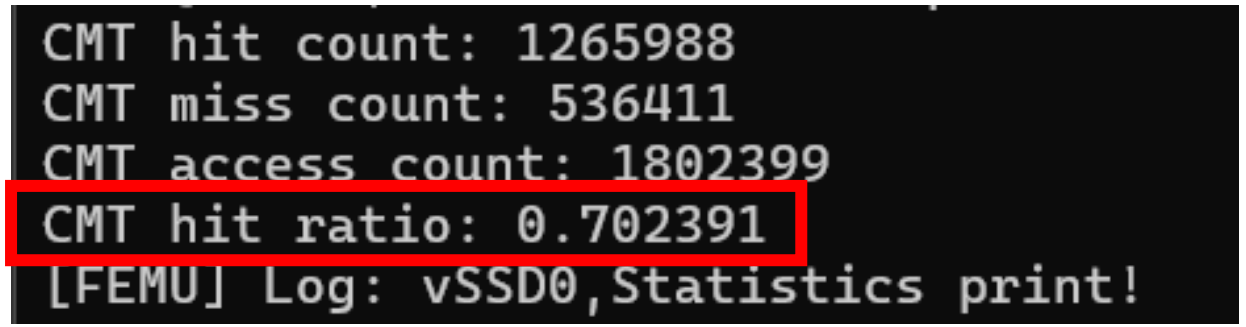
Motivation

- **기존의 DFTL cmt히트율**

실험환경: FIO 워크로드 4KB에서 총 1GB로 60초 실행



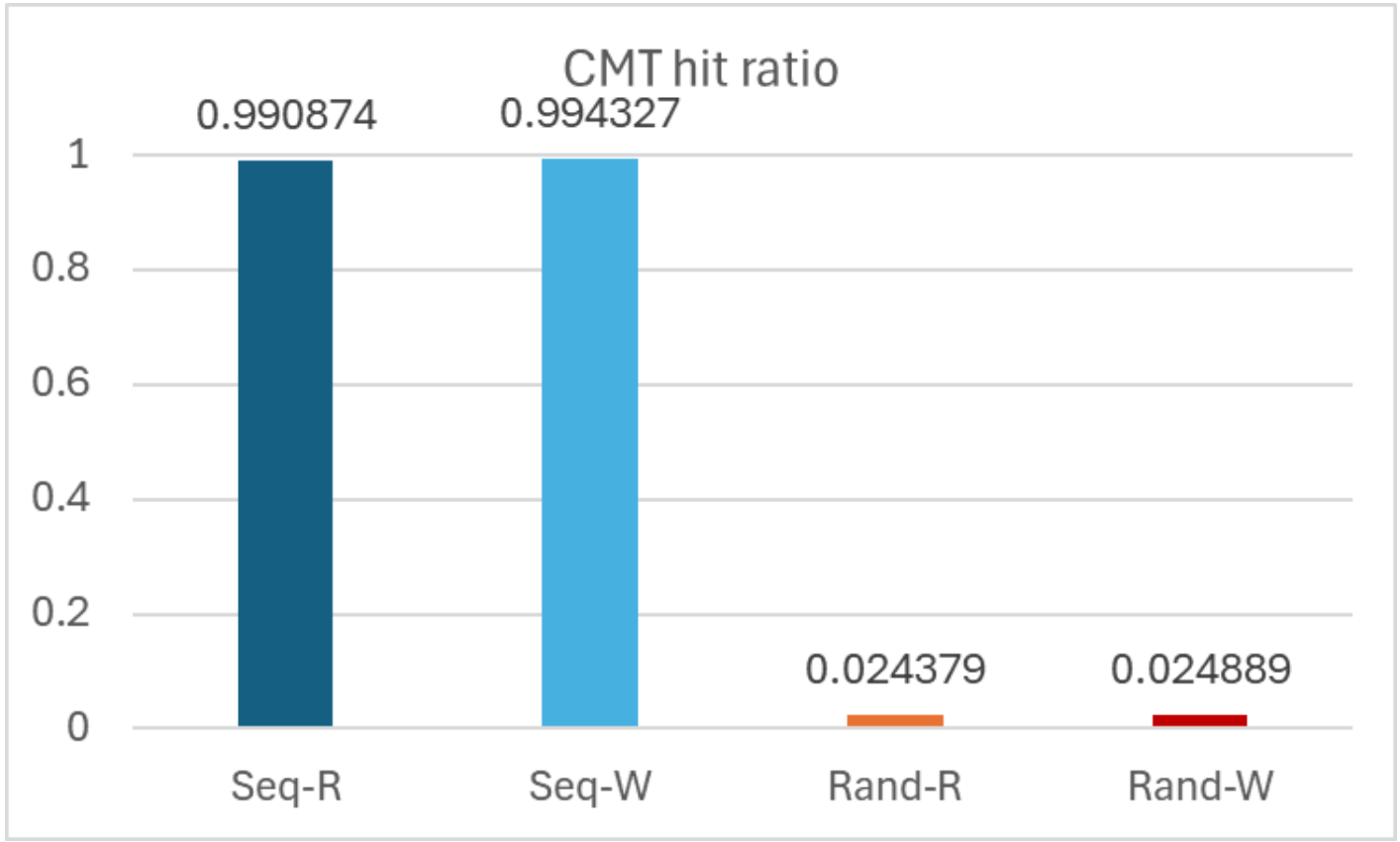
Fio random r/w 결과



Fio sequential r/w 결과

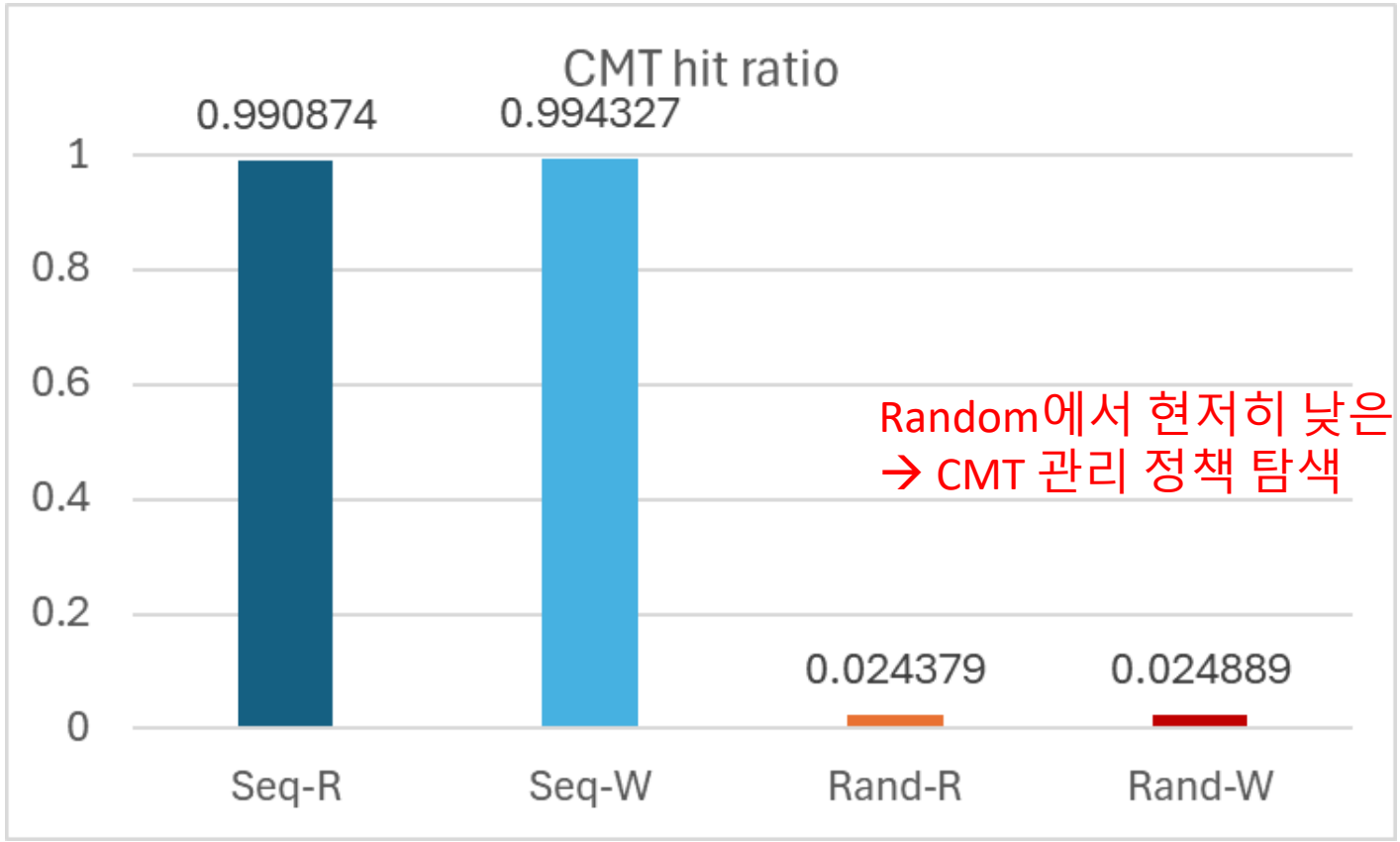
Motivation

- 기존의 DFTL cmt히트율



Motivation

- 기존의 DFTL cmt히트율

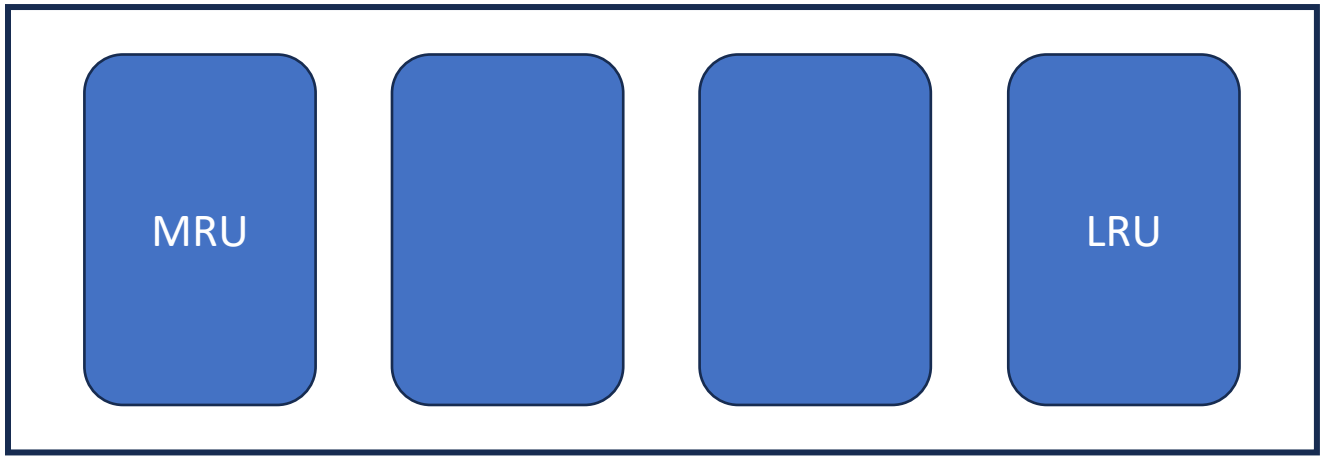


Random에서 현저히 낮은 히트율
→ CMT 관리 정책 탐색

Motivation

- 기존의 DFTL cmt관리 방식 → LRU 알고리즘

LRU_Q

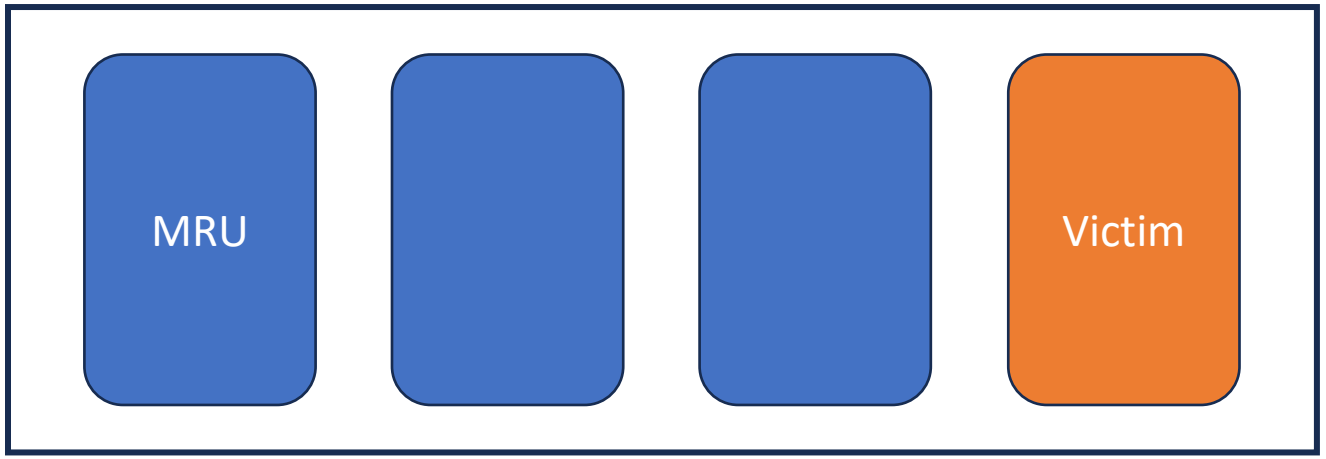


LFU 알고리즘을 함께 적용하면 다양한 워크로드에서 cmt히트율을 높일 수 있을 것이라 생각함

Motivation

- 기존의 DFTL cmt관리 방식 → LRU 알고리즘

LRU_Q

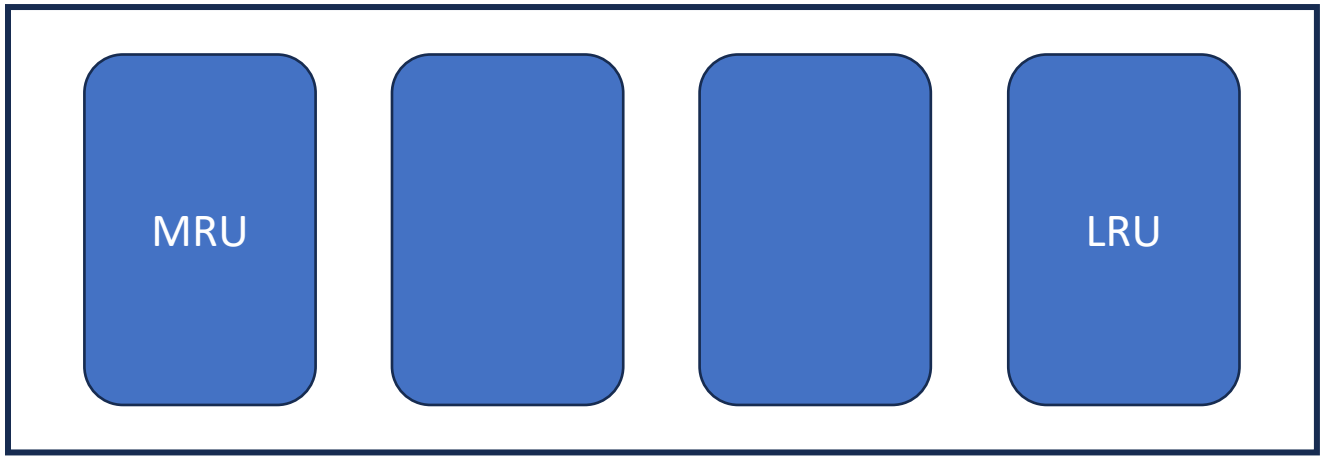


LFU 알고리즘을 함께 적용하면 다양한 워크로드에서 cmt히트율을 높일 수 있을 것이라 생각함

Motivation

- 기존의 DFTL cmt관리 방식 → LRU 알고리즘

LRU_Q



2Q 알고리즘 적용 시도

LFU 알고리즘을 함께 적용하면 다양한 워크로드에서 cmt히트율을 높일 수 있을 것이라 생각함

Motivation

버퍼 크기가 늘어날 때 2Q에서 더 큰 히트율 향상

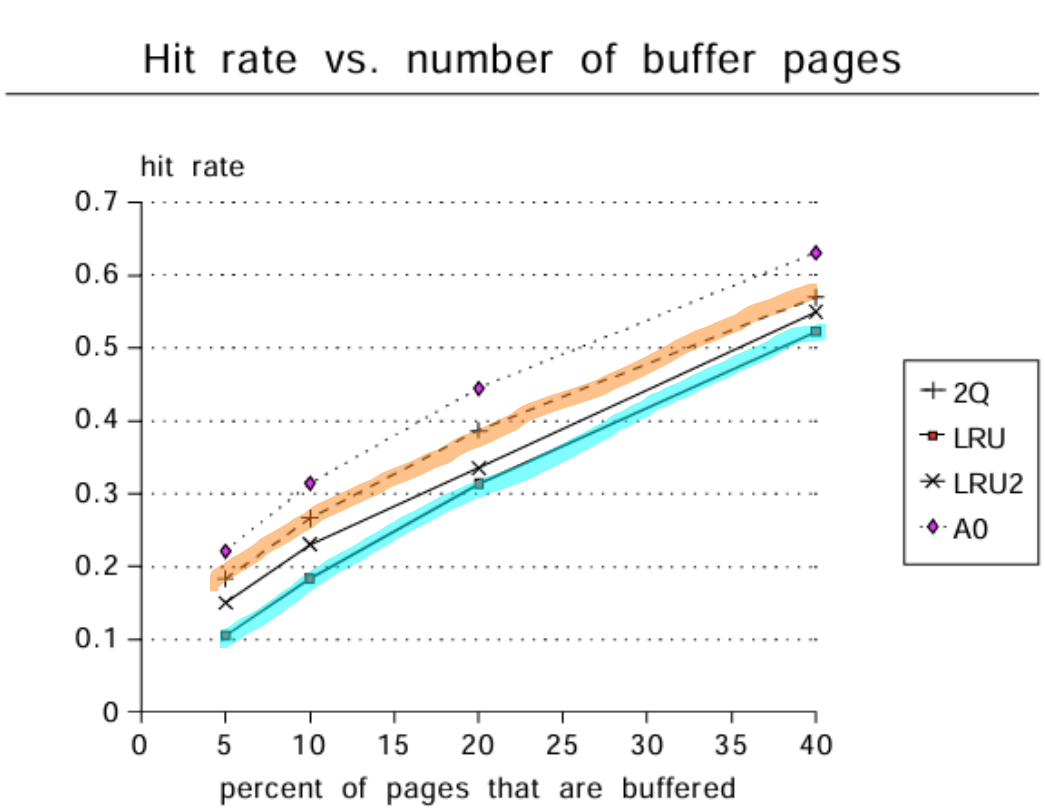


Figure 1: Zipf input comparison with parameter 0.5

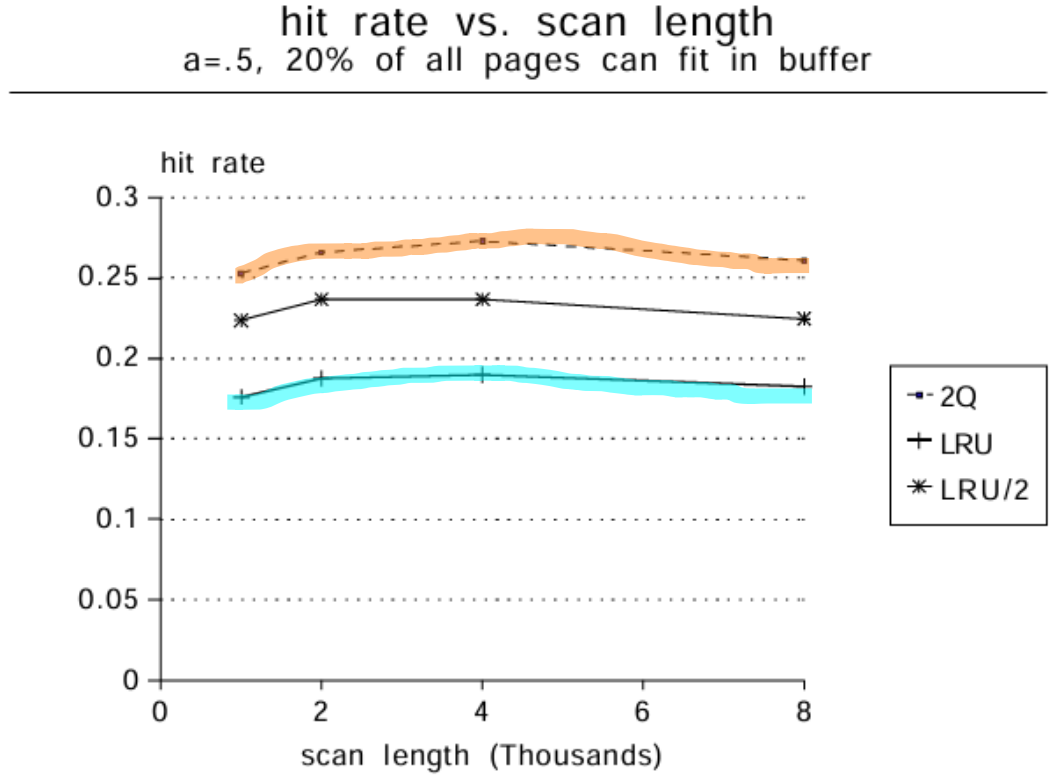


Figure 3: 1/3 Scan Input Mixed with Zipf Input Having Parameter 0.5

Motivation

버퍼 크기가 늘어날 때 2Q에서 더 큰 히트율 향상

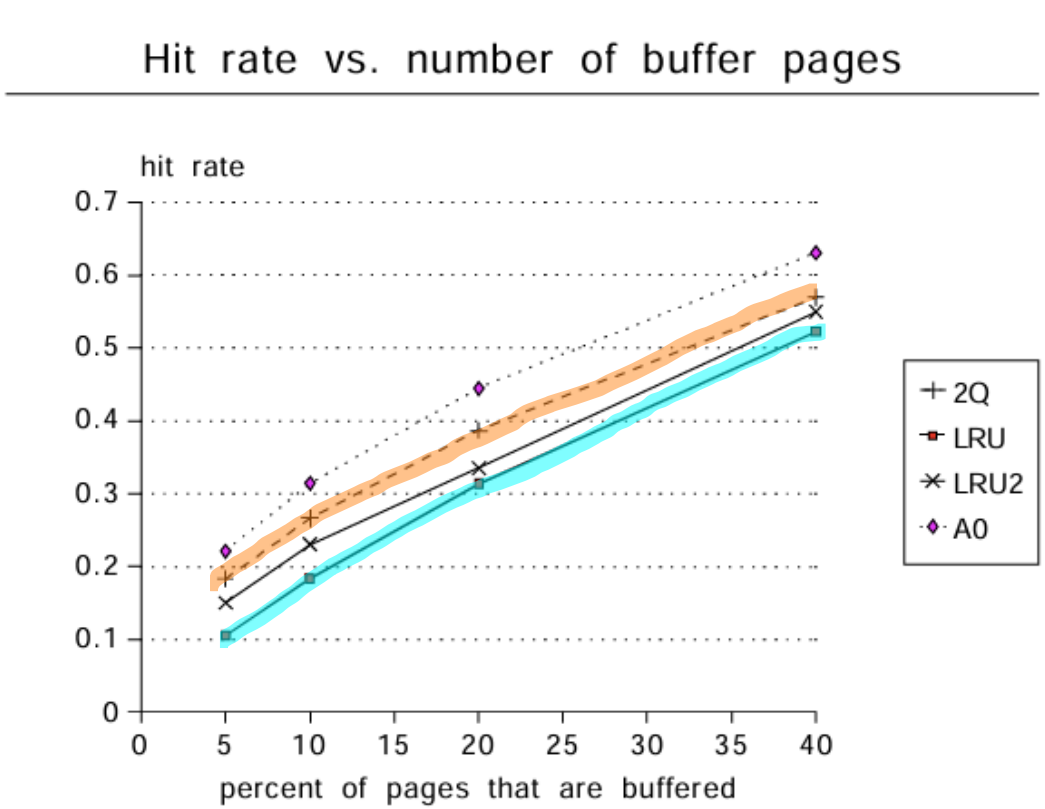


Figure 1: Zipf input comparison with parameter 0.5

Scan 방식 접근: 지나온 것에 다시 접근하지 않기 때문에 최근 entry를 유지하는 LRU는 비효율적

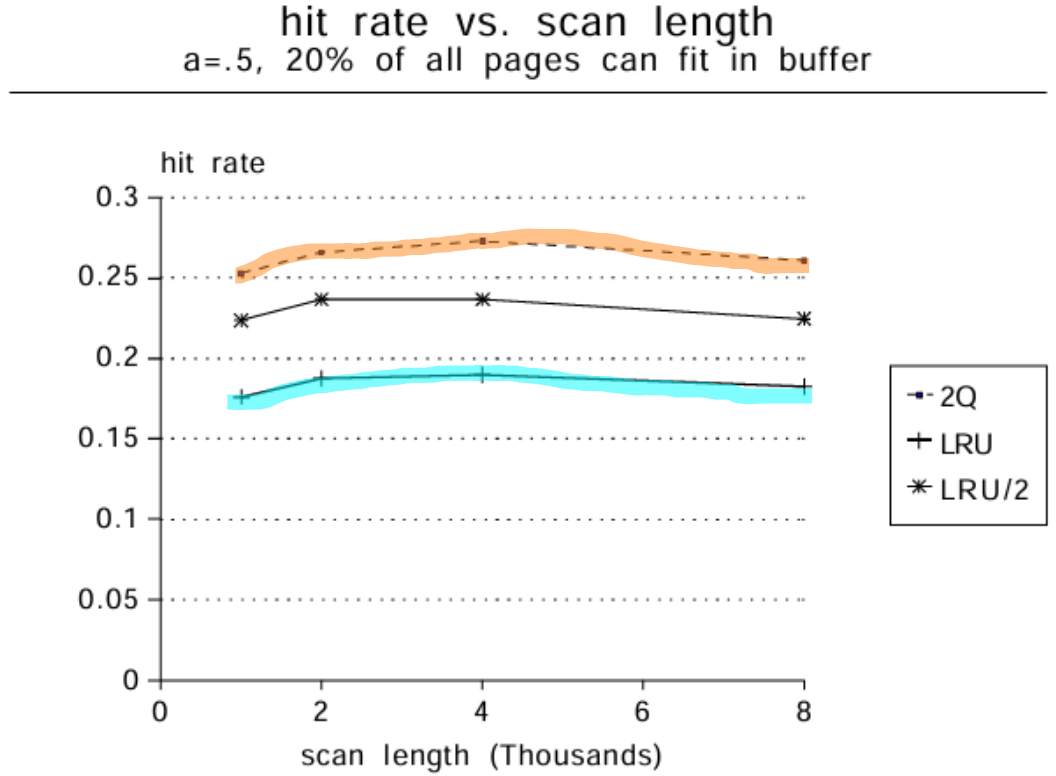
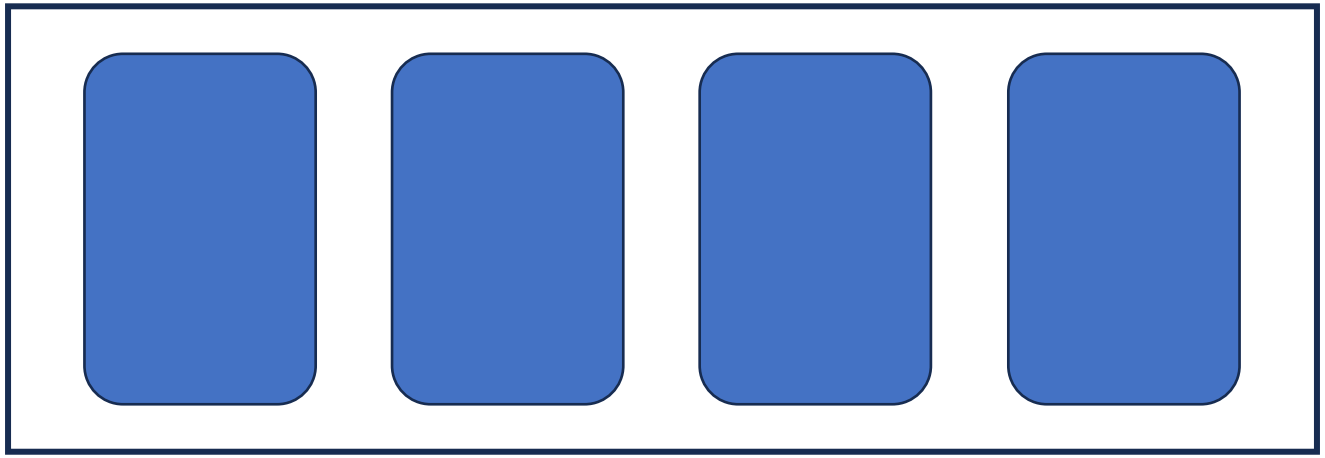


Figure 3: 1/3 Scan Input Mixed with Zipf Input Having Parameter 0.5

Structure

Queue1
(LRU)

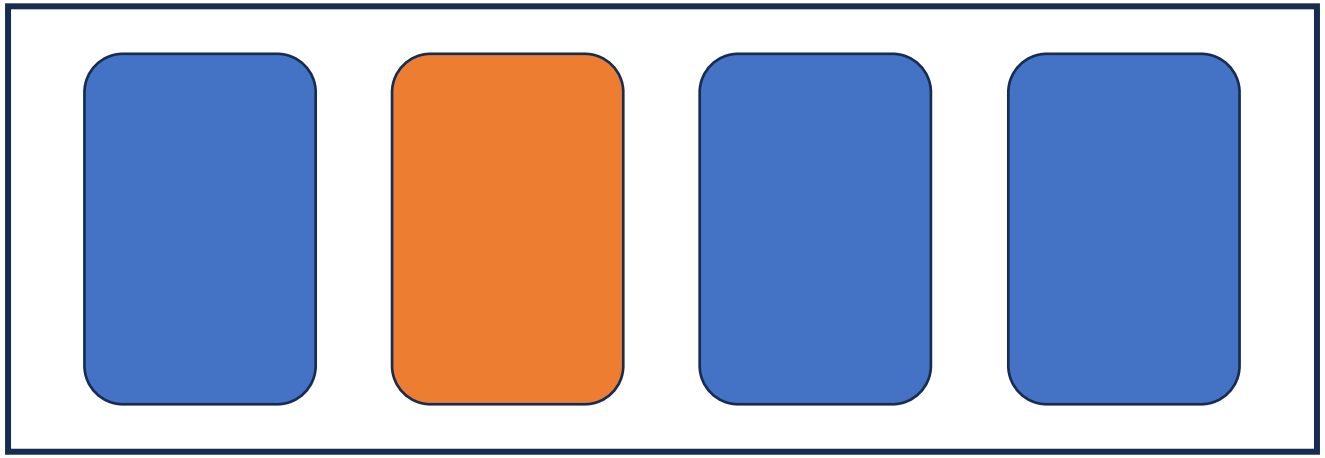


: 최근에 접근한 entry 관리

Queue2
(LFU)

Structure

Queue1
(LRU)



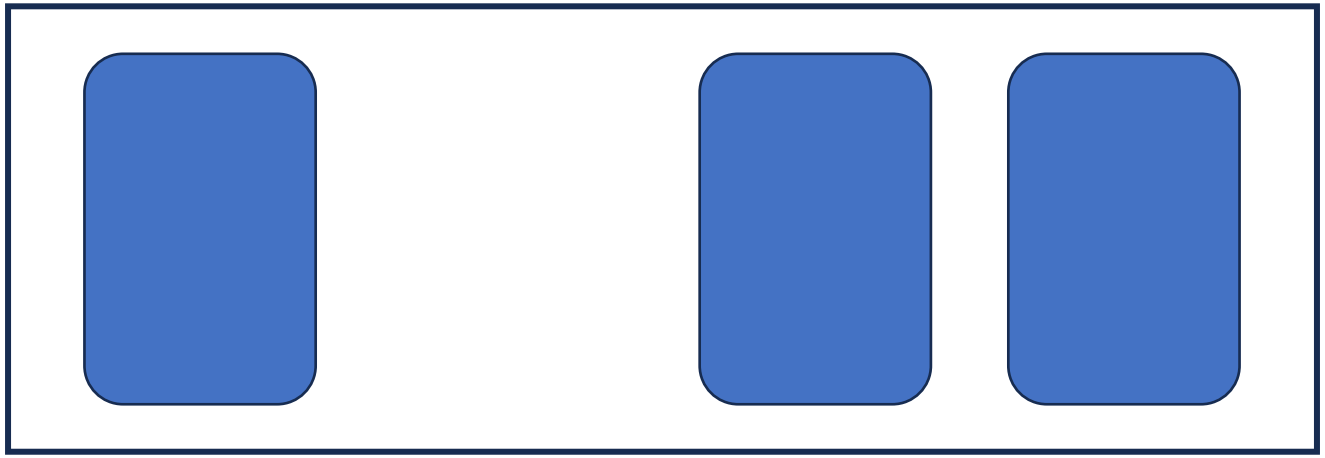
: 최근에 접근한 entry 관리

Queue2
(LFU)

Q1에서 다시 접근된 데이터는
Q2로 이동

Structure

Queue1
(LRU)



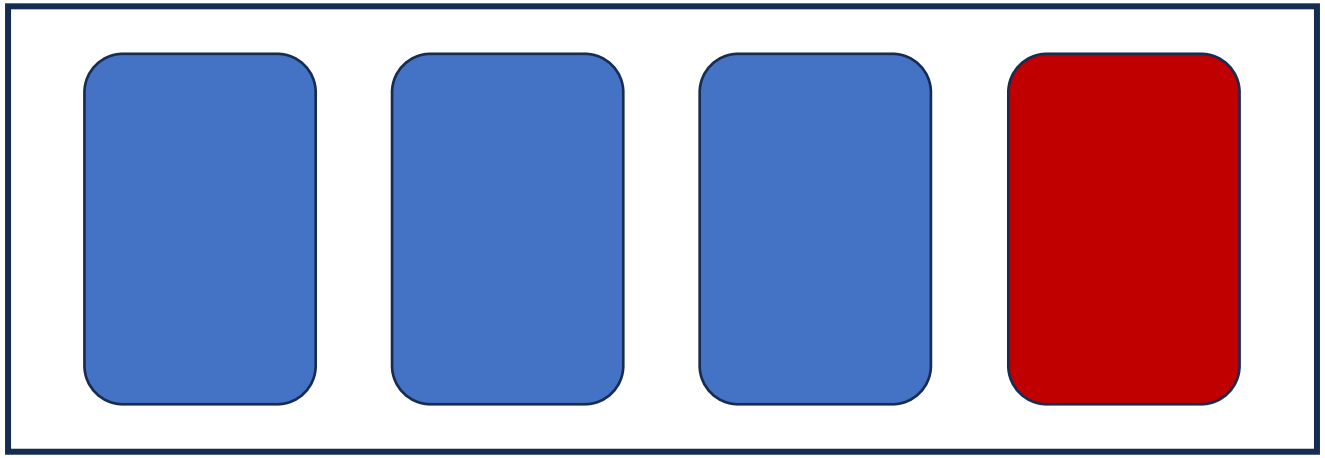
: 최근에 접근한 entry 관리

Queue2
(LFU)

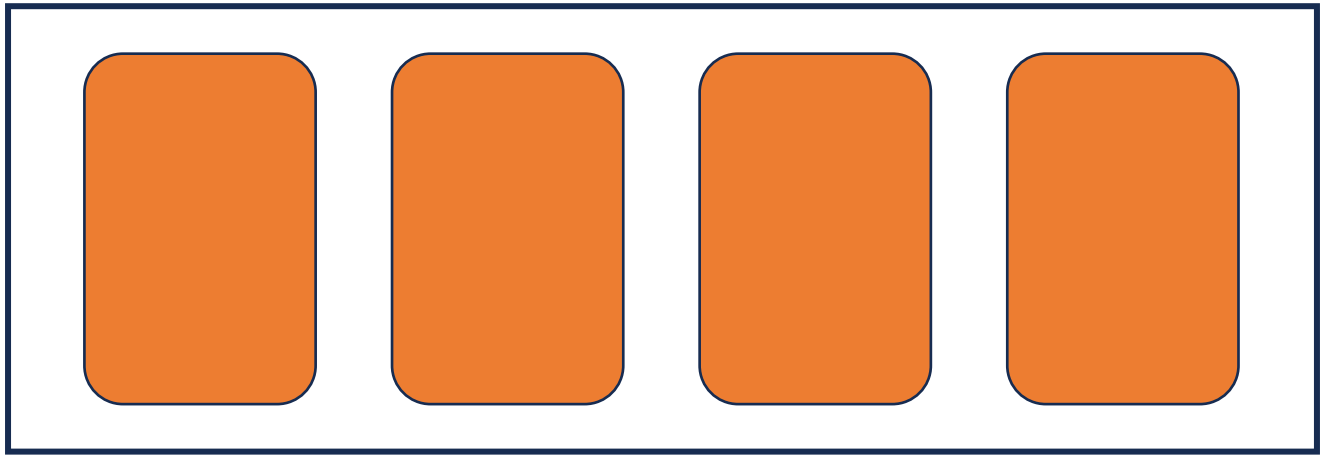
Q1에서 다시 접근된 데이터는
Q2로 이동

Structure

Queue1
(LRU)



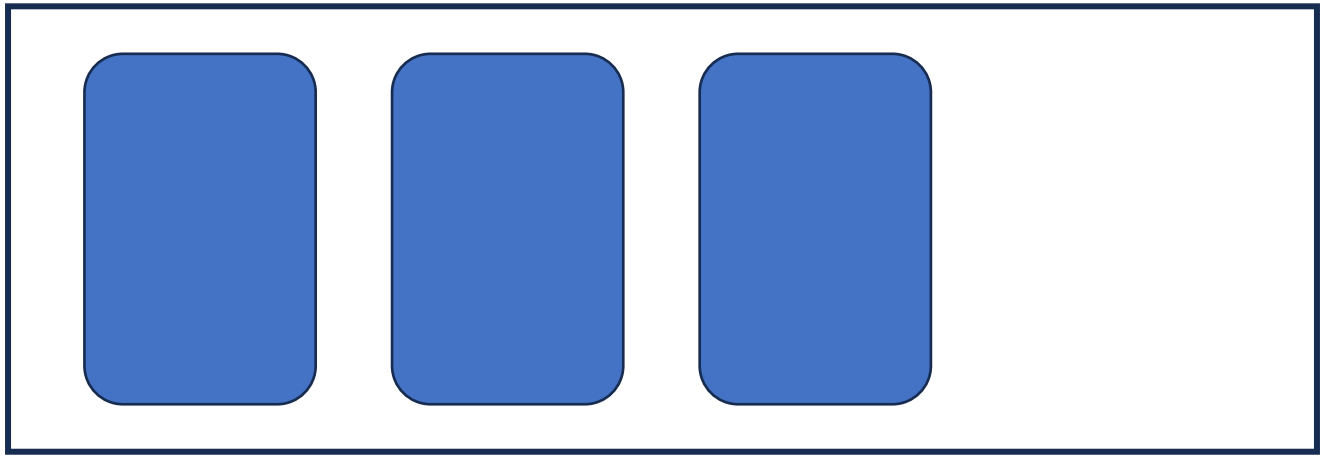
Queue2
(LFU)



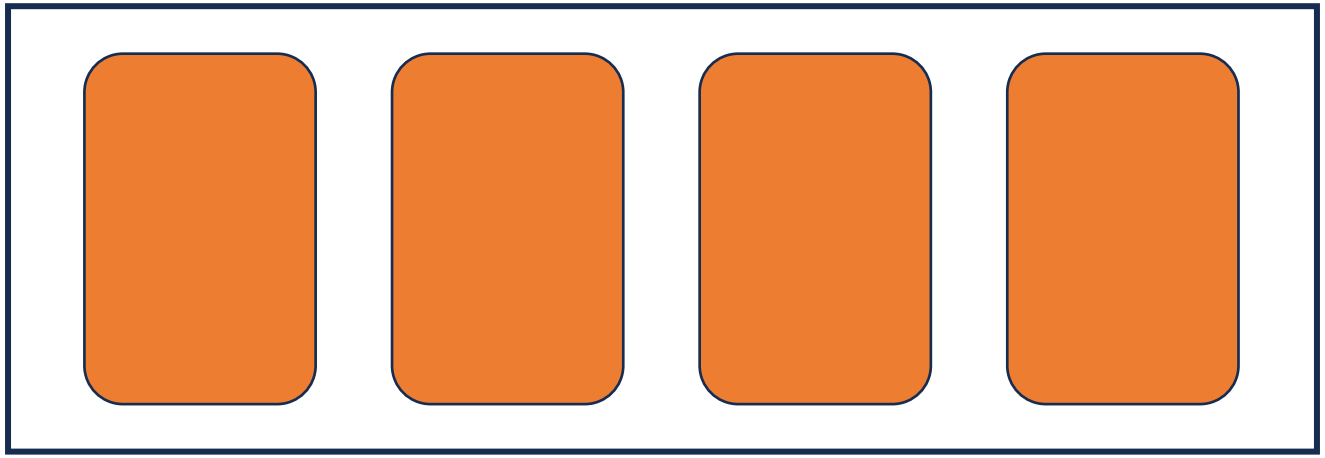
Entry evict시 Q1 우선 선택
(자주 접근한 데이터 유지)

Structure

Queue1
(LRU)



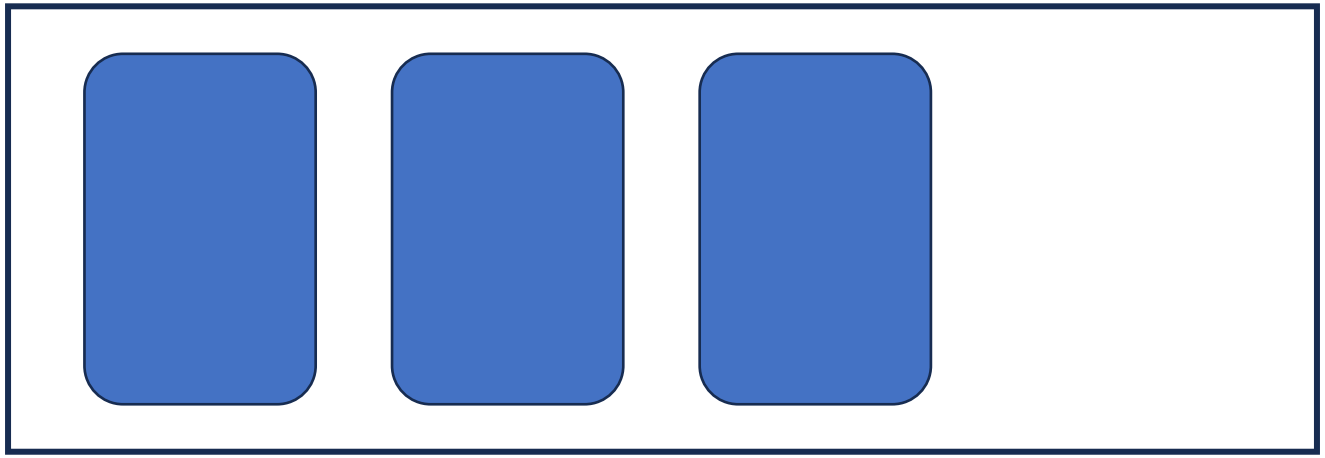
Queue2
(LFU)



Entry evict시 Q1 우선 선택
(자주 접근한 데이터 유지)

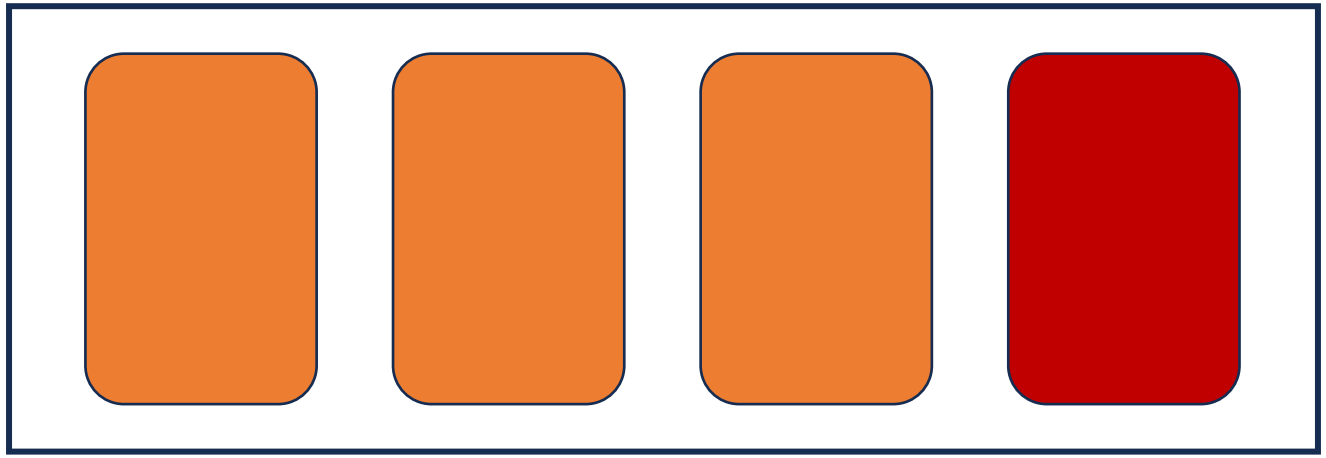
Structure

Queue1
(LRU)



Entry evict시 Q1 우선 선택
(자주 접근한 데이터 유지)

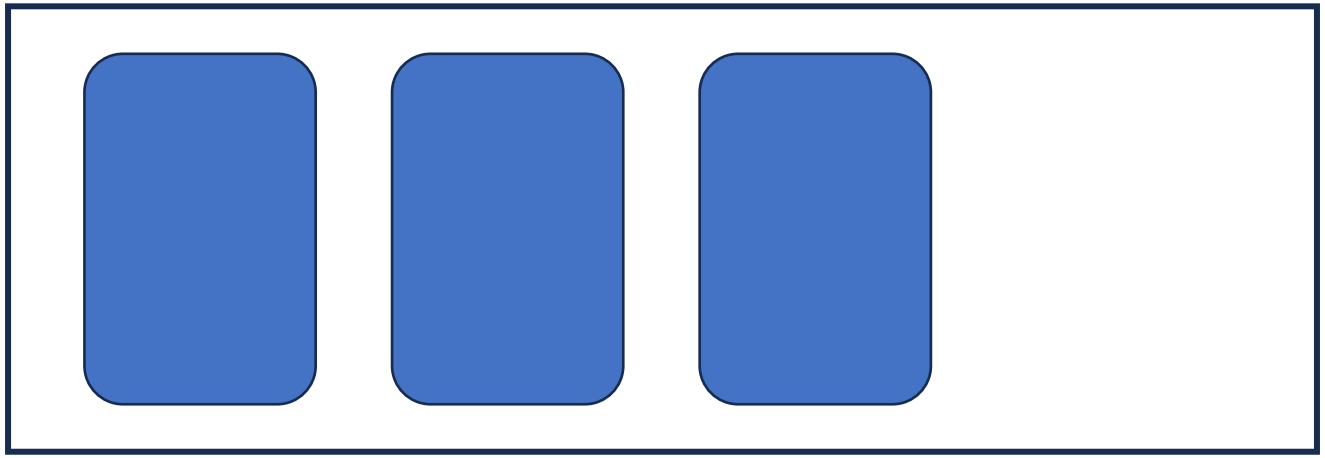
Queue2
(LFU)



Q2 꽉 차면 LRU 방식으로 제거

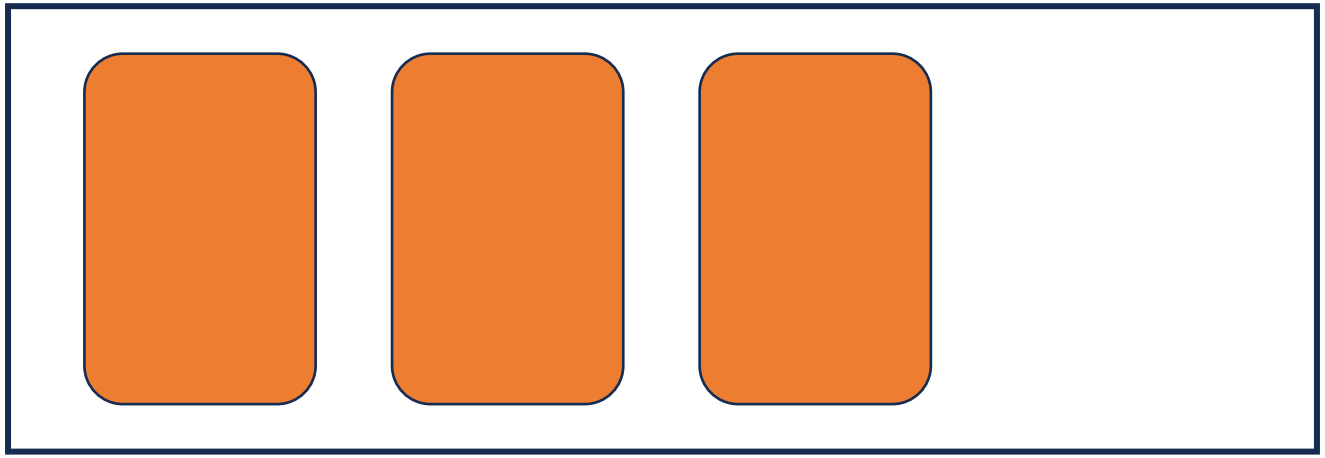
Structure

Queue1
(LRU)



Entry evict시 Q1 우선 선택
(자주 접근한 데이터 유지)

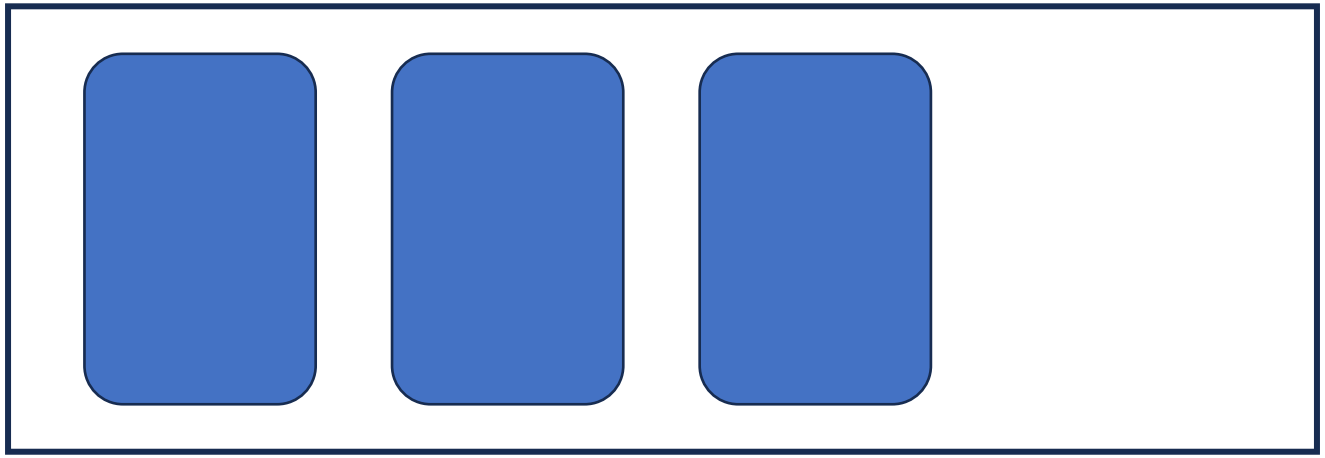
Queue2
(LFU)



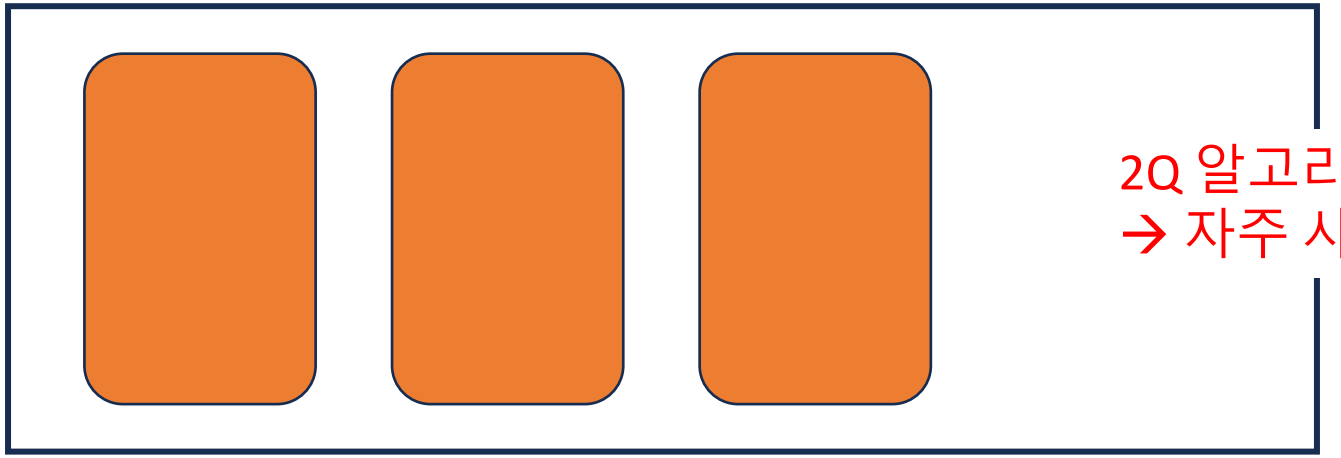
Q2 꽉 차면 LRU 방식으로 제거

Structure

Queue1
(LRU)



Queue2
(LFU)



2Q 알고리즘
→ 자주 사용하는 entry를 cmt에 더 유지시킴

Code

cmt_mgmt: 구조체에 Q1, 2 추가

```
struct cmt_mgmt {
    struct cmt_entry *cmt_entries;
    struct hash_table ht;
    QTAILQ_HEAD(, cmt_entry) free_cmt_entry_list;
    QTAILQ_HEAD(, cmt_entry) A1_list;
    QTAILQ_HEAD(, cmt_entry) Am_list;
    QTAILQ_HEAD(, TPnode) TPnode_list;
    uint64_t tt_entries;
    uint64_t tt_TPnodes;
    int free_cmt_entry_cnt;
    int used_cmt_entry_cnt;
    int counter;
};
```

cmt_entry: 구조체에 frequency 추가

```
typedef struct cmt_entry {
    uint64_t lpn;
    uint64_t ppn;
    int dirty;
    // int hotness;
    QTAILQ_ENTRY(cmt_entry) entry;
    bool prefetch;
    uint64_t next_avail_time;
    struct cmt_entry *next;     /* for hash */
    int frequency;  // LFU에 사용
    time_t last_access_time;  // LRU에 사용
} cmt_entry;
```

Code

Insert_entry_to_cmt함수: 먼저 Q1에 넣음

```
QTAILQ_INSERT_HEAD(&cm->A1_list, cmt_entry, entry);
```

Evict_entry_from_cmt함수: Q1 마지막에 엔트리 없으면 Q2의 마지막에서 선택함

```
cmt_entry = QTAILQ_LAST(&cm->A1_list);
if (cmt_entry == NULL) {
    cmt_entry = QTAILQ_LAST(&cm->Am_list);
}
if (cmt_entry == NULL) {
    ftl_err("no entry to evict");
}
```

Code

*cmt_hit: Q1에서 히트 시 Q2로 옮김

```
if (cmt_entry != NULL) {
    if (QTAILQ_FIRST(&curTP->cmt_entry_list) != cmt_entry) {
        QTAILQ_REMOVE(&curTP->cmt_entry_list, cmt_entry, entry);
        QTAILQ_INSERT_HEAD(&curTP->cmt_entry_list, cmt_entry, entry);
    }
    QTAILQ_REMOVE(&cm->A1_list, cmt_entry, entry);
    QTAILQ_INSERT_TAIL(&cm->Am_list, cmt_entry, entry);
}

return cmt_entry;
```

Code

*cmt_hit: Q1에서 히트 시 Q2로 옮김

```
if (cmt_entry != NULL) {
    if (QTAILQ_FIRST(&curTP->cmt_entry_list) != cmt_entry) {
        QTAILQ_REMOVE(&curTP->cmt_entry_list, cmt_entry, entry);
        QTAILQ_INSERT_HEAD(&curTP->cmt_entry_list, cmt_entry, entry);
    }
    QTAILQ_REMOVE(&cm->A1_list, cmt_entry, entry);
    QTAILQ_INSERT_TAIL(&cm->Am_list, cmt_entry, entry);
}

return cmt_entry;
```

- 구현 문제로 인한 segmentation falut 발생

Plan

- 코드 문제 해결하여 cmt히트 측정
- 실시간 random/sequential 워크로드 판단 알고리즘 탐색
  - Markov-based technique(전이 확률 기반)
  - Dynamic Time Warping(이상 감지 기법)
  - 가중 표준 편차 사용한 변화 감지
- Random 처리하는 방법 구상

Thank you!
Q & A ?

2024.08.28
Presentation by Seonju Koo
pigeon99@dankook.ac.kr