**버퍼 캐시에서의 대기 이벤트들**

**- latch: cache buffers chains**

버퍼 캐시를 사용하기 위해 해시 체인을 탐색하거나 변경하려는 프로세스는 반드시 해당 체인을 관리하는 cache buffers chains 래치를 획득해야 한다. 읽기 전용의 목적으로 체인을 탐색하는 경우에는 cache buffers chains 래치를 Shared 모드로 공유할 수 있어 경합을 줄이는데 도움이 된다.

cache buffers chains 래치 경합이 발생하는 대표적인 경우는 다음과 같다.

1) 비율적인 SQL 2) 핫블록 (Hot Block)

**- 비효율적인 SQL**

비효율적인 SQL 문장이 cache buffers chains 래치 경합의 가장 중요한 원인이다. 동시에 여러 프로세스가 넓은 범위의 인덱스나 넓은 범위의 테이블에 대해 스캔을 수행할 경우 cache buffers chains 래치 경합이 광범위하게 발생할 수 있다. 래치 경합이 핫블록에 의해 발생하는지, 비효율적인 SQL 문장에 의해 발생하는지를 판단하는 가장 중요한 근거는 SQL 문장 자체이다. SQL 문장 자체가 비효율적으로 작성되어 있는 것으로 판단할 수 있는 근거가 확실하다면 SQL 문장을 튜닝함으로써 문제를 해결할 수 있다.

만일 SQL 문장에 대한 정보가 없다면 간접적으로 핫블록에 의한 문제인지, 비효율적인 SQL 문장에 의한 문제인지 판단할 수 있는 방법이 있다. V$LATCH\_CHILDREN 뷰에서 자식 cache buffers chains 래치에 해당하는 CHILD# 과 GETS, SLEEPS값을 비교하여 특정 자식 래치에 사용하는 횟수와 경합이 집중되는지 판단하는 것이다. 핫블록 여부를 판단하는 또다른 방법은 V$SESSION\_ WAIT 뷰로부터 래치의 주소를 얻어서 비교하는 것이다. Parallel Query를 적절히 시용하는 것도 cache buffers chains 래치 경합을 줄이는 방법이 될 수 있다.

**- 핫블록**

SQL 문의 작동방식이 소수의 특정 블록을 계속해서 스캔하는 형태로 작성되었다면 여러 세션이 동시에 이 SQL 문을 수행히는 경우, 핫블록에 의한 cache buffers chains 래치 경합이 발생한다. 이 문제는 통일 블록을 반복해서 스캔하는 설계상의 문제를 해소하면 자연스럽게 해결된다. 하지만 불행히도 어플리케이션을 수정하는 것이 불가능한 경우가 많다. 우회적으로 핫블록에 의한 cache buffers chains 래치 경합을 해소하는 방법은 핫블록에 속한 로우들을 되도록이면 다른 블록으로 분산시키는 것이다. 보통 다음과 같은 방법을 권장하고 있다.

1) PCTFREE를 높게 주거나 작은 크기의 블록을 사용함으로써 블록 경합을 줄인다.

2) 파티셔닝 (Partitioning) 기법을 사용해서 로우가 물리적으로 다른 블록으로 들어가게끔 한다.

3) 문제가 되는 블록의 로우들에 대해서만 삭제 후 재삽입 작업을 한다. 이 방법은 테이블에 대해서만 기능하다.

**- latch: cache buffers lru chain**

Working Set (LRU + LRUW)을 탐색하거나 변경하려는 프로세스는 항상 해당 Working Set을 관리하는 cache buffers lru chain 래치를 획득해야 한다. 오라클은 다음과 같은 경우에 cache buffers lru chain 래치 를 획득해야 한다.

1) 프로세스가 아직 메모리에 올라오지 않은 블록을 얽고자 할 경우에는 프리 버퍼를 할당받기 위해 LRU 리스트를 탐색하게 되는데，이 과정에서 cache buffers lru chain 래치를 획득해야 한다.

2) DBWR이 더티 버퍼를 파일에 기록하기 위해 LRUW 리스트들 탐색하고, 해당 버퍼를 LRU 리스트로 옮기는 과정에서도 cache buffers lru chain 래치를 획득해야 한다.

cache buffers lru chain 래치 경합의 가장 주된 원인은 과도한 프리 버퍼의 요청이다. 비효율적인 SQL 문장이 프리 버퍼를 과도하게 요청하는 가장 전형적인 경우인데, 동시에 여러 세션이 비효율적인 SQL 문장을 수행하게 되면 프리 버퍼를 탐색하는 과정에서 그리고 더티 버퍼를 기록하는 과정에서 cache buffers lru chain 래치를 획득하기 위해 경쟁하게 된다. 비효율적인 SQL에 의해 cache buffers lru chain 래치 경합이 발생할 경우에는 SQL문의 적절한 튜닝이 최선의 해결책이다.

**- buffer busy waits / read by other session**

블록 단위의 락은 존재하지 않지만, 오라클의 I/0가 블록 단위로 이루어진다는 사실 때문에 블록 단위의 락은 반드시 필요하다. 현재 자신만이 블록을 변경하고 있다는 것을 보장받아야 한다. 이 경우에 획득해야 하는 락을 buffer lock이라고 부른다.

Buffer lock을 획득하는 모드에는 Shared 모드와 Exclusive 모드가 있다 버퍼를 읽는 과정에서는 Shared 모드의 락을 획득해야 하고, 변경하는 과정에서는 Exclusive 모드의 락을 획득해야 한다. Buffer lock은 cache buffers chains 래치, TX 락과 함께 버퍼의 변경을 동기화하는 역할을 한다. Buffer lock을 획득하기 위해 기다리는 동안 일반적으로 buffer busy waits 이벤트를 대기하게 된다. 오라클 10g부터는 buffer busy waits 대기 이벤트의 정의가 크게 변경되었다.

1) 기존의 buffer busy waits 이벤트가 read by other session과 buffer busy waits 두 개의 이벤트로 분화되었다.

2) 원인 코드는 더 이상 제공되지 않으며, 그 값은 경합이 발생한 블록의 클래스를 나타낸다.

**- Select / Select에 의한 read by other session**

Select / Select에 의한 buffer lock 경합은 동시에 같은 블록을 메모리에 올리는 과정에서 발생하며 read by other session 이벤트 대기로 관찰된다. read by other session 대기이벤트는 db file sequential read, db file scattered read 대기이벤트와 밀접한 관련이 있다. Select / Select에 의한 read by other session 대기를 줄이는 방법은 다음과 같이 정리할 수 있다.

1) SQL 최적화를 통해서 최소한의 I/0만으로 원하는 결과를 가져올 수 있도록 해야 한다.

2) SGA 사이즈(또는 버퍼 캐시 사이즈)가 시스템 전반적인 I/0에 비해 작다면 SQL 튜닝만으로는 문제를 해결할 수 없으며 SGA의 물리적 크기를 늘려 주어야한다.

**- Select / Update에 의한 buffer busy waits / read by other session**

이것에 의한 buffer lock 경합은 다음과 같은 상황에서 발생한다.

1) 특정 프로세스가 특정 테이블을 변경한다. 데이터의 이전 이미지가 언두 블록에 기록된다.

2) 동시에(혹은 이후에) 많은 프로세스들이 변경된 데이터에 대해 읽기를 시도한다. 언두 블록에 대한 경합은 오라클의 가장 기본적인 메커니즘 중 하나인 일관된 Consistent Read에 그 원인이 있다. Select 세션들이 데이터 블록을 읽을 때，Update에 의해 변경된 상태인 경우에는 기본적으로 언두 블록을 이용해서 과거의 상태를 읽어 와야 한다. 이것을 일관된 읽기(Consistent Read) 라고 부르며 이 과정에서 CR 블록이 만들어진다. 이 때 언두 블록에 대해 동시에 많은 세션들이 읽기를 시도하면서 언두 블록에 대해서 Select / Select 의 경우와 동일한 상황의 buffer lock 경합이 발생하고 이로 인해 read by other session 이벤트에 대한 대기가 발생하게 된다.

해결책은 SQL 문을 적절히 튜닝해서 논리적 I/O를 줄여야 한다. 논리적인 I/O가 줄면 일관된 읽기를 위해 언두 블록을 읽는 회수도 줄어든다. SGA 크기가 지나치게 작으면 버퍼 캐시에 CR 블록이 상주하지 못해 물리적 읽기 작업이 발생하고, 이로 인해 read by other session 대기가 증가할 수 있다. 따라서 SGA의 크기를 적절히 유지해줘야 한다.

**- Insert / Insert에 의한 buffer busy waits**

Insert / Insert에 의한 buffer lock 경합은 대부분 잘못된 프리리스트 값에 기인한다. FLM을 사용하는 경우 FREELISTS 속성의 기본값은 1이 된다. 이 경우 마스터 프리리스트 하나만이 생성되기 때문에, 동시에 여러 프로세스가 동일 프리 블록을 사용하게 되고, 이로 인해 buffer lock 경합이 발생한다. 또한 Insert를 수행하면서 세그먼트 헤더 블록을 변경하는 것은 프리리스트 정보를 변경하거나 HWM을 변경하기 위한 것이다. HWM을 변경하는 과정에서 HW 락 경합 현상이 같이 발생하며, 이로 인해 enq: HW - contention 이벤트를 대기하게 된다.

ASSM을 사용한 경우, buffer busy waits 대기와 enq: HW - contention 대기가 모두 줄어든 것을 확인할 수 있다. 하지만 여전히 buffer busy waits 대기가 성능에 영향을 주고 있다. ASSM을 사용하는 경우에는 비트맵 블록에 대한 buffer busy waits 대기가 주로 발생한다. ASSM을 사용하는 경우에도 buffer lock에 대한 경합은 여전히 존재하지만, buffer busy waits 대기가 전반적으로 줄어드는 효과가 있으며 특히 HW 락 경합이 크게 줄어드는 효과가 있다.

**- Update / Update에 의한 buffer busy waits**

동시에 여러 개의 세션이 서로 다른 로우를 업데이트하는 경우에도 만일 해당 로우들이 같은 블록 안에 있다고 하면 buffer lock에 의한 동기화가 필요하다. 동일 블록을 변경하는 과정에서 발생하는 buffer lock 경합은 그 해결책이 다르다. 문제는 크게 2가지가 있다.

1) 프로세스들이 각각 서로 다른 로우를 업데이트하지만 같은 블록을 변경하는 것만으로 매우 심각한 성능문제를 야기하게 된다. 인덱스의 경우에도 buffer lock 경합이 발생할 수 있다.

2) 언두 헤더 블록에 대한 경합도 많이 나타난다. 이것은 동시에 여러 프로세스가 Update에 의한 언두 데이터를 생성하면서 언두 헤더 블록을 변경하는 과정에서 발생한다. Buffer lock 경합이 발생하는 원인이 서로 다른 로우가 같은 블록에 있다는 것에서 기인하므로, 서로 다른 로우를 서로 다른 블록에 흩어지게끔 분산시키는 방법이 가장 보편적으로 사용된다. 로우를 분산시키는 방법은 여러가지가 있다.

1) PCTFREE 값을 높게 준다. 이 방법은 로우를 분산시키는 가장 확실한 해결책이지만, 공간 낭비를 초래하고 그로 인해 풀테이블 스캔이나, 인덱스 풀 스캔, 인덱스 범위 스캔의 성능에 영향을 주게 된다. 또한 SGA 내의 버퍼 캐시에 대한 낭비를 초래해 cache buffers chains 래치 경합이 생길 가능성이 있다.

2) 파티셔닝 기법을 사용하여 각 로우들을 물리적으로 다른 블록으로 흩어지게끔 한다.

3) 작은 블록 크기를 사용한다. 하지만 이 경우에도 PCTFREE 값을 높게 주는 것과 같은 부작용이 있다.

**- write complete waits**

write complete waits 대기는 buffer busy waits 대기와 마찬가지로 buffer lock 경합에 의한 대기로 분류할 수 있다. DBWR은 더티 버퍼를 디스크에 기록하는 동안에 버퍼에 대해 buffer lock을 Exclusive하게 획득한다. 이때 버퍼를 읽거나 변경하려는 다른 프로세스는 이 작업이 끝나기를 기다려야 하고 그동안 write complete waits 이벤트를 대기하게 된다. write complete waits 대기 가 보편적으로 나타나는 경우, 어플라케이션의 문제보다는 DBWR의 성능 문제일 가능성이 매우 높다. DBWR의 성능이 느린 이유는 매우 다양하지만 대부분 다음 범주에 들어온다.

1) I/O 시스템의 성능이 느린 경우: DBWR에서 db file parallel write 대기시간이 길게 나타난다면 시스템에 문제가 있다고 판단할 수 있다. DBWR에서 db file parallel write 대기시간이 길어지면, 서버 프로세스는 연쇄적으로 free buffer waits 대기나 write complete waits 대기를 겪게 된다.

2) DBWR의 작업량이 너무 많은 경우: 잦은 체크포인트가 발생하는 경우 DBWR의 활동량이 지나치게 많아지고, 이로 인해 DBWR의 성능이 저하될 수 있다. DBWR의 성능 저하는 시스템 전체적인 성능 저하로 직결된다.

간접적으로 DBWR의 성능을 개선시키는 방법으로 다중 버퍼 풀을 적절히 사용하는 것도 권장되고 있다. 다중 버퍼 플은 DBWR의 성능과는 직접적인 관계는 없지만 버퍼 캐시의 효율성을 높임으로써 메모리에 상주해야 할 데이터가 불필요하게 디스크에 기록되는 빈도를 낮추고, 이로 인해 write complete waits 대기가 발생할 확률을 낮춘다. 또한 잦은 체크포인트를 줄이기 위해서는 FAST\_START\_MTTR\_TARGET 값을 크게 해주어서 체크포인트의 횟수를 줄일 수 있어, DBWR의 작업량을 줄여줄 수 있다. 하지만 이 경우, 복구에 더 많은 시간이 소요될 수 있다.

**- free buffer waits**

free buffer waits 대기가 발생하는 이유는 보통 다음과 같다.

1) 비효율적인 SQL

2) 지나치게 작은 버퍼 캐시

3) DBWR의 성능 저하

비효율적인 SQL 문장은 많은 프리 버퍼를 요청하기 때문에 이와 비례해서 free buffer waits 대기가 증가하게 된다. 버퍼 캐시의 크기가 너무 작은 경우에는 적은 회수의 프리 버퍼 요청으로도 free buffer waits 대기가 유발된다. SQL 문이나 버퍼 캐시의 크기에 문제가 없다면 free buffer waits 대기의 궁극적인 발생원인은 DBWR의 성능문제로 볼 수 있다. DBWR의 성능을 개선시켜서 최악의 상황에서도 어느 정도의 성능이 보장되도록 하는 것이 최선의 해결책이다.

**- enq: TC – contention**

인위적인 체크포인트를 수행하는 작업들 중 일부는 TC 락 (Thread Checkpoint Lock, 흑은 Tablespace Checkpoint Lock)을 획득해야 한다. TC 락을 획득하는 과정에서 경합이 발생하면 enq: TC -contention 이벤트를 대기하게 된다. 실제로 TC 락을 획득하는 과정은 다음과 같다.

1) 서버 프로세스가 우선 TC락을 X모드로 획득한다.

2) 서버 프로세스는 획득한 TC락을 SSX모드로 변경하고, 동시에 CKPT 프로세스가 SS모드로 해당 락을 획득한다. CKPT는 락을 획득하고, 체크포인트 작업을 수행하게 된다.

3) 서버 프로세스는 다시 TC락을 X모드로 획득하려고 시도하는데, 해당 락이 CKPT에 의해 해제될 때까지 대기하게 된다. 이 때의 대기 이벤트가 enqueue: TC - contention이다.

4) 체크포인트 작업이 완료되면 CKPT 프로세스는 TC락을 해제하고, 서버 프로세스는 TC 락을 획득함으로써 체크포인트 작업이 끝났다는 것을 알게 된다.

enq: TC - contention 대기는 여러 프로세스에 의해 경합이 발생하지 않더라도 관찰이 된다는 점에서 락 경합에 의한 다른 대기현상들과는 구별이 된다. 체크포인트가 발생하는 경우는 매우 다양하지만, 모든 경우에 의해 TC 락에 의한 대기가 발생하는 것은 아니며, 서버 프로세스에 의해 유발된 체크포인트 작업을 동기화 시키는 과정에서만 발생한다 enq: TC - contention 대기가 발생하는 대표적인 사례는 다음과 같다.

1) Parallel Query 2) 테이블 스페이스 핫백업(Tablespace hot backup)

**- Parallel Query**

Parallel Query (이하 PQ)에서 체크포인트가 발생하는 이유는 슬레이브 세션 (Slave Session) 에 의한 direct path read 때문이다 direct path read는 버퍼 캐시를 거치지 않고, 데이터 파일을 직접 읽는 것을 말한다. 이 경우 SGA를 경유하지 않기 때문에, 현재 SGA에 있는 블록과 데이터 파일에 있는 블록 사이에 버전 불일치 현상이 생길 수 있다. 이러한 현상을 방지하기 위해 오라클은 데이터 파일에 대해 direct path read를 수행하기 전에 체크포인트를 수행한다. 코디네이터 세션은 슬레이브 세션을 구동하기전에 direct path read를 수행할 객체에 대해 세그먼트 레벨의 체크포인트를 요청하게 되고, 체크포인트 작업이 끝날 때까지 enq: TC - contention 이벤트를 대기한다. 오라클 10g R2 부터는 PQ 수행 시 수행되는 체크포인트 알고리즘이 크게 개선되어, TC락에 의한 경합이 이전 버전에 비해 크게 줄어들었다.

**- 테이블스페이스 핫백업**

ALTER TABLESPACE .. . BEGIN BACKUP을 수행하면 오라클은 테이블스페이스 내에 속한 모든 데이터 파일에 대해 더티 블록을 디스크에 기록 (Checkpoint) 하게 되는데 이 과정에서 enq: TC - contention 대기를 겪는다. TC 락 경합에 의한 성능상의 문제는 대기 그 자체보다는 인위적인 체크포인트를 수행한다는데 있다. PQ는 해당 시스템과 업무를 고려하여 꼭 필요한 경우에만 사용하도록 할 필요가 있다 다만. PQ에 의한 체크포인트는 PQ가 수행되는 객체에 대해서만 수행된다는 점에서 일반적인 체크포인트보다는 부하가 적다.

**- enq: CI - contention, enq: RO - contention**

체크포인트, 로그 스위치(log switch), 셧다운(Shutdown) 등의 작업을 호출하는 것을 오라클에서는 Cross Instance Function이라고 부른다. 총 11 개의 Cross Instance Function이 정의되어 있으며 Function마다 call, parameter, queue, return이라는 네 개의 단계마다 락을 획득해야 한다. 이때의 락을 CI 락이라 부르며， CI 락을 획득하기 위해 기다리는 동안 enq: CI - contention 이벤트를 대기하게 된다. 이것은 경합에 의해 발생하는 것이 아니기 때문에, 대부분 상황에서는 성능이슈와 연결되지 않으나, 체크포인트와 관련된 일부 작업 등에서는 체크포인트 자체의 성능 문제가 발생하는 경우 이 대기에 의한 성능 저하가 종종 발생한다. 해결책으로는 다음과 같이 3가지가 있다.

1) 오라클 10g R2로 업그레이드한다.

2) DBWR의 성능을 개선시킨다.

3) SGA, 즉 버퍼 캐시의 크기가 불필요하게 크지 않은 지 확인한다.

Truncate와 관련해서 자주 관찰되는 대기현상이 한 가지 더 있는데 바로 enq: RO fast obiect reuse 대기 현상이다. Truncate나 Drop과 같이 객체가 사용하는 공간(세그먼트)을 삭제하는 작업은 CKPT와 DBWR에 의해 실제로 삭제가 이루어질 때까지 기다리게 되 는데 그동안 enq: RO - fast object reuse 이벤트를 대기하게 된다. RO 락은 오라클 10g 부터 객체 삭제 시 객체 정보를 삭제 후 재사용하는 작업을 동기화하는데 주로 사용된다.