**트랜잭션에서의 대기 이벤트들**

**- enq: TM – contention**

DML이 수행되는 동안，DML과 관련된 객체에 대한 변경을 방지하기 위해 DML을 수행하는 프로세스는 반드시 해당 테이블에 대해 TM 락을 획득해야 한다. TM락에는 DML락과 DDL락이 있다. DML 락은 실제로는 TM 락과 일치한다 DML 락은 DBA\_DML\_LOCKS 뷰를 통해 관찰 가능한데 이 뷰는 V$LOCK 뷰에서 락 유형이 TM인 것만을 추출해서 보여주는 역할을 한다.

DDL 락은 실제로는 library cache lock과 일치한다 DDL락은 DBA\_DDL\_LOCKS 뷰를 통해 관찰가능한데, 이 뷰는 실제로는 X$KGLLK 뷰를 적절히 가공해서 보여주는 역할을 한다. 일반적인 DML 문은 테이블에 대해 TM 락을 Sub-Exclusive (SX) 모드로 획득한다.

Sub-Exclusive 모드 간에는 상호호환성이 있기 때문에 여러 세션이 동일 테이블에 대해 동시에 DML을 수행하는 것이 가능하다. DML을 수행한 세션은 테이블에 대해서는 TM 락을 Sub-Exclusive 모드로 획득하고 변경된 데이터에 대해서는 TX 락을 Exclusive모드로 획득한다. 즉, DML간에는 TM락을 둘러싼 경합이 발생하지 않는다. TM Lock 경합이 발생하는 경우는 다음과 같다.

1) 인덱스가 없는 Foreign key의 부모키를 변경하는 경우 (현재는 발생하지 않음)

2) DML과 DDL 간의 TM 락 경합: DDL이 수행중인 테이블에 대해서 DML을 실행하는 경우 나타난다. 이를 방지하기 위해선 데이터량이 많은 테이블에 대한 부적절한 DDL을 방지해야 하고, DDL 수행 시 가능하면 온라인 옵션을 사용해야 한다. SS모드로 TM락을 획득하기 때문이다. 또한 Parallel DDL을 사용해 DDL의 수행 속도를 극대화해야 한다.

3) Lock table .. 에 의한 TM 락 경합: Lock table 문을 이용해 의도적으로 TM 락을 획득하는 경우에 TM 락 경합이 발생할 수 있다. 동기화를 위해서 테이블 전체에 락을 거는 것보다는 DBMS\_LOCK 패키지를 사용하거나 select ... for update 등을 사용해서 락의 범위를 줄이는 방법을 고려해야 한다.

4) Direct load 작업에 의한 TM 락 경합: INSERT /\*+ APPEND \*/ INTO 나 SQL\*Loader의 direct path load와 같은 일부 기능들은 해당 테이블에 대해 TM 락을 Exclusive하게 획득한다. Direct load 작업은 SGA를 경유하지 않고 데 이 터파일에 직접 쓰기를 수행하므로 작업 이 수행되는 동안 테이블에 어떠한 변화도 허용해서는 안된다. 따라서 TM 락을 Exclusive하게 획득함으로써 테이블에 대한 어떠한 변경도 허용되지 않는다는 것을 보장받아야만 작업이 가능하다.

**- enq: TX - row lock contention, enq: TX - allocate ITL Entry, enq: TX - index contention**

프로세스가 TX 락을 획득해야 하는 상황과 이 과정에서 경합이 발생할 경우 관찰되는 대기 현상은 다음과 같다.

1) 특정 로우를 변경하고자 하는 경우. 관련된 대기이벤트는 enq:TX - row lock contention이다.

2) 특정 로우에서 Unique Key 나 Primary Key에 해당하는 컬럼을 변경하고자 하는 경우 - 관련된 대기이벤트는 enq: TX - row lock contention이다.

3) 변경하고자 하는 블록의 ITL에 자선에 해당하는 트랜잭션 엔트리 (Transaction Entry) 를 등록하고자 하는 경우 관련된 대기이벤트는 enq: TX - allocate ITL entry이다.

4) 비트맵 인덱스가 생성되어 있는 컬럼 값을 변경하고자 하는 경우 관련된 대기이벤트는 enq: TX - row lock contention이다.

5) 인덱스 리프 노드에 서 Split 이 발생하는 경우 관련된 대기 이벤트는 enq: TX - index contention이다.

6) 기타의 경우 관련된 대기이벤트는 enq: TX - contention이다.

**- 여러 세션이 동일 로우를 변경하는 경우 (enq : TX - row lock contention, mode=6)**

프로세스가 특정 로우를 변경하기 위해 해당 로우를 방문했을 때, 현재 로우가 변경된 상태라면 ITL로부터 해당 로우를 변경한 트랜잭션을 확인하고, 자기 자신을 TX Enqueue 목록에 추가하고 enq: TX - row lock contention 이벤트를 대기한다. 이 대기는 TX 락을 보유한 프로세스가 락을 해제할 때까지 계속된다. 동일 로우 변경에 따른 TX 락 경합은 철저하게 어플리케이션 이슈이다. 장시간 수행되는 Update나 Delete 명령은 트랜잭션이 적은 시간대에 수행하는 것이 좋다.

**- 여러 세션이 Unique Key 충돌을 일으키는 경우 (enq: TX - row lock contention, mode=4)**

Unique Key 또는 Primary Key 충돌이 발생할 때도 TX 락 경합이 발생하게 된다. 프로세스 A가 Insert를 수행한 후, 프로세스 B가 Unique Key 충돌이 발생하게끔 Insert를 수행하면, 프로세스 B는 프로세스 A의 트랜잭션에 대해 TX 락을 Shared 모드로 획득하기 위해 대기한다. 이때의 대기현상은 enq TX - row lock contention 이벤트로 나타난다. 프로세스 B는 프로세스 A가 커밋하거나 롤백할 때까지 대기하게 된다. 프로세스 A에서 커밋이 이루어지면，ORA-000l 에러상황이 되며, 롤백이 이루어지면 프로세스 B의 Insert는 성공적으로 이루어지게 된다. Unique Key 충돌에 의한 TX 락 경합은 철저하게 어플리케이션 이슈이다. 최선의 해결책은 시퀀스를 사용해서 Unique Key를 생성하는 것이다.

**- ITL 엔트리 부족 (enq: TX - allocate ITL entry, mode=4)**

ITL이 약속된 최대치, 즉 MAXTRANS에 의해 지정된 값을 초과하거나 블록 내의 여유공간이 부족해서 엔트리를 등록하는 것이 불가능한 경우, 프로세스는 이미 ITL에 엔트리를 등록한 프로세스의 트랜잭션에 대해 TX 락을 Shared 모드로 획득하기 위해 대기하게 된다. 이때의 대기현상은 enq: TX- allocate ITL entry 이벤트로 관찰된다.

테이블을 생성할 때 부여하는 INITRANS, MAXTRANS, PCTFREE의 세가지 속성값이 ITL에 영향을 준다. 높은 INITRANS 값으로 테이블을 생성하는 것이 ITL 공간 부족에 따른 TX 락 경합 현상을 해결하는 일반적인 방법이다.

Row Chaining이나 Row Migration이 발생한 로우의 경우 하나의 로우를 업데이트할 때 여러 개의 블록에 대해 각각 ITL 엔트리를 할당해야 하므로 이로 인해 ITL 엔트리 부족에 의한 TX 락 경합이 발생할 확률이 높아진다.

**- 여러 세션이 비트맵 인덱스 충돌을 일으키는 경우 (enq: TX - row lock contention, mode=4)**

동시에 두 세션이 같은 리프 노드에 대해 비트맵 연산을 수행할 경우, 순서를 보장하기 위해 TX 락을 획득해야 한다. 즉， 특정 세션이 TX 락을 Exclusive하게 획득한 후, 비트맵 연산을 수행하고 아직 커밋을 하지 않았다면, 다른 세션들은 선행한 트랜잭션에 대해 TX 락을 Shared 모드로 확보하기 위해 대기함으로써 비트맵 연산이 끝나기를 기다리게 된다. 하나의 리프 노드가 넓은 범위의 ROWID를 관리하기 때문에 TX 락 경합이 광범위하게 나타날 수 있다. 즉, 비트맵 인덱스는 읽기 작업은 빈번하고, 쓰기 작업은 드문 테이블에 대해 DSS성 Select 문의 성능을 극대화하기 위해 고안된 것이다. DML이 빈번한 테이블에 대해 비트맵 인덱스를 함부로 사용하는 것은 대단히 위험하다.

**- 인덱스 리프 노드에서 Split이 발생하는 경우 (enq: TX - index contention, mode=4)**

여러 세션이 인덱스가 생성되어 있는 테이블에 대해서 많은 양의 DML을 수행하는 경우에 주로 발생한다. 이 대기현상은 자주 발생하지 않지만, 생성된 인덱스의 수가 많고 인덱스를 이루는 컬럼들의 값이 커서 리프 노드 블록이 빈번하게 Split되는 경우에는 상당한 성능 저하의 원인이 된다. 인덱스 Split에 의해 발생하는 경합을 줄이는 기본적인 방법은 동일한 리프 노드 블록에 집중적으로 데이터가 추가되는 현상을 막는 것이다. 하지만 반드시 특정 키를 기준으로 정렬이 된 형태를 유지해야 한다는 제약조건이 있다면 이런 방법을 사용할 수 없다. 인덱스의 블록 크기를 크게 설정하는 것도 하나의 해결책이 된다. 큰 크기의 블록을 사용할 경우 하나의 블록에 들어가는 엔트리의 수가 많으므로 Split이 그만큼 덜 발생하기 때문이다. 하지만 블록 크기가 증가하게 되면 buffer lock 경합에 의해 buffer busy walts 대기현상이 유발될 수 있으므로 조심해야 한다.

**- 기타 (enq: TX - contention)**

1) 분산 트랜잭션 환경에서 Prepared Transaction에 의해 락이 획득된 로우를 읽는 경우, 트랜잭션이 종료될 때까지 TX락을 Shared 모드로 획득하기 위해 대기해야 한다.

2) FLM을 세그먼트 공간관리기법으로 사용하는 경우, TFL을 확보하려는 프로세스는 TFL을 확보하지 못할 경우 이미 TFL을 확보한 Transaction의 TX락을 Shared 모드로 확보하기 위해 대기해야 한다.

3) 언두 세그먼트 헤더의 트랜잭션 테이블에서 새로운 슬롯을 할당 받고자 하는 경우 TX락을 Exclusive하게 획득해야 한다.

**- enq: UL - contention, PL/SQL lock Timer**

DBMS\_LOCK 패키지를 이용해 획득하는 락을 UL(User-defined Lock)락이라고 부른다. UL 락을 획득하기 위해 대기하는 세션은 enq: UL – contention 이벤트를 대기한다. DBMS\_LOCK.REQUEST 함수를 이용하면 UL 락을 Exclusive하게 획득하고, DBMS\_LOCK.RELEASE 함수를 이용하면 락을 해제한다.

UL 락에 의한 경합은 DBMS\_LOCK 패키지를 효율적으로 사용하지 못할 때 발생한다. UL 락의 해제는 기본적으로 락을 보유한 세션에서만 가능하다. 따라서 특정 세션이 UL 락을 장시간 보유함으로써 동시성 문제를 일으키고 있다면 세션을 강제로 종료하는 것 외에는 대안이 없다. DBMS\_LOCK.REQUEST 함수 사용시 가능하면 RELEASE\_ON\_COMMIT 옵션을 사용해서 불필요하게 락을 보유하지 않도록 하는 것이 좋다. DBMS\_LOCK 패키지와 관련된 또 하나의 대기 이벤트가 있다. DBMS\_LOCK.SLEEP 프로시저를 이용해 트랜잭션을 잠시 중단할 경우 해당 프로세스는 PL/SQL lock timer 이벤트를 대기한다.