더블 라이트 버퍼

Double Write Buffer

문서 정보

본 문서는 CUBRID 소스코드를 기반으로 하여서 Double Write Buffer의 구조와 DB가 사용하는 매커니즘에 대해서 설명한다.

문서 변경 내역

버전	일자	이력사항	작성자
11.1.0.0272-992b024	2021.07.02		김명규(eido5@cubrid.com)

목차

관련 파일

double_write_buffer.c

1. 개요

- DB 시스템은 Page replacement를 위한 강제적인 flush부터 주기적으로 flush하는 작업을 진행한다. 이러한 flush는 Double Write Buffer를 사용하지 않고 flush 하거나 Double Write Buffer 사용하여 flush 하는 방법이 있다.
- DWB를 사용해서 flush 하기로 결정한 DB 시스템은 Double Write Buffer(이하 **DWB**)를 사용하여, System crash가 발생해서 일어난 Partial Write에 대해서 Page를 recovery를 할 수 있다.
- **Partial Write**이란, OS의 File I/O 단위인 block과 DB시스템의 Page단위의 크기가 같지 않기 때문에 발생 가능한 문제이다. (일반적으로 DB Page의 크기 더 크다.) 정상적인 DWB 동작에는 발생하지 않지만, DWB에서 DWB volume 혹은 DB로 Page를 flush 하는 도중 system crash가 발생하게 된다면, DB의 페이지의 일부분만(partial) write가 일어나는 상황이 발생하고 그 상태를 partial write이라고 부르게 된다.
- DB 시스템이 DWB를 사용하는 방식에는 크게 **Page를 저장할 공간 탐색 및 저장**, **특정 개수의 Page를 disk로 한번에 flush, Partial write가 일어난 DB 복구**로 구분된다.
- 2장에서는 DWB의 구조, 3장에서는 Page를 저장할 slot 탐색 및 저장, 4장에서는 DWB Flush, 5장에는 Corrupted Data Page Recovery에 대해서 알아보겠다.

NOTATION

- Flush: 일반적으로 Memory에 있는 Data를 Disk에 쓰고 동기화하는 행위
- Buffer pool: 메모리에 있는 Data를 Page로 관리하는 Data Cache 영역

2. DWB 구조

- 위 문서에서 DWB라고 부르는 것은 메모리에 할당되어 있는 DWB block들을 의미한다. 이러한 Block은 Slot들로 구성되어 있다.
- DWB volume은 disk에 저장되어있는 DWB의 단일 블록이다.
- Block은 매크로에서 개수는 최소 1개부터 32개까지[**DWB_MIN_BLOCKS, DWB_MAX_BLOCKS**] 크기는 0.5M에서 32M로 제한되어있다 **[DWB_MIN_SIZE, DWB_MAX_SIZE]**. 시스템 파라미터로서는 기본적으로 2M의 크기(**PRM_ID_DWB_SIZE**)로 2개(**PRM_NAME_DWB_BLOCKS**)의 Block이 설정되어있다.
- Slot의 총 개수는 기본적으로 256개(DWB size/size of io_page)

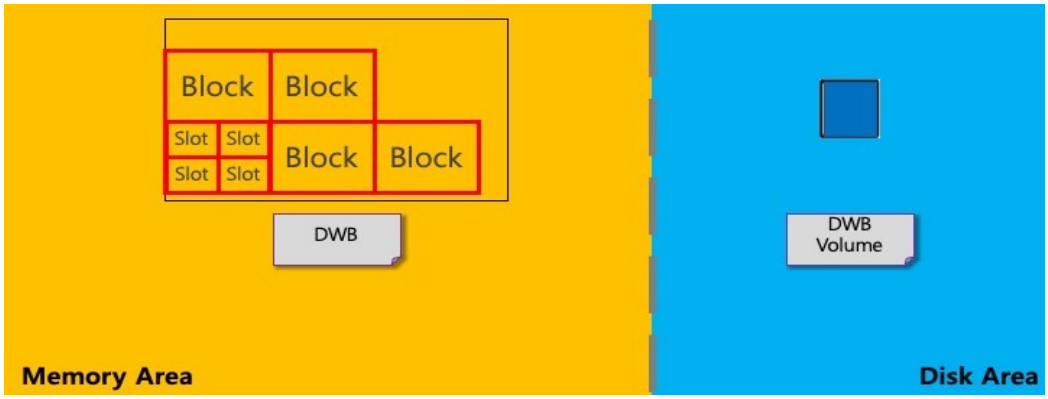

Figure 1 The Structure of DWB

2.1 DWB block, DWB slot

- DWB는 Block과 전역 변수 그리고 Block은 지역변수와 Slot으로 이루어져있다.
  - 전역 변수에는 position_with_flags라는 변수가 있는데, 현재 Page를 저장해야 할 Block과 Slot의 index, 현재 Block 혹은 전체 DWB의 상태를 나타낸다.(DWB 생성 유무, 현재 해당 DWB의 구성요소 변경 상황)
  - 또한 전역변수에는 챕터 4.2에서 다루는 daemon이 fsync를 주기적으로 호출할 때 사용하는 단일 block 포인터 file_sync_helper_block 가 존재한다.
  - 각 Block에서 지역변수 write_buffer라는 포인터를 가지고 있다. 블록 내 모든 Slot들이 참조하여 실제 Page의 내용이 저장된다.
  - 각 Block의 지역변수에 '**wait_queue**'라는 이름의 queue가 있고, 이 queue는 slot 탐색 중 대기가 일어나는 thread를 저장하는 역할을 한다.
- Slot은 해당 Slot의 index와 몇 번째 Block에 속하는지 알 수 있다. Slot은 Page를 관리 하는 곳이다. Page의 VPID, LSA를 저장할 수 도 있다.
- DWB volume은 DB에 flush하려는 DWB Block을 Disk에 먼저 flush한 Block이다.

3. Slot 탐색 및 Page 저장

- 이번 챕터는 Slot의 정확한 위치를 파악해서, Page를 저장하는 일련의 과정을 설명하려고 한다. 또한 Slot에 Page를 저장하기 위해 대기해야 하는 경우에 대해서도 챕터 후반부에 알아보려고 한다.
- Slot의 위치 탐색 및 탐색 대기 역할은 dwb_acquire_next_slot() 함수
- Data Page를 저장하는 역할은 dwb_set_slot_data() 함수

3.1 Slot 위치 탐색 및 Page 저장

- Slot 탐색을 진행하고, Page를 해당 순서의 Slot에 저장하는 메커니즘

3.1.1 Block 및 Slot의 위치 탐색

- DWB의 전역변수 '**position_with_flags**'를 비트 연산을 사용해서 Block 및 Slot의 index 탐색. 이를 사용하여서 Slot의 시작주소를 획득
- Block의 index는 순환적이다.

3.1.2 Page 저장

- 3.1.1에서 구한 Slot의 index에 Page 저장(memory copy)
  - ◆ 실제로는 Block의 지역변수 write_buffer에 slot순서대로 저장
- Vpid, LSA정보를 함께 저장

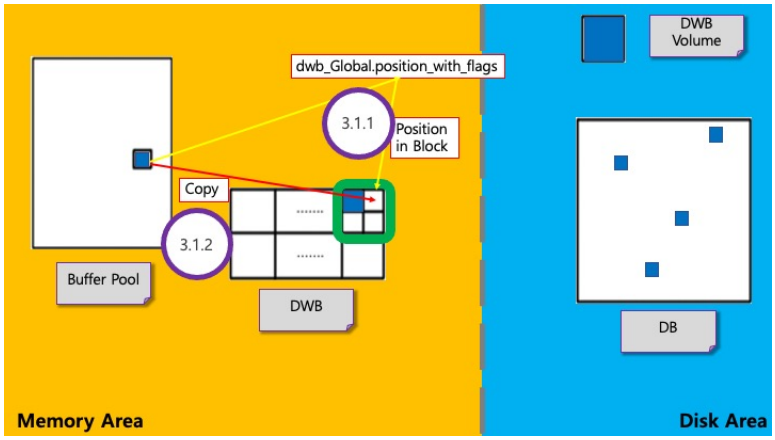

Figure 2 slot 탐색 및 Page 저장

3.2 탐색 및 저장 실패의 케이스

- 아래의 설명은 slot의 index를 찾기 전에 해당 Block에 다른 thread가 write작업을 진행 중일 때 나타나는 현상을 보여준다. Write 작업이란 다른 thread 가 해당 Block에 대해서 현재 slot 탐색부터, DWB flush사이의 작업을 진행중이다.

3.2.1 Flush 순서 및 대기

- Page를 저장해야 할 slot이 Block의 첫 번째 slot이고, Block이 다른 thread에 의해서 writing중인 상태이다.
- 해당 work thread를 대기 상태로 설정하려고 할 때 , Block의 지역변수에 존재하는

wait_queue에 work thread를 저장시키고 대기시킨다.

- 해당 block이 writing 상태가 끝날 때까지 work thread는 dwb_acquire_next_slot() 함수 내부 "start" goto phase부터 다시 진행한다.
- 해당 block에 대해서 다른 thread가 write작업을 마쳤다면, 다시 함수내부에서 처음부터 시작할 때, 현재 thread가 전역변수 '**position_with_flags**'에서 현재 block의 상태를 writing으로 바꾼 다음 다시 slot의 index를 찾는 작업을 진행한다.

4. DWB Flush

- 하나의 Block의 slot이 page로 가득 찼을 때 해당 Block의 Flush를 시작한다. 그렇게 되면 DWB에서 DWB volume로 그 다음에는 DWB에서 DB로 Flush 하는 과정을 거친다.
- dwb_add_page()함수에서 Block의 slot이 page로 가득찼다고 판별이 되면, dwb_flush_block() 함수를 실행해서 DWB flush를 실행한다.

4.1 Flush

- DWB를 사용해서 Disk로 Flush가 일어나는 경우를 살펴보도록 하겠다.
- 아래의 Flush는 flush daemon을 (4.2)통하여 flush를 진행하거나 수동으로 직접 dwb_flush_block()를 호출하여 진행한 경우에 있어서 모두 해당한다.
- Flush가 끝나면, Block의 상태 및 다음 순서의 Flush Block을 수정한다.

4.1.1 DWB volume

- System crash가 일어나기 전에 DWB volume으로 먼저 DWB Block을 Flush한다.
- Block의 지역변수 '**write_buffer**'을 사용하여서, slot의 순서대로 DWB volume에 write을 진행한다.
- Block 전체의 Write이 끝나면, fsync()를 호출하여서 DWB volume에 Flush를 마무리한다.

4.1.2 DB

- Block내부의 slot들을 정렬 후, DB에 해당하는 Page마다 write() 함수를 호출하여 write를 진행한다.
- Write가 끝난 뒤, sync daemon을 사용할 수 있다는 전제하에, 전역변수

'**file_sync_helper_block**'에 현재 Flush하려는 Block을 참조 시킨다. Daemon을 호출해도 되는 이유는 이미 DWB volume에 flush가 되었기 때문에 해당 Flush가 급한 작업이 아니기 때문이다.(system crash가 발생해도 recovery 가능)

- 각 Page마다 sync daemon(4.2)을 호출하거나 불가능 할 경우 fsync()를 직접 호출한다.

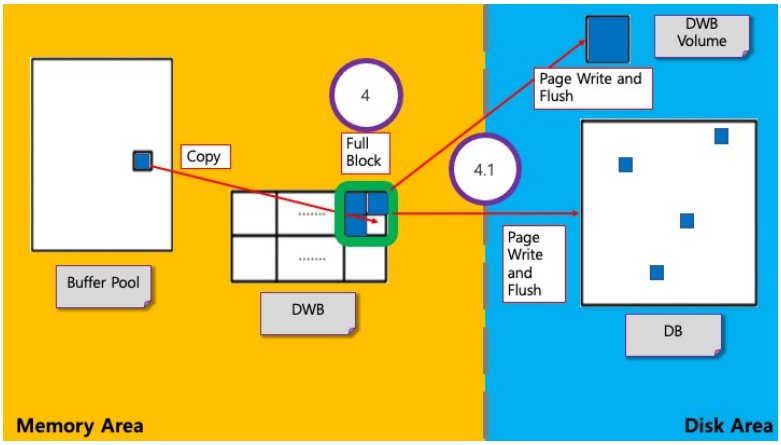

Figure 3 DWB Flush

4.1.2.1 Slot ordering

- Slot 정렬은 다음과 같은 이유의 장점 때문에 진행한다.
  1. DB에 있는 volume들이 VPID순서로 정렬되어있다.
  2. 같은 Page의 경우 Page LSA를 통해서 최신 버전만 Flush할 수 있다.
- Slot ordering은 Slot행렬을 따로 만들어서 Slot들을 복사한다.
- 그 다음 VPID, LSA기준으로 정렬 후 이전 시점의 Page LSA를 가진 slot들은 초기화한다.

4.2 (Flush, Sync) daemon

- dwb flush block daemon 은 주기적으로 Block의 모든 slot에 page가 저장되어 있는지 확인하여서, 꽉 차있다면 dwb_flush_block()을 호출한다.
- dwb file sync helper daemon은 dwb flush block daemon가 호출하는 daemon으로서, 주기적으로 dwb_file_sync_helper() 을 호출해서 DB로 Page를 flush한다.(추후 내용 추가)
  - '**file_sync_helper_block**'가 참조한 Block을 fsync 한 뒤, NULL값으로 초기화

5. Corrupted Data Page Recovery

- 위 문서에서 "**Page가 Corrupted하다**"라는 뜻은 Partial Write이 일어난 Page를 뜻한다. 또한 DWB를 통해 일어나는 recovery는 log와 관련 없는 내용을 밝힌다. 정확히

는 log recovery를 진행하기 전에 실행한다. Recovery르가 시작되면, corruption test를 진행하여서 recovery를 진행할 Page를 선별하고 recovery가 불가능한 경우에는 recovery를 그대로 종료한다.

- Recovery가 시작할 때 recovery block이 만들어지며 DWB volume에 저장된 내용을 메모리에 할당시킨다. 할당된 Block은 slot ordering 을 통해서 정렬시킨 뒤, 같은 Page의 최신 Page LSA를 가진 Page만 Recovery에 사용된다..
- dwb_load_and_recover_pages() 함수를 통해 전체적인 recovery를 진행한다.
- dwb_check_data_page_is_sane() 함수를 통해 corruption test를 진행한다.

5.1 Corruption Test

- 같은 volume fd, page id를 가진 recovery block의 Page와 DB의 Page의 corruption test를 각각 진행한다.
- LSA를 통해서 Partial Write이 일어났는지 확인한다.
- Recovery block에서 corruption이 발생했다면, recovery를 잘못된 data로 진행하는 것이 되기 때문에 recovery를 중지한다.
- Recovery block에서 corruption이 발생하지 않았다면, 해당 page는 recovery에 사용 가능하다.
- DB Page가 corruption이 발생했다면, 그 다음 Page의 corruption test를 진행한다.
- DB Page가 corruption이 발생하지 않았다면, 해당 slot은 NULL로 초기화 시켜서 recovery 속도를 향상시킬 수 있게 한다.

5.2 Recovery

- 정렬된 Recovery block을 DB에 Flush를 진행한다.
- 4.1.2에서 DWB block을 DB에 write하는 방식처럼 slot을 정렬해서 한 뒤 write을 진행한다. Write을 진행한 다음 곧바로 Page에 대해서 Flush를 진행한다.

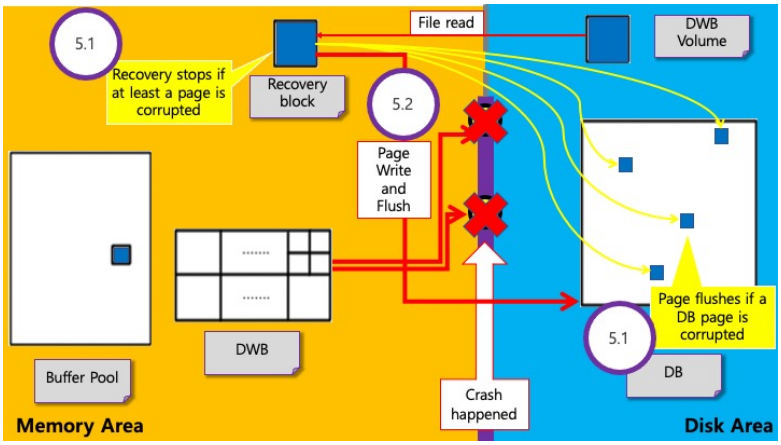

Figure 4 Recovery corrupted Pages

6. Appendix

DWB 역할 그 자체보다는 DB 시스템에서 DWB를 사용한 부수적인 역할에 대해서 알아보려고 한다.

6.1 Slot Hash Entry

- Page replacement에서 Cache 역할을 담당. Memory에 찾으려는 page가 없다면, DWB에서 찾는다.(I/O cost 감소)
- 생성 시점 : add_page() 함수에서 block과 slot의 index를 구한 다음 생성
- 제거 시점 : DB에 Flush 마치고 제거한다.
- Key, Value값으로서 저장