(1) Query Coordinator와 병렬 서버 프로세스

- Query Coordinator(QC) : 병렬 SQL문을 발생한 세션
- 병렬 서버 프로세스 : 실제 작업을 수행하는 개별 세션

↕ oracle 병렬처리는 'multi processing'이다.

병렬처리 수행 과정 및 QC의 역할

1. 병렬 SQL이 시작되면 QC는 사용자가 지정한 병렬도(DOP, Degree Of Parallelism)와 오퍼레이션 종류에 따라 하나 또는 두개의 병렬 서버집합(Server Set)을 할당한다. 우선 서버풀(Parallel Execution Server Pool)로부터 필요한 만큼 서버 프로세스를 확보하고, 부족분은 새로 생성한다.
   - parallel_min_servers 파라미터로 설정된 수만큼의 병렬프로세스를 오라클이 기본적으로 생성해 서버풀에 담아 둔다.
   - parallel_max_servers 파라미터에 의해 생성가능한 최대 병렬서버 갯수가 결정된다.
2. QC는 각 병렬서버에게 작업을 할당한다. 작업을 지시하고 일이 잘 진행되는지 관리,감독하는 작업반장의 역할이다.
3. 병렬로 처리하도록 사용자가 지시하지 않은 테이블은 QC가 직접 처리한다. 예를 들어, 아래와 같은 실행계획에서 dept테이블을 직렬로 읽어 병렬서버에 전송하는 8~9번 오퍼레이션은 QC의 몫이다.
4. QC는 각 병렬서버로부터의 산출물을 통합하는 작업을 수행한다. 예를 들어 집계함수(sum, avg, min, max 등)가 사용된 아래와 같은 병렬쿼리를 수행할 때, 각 병렬서버가 자신의 처리범위 내에서 집계(4번 단계)한 값을 QC에게 전송(3번 단계)하면 QC가 최종 집계 작업을 수행(1번 단계)한다.
5. QC는 쿼리의 최종 결과집합을 사용자에게 전송하며, DML일때는 갱신건수를 집계해서 전송해 준다. 쿼리결과를 전송하는 단계에서 수행되는 스칼라서브쿼리도 QC가 수행한다.

- 병렬 처리가 전혀 발생하지 않더라도 parallel_min_servers에 지정된 최소 개수를 유지 한다고 함.
  확인결과 parallel_min_servers 의값은 0 이였음.

```
SET AUTOT ON

SELECT /*+ ordered use_hash(d) full(d) full(e) noparallel(d) parallel(e 4) */
       COUNT(*)
     , MIN(sal)
     , MAX(sal)
     , AVG(sal)
     , SUM(sal)
  FROM dept d, emp e
 WHERE d.loc = 'CHICAGO'
   AND e.deptno = d.deptno
;

Execution Plan
----------------------------------------------------------
Plan hash value: 321505112

해당 operation 아랫은 'QC'가 처리함
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |     1 |    47 |     6  (17)| 00:00:01 |        |      |            |
|   1 |  SORT AGGREGATE          |          |     1 |    47 |            |          |        |      |            |
|   2 |   PX COORDINATOR         |          |       |       |            |          |        |      |            |
|   3 |    PX SEND QC (RANDOM)   | :TQ10002 |     1 |    47 |            |          | Q1,02  | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE       |          |     1 |    47 |            |          | Q1,02  | PCWP |            |
|*  5 |      HASH JOIN ← driving table에서도 병렬 처리 가능.|          |     5 |   235 |     6  (17)| 00:00:01 | Q1,02  | PCWP |            |
|   6 |       BUFFER SORT        |          |       |       |            |          | Q1,02  | PCWC |            |
|   7 |        PX RECEIVE        |          |     1 |    21 |     3   (0)| 00:00:01 | Q1,02  | PCWP |            |
|   8 |         PX SEND HASH     | :TQ10000 |     1 |    21 |     3   (0)| 00:00:01 |        | S->P | HASH       |
|*  9 |          TABLE ACCESS FULL| DEPT     |     1 |    21 |     3   (0)| 00:00:01 |        |      |            |
|  10 |       PX RECEIVE         |          |    14 |   364 |     2   (0)| 00:00:01 | Q1,02  | PCWP |            |
|  11 |        PX SEND HASH      | :TQ10001 |    14 |   364 |     2   (0)| 00:00:01 | Q1,01  | P->P | HASH       |
|  12 |         PX BLOCK ITERATOR|          |    14 |   364 |     2   (0)| 00:00:01 | Q1,01  | PCWC |            |
|  13 |          TABLE ACCESS FULL| EMP      |    14 |   364 |     2   (0)| 00:00:01 | Q1,01  | PCWP |            |
-------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   5 - access("E"."DEPTNO"="D"."DEPTNO")
   9 - filter("D"."LOC"='CHICAGO')
```

용어 설명:

*** Query Coordinator(QC)**

parallel execution을 수행한 foreground process, 즉 query를 수행한 session으로 query slave로 부터 데이터를 받게 된다.

*** Slaves** ← slave process

slave는 disk로 부터 바로 데이터를 읽거나 다른 slave에 의해 만들어진 table queue 구조로 부터 읽어 그것을 자신의 queue에 write한다. slave 가 disk로 부터 데이터를 읽을 때 buffer cache를 거치지 않는 direct I/O path read를 수행한다. slave는 이미 변경되었으나 disk에 반영되지 않은 데이터를 buffer cache에서 disk로 강제로 flush 한다. 그후 data를 direct path I/O로 읽는다.

slave는 producers slave와 consumer slave의 두개의 종류가 있다.

1) **Producers slave**는 QC로 부터의 주어진 ROWID range나 partition을 통해 data block을 읽어 관련 데이터를 읽어 온다. 이 데이터는 table queue에 보내지며 이를 다시 QC나 consumer slave가 처리하게 된다.

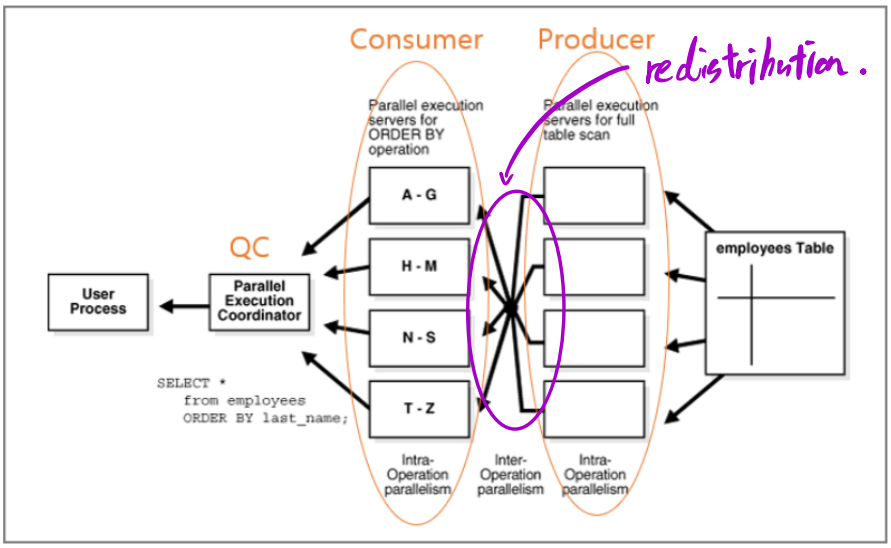
2) **Consumer slave**는 producer slave에 의해 보내진 table queue의 데이터를 처리하여 QC로 "dequeue"하게 된다.

consumer slave와 producer slave로 나눠져있기 때문에 sort가 필요한 parallel execution에는 두배의 parallel query slave가 필요하게 된다.

*** Table Queues**

(TQ)는 process가 다른 process에게 row를 보내기 위한 queue이다. Table queue는 producer slave가 consumer slave에게, consumer slave가 query coordinator에게 데이터를 보내기 위해 사용된다.

병렬처리에 대해 설명된 문서들을 찾아보다 보면 자주 접하게되는 용어가 있습니다.
QC, Consumer, Producer 라는 용어입니다.



(출처 : Oracle Document)

테이블에서 데이터를 읽는 작업을 하는 병렬프로세서들을 **Producer** 라고 하고, 이 읽은 데이터를 받아서 Sorting(정렬), DML DDL, Join 등을 수행하는 작업을 하는 병렬프로세들을 **Consumer** 라고 합니다.
그리고, 작업된 결과들을 취합(통합)하는 작업을 하는 프로세서가 있는데 이를 **QC(Query Coordinator)** 라고 합니다.
병렬처리 힌트 쓰는데, 당장 이런걸 꼭 알 필요는 없지만,
병렬처리 관련해서 구글링하다보면 영어문서에는 이런 용어들이 빈번하게 나오기 때문에 잘 알아둘 필요가 있겠습니다.

3. Database Parameter Setup for Parallel Execution

PARALLEL_MAX_SERVERS

인스턴스 당 최대 사용가능한 slave 개수. 0 설정시 parallel query 사용 불가.

PARALLEL_MIN_SERVERS

instance startup 시 미리 띄워 놓을 slave 개수.

PARALLEL_MIN_PERCENT

기본적으로 optimizer가 query를 parallel로 수행하도록 SQL 수행 계획을 생성되더라도 수행시 요구되는 parallel slave를 띄우기 위한 충분한 resource가 없을 경우, 사용자에게 특별한 메세지 없이 serial하게 query를 수행한다. 이러한 경우가 발생하면 query 수행시간이 예상보다 더 늦어지는 일이 발생할 수 있다.

PARALLEL_MIN_PERCENT는 수행시 충분치 못한 resource로 인한 parallel execution이 무시되는 경우를 방지하고 에러를 출력한다. PARALLEL_MIN_PERCENT는 가능한 parallel execution slave의 percent로 설정한다.

만약 설정한 percentage 만큼의 query slave를 띄울 수 없다면 serial로 수행하지 않고 ORA-12827 에러를 발생한다.

예) 만약 resource 부족으로 slave를 띄우지 못했을 경우:

0	에러 없이 serial execution을 수행한다.
50	best parallel execution time의 2배 정도까지의 execution time은 accept하고 에러 없이 수행
100	주어진 parallel query를 수행할 수 있는 resource가 없는 경우 ORA-12827 에러 발생.

OPTIMIZER_PERCENT_PARALLEL

PARALLEL_ADAPTIVE_MULTI_USER

PARALLEL_ADAPTIVE_MULTI_USER init parameter가 TRUE로 설정되어 있을 경우 parallel execution 사용할 때 multi-user 환경에서의 성능 향상을 위한 algorithm을 사용하게 된다. 이 algorithm은 query가 수행되는 시점에 system load에 따라 자동으로 요청하는 parallel의 degree를 줄여 query를 수행한다.

예를 들어 17 CPU 시스템에서 default parallel degree가 32로 설정되어 있다면 첫번째 사용자는 32개의 parallel slave process를 사용해 query가 수행된다. 그러나 두번째 사용자가 query를 수행할 경우 16개의 parallel slave process가 사용되며, 세번째 사용자는 8개, ..

결국 32번째 user는 1개의 parallel slave process를 사용하게 된다.

1. Introduction to Parallel Execution

대용량 table에 대한 Full table scan이나 큰 size의 index 생성 등 많은 데이터를 handle할 때 작업들은 여러 process에게 나누어 수행하도록 할 수 있습니다.

이러한 작업 방식을 **parallel execution** 또는 **parallel processing**라고 합니다.

parallel execution은 여러가지 유형의 작업에서 쓸만합니다.

1. large table scan이나 join 또는 partitioned index scan등의 query
2. large table이나 index 생성
3. Bulk insert나 update, delete
4. Aggregations

2. How Parallel Execution works

(1) 처음 query가 oracle server로 들어오면 query를 분석하는 **parse 단계**를 거치게 됩니다. 이때 여러개의 access path 중 가장 성능이 좋다고 판단되는 access path가 결정되게 됩니다. parallel execution이 선택되게 되면,

(2) **수행 단계**에서 query를 수행한 user shadow process는 query coordinator(QC)가 되고 parallel slave 들이 요청한 session에 할당됩니다.

(3) query coordinator는 할당된 slave process에 ROWID나 partition 단위로 데이터를 나눠줍니다.

(4) "producer slave"는 데이터를 읽어 "table queue"로 데이터를 보내는데, "consumer slave"나 query coordinator가 데이터 처리를 위해 이 table queue의 데이터를 대기하게 됩니다.

(5) 만약 sort가 필요한 parallel execution이라면 이들 table queue의 데이터는 "consumer slave process"에 의해 읽혀져 sort 되며,sort 된 데이터는 새로운 "table queue"에 보내지게 됩니다. 이들 sort 된 데이터는 다시 Query coordinator에 의해 읽혀집니다.

만약 sort가 필요 없는 parallel execution이라면 producer slave가 보낸 table queue데이터를 query coordinator가 직접 읽어 처리합니다.

4. Parallel Execution Performance

Parallel query의 수행은 performance 상의 이점을 얻을 수 있으나 parallel queyr를 수행하기 앞서 몇가지 고려할 만한 사항이 있다.

multi-slave process는 당연한 얘기지만 single process 보다 많은 CPU resource와 slave process 각각의 private memory를 사용하게 된다. 만약 CPU 자원이 부족하게 되면 oracle은 parallel operation을 serial operation으로 변경해 작업을 수행한다. 따라서 parallel execution은 현재 system의 resource 상태를 고려해 parallel degree를 고려해야 한다.

또 I/O stress가 많은 시스템이라면 slave process에 의한 추가적인 I/O요구가 부담이 될 수 있습니다.
특히 I/O가 특정 disk에 집중된다면 disk I/O의 bottleneck이 발생할 수 있으므로 I/O의 분산 등도 고려되어야 한다.

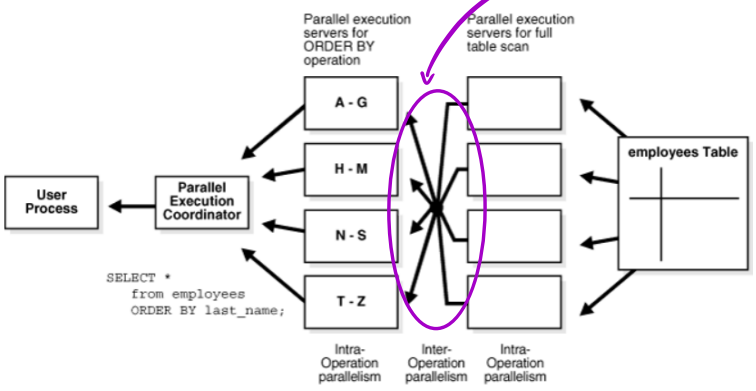
Parallel query는 Full Table Scan으로 데이터를 처리한다. 따라서 index의 사용이 유리한 경우에는 오히려 parallel execution의 성능이 더 나쁠 수 있다.

이러한 성능의 차이는 index에 의해 query 조건의 선처리로 대상 데이터량을 줄이는데서 발생한다.

- Nested Loops vs. Hash/Sort Merge Joins
  Nested loop join의 경우 query 조건에 의한 "row elimination"으로 driviing table의 대상 row를 줄이기 때문에 FTS 보다는 index scan에 적합하다. 반면 Hash join과 sort merge의 경우 일반적으로 대량의 데이터를 처리하는데 더 효과적이다. 이는 Hash join과 Sort Merge join은 driving row source에 대해서 조건에 의한 데이터의 "row eliminate"를 하지 않기 때문이다.
- slave process를 생성하고, data를 분할하고, 여러 slave process로 데이터를 전송하고 결과를 취합하는 등의 비용이 data를 serial하게 처리하는 것보다 많을 수 있다.
- Data skew
  parallel query는 데이터를 ROWID range를 기본으로 slave process 간에 할당한다. 각각의 slave에게 같은 개수의 block을 할당한다는 것은 같은 수의 row를 할당한다는 말과는 다른 의미이다. 예를 들어 대량의 데이터가 수집되고 삭제되는 업무의 경우 특정 블록들에는 데이터가 전혀 들어 있지 않을 수 있다. 이러한 균등하지 않은 데이터 분할로 인해 특정 slave query의 성능이 늦어질 수 있으며 이는 전체 PQ 처리 시간에 영향을 미치게 된다.

Figure 8-2 illustrates the parallel execution of the example query.

Figure 8-2 Inter-operation Parallelism and Dynamic Partitioning



Description of "Figure 8-2 Inter-operation Parallelism and Dynamic Partitioning"

As illustrated in Figure 8-2, there are actually eight parallel execution servers involved in the query even though the DOP is `4`. This is because a producer and consumer operator can be performed at the same time (inter-operation parallelism).

Also all of the parallel execution servers involved in the scan operation send rows to the appropriate parallel execution server performing the `SORT` operation. If a row scanned by a parallel execution server contains a value for the `last_name` column between `A` and `G`, that row is sent to the first `ORDER BY` parallel execution server. When the scan operation is complete, the sorting processes can return the sorted results to the query coordinator, which, in turn, returns the complete query results to the user.

How Parallel Execution Servers Communicate

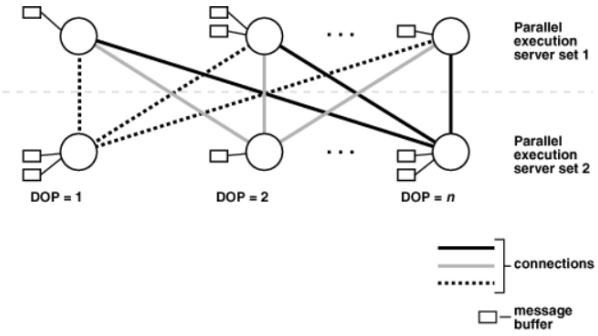
To execute a query in parallel, Oracle Database generally creates a set of producer parallel execution servers and a set of consumer parallel execution servers. The producer server retrieves rows from tables and the consumer server performs operations such as join, sort, DML, and DDL on these rows. Each server in the producer set has a connection to each server in the consumer set. The number of virtual connections between parallel execution servers increases as the square of the degree of parallelism.

Each communication channel has at least one, and sometimes up to four "memory buffers," which are allocated from the shared pool. Multiple memory buffers facilitate asynchronous communication among the parallel execution servers.

(=message buffer)

A single-instance environment uses at most three buffers for each communication channel. An Oracle Real Application Clusters environment uses at most four buffers for each channel. Figure 8-3 illustrates message buffers and how producer parallel execution servers connect to consumer parallel execution servers.

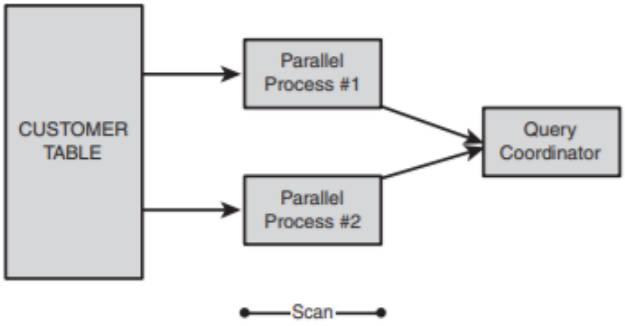
Figure 8-3 Parallel Execution Server Connections and Buffers



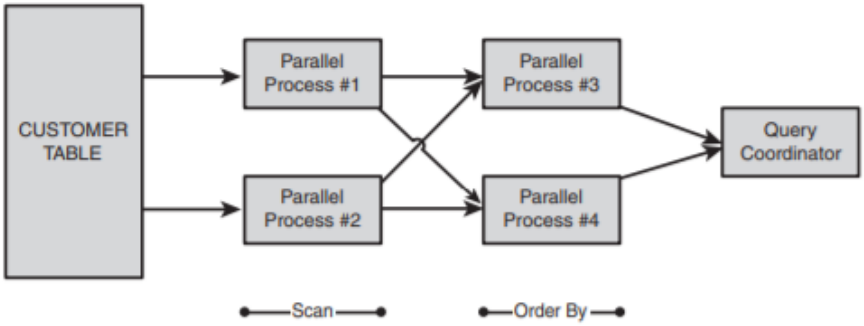
Description of "Figure 8-3 Parallel Execution Server Connections and Buffers"

When a connection is between two processes on the same instance, the servers communicate by passing the buffers back and forth in memory (in the shared pool). When the connection is between processes in different instances, the messages are sent using external high-speed network protocols over the interconnect. In Figure 8-3, the DOP equals the number of parallel execution servers, which in this case is n. Figure 8-3 does not show the parallel execution coordinator. Each parallel execution server actually has an additional connection to the parallel execution coordinator. It is important to size the shared pool adequately when using parallel execution. If there is not enough free space in the shared pool to allocate the necessary memory buffers for a parallel server, it fails to start.

```
SELECT /*+ parallel(c,2) */ *
  FROM customers c
```




```
SELECT /*+ parallel(c,2) */ *
  FROM customers c
 ORDER BY cust_last_name,cust_first_name
```




```
SELECT /*+ parallel(c,2) */ cust_last_name,count(*)
  FROM customers c
 GROUP BY cust_last_name
 ORDER BY 2 desc
```

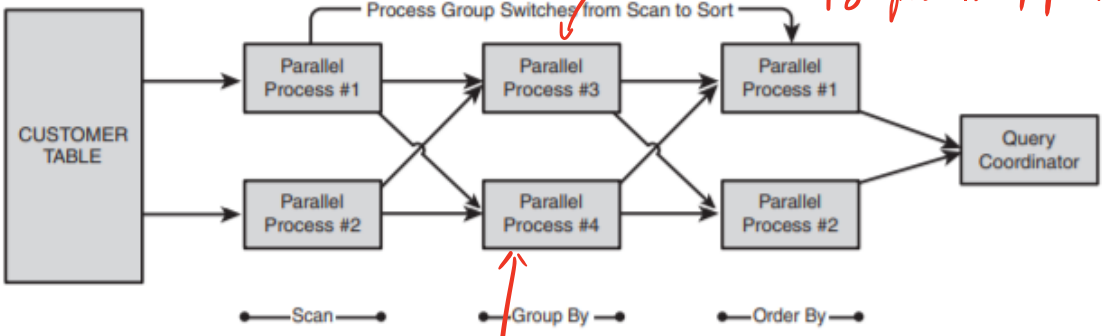


FIGURE 13-3 Parallel process allocation for a DOP of 2.

'Group 1'과 'Group 2'는 해당 process에서 담당

'Group 3'과 'Group 4'는 해당 process에서 담당

Figure 25-2 Data Flow Diagram for Joining Tables