**리두에서의 대기 이벤트들**

**- latch: redo writing, latch: redo allocation, latch: redo copy**

1) redo writing 래치: 리두 버퍼내의 공간을 확보하기 위해 LGWR에게 쓰기 요청을 하려는 프로세스는 redo writing 래치를 획득해야 한다. LGWR에 의한 쓰기 작업은

동시에 수행될 수 없으므로, 자연스럽게 이 래치는 전체 인스턴스에 하나만 존재한다. redo writing 래치는 Willing-to-wait 모드로 획득된다 redo writing 래치를 획득하는 과정에서 경합이 발생하면 latch: redo writing 이벤트를 대기하게 된다.

2) redo copy 래치: PGA내의 체인지 벡터를 리두 버퍼로 복사하려는 프로세스는 작업의 전체과정 동안 redo copy 래치를 보유해야 한다. redo copy 래치는 기본적으로 No-wait 모드로 획득한다. 만일 프로세스가 redo copy 래치를 획득하는데 실패하면 연속적으로 다음 redo copy 래 치를 획득하기 위해 시도한 후 마지막 redo copy 래치를 획득하는 과정에서는 Willing-to-wait 모드가 사용된다. redo copy 래치를 획득하는 과정에서 경합이 발생하면 latch: redo copy 이벤트를 대기하게 된다.

3) redo allocation 래치: 체인지 벡터를 리두 버퍼에 복사하기 위해 리두 버퍼 내에 공간을 확보하고자 하는 과정에서 redo allocation 래치를 획득해야 한다. 전체 리두 버퍼를 복수개의 Redo strands라는 공간으로 분할해서 사용할 수 있으며, 하나의 redo allocation 래치가 하나의 redo strand를 관리한다 복수개의 redo allocation 래치를 사용 가능하기 때문에 래치 경합을 다소 줄일 수 있다. 또한 Dynamic Parellelism에 의해 동적으로 redo strands의 개수를 조정해서 redo allocation 래치의 수가 크게 늘어났다.

SGA에 추가적으로 private redo strands라는 기능을 도입하여 다중 strands의 개념을 더욱 확장하였다. Private strands 영역은 프로세스 간에 공유되지 않기 때문에 리두를 생성하는 과정에서 래치를 획득할 필요가 없으며, LGWR에 의해서 리두 로그 파일로 바로 쓰여 지기 때문에 PGA에서 리두 버퍼로 복사하는 과정이 불필요하다. 이것은 zero copy redo라고 표현하기도 한다.

일반적인 환경에서는 리두 관련 래치 경합은 잘 발생하지 않는다. 래치 경합의 횟수보다는 래치 경합에 따른 대기시간을 경합의 발생 증거로 활용하는 것이 바람직하다. 리두 버퍼의 크기가 리두 래치 경합을 일으키는 요인이 될 수도 있으며, 시스템 전체적으로 불필요하게 많은 리두 데이터가 생성된다면 Nologging 기능을 활용함으로써 리두 데이터를 줄이는 것도 래치 경합을 줄일 수 있으나, 이 작업은 기본적으로 복구가 되지 않는다.

다행히 리두와 관련된 대부분의 성능문제가 래치 경합보다는 어플리케이션의 작동 방식이나 I/O 성능과 관련이 있다. "다행히" 라는 표현을 쓴 이유는 리두 래치에서 발생하는 경합에 대해서는 뚜렷한 해결책이 없는 경우가 많기 때문이다. 하지만 버전이 올라감에 따라 리두 버퍼 관련 알고리즘이 개선되어서 대부분의 경우 래치 경합은 더 이상 문제가 되지 않는다.

**- log file sync**

서버 프로세스가 커밋 또는 롤백을 수행하면 LGWR은 리두 버퍼에서 가장 최근에 기록이 이루어진 이후 시점부터 커밋 지점까지의 리두 레코드를 리두 로그 파일에 기록한다. 이것을 "sync write"라고 부르며 redo synch writes 통계값을 통해 조회 가능하다. 트랜잭션의 존재 이유가 데이터를 변경하고 커밋을 수행하는 것이기 때문에 log file sync 대기이벤트는 오라클에서 가장 보편적인 대기이벤트 중 하나이다. 보통 sync write가 수행되는 시간은 매우 짧기 때문에 log file sync 대기는 문제가 되지 않는다. 하지만 일단 발생하게 되면 상당히 해결하기 어려운 대기현상이다. 만일 log file sync 대기가 광범위하게 나타난다면 아래와 같은 사실을 의심해야한다.

1) 커밋 회수와 log file sync: 지나치게 잦은 커밋은 log file sync 대기의 주범이다. 여러 세션에서 동시에 대량의 커밋을 수행하면 log file sync 대기가 광범위하게 나타날 수 있다. 오라클은 여러 세션에서 동시에 실행하거나, PL/SQL 블록에서 반복적으로 커밋이 이루어지거나, Recursive SQL에 의해서 커밋이 이루어지는 경우 그룹 커밋 기능이 사용된다. 이때, user commits 통계 값은 커밋 명령을 호출한 횟수와 동일하게 증가하지만, log file sync는 한번만 발생하게 된다. nocache 속성의 시퀀스 또한 log file sync를 유발한다. Sequence.nextval을 호출하면 매번 딕셔너리 테이블의 정보를 갱신하고 커밋을 수행해야하기 때문이다.

2) I/O 시스템의 성능과 log file sync: 리두 로그 파일이 위치한 I/O 시스템의 성능이 느린 경우에는 LGWR의 synch write를 수행하는 시간이 늘어나고 이로 인해 log file sync 대기시간이 증가할 수 있다. 이 경우, 서버 프로세스가 log file sync 이벤트를 대기하는 동안 LGWR는 log file parallel write 이벤트를 대기하게 된다. 일반적인 가이드는 리두 로그 파일올 가장 빠른 다바이스에 위치시키라는 것이다. 그리고 디스크 경합을 피하기 위해 서로 다른 그룹의 리두 로그 파일은 서로 다른 디스크에 분산시켜야 한다. 리두 로그 파일을 데이터 파일이나 컨트롤 파일과 다른 디스크에 배치시키는 것 또한 필요하다.

3) 리두 데이터의 양과 log file sync: 커밋 수행시 리두 로그 파일에 기록해야 할 데이터양을 줄이면 log file sync 대기시간 또한 줄일 수 있다. 특히 크고 긴 트랜잭션에서 리두 데이터양을 줄이게 되면 LGWR의 백그라운드 기록 작업이 줄게 되고 자연스럽게 리두와 관련된 경합이 해소될 수 있다. 대량의 데이터를 생성하는 작업에서는 다음과 같은 기법을 쓸 수 있는지 검토해 보아야한다.

(1) Nologging 옵션을 사용할 수 있는지

(2) SQL\*Loader로 대량의 데이터를 적재할 때는 Direct load option을 사용한다.

(3) 임시 작업이 필요할 때는 가능하면 임시 테이블을 사용한다.

(4) 인덱스가 있는 테이블에 대해 Direct load 작업을 수행할 때는 인덱스를 Unusable로 변경하고 데이터 생성하고 인덱스를 Nologging 모드로 재구성과 같은 절차를 통해서 인덱스에 의해 리두가 생성되는 것을 방지한다.

(5) LOB 데이터의 경우 데이터의 크기가 크다면 가급적 Nologging 속성을 부여한다.

4) 리두 버퍼의 크기와 log file sync: 일반적으로 리 두 버퍼의 크기가 지나치게 큰 경우에 log file sync 대기가 증가하는 경향이 있다. 리두 버퍼의 크기가 큰 경우, LGWR에 의한 백그라운드 기록 횟수가 줄어 드는데, 이 경우 유저 세션에서 sync write를 수행할 때 기록해야 할 데이터의 양이 많아지고 이로 인해 log file sync 대기 시간이 길어질 수 있기 때문이다.

반대로 log buffer space 대기가 증가하는 경향을 보인다면 리두 버퍼의 크기를 늘리는 것이 일반적인 해결책이다. 이런 경우의 해결책은 \_LOG\_IO\_SIZE 히든 파라미터를 이용하는 것이다. \_LOG\_IO\_SIZE 파라미터는 LGWR이 리두 버퍼의 내용을 리두 로그 파일에 저장하는 임계치 값을 나타낸다. log buffer space 대기를 줄이기 위해 리두 버퍼의 크기를 키우고 이와 동시에 log file sync 대기를 줄이기 위해\_LOG\_IO\_SIZE 값을 적당히 작은 크기로 조정하면 된다.

**- log file parallel write**

LGWR 프로세스에만 발생하는 대기 이벤트이다. LGWR은 리두 버퍼의 내용을 리두 로그 파일에 기록하기 위해 필요한 I/O 콜을 수행한 후 I/O 작업이 완료되기를 기다리는 동안 log file parallel write 이벤트를 대기하게 된다. I/O 시스템의 성능에는 아무런 이상 징후가 없는데도 리두 데이터양이 지나치게 많은 경우 log file parallel write 대기가 증가할 수 있다. 이것을 해결하기 위한 방법은 아래와 같다.

(1) 불필요한 커밋을 줄인다.

(2) Nologging 옵션을 활용하여 리두 데이터의 양을 줄인다.

(3) 리두 로그 파일이 위치한 I/O 시스템의 성능을 높인다.

(4) 핫백업을 너무 자주 수행하지 말고, 특히 트랜잭션이 많은 시점에 수행하지 않아야 한다. 핫백업 모드에서는 리두 데이터가 로우 레벨이 아닌 블록 레벨로 생성되기 때문이다.

**- log buffer space**

redo allocation 래치를 획득한 상태에서 리두 버퍼에 공간을 확보하려는 순간에 적절한 여유 공간이 없는 경우, 공간이 확보되기를 기다려야 한다. 이때 경우에 따라 두 가지 종류의 이벤트를 대기하게 된다.

만일 현재 시용 중인 리두 로그 파일이 꽉 차서 더 이상 여유 공간을 확보할 수 없다면， LGWR은 로그 파일 스위치를 수행하고, 서버 프로세스는 log file switch completion 이벤트를 대기한다. 그 외의 경우에는 log buffer space 이벤트를 대기하게 된다. log buffer space 대기는 트랜잭션에 의해 생성되는 리두의 양에 비해 리두 버퍼의 크기가 작을 때 발생한다. log buffer space 대기를 줄이기 위해 리두 버퍼 크기를 늘리는 경우, log file sync 대기가 증가할 수 있다.

log buffer space 대기를 줄이는 또 하나의 방법은 리두 데이터를 적게 생성하는 것이다. Direct load 기능을 적당히 사용하고 Nologging 옵션을 부여하는 것 등이 이 방법에 속한다. 또한 더 빠르고 효율적인 I/O 시스템을 리두 로그 파일에 사용함으로써 LGWR의 쓰기 성능을 개선하고, log buffer space 대기를 줄일 수 있다.

**- log file switch completion, log file switch (checkpoint incomplete), log file switch (archiving needed), log file switch (private strand flush incomplete)**

서버 프로세스는 LGWR에 의해 로그 파일 스위치가 끝날 때까지 log file switch completion 이벤트를 대기하게 된다. 하지만 로그 파일 스위치가 끝나는 시점에, 새로 사용할 리두 로그 파일에 대해 아직 종료되지 않은 작업이 있다면 아래와 같이 추가적으로 이벤트를 대기해야 한다.

1) 새로 사용할 리두 로그 파일에 대한 체크포인트 작업이 끝나지 않았다면 프로세스는 DBWR에 의해 체크포인트가 끝나기를 대기해야 한다. 이 경우 log file switch (checkpoint incomplete) 이벤트를 대기한다.

2) 새로 사용할 리두 로그 파일에 대해 아카이브 작업이 아직 끝나지 않았다면 프로세스는 아카이브 작업이 끝나기를 기다려야한다. 이 경우 log file switch (archiving needed) 이벤트를 대기한다.

3) 새로 사용할 리두 로그 파일에 대한 Private strands에 대한 플러시 작업이 아직 끝나지 않았다면, 이 작업이 끝나기를 기다려야한다. 이 경우 log file switch (private strand flush incomplete)이벤트를 대기하게 된다.

서버 프로세스는 먼저 log file completion이벤트를 대기하고 특수한 경우 위의 이벤트를 대기하게 된다. 위 방법들에 대한 해결책은 동일하다. 발생 원인은 트랜잭션이 생성하는 리두 데이터 양에 비해 리두 로그 파일이 지나치게 작다는 것이다. 따라서 리두 로그 파일의 크기를 충분히 키워주고, Direct load operation이나 Nologging 옵션을 사용하여 리두 데이터의 양을 줄여야 한다.