

Oracle Technical Note

Troubleshooting 시리즈(1)

오라클 메모리 심층분석

이번 호부터 연재하는 Trouble Shooting 시리즈는 필자가 한국오라클 서버지원팀에서 근무하면서 실제 고객들이 많이 문의하는 부분들을 정리한 것으로, 단편적인 해결책이 아닌 보다 근본적이고 심도 있는 내용을 제공하게 될 것이다. 각 호에서는 특정 주제를 설정, 기본적인 개념과 메커니즘을 설명하고 실제 업무에서 발생 빈도가 높은 에러를 처리하는 과정을 소개할 것이다.

- 앞으로 다룰 주제는 이번 호의 오라클 메모리를 시작으로 오라클 기본 메커니즘, 시스템 튜닝, 데이터베이스 LOCK, 분산 데이터베이스 등 고객의 관심이 많은 부분들이 될 것이다.
- 이 연재를 통해 많은 분들이 오라클을 더욱 확실히 이해할 수 있는 기회가 되었으면 한다.

오라클이 사용하는 4가지 메모리 Software Code Area와 SGA, PGA, Sort Area에 대해 설명하고, 메모리 구조와 기본 메커니즘에 대해 알아본다. 그리고 메모리와 관련된 문제 형태들과 해결책도 소개한다.

글 | 박경희(한국오라클 서버지원팀) | khpark@kr.oracle.com |

오라클 메모리의 구성

오라클이 사용하는 메모리는 크게 Software Code Area, SGA(System Global Area), PGA(Program Global Area) 그리고 Sort Area로 나누어진다.

Software Code Area란 실제로 오라클 데이터베이스가 작동되는 Oracle Compiled Code가 차지하는 영역이며, SGA(System Global Area)는 오라클 데이터베이스가 임의의 서버에서 Startup 될 때 생성되어 Shutdown 될 때까지 유지되는 메모리이다. 이 SGA는 오라클 데이터베이스가 정상적인 작동을 하기 위한 여러 가지 정보와 데이터를 유지하는 부분으로, 여러 사용자는 동시에 같은 오라클 데이터베이스에 접속하여 이 SGA 안의 데이터를 공유한다. 이러한 이유로 SGA를 Shared Global Area라고도 하는데, 이러한 공유는 여러 가지 Latch, Enqueue 등의 Oracle Lock에 의해 가능하다. 그리고 이 SGA는 크게 Shared_Pool 영역, Database Buffer Cache 영역, Redo Log Buffer 영역으로 나뉘어지는데, 이에 대해서는 뒷부분에서 자세히 다루도록 한다.

반면, PGA(Program Global Area)는 오라클 프로세스당 한 개씩 할당되는 메모리 부분으로, 다른 사용자 프로세스의 접근이 금지된다. 그리고 이는 다시 Stack Area와 Data Area로 나뉘어진다. 이 PGA는 사용자 프로세스가 사용하는 프로그램의 인수나 Program Interface Call(| 그림 2 | 의 UPI/OPI)을 위한 부분, 세션 및 사용자 정보를 저장하기 위해 할당하는 부분으로, 사용자 프로세스가 오라클 데이터베이스에 접속하는 방법에 따라 이 PGA의 일부분이 SGA에 포함될 수 있다.

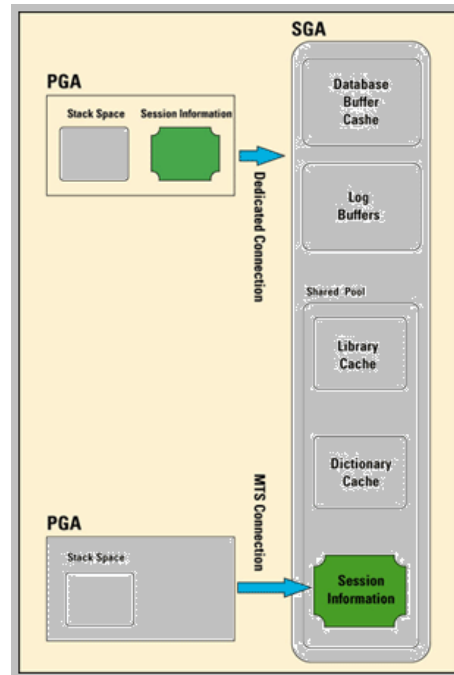


그림 1 SGA와 PGA의 관계

그리고 Sort Area는 Oracle User Process가 Sort를 필요로 할 때 할당되는 영역으로, 이는 init.ora 파일에 설정한 SORT_AREA_SIZE만큼 늘어날 수 있으며, 한 프로세스에서의 작업이 끝나면 SORT_AREA_RETAINED_SIZE로 줄어들게 된다. 이 Sort Area는 위의 PGA의 영역 중의 하나로 MTS Connection의 경우 SGA 영역에 잡히게 된다.

오라클에서 사용하는 메모리를 그림으로 살펴보면 | 그림 2 |, 가장 아래 부분은 실제 오라클 프로그램의 Compiled Code이며, 중간 부분이 오라클에 Connect하여 사용되는 사용자 프로세스 메모리이다. 그리고 그 윗 부분이 오라클 Instance가 Startup 될 때 사용되는 메모리인 SGA 영역으로, 이 SGA가 차지하는 전체 메모리 사이즈는 SVRMGRL > CONNECT INTERNAL: SVRMGRL> SHOW SGA로 전체 사이즈를 참조할 수 있다 .

이 메모리 사이즈는 서버의 Real Memory 안에서 차지하는 값인 만큼 그 크기는 서버의 사용용도에 따라 결정되며 오라클 전체 시스템 튜닝 작업을 통해 적절한 값을 설정해 주는 것이 바람직하다. 그러나 오라클도 OS 측면에서 보면 하나의 프로세스로 간주될 수 있으므로 오라클이 차지하는 메모리가 50%를 넘지 않도록 한다.

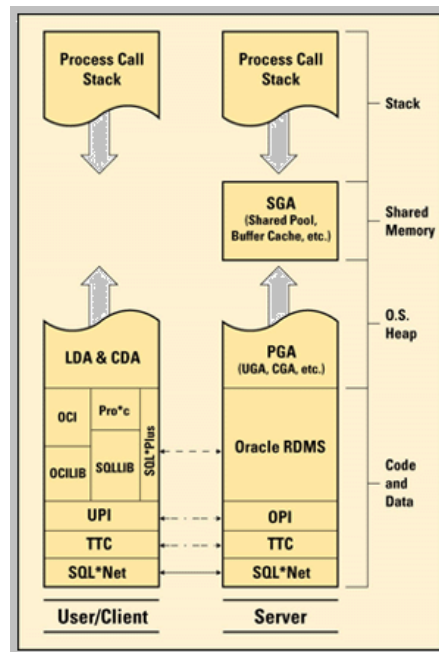


그림 2 오라클 메모리 구성

오라클 메모리의 구조와 메커니즘

그러면 오라클에서 사용하는 메모리의 각 구조와 기본 메커니즘에 대해 하나씩 살펴 보도록 하자

SGA(System Global Area)

SGA는 사용되는 용도에 의해 크게 Database Buffer Cache, Shared_Pool, Redo Log Buffer로 나눌 수 있다 .

Database Buffer Cache

Database Buffer Cache는 오라클 서버 프로세스가 데이터의 처리를 위해 사용하는 부분으로, 최근에 사용된 데이터베이스의 데이터 블록을 저장하고 있는 Database Buffer Set을 일컫는 것이다. 이 Buffer Cache는 변경된 데이터뿐만 아니라 변경되지 않는 데이터도 가지고 있는데, 이는 수행속도 향상을 위해 디스크 입출력보다는 메모리 액세스를 하게 하기 위함이다.

이 Database Buffer Cache는 Cache Buffer Chain List, Dirty List, LRU (Least Recently Used list)로 나누어 관리되고, 이들의 3가지 List를 관리하면서

사용자의 작업시 필요한 Buffer를 제공하는 역할을 하는 것이 DB Writer(DBWR)이다.

이 3가지 List를 통해 Free Buffer는 오라클 서버 프로세스에 할당되어 사용되고, 사용 후 Dirty Buffer가 된 Buffer들은 DBWR에 의해 디스크에 쓰여진 후 다시 Free Buffer가 되어 오라클 서버 프로세스에 의해 재사용되는 작업을 반복하게 된다. 이를 그림으로 표현해보면 | 그림 3 |과 같다.

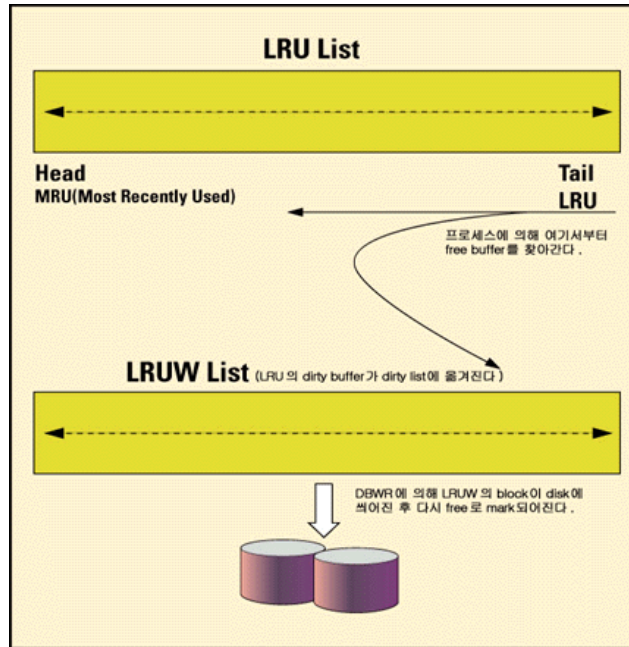


그림 3 Database Buffer Cache에서 DBWR의 역할

이 Database Buffer의 개수는 \$ORACLE_HOME/dbs/initSID.ora 파일의 DB_BLOCK_BUFFERS의 변수 값만큼 오라클 인스턴스가 시작될 때 할당되는 것으로, 이 Buffer들은 앞으로 설명할 LRU List나 LRUW List(또는 Dirty List)의 어느 한 군데에 속하게 된다.

Cache Buffer Chain List

각 Database Buffer는 Data Block Address(DBA)가 Hash Function에 의해 해시되어 Hash Table에 할당되어 관리되는데, 이를 Cache Buffer Chain List라 한다.

즉 Cache Buffer Chain List란 양방향의 링크된 리스트로, 인스턴스가 시작될 때 할당되는 Hash Table로 구성된다. 이 Hash Table 안의 Bucket은 각 Database Block Buffer들의 Header 정보를 가지며, 이 각 Buffer들은 뒤에 설명할 LRU List 나 Dirty List의 한 가지에 속하게 되는 것이다.

Hash Bucket의 개수는 initSID.ora 파일에 그 개수를 명시적으로 지정할 수도 있으나 기본적으로 db_block_buffers/4보다 큰 최소의 소수(prime number)가 할당되어 Database Buffer들을 관리한다.

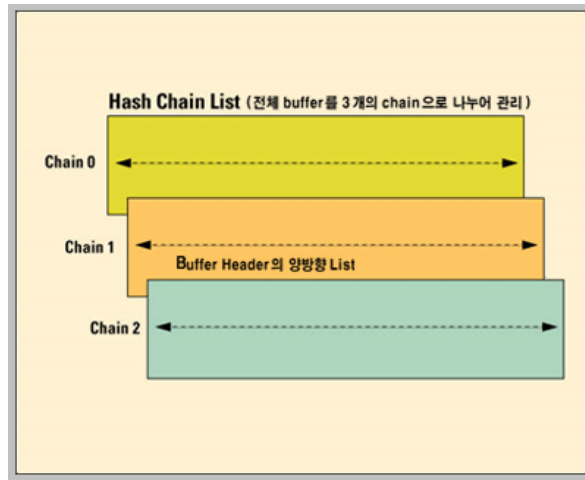


그림 4 Cache Buffer Chain List

LRU(Least Recently Used list)

Database Buffer Cache의 Buffer들은 관리를 위해 LRU와 LRW(Dirty List) 중의 하나에 할당되는데, 이 중 LRU List에 속한 Buffer들은 Free Buffer, Pinned Buffer, Dirty Buffer 중의 한 가지 특성을 갖게 된다.

처음에는 Free 상태이던 Buffer들이 오라클 프로세스에 의해 사용된 이후에는 상태가 Pinned로 바뀌어 LRU List의 Header 부분인 MRU(Most Recently Used) 부분에 위치하게 된다. 그리고 이 Buffer들이 더 이상 사용되지 않게 되면 차츰 LRU List의 꼬리 부분인 LRU 부분으로 옮겨오게 되는 것이다.

그러므로 오라클 프로세스는 특정 작업을 위해 현재 메모리에 있지 않는 데이터는 디스크에서 데이터를 읽어오기 위해서 Free Buffer를 필요로 하고, 이 Free Buffer Block을 얻기 위해 LRU List에서 제일 사용된 빈도가 작고 오래 전에 사용되었던 Buffer를 사용하기 위해 LRU List의 꼬리 부분부터 찾기 시작하는 것이다. LRU List의 각 Buffer의 성격을 자세히 살펴보면, 다음과 같다.

- Pinned Buffer : 현재 유저가 사용중이어서 재사용할 수 없는 상태
- Free Buffer : Dirty Buffer가 디스크에 쓰여진 후 Free로 마크되어 사용할 수 있는 상태로 되었거나 내용이 변경되지 않아 사용 가능한 Buffer
- Dirty Buffer : 유저가 사용하여 내용이 변경된 Buffer이나, 현재 유저나 웨이터(waiter)가 없는 상태이지만 아직 디스크에 쓰여지지 않은 Buffer

LRW(Dirty List)

LRW(또는 Dirty List)는 오라클 프로세스들에 의해 모아진 Dirty Buffer들을 모아놓은 List이다. 이 Dirty Buffer들은 오라클 프로세스들이 자신이 사용할 Free Buffer를 찾기 위해 LRU List를 꼬리 부분부터 찾아나가다 Dirty Buffer를 만나면 이를 LRW List에 옮겨놓은 것들로, 이 Buffer들은 결국 DBWR(DB Writer)에 의해 디스크에 쓰여지고 다시 Free로 마크되어 LRU List로 가게 되는 것이다.

이제 이러한 기본적인 관리체계를 이해하였다는 가정하에 실제 오라클 프로세스가 Free Buffer를 찾는 과정을 살펴보기로 한다.

오라클 프로세스는 우리에게 결과를 주기 위해 디스크나 Buffer Cache에서 해당 데이터를 읽어와야 하고, Buffer Cache에 데이터가 없는 경우 디스크에서 해당 블록의 데이터를 Free Buffer에 읽어들이는 것이다.

이 Free Buffer를 찾는 순서를 간략하게 살펴보면 다음과 같다.

1. 오라클 프로세스가 LRU List에 Lock을 걸고 LRU의 끝부분부터 Free Buffer 블록을 찾기 시작한다. 이를 찾는 중에 Dirty Buffer를 만나면, 이 Dirty Buffer를 LRUW List에 옮긴다.
2. LRU List를 찾을 때도 끝까지 다 찾는 것은 아니고, Kernel에서 지정하는 어느 특정 값만큼 Depth를 정하여 스캔하며, 이 개수(foreground scan depth) 내에서 찾지 못하면 오라클 프로세스는 더 이상 LRU List는 읽지 않고, DBWR에게 Dirty Buffer를 모으도록 하는 메시지를 보내는 작업이 수행되고 LRU Latch는 해제된다.
3. 메시지를 받은 DBWR는 LRU Latch Lock을 걸고, LRU 꼬리 부분에서 DBWR Scan Depth만큼의 디스크에 쓰여져야 할 Dirty Buffer들을 모으는 Large Batch Write를 수행한다.
4. LRUW에 모인 Dirty Buffer는 DBWR에 의해 디스크로 쓰여지고 나면 이 Buffer는 Free Buffer로 되어 다시 사용될 수 있도록 LRU의 끝부분에 위치하게 된다.

Shared_Pool

SGA 영역의 하나인 Shared_Pool은 Library Cache 영역, Dictionary 영역, Control Structure의 3 가지로 구성된다.

그리고 이들의 전체 사이즈는 initSID.ora 파일에 설정한 SHARED_POOL_SIZE 만큼 잡히게 된다. 이를 그림으로 살펴보면 | 그림 5 |와 같다 .

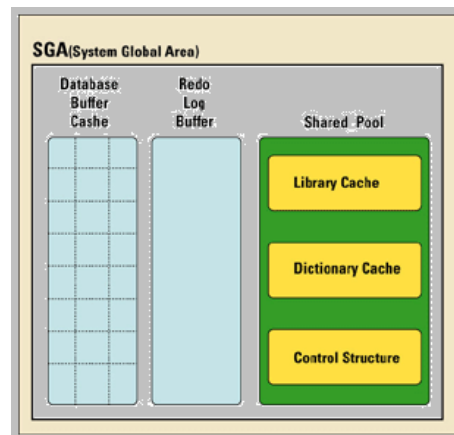


그림 5 SGA의 구성

Library Cache

Library Cache의 가장 중요한 목적은 Library Cache 안에 저장된 Object를 가장 빠르게 찾고 저장하는 기법을 제공하는 것으로, 오라클의 모든 동적인 메모리를 관리하는 Heap Manager와 Library Cache Manager에 의해 관리된다.

이 목적을 위해 Library Cache Manager도 Hashing 기법을 이용하여 Object에 대한 이름을 갖는 Handle을 찾고, 이 Handle은 다시 해당 Object를 가리키고 있다.

먼저 Library Cache에서 찾는 Library Object 는 다음과 같다.

- Package
- Procedure
- Function
- Trigger
- Shared Cursor
- Anonymous PL/SQL Block
- Table Definition
- View Definition
- Form Definition

이처럼 Library Cache 안에 저장되어 사용되는 Object를 보면 오라클 서버는 SQL 문과 PL/SQL 블록을 저장하기 위해 사용한다. 또한 이 Library Cache에 존재하는 문장을 실행하면 그 문장을 다시 Parsing 하지 않고 사용하며, 이 Library Cache를 관리하는 데는 Database Buffer Cache와 마찬가지로 LRU 알고리즘을 사용하여 관리한다. 다음은 Library Cache Manager가 원하는 Object를 찾는 순서이다.

1. 필요한 Object의 Namespace, Object Name, Owner, Database Link 값을 Hash Function을 적용하여 해당 Object가 존재하는 Hash Bucket을 찾아낸다(SQL 문장의 경우 앞뒤 64바이트의 글자를 이용한다).
2. Hash Bucket의 Linked List를 따라 원하는 Object가 실제 존재하는지를 체크한다.
3. 만약 Object가 존재한다면 찾은 Object를 사용하고, 존재하지 않는다면 Library Cache Manager는 주어진 이름으로 Empty Object를 생성한다.
4. 위에서 생성된 Empty Object를 Hash Table에 포함시킨 후 오라클 프로세스에게 해당 Object를 로드하도록 요청한다.
5. 오라클 프로세스가 해당 Object를 디스크에서 읽어 Heap Manager에 의해 새로 할당된 메모리에 올려놓는다.
6. 해당 Object를 사용한다.

| 그림 6 |은 SQL 문이 실제 Library Cache에서 공유되는 과정을 보여준다. SQL 문이 공유되기 위해서는 유사한 SQL 문에 대해서는 다음의 예제처럼 Bind Variable을 사용하여 같은 SQL을 사용할 수 있도록 하며, 대소문자, 스페이스까지 같도록 유의하여야 한다.

```

SQL>SELECT CUST_NAME FROM CUST WHERE
      CUST_NO=10000;
SQL>SELECT CUST_NAME FROM CUST WHERE
      CUST_NO=10001;
→SQL>SELECT CUST_NAME FROM CUST WHERE
      CUST_NO= :V_CUST_NO

```

더욱이 실제 Library Cache 안에 공유되는 부분은 SQL 텍스트뿐만 아니라 Execution Plan, Bind Variable Data Type과 Length여서 가능한 SQL 문을 공유할 수 있도록 하여야 한다.

위의 알고리즘을 이용해 아래의 SQL 문을 찾아가는 순서를 살펴보자.

1. SQL 텍스트를 Hash Function을 적용해 이 SQL 문의 Handle이 있는 Bucket 을 찾아간다.
2. 해당 Bucket 안에는 다른 SQL 문이면서 Hash Function 값만이 같은 여러 SQL들이 있을 수 있으므로 이 Bucket List를 따라가면서 공유 가능한 같은 SQL이 있는지를 체크한다.
3. 같은 문장을 발견하였지만 서로 다른 Version이 있을 수 있다. 이 Version이란 SQL 문은 같지만 서로 다른 유저의 Object를 사용한 경우이거나, Bind Variable Data Type이 다른 경우, 서로 다른 Application Info를 사용하는 경우 서로 다른 Version으로 존재하며, 이는 공유되지 않는다.
4. 같은 문장을 찾지 못한 경우는 SQL 문은 다시 Parsing 된다.

Dictionary Cache

오라클은 SQL 문을 Parsing 할 때 Data Dictionary를 계속적으로 참조하는데, Dictionary Cache란 이들 Dictionary Data를 올려놓고 사용하는 메모리 부분이다.

여기에서 Data Dictionary란 참조되는 데이터베이스의 모든 정보와, 구조, 그들의 사용자 등이 정의된 모든 Table과 View들의 모음을 일컫는다. Dictionary Cache는 각 참조 데이터를 ROW 단위로 가지므로 ROW Cache라고도 한다.

이 부분의 튜닝을 위해서는 V\$ROWCACHE의 Hit율을 참조하면 가능하고, SHARED_POOL_SIZE 패러미터를 사용하여 간접적으로 크기를 조절할 수 있다 .

체크포인트

이제까지 SGA가 관리되는 부분을 간략히 살펴보았는데, 이제 이러한 부분이 전체 데이터베이스 부분의 성능에 영향을 미칠 수 있는 경우에 대해 살펴보도록 한다.

Buffer 오퍼레이션을 줄이기 위해 다음의 2가지 방법을 사용한다.

1. Temporary 타입의 Tablespace 사용

Permanent 타입의 Temp Tablespace를 사용하는 경우는 Temporary Space를 요구하는 경우, 오라클은 항상 새로운 Temporary Segment를 생성하여야 한다. 더욱이 같은 사용자가 이전에 수행하였던 작업과 동일한 작업을 수행하여도 새로운 세그먼트를 생성한다. 그러나 Temporary 타입은 작업 종료 후에도 De-allocate 되지 않고, Temporary Segment를 구성하는 Extent에 대한 정보를 Sort Extent Pool(SEP) element로 구성하여 SGA 내에 유지하며, 이후 재사용시 이용함으로써 작업이 끝난 후 계속 Cleanup 하는 SMON의 작업을 줄일 수 있는 이점이 있다.

이를 위해 Buffer 관리시 DBWR이 Disk에 쓸 때 Dirty Buffer를 찾거나 Buffer Header를 찾을 때 Sort block이나 Temporary 타입의 세그먼트는 Buffer의 핸들링을 많이 줄일 수 있다.

2. Direct option을 이용한 Sort, Sql*load, Export

Sort, Data Load, Export 시 Direct option을 사용하면 Buffer Cache를 이용하지 않고 디스크 블록 구조와 같은 메모리를 이용해 바로 데이터 파일에 쓰므로 Buffer의 핸들링을 줄일 수 있다.

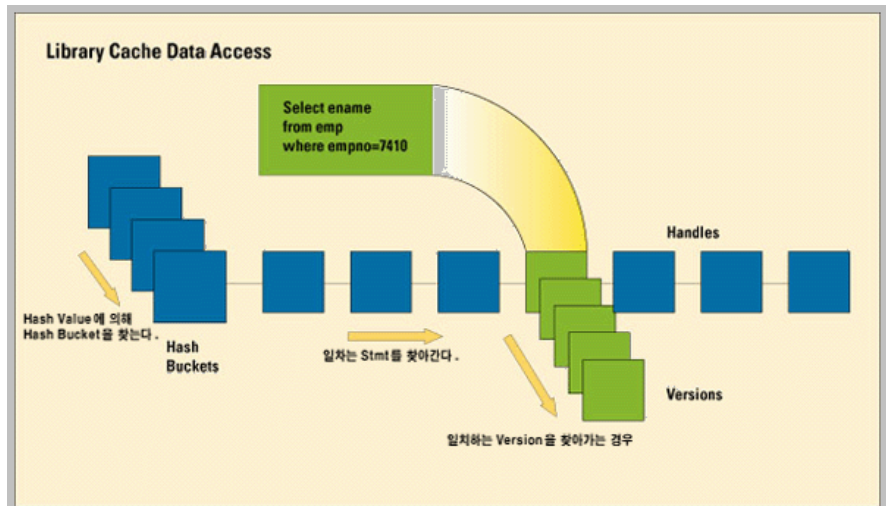


그림 6 Library Cache데이터 액세스

앞에서 살펴보았듯이 Buffer의 관리를 위해 사용된 LRU, LRU List를 오라클 프로세스가 사용하기 위해서는 먼저 오라클에서 사용하는 Lock의 일종인 Latch를 얻어야 가능하다. 이 Latch는 기본적으로 1개만 잡히도록 되어 있으나, 여러 사용자 등으로 인해 이의 경쟁이 발생할 수 있으므로 여러 개의 CPU를 사용하는 경우에는 DB_BLOCK_LRU_LATCHES를 CPU*2개만큼 설정함으로써 오라클 프로세스가 Buffer Cache List를 액세스하기 위해 얻는 Latch의 경쟁을 줄이도록 한다.

V8.x부터는 BUFFER_POOL_KEEP, BUFFER_POOL_RECYCLE, DEFAULT의 Multiple Buffer Pool을 제공하여 각 세그먼트마다 특성에 따라 다른 Buffer를 사용함으로써 Buffer를 효율적으로 사용할 수 있으므로 이의 사용을 신중히 고려해본다.

이의 사용을 위해서는 initSID.ora 파일에

```
DB_BLOCK_BUFFERS=1000
DB_BLOCK_LRU_LATCHES=6
BUFFER_POOL_KEEP=(Buffers:400,lru_latches:2)
BUFFER_POOL_RECYCLE=(Buffers:100,lru_latches:1)
```

처럼 설정 후 실제 Object 생성시 Storage 절의 BUFFER_POOL에 실제 Object가 사용할 Buffer Cache의 Pool을 지정할 수 있다.

```
SQL>CREATE TABLE cust (cust_no number,...)
STORAGE(BUFFER_POOL KEEP) ;
```

이의 의미는 전체 Buffer Cache 1000 블록 중 Keep Pool은 400 블록, Recycle Pool은 100개의 블록을 가지며, 나머지 Default Pool에 500개의 블록이 할당된다는 것이다. Latch는 5개 중 Keep에 2개, Recycle에 1개, Default에 3개가 할당되어 사용된다. 이 Keep Pool은 LRU 알고리즘에 의해 관리되므로 해당 세그먼트가 100% 유지되지는 않지만, 오랫동안 Cache에 유지하고자 하는 경우 사용된다. 반면 Recycle은 자주 사용되지 않는 세그먼트의 유지를 위해 사용하며, 그 외 나머지가 기존의 Buffer Cache와 동일한 Default Pool이다.

적정한 양의 DB_BLOCK_BUFFERS 개수 사용으로 Buffer Cache의 Hit율을 70~85% 이상을 유지하도록 한다. 이 Hit율을 체크하기 위한 스크립트는 다음과 같다.

```
select trunc((1-(phy.value/(cur.value +
con.value))) * 100,5)
from v$sysstat phy, v$sysstat
cur,v$sysstat con
where phy.name='physical reads'
and cur.name='db block gets'
and con.name='consistent gets'
/
```

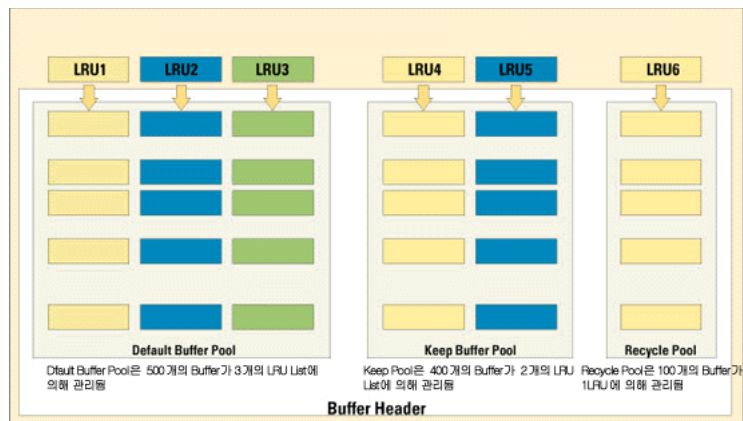


그림 7 3개의 Buffer Pool과 6개의 LRU List에 의해 관리됨

Library Cache의 튜닝을 위해

```
SQL> select cum(pins), sum(reloads)
       From v$libraryCache;
```

을 점검하여 pins에 대한 reload가 1% 이상이면 Library Cache Miss를 줄일 수 있도록 Shared_Pool_Size를 늘리도록 한다.

Dictionary Cache의 튜닝을 위해서

```
SQL> select sum(gets) / (sum(gets) +
       sum(getmisses)) from v$rowCache;
```

의 Hit율이 90% 이하인 경우 Shared_Pool_Size를 늘리도록 한다.

Troubleshooting

메모리 할당 에러

때때로 임의의 작업 수행시 OUT OF MEMORY를 접하는 경우가 있다. 이는 사용자 프로세스가 사용하고자 하는 메모리를 할당하지 못하는 경우이며, 이 경우는 대부분 Parallel 작업을 Dedicated Connection 방법을 이용하여 오라클에 접속한 경우이다.

이는 사용자 프로세스당 할당되는 PGA 영역 중 Dedicated Connection을 이용해 오라클에 접속한 경우 실제 사용된 후의 메모리는 OS에 반환되는 것이 아니고, 해당 프로세스에 할당된 메모리(UGA)에 반환된다.

실제로 Sort_area_size가 크게 할당되고, Sort_area_retained_size가 작은 경우 Sorting 작업은 끝났지만 이 메모리는 자기 프로세스만이 사용 가능한 부분으로 반환되어 프로세스가 끝날 때까지 사용되지 못하여 전체 메모리 고갈 현상을 유발하거나 Performance 부분에 심각한 문제를 야기할 수 있다.

Ora-4031

이 에러는 사용자가 특정 Object를 사용하고자 할 때 해당 Object를 위해 Shared_Pool에 공간을 할당하려 할 때 충분한 공간을 얻지 못하는 경우이다. Shared_Pool 영역에 차지하는 크기가 큰 PL/SQL 사용시 오라클은 공간 확보를 위해 현재 사용되지 않는 Object를 Flush시킨다. 기존 V7.2. 이하에서는 특정 프로시저를 Shared_Pool에 올리기 위해 연속된 공간을 필요로 하였으나, 이후 버전에서는 꼭 연속되지 않아도 사용이 가능하여 ORA-4031 에러가 발생하는 경우가 많이 줄어들었다. 이의 해결을 위해서는 Shared_Pool을 늘려주도록 한다 .

또한 V7.2 이하 버전에서 이 에러를 유발시키는 문제인 메모리의 단편화(fragmentation)를 줄이기 위해서, 또는 일반적인 경우 자주 사용하는 Object가 메모리에서 밀려 내려가는 경우를 없애기 위해서 Object를 미리 Shared_Pool에 Keep 한다.

Shared_Pool에 Procedure를 Keep 하기

Shared_Pool에 크기가 큰 프로그램을 Keep 하기 위해서는 아래의 예처럼 DBMS_SHARED_POOL 패키지를 이용할 수 있다. 단 이처럼 DBMS_SHARED_POOL을 사용하기 위해서는 다음의 단계를 통해 이 패키지를 생성하여야 한다.

```
SQL> @$ORACLE_HOME/rdbms/admin/dbmspool.sql
Package created.
```

또 Package body를 생성하기 위해

```
SQL> @$ORACLE_HOME/rdbms/admin/prvtpool.plb
View created.
Package body created.
SQL> GRANT EXECUTE ON DBMS_SHARED_POOL TO
user_name ;
Grant succeeded.
```

각 Object를 Keep 하는 방법은 다음과 같다.

Procedure, Function, Package의 Keep

1. Stored PL/SQL Object를 Call 한다.
2. PL/SQL Object 이름이 PRINT_NAME 인 경우

```
SQL> exec dbms_shared_pool.keep('print_name', 'p')
PL/SQL procedure successfully completed.
```

Trigger의 Keep

1. Trigger를 생성한다.
2. Trigger name이 INSERT_ANOTHER 인 경우

```
SQL> exec dbms_shared_pool.keep('insert_another', 'r')
PL/SQL procedure successfully completed.
```

Sequence의 Keep

1. 다음처럼 Sequence를 Reference 한다 .

```
select khpark.currval from dual;
```

2. Sequence name이 KHPARK인 경우

```
SQL> exec dbms_shared_pool.keep('khpark', 'q')
PL/SQL procedure successfully completed.
```

Shared Cursor에 SQL 문의 Keep

여러 사용자간에 주로 사용하는 SQL 문을 공유하기 위해서는 다음의 방법을 이용한다.

예를 들어, SQL> select ename from emp where empno = 10000문을 Library Cache 안의 Shared Cursor 부분에 Keep 하고자 하면,

```
SQL> SELECT ADDRESS, HASH_VALUE FROM V$SQLAREA
WHERE SQL_TEXT='select ename from emp
where empno=10000'
```

를 이용하여 원하는 SQL 문에 대해

```
ADDRESS = 70378A10
HASH_VALUE = -614308548
```

를 구한다.

```
SQL> EXEC DBMS_SHARED_POOL.KEEP('70378A10, -  
614308548', 'c')
```

으로 Shared Cursor에 Keep 하도록 한다. 또 이의 Keep 상태는

```
SQL> select * from v$db_object_cache where kept='  
YES';
```

이거나

```
SQL> exec dbms_shared_pool.sizes(0) ;
```

로 체크가 가능하다.

DBMS_SHARED_POOL.SIZES(size)는 제시한 Size 이상의 Keep 된 Object를 나타내주는 프로시저이다.

각 Object를 Shared_Pool에 유지하던 기능을 해제할 때는 UnKeep 프로시저를 사용한다. 이는 dbms_shared_pool.unkeep 을 이용하여

```
SQL> execute dbms_shared_pool.unkeep ('print_name','p')
```

처럼 사용한다.

이상에서 대략적으로 오라클의 메모리 구조와 이에 의한 문제 형태를 살펴보았다. 여기에서 메모리 튜닝 부분은 거의 다루지 않았는데, 이는 다음 기회의 시스템 튜닝 부분에서 자세히 다룰 예정이다.



한국오라클(주)

서울특별시 강남구 삼성동 144-17
삼화빌딩
대표전화 : 2194-8000
FAX : 2194-8001

한국오라클교육센터

서울특별시 영등포구 여의도동 28-1
전경련회관 5층, 7층
대표전화 : 3779-4242~4
FAX : 3779-4100~1

대전사무소

대전광역시 서구 둔산동 929번지
대전둔산사학연금회관 18층
대표전화 : (042)483-4131~2
FAX : (042)483-4133

대구사무소

대구광역시 동구 신천동 111번지
영남타워빌딩 9층
대표전화 : (053)741-4513~4
FAX : (053)741-4515

부산사무소

부산광역시 동구 초량동 1211~7
정암빌딩 8층
대표전화 : (051)465-9996
FAX : (051)465-9958

울산사무소

울산광역시 남구 달동 1319-15번지
정우빌딩 3층
대표전화 : (052)267-4262
FAX : (052)267-4267

광주사무소

광주광역시 서구 양동 60-37
금호생명빌딩 8층
대표전화 : (062)350-0131
FAX : (062)350-0130

고객에게 완전하고 효과적인
정보관리 솔루션을 제공하기 위하여
오라클사는 전 세계 145개국에서
제품, 기술지원, 교육 및
컨설팅 서비스를
제공하고 있습니다.

<http://www.oracle.com/>
<http://www.oracle.com/kr>