❓

Disk Manager 3주차 분석 QnA

`storage/disk_manager.c`의 `disk_reserve_sectors` 함수에서 궁금한 점이 있습니다.

해당 함수에서 오류 발생에 따른 처리 로직 (4302 라인) `error` 레이블 부분입니다.

질문 제시를 위해 이해한 상태를 기술했습니다.

(과정 중에 틀린 부분이 있는지도 봐주시면 감사드리겠습니다!)

현재까지 이해한 사항 입니다.

1) 디스크 섹터 예약 진행 도중에 오류가 발생하여 `error` 레이블로 진입하는 경우, 예약된 섹터를 예약 이전의 상태로 돌려 놓아야 하는 것으로 이해했습니다.

2) `disk_reserve_from_cache` 함수 → `disk_reserve_from_cache_vols` 함수에서 `disk_reserve_from_cache_volume` 함수의 반복되는 호출로 `n_cache_vol_reserve`를 증가시켜 예약하려는 볼륨 수를 알게 되고, 이 때 예약할 볼륨에 대한 volid와 섹터 수를 알게 되는 것 (`context->cache_vol_reserve[context->n_cache_vol_reserve].volid`와 `context->cache_vol_reserve[context->n_cache_vol_reserve].nsect = nsects`)으로 이해했습니다.

3-1) 이 때, `disk_reserve_sectors` 함수의 `error` 레이블을 보면 목적이 `Temporary`인 경우에만 `if` 문으로 분기되어 `qsort` 함수 호출 → `disk_unreserve_ordered_sectors_without_csect` 함수 호출 과정이 이뤄지는 것을 볼 수 있었습니다. (4308 라인 ~ 4318 라인)

```
4302    error:
4303      nreserved = (int) (context.vsidp - reserved_sectors);
4304      if (nreserved > 0)
4305        {
4306          int iter_vsid;
4307
4308          if (purpose == DB_TEMPORARY_DATA_PURPOSE)
4309            {
4310              /* nothing was logged. we need to revert any partial allocations we may have made. */
4311              bool save_check_interrupt = logtb_set_check_interrupt (thread_p, false);
4312
4313              qsort (reserved_sectors, nreserved, sizeof (VSID), disk_compare_vsids);
4314              if (disk_unreserve_ordered_sectors_without_csect (thread_p, purpose, nreserved, reserved_s
4315                {
4316                  assert_release (false);
4317                }
4318              (void) logtb_set_check_interrupt (thread_p, save_check_interrupt);
4319            }
```

3-2) 목적이 `Temporary` 든, `Permanent`이든 예약된 섹터들은 예약되기 이전의 상태로 만들어야 하기 때문에, 목적에 관계 없이 제시된 사진의 (4324 라인 ~ 4336 라인) 볼륨 당 예약된 섹터 수를 감소 시키는 과정을 거치는 것으로 이해했습니다.

```
4321      /* we'll need to remove reservations for the rest of sectors (that were not allocated from dis
4322       * let's avoid removing the reservations for the ones we allocated from disk and rollbacked (t
4323       * removed from cache too) */
4324      for (iter_vsid = 0; iter_vsid < nreserved; iter_vsid++)
4325        {
4326          /* search vsid in volumes */
4327          for (iter = 0; iter < context.n_cache_vol_reserve; iter++)
4328            {
4329              if (reserved_sectors[iter_vsid].volid == context.cache_vol_reserve[iter].volid)
4330                {
4331                  context.cache_vol_reserve[iter].nsect--;
4332                  break;
4333                }
4334            }
4335          assert (iter < context.n_cache_vol_reserve);
4336        }
4337    }
```

3-3) `disk_cache_free_reserved` 함수에서 목적에 맞는 `Lock`을 취득하여 볼륨 당 가용 섹터 수 (기존에 예약해뒀던 섹터 수를 이용)를 원래대로 돌리고 로그를 찍는 과정을 거치는 작업을 진행하는 것으로 이해했습니다.

여기서 질문이 생겼습니다.

1) 목적이 `Temporary`인 부분에서만 `qsort` 함수를 이용하여 볼륨 아이디로 오름차순 정렬 (볼륨 아이디가 동일하다면 섹터 아이디로 오름차순 정렬)을 진행하게 되던데, 이것이 혹시 `disk_reserve_from_cache` 함수에서 `Permanet` 타입 볼륨에서 먼저 예약 후 `Temporary` 타입 볼륨에서 예약하는 과정과 관련이 있나요?

`disk_unreserve_ordered_sectors_without_csect` 함수 내에서 반복문을 진행할 때 인접한 볼륨끼리는 연속되게 하여 볼륨 당 예약된 섹터 수 (`context.cache_vol_reserve[context.n_cache_vol_reserve].nsect`와 `context.cache_vol_reserve[context.n_cache_vol_reserve].volid`)를 기록하기 위한 것으로 이해했는데, 그렇다면 목적이 `Temporary`일 때는 인접한 볼륨끼리는 연속된 볼륨 아이디로 배치가 안 되어서 `qsort` 함수를 호출한 것 같았습니다.
(여기서 언급한 연속된이란 1, 2, 3, 4와 같은 것이 아니라 1, 1, 3, 3, 3, 4, 4, 2, 2와 같은 형태를 말하고 싶었습니다.)

> **Mentor**
> 모든 Purpose에 대해서 `disk_reserve_from_cache` 함수에서 Permanent 타입(이하 P타입) 볼륨 먼저 예약 후 Temporary 타입(이하 T타입) 볼륨을 예약하지 않습니다.
>
> 단, Temporary 목적으로 Temporary Data를 저장할 때는 디스크의 공간을 p/t (permanent type / temporary purpose), t/t (temporary purpose / temporary purpose) 순서로 이뤄집니다.
>
> 이 때 Error Phase에서 정렬 후 un-reserve하는 부분은 p/t에 대한 것입니다.
>
> 정렬을 수행하는 이유는 reserve 할 때, 섹터마다 사용가능한 page수가 다르기 때문입니다.
>
> 예를 들면, 26페이지를 예약해야하고, [1섹터] [14페이지] 예약가능, [2섹터] [36페이지] 예약가능이라고 가정해 봅시다. 그러면 예약되는 순서는 2섹터가 예약되고 나중에 13페이지를 예약할 때 1섹터를 예약하겠죠? 그럼 [2,1] 순서로 섹터예약이 되는 겁니다.
>
> 하지만 예약 해제할때는 오름차순 혹은 내림차순으로 정렬되어 있어야 예약해제하
> 기에 비용이 적게 사용되니 정렬 후 예약해제를 하는 것입니다

2) `disk_reserve_from_cache_volume`에서 `context.cache_vol_reserve[context.n_cache_vol_reserve].nsect`와 `context.cache_vol_reserve[context.n_cache_vol_reserve].volid` 를 받아내는 것으로 알고 있는데 목적이 `Temporary`인 경우에는 왜 `disk_unreserve_ordered_sectors_without_csect` 함수에서 이들을 다시 기록을 하게 되는 것인지 궁금합니다. (4692 라인 ~ 4693 라인)

> **Mentor**
>
> `disk_unreserve_ordered_sectors_without_csect` 함수 내 지역변수 context에 내용을 복사하여서, 예약해제 하기 위해서입니다. 해당 행렬을 계속 사용하기에는 multithread 환경에서 데이터 변경의 위험도 있습니다.

3) `disk_unreserve_ordered_sectors_without_csect` 함수의 마지막 반복문에서 (4700 라인 ~ 4711 라인) `disk_unreserve_sectors_from_volume` 함수를 호출하여 예약된 섹터의 반환을 볼륨에도 반영하는 것으로 이해를 했고, 이 때 함수 내에서 `disk_stab_iterate_units`를 호출하면서 `disk_stab_unit_unreserve`를 통해 `stab`에도 반영하는 것을 보았습니다. 이와 같은 `stab`의 반영은 목적이 `Temporary` 일때만 이뤄지는 것을 확인할 수 있었습니다. 목적이 `Permanent`일 때는 어째서 반환된 `sector`들을 `stab`에 반영하는 코드를 찾을 수 없는 것인가요?

제공해주신 `Disk Manager` 분석 문서의 예약 해제를 살펴보면 `유닛단위로 예약을 해제할 때의 루틴은 영구/임시 목적에 따라 다른데, 임시목적의 경우는 즉시 예약해제를 하지만 영구목적의 경우는 log_append_postpone()을 통해 트랜잭션이 commit(log_commit())될 때까지 해당 작업을 미룬다.`라는 것이 이와 관련이 있는 것인지도 궁금합니다.

혹시 위의 `log_append_postpone`의 복구 정보가 `MVCC`와 관련이 있는 것인지, 만약 그렇다면 `MVCC` 덕분에 목적이 `Permanent`일 때는 `stab`에 즉시 반영을 하지 않아도 되는 것인지도 궁금합니다.

```
4699
4700    for (index = 0; index < context.n_cache_vol_reserve; index++)
4701      {
4702        /* unreserve volume sectors */
4703        context.nsects_lastvol_remaining = context.cache_vol_reserve[index].nsect;
4704
4705        error_code = disk_unreserve_sectors_from_volume (thread_p, context.cache_vol_reserve[index].vo
4706        if (error_code != NO_ERROR)
4707          {
4708            ASSERT_ERROR ();
4709            return error_code;
4710          }
4711      }
4712
4713    return NO_ERROR;
4714  }
```

> **Mentor**
>
> Disk Manager에서는 un-reserve를 곧바로 하지 않습니다. Temporary 목적은 로그를 기록하지 않아서, 바로 un-reserve 함수를 호출합니다. 하지만, Permanent 목적의 경우에는 `log_sysop_start` 함수를 통해서 로그 기록을 합니다.
>
> 따라서 에러가 난다면, un-reserve를 실행하지 않고 로그를 되돌려서 실행했던 작업을
> 취소합니다.
>
> MVCC와는 관련이 없습니다.