**I/O에서의 대기 이벤트들**

**- db file scattered read**

멀티 블록 I/O를 한번 수행할 때마다 물리적인 I/O가 끝나기를 기다리게 되며, db file scattered read 이벤트를 대기하게 된다. 오라클은 DB\_FILE\_MULTIBLOCK\_READ\_COUNT (MBRC) 파라미터로 지정된 값만큼 멀티 블록 I/O를 수행한다. 풀 테이블 스캔(FTS)에서 싱글 블록 I/O를 수행하거나, MBRC보다 작은 수의 블록을 읽어 들이는 경우는 다음과 같다.

1) 익스텐트 경계에 도달한 경우

2) 스캔 도중에 캐시된 블록이 있을 경우

3) Chained Row가 있는 경우

**- 오라클의 I/O 레이어를 기준으로 db file scattered read 대기 문제에 대한 해결책**

1) 어플리케이션 레이어: 대기가 주로 발생하는 SQL문을 추출해야 한다. 불필요하게 FTS나 인덱스 풀스캔을 수행하고 있다면 SQL 문을 수정하거나 좀 더 합리적인 인덱스를 생성해주면 된다.

2) 오라클 메모리 레이어: 버퍼 캐시의 크기가 지나치게 작다면 그만큼 물리적 I/O가 반복해서 필요해서 대기도 늘어난다. 이런 관점에서 FTS를 다루는 효과적인 방법은 다중 버퍼 풀을 사용하는 것이다. 다중 버퍼풀은 세가지 측면에서 버퍼 캐시의 성능을 개선시킨다.

(1) 자주 액세스되는 객체를 메모리에 상주시킴으로써 물리적인 I/O를 최소화한다.

(2) 휘발성의 데이터는 빠른 속도로 메모리에서 재활용함으로써 메모리의 낭비를 최소화한다.

(3) 각 버퍼마다 별도의 cache buffers lru chain 래치를 사용하기 때문에 래치 경합을 감소시키는 효과가 있다.

또 다른 방법으로는 DB\_FILE\_MULTIBLOCK\_READ\_COUNT 파라미터 값을 높이는 것이다. 이 값을 높이면 FTS 속도가 향상되며 대기도 비례해서 줄어든다. 주의해야 할 것은, 이 값은 시스템 전체 레벨에서 사용하는 것보다는 해당 SQL을 수행하는 구간 동안만 높이는 것이 좋다. 추가적으로 큰 크기의 블록을 사용하는 것 또한 FTS의 성능을 향상시킬 수 있다. OLTP 시스템에서는 표준 블록 크기만 사용하는 것이 일반적이지만, DSS 시스템의 경우에는 더 큰 크기의 블록을 사용함으로써 성능개선 효과를 얻을 수 있다.

3) 오라클 세그먼트 레이어: 파티셔닝을 적절히 수행함으로써 FTS의 범위를 줄일 수 있는지 검토해야한다.

4) OS/디바이스 레이어: I/O 시스템 자체의 성능을 의심해보아야 한다.

**- db file sequential read**

db file sequential read 이벤트는 싱글 블록 I/O와 함께 발생하는 대기 이벤트이다. 한번의 싱글 블록 I/O가 발생할 때마다 한번의 db file sequential read 이벤트 대기가 발생한다. db file sequential read 대기에 의해 성능이 문제가 생기는 경우는 대부분 비효율적인 인덱스스캔이나 Chained Row나 Migrated Row에 의해 추가적인 I/O가 발생하는 경우이다.

**- 오라클의 I/O 레이어를 기준으로 db file sequential read 대기 문제에 대한 해결책**

1) 어플리케이션 레이어: 비효율적인 SQL 문장이나 비효율적인 인덱스 스캔이 자주 수행되는 경우 불필요한 물리적 I/O로 인해 대기가 증가할 수 있다. 선택도가 좋지 않은 인덱스의 사용이 대기의 주범이다. 부적절한 인덱스 사용은 I/O 뿐만 아니라 버퍼 캐시의 경합을 유발할 수 있다.

2) 오라클 메모리 레이어: 버퍼 캐시의 크기가 지나치게 작은 경우 반복적으로 물리적 I/O가 발생하고 이로 인해 대기가 증가할 수 있다. 이 경우 free buffer waits 대기가 함께 나타날 확률이 높다. 또한 이 대기 역시 다중 버퍼 풀을 사용해서 캐시를 효율적으로 사용하는 것을 고려해야한다. 만약 인덱스가 효율적으로 생성되었음에도 불구하고 기대이상으로 높다면, 클러스터링 팩터(CF)이 지나치게 높지 않은지, Row Chaining이나 Row Migration이 지나치게 많이 발생하지 않았는지 확인해야 한다.

CF는 인덱스 테이블에 대한 군집도를 의미한다. CF가 높을수록 테이블 블록을 읽는 횟수가 증가하고 이로 인해 물리적 I/O가 증가한다. 이 경우 FTS을 하거나 다른 인덱스를 이용하거나, 최후의 수단으로는 테이블 재생성을 해야 한다. 그러나, CF가 좋지 않다고 해서 항상 성능이 느린 것은 아니며, ASSM과 같은 관리기법을 사용할 경우 기존에 비해서 CF값이 높아지는 경향이 있으므로 판단에 주의를 요할 필요가 있다.

Row Chaining은 로우의 크기가 블록보다 큰 경우에 발생한다. 따라서 테이블 정의를 변경하거나 PCTFREE 값을 작게 해서 테이블을 재생성 하거나, 혹은 더 큰 크기의 블록을 사용하는 등의 방법을 사용해야한다. Row Migration은 row chaining과 달리 최초에는 하나의 블록안에 정상적으로 삽입되었으나, 이후 블록의 여유 공간이 소진된 상태에서 Update에 의해 로우 크기가 늘어난 경우에 발생한다. 이 때 로우 전체가 새로운 블록으로 옮겨가고 최초의 위치에는 옮겨간 위치의 ROWID를 기록하게 된다. 이 문제를 해결하기위해서는 PCTFREE 값을 합리적으로 부여하거나, export 후 import 하거나, alter table xxx move ..., analyze table xxx list chained rows into yyy를 수행해서 로우들을 추출하고 해당 로우들에 대해 Delete/Insert를 수행한다.

3) OS/디바이스 레이어: 위 방안으로 되지 않는다면 I/O 시스템 자체의 성능을 의심해보아야 한다.

**- direct path read**

Parallel Query 수행 시 슬레이브 세션이 수행하는 direct path I/O에 의해 발생한다. Parallel Query 수행 시 발생하는 direct path read 대기는 필연적인 것이다. Parallel Query 자체의 성능을 높이거나 I/O 시스템 자체의 성능을 높이는 것이 튜닝 포인트이다. direct path I/O를 사용할 경우 버퍼 캐시를 경유하지 않기 때문에 버퍼 캐시와 관련된 경합이 발생하는 경우, 이를 피하기 위한 방법으로 활용 가능하다.

하지만 Parallel Query를 사용하는 경우 CPU와 메모리 사용의 증가, 체크포인트 작업 증가 등의 부작용이 발생할 수 있으므로 이에 대한 충분한 고려가 필요하다. 또한 direct path read는 비록 데이터 파일에서 직접 데이터를 읽지만, 언두를 참조하는 메커니즘은 동일하며 direct path read가 임시 영역이 아닌 데이터 파일에 대한 direct path read인 것을 구분해야 한다.

**- direct path write**

Direct load 작업 (CTAS, insert /\*+ append \*/ .. 등)이 발생함을 의미한다. 이러한 작업이 요청될 경우 SGA를 경유하지 않고 데이터 파일에 직접 쓰기 작업을 수행한다. 특징은 아래와 같다.

1) SGA를 거치지 않고, 데이터파일에 직접 쓰기를 수행한다.

2) HWM 이후에 블록을 추가한다. 즉, 프리리스트나 비트맵 블록에서 관리하는 프리 블록들을 사용하지 않는다.

3) 추가된 데이터에 대해 언두를 생성하지 않는다. (CTAS는 딕셔너리 변경에 대한 언두는 생성된다.)

4) 테이블에 nologging 옵션이 주어진 경우에 리두가 생성되지 않는다.

5) 테이블에 대해 TM락을 Exclusive하게 획득하기 때문에 다른 세션에서의 DML이 허용되지 않는다.

또한 병렬 모드와 함께 사용하면 속도를 높일 수 있다. Direct 모드인 경우에는 데이터가 직접 테이블 세그먼트로 기록되지만, Parallel 모드와 병행되는 경우에는 일단 테이블 세그먼트가 속한 영구 테이블스페이스 (Permanent Tablespace) 내의 임시 세그먼트 (Temporary Segment)에 직접 기록한 다음 모든 작업이 성공적으로 끝난 후에 테이블 세그먼트에 병합된다는 것에 유의해야한다.

**- direct path read temp / direct path write temp**

정렬 작업을 위해 임시 영역을 읽고 쓰는 경우에 발생한다.

1) 어플리케이션 레이어: 정렬이 필요한 SQL 문장들이 최적화되어 있는지 검토해 보아야한다.

2) 오라클 메모리 레이어: PGA\_AGGREGATE\_TARGET 값을 적절히 변경해야 한다. 특정 세션에 대해서만 작업 공간의 크기를 높여주고 싶다면 alter session set workarea\_size\_policy = manual로 변경한 후, alter session sort\_area\_size = .. 을 이용해서 필요한만큼의 값을 주면 된다.

**- direct path read(lob) / direct path write(lob)**

LOB 컬럼을 생성할 때 NOCACHE 옵션을 시용하면 LOB 세그먼트를 읽고 쓰는 작업은 direct path I/O를 사용하며 이 경우 direct path read (Job), direct path write (Job) 이벤트를 대기하게 된다. SGA의 크기가 여유가 없거나 LOB 데이터의 크기가 크다면 NOCACHE를 사용하는 것이 유리하다. LOB 데이터가 크지 않고, 자주 액세스되는 범위가 정해져 있다면 CACHE 옵션을 사용함으로써 성능 개선 효과를 얻을 수 있다. LOB 생성 시 다음과 같은 사항들에 주의해야한다.

1) enable storage in row 옵션올 사용하는 경우 4000 bytes 보다 작은 LOB 데이터는 로우와 같은 블록에 저장된다. 따라서 Row chaining을 유발할 가능성이 높다.

2) disable storage in row 옵션을 사용하는 경우 LOB 데이터는 별도의 LOB 세그먼트에 저장되며 Row 내에는 20 bytes의 LOB Locator 정보만 저장된다 이 경우 언두 데이터는 LOB Locator에 대해서만 생성된다 LOB 데이터에 대한 실제적인 언두 정보는 언두 테이블스페이스에 저장되지 않고 같은 LOB 세그먼트 내에 저장된다. LOB 세그먼트의 언두는 PCTVERSION 옵션에 의해 제어된다. 따라서 PCTVERSION을 낮게 주는 경우 예기치 않은 ORA-01555: snapshot too old 에러가 생길 수 있다.

3) CACHE / NOCACHE, LOGGING / NOLOGGING : 만일 LOB 데이터의 크기가 크고 자주 액세스되지 않거나， 무작위적으로 액세스된다면 NOCACHE 옵션을 사용하는 것이 바람직하다. NOCACHE 옵션을 사용하는 경우 리두를 생성할 지의 여부도 지정할 수 있다. 만일 반드시 복구가 될 필요가 없는 데이터라면 Nologging 옵션을 사용함으로써 성능개선효과를 얻을 수 있다. CACHE 옵션을 사용하는 경우에는 반드시 Logging 속성을 지니게 된다. NOCACHE 옵션을 사용하는 경우 버퍼 캐시를 경유하지 않기 때문에 direct path read (Job)，direct path write (Job) 이벤트를 대기하게 된다.

4) 청크의 크기 : Out-of-line LOB인 경우 청크 단위로 LOB 데이터를 저장하게 된다. 큰 청크 사이즈의 문제는 공간이 낭비될 가능성이 높다는 것이다. 따라서 청크 사이즈를 결정할 때는 대상이 되는 LOB 데이터의 크기를 고려해서 결정해야 한다. 청크의 크기를 지나치게 작게 하는 경우에는 연속적으로 청크를 할당 받는 과정에서 오버헤드가 발생하게 된다.

**- db file parallel write**

DBWR이 더티 블록를 기록하기 위한 I/O 요청을 보낸 후 요청이 끝나기를 기다리는 동안 db file parallel write 이벤트를 대기하게 된다. db file parallel write 대기는 기본적으로 I/O 이슈라고 보면 된다. 만일 DBWR 프로세스에서 이 대기가 광범위하게 나타난다면 데이터 파일과 관련된 I/O 시스템에 심각한 성능저하 현상이 발생하는 것으로 판단할 수 있다.

만일 I/O 시스템의 성능에 문제가 없는데도 db file parallel write 대기가 사라지지 않는다면 그 때는 I/O 시스템이 감당할 수 없을 정도의 많은 쓰기 요청이 발생하는 것으로 간주할 수 있다. I/O 성능 향상은 로(raw)디바이스와 비동기 I/O를 조합하거나, OS 차원에서 Direct I/O를 사용하면 된다. I/O 작업이 너무 많은 경우 FAST\_START\_MTTR\_TARGET 파라미터 값을 크게 주거나, DBWR에 불필요한 부하를 주는 요소가 없는지 살펴볼 필요가 있다. 또한 다중 버퍼 풀을 사용하여 버퍼 캐시를 효율적으로 사용함으로써 DBWR의 부담을 줄여줄 수 있다.

**- control file parallel write**

컨트롤 파일의 갱신을 요청한 프로세스들은 갱신이 완료될 때까지 control file parallel write 이벤트를 대기하게 된다. 일반적인 환경에서는 컨트롤 파일을 갱신하는 회수가 많지 않기 때문에 control file parallel write 대기현상은 잘 발생하지 않는다. 하지만 로그 파일 스위치가 자주 발생하거나, 체크포인트가 자주 발생하는 경우, nologging에 의한 데이터파일 변경이 잦은 경우, I/O 시스템의 성능이 느린 경우 발생한다. control file parallel write 대기는 흔히 control file sequential read 대기나 enq:CF - contention 대기와 함께 발생하는 경우가 많다.