TIBERO

TECHNICAL WHITE PAPER

Tibero Optimizer SQL Execution Plan

October 2012

목차

Summary

Optimizer 에 의해 만들어진 SQL 플랜을 확인한는 여러 방법들을 소개하고 플랜에서 보여주는 정보의 의미에 대해 알아본다. 그리고, 운영 중인 시스템에서 비효율적으로 수행 중인 SQL 과 플랜을 찾아내는 방법을 이해함으로써 튜닝 대상 SQL 을 효율적으로 찾을 수 있는 방법을 소개한다.

1. Introduction

Tibero의 optimizer는 SQL 을 수행하기 위한 플랜을 만들 때, cost 기반으로 가장 효율적인 플랜을 만들게 된다. cost란 SQL을 수행하면서 발생하는 디스크 IO와 CPU 연산을 모두 고려해서 만든 수치로서 절대 비교가 가능한 값이다. optimizer는 각 플랜 단계 별로 발생할 디스크 IO, CPU 연산을 테이블의 통계 정보를 기반으로 예측하면서 cost 값을 계산하여 가장 효율적인 플랜을 만들게 된다. 하지만, 이 cost 값 계산은 다음과 같은 상황에 의해 잘못 예측되는 경우 전체적으로 비효율적인 플랜을 만들 수도 있다.

- cost 모델의 잘못된 가정
- 오래된 통계 정보
- 통계 정보가 없을 시 데이타 sampling 오류
- 조건문 간의 의존성
- 예측 불가능한 함수가 포함된 조건문
- 조건문에 사용된 사용자 파라미터
- join 순서 결정 시 너무 많은 조합에 따른 pruning 에 의한 local optimization

Tibero 에서는 optimizer 에 의해 만들어진 플랜의 단계 별 cost 값들을 쉽게 조회하고, 분석할 수 있는 기능을 다양한 방법으로 제공하고 있다. 이 문서에서는 플랜을 확인하는 여러 방법과 플랜에 대한 기본적인 정보와 값들의 의미에 대해서 설명하도록 하고, 마지막으로는 현재 운영 중인 시스템에서 비효율적으로 수행되는 쿼리를 찾을 수 있는 여러 방법을 소개하도록 하겠다.

2. Watching SQL Plan

2.1. SQL Plan 이란?

optimizer에서 만들어낸 플랜은 여러 개의 수행 operation의 트리 형태로 이루어져 있고, SQL 수행을 하게 되면 이 플랜을 통해 제일 하단에서 만들어진 row들이 위 operation으로 전달되면서 최상단 operation의 결과 row가 쿼리의 최종 결과가 된다.

각 단계 별 operation을 플랜 노드라고 부르고 노드의 종류는 현재 Tibero 5.0에서 65개가 있는데, 노드를 크게 분류해보면 다음과 같다. (다음 장에 operation 종류에 대한 설명이 있다)

- row 생성 노드 (table, index scan, ...)
- join 노드 (nested join, sort merge join, hash join, index join, ...)
- dml 노드 (insert, update, delete, merge, ...)
- 그 밖의 노드 (orderby, groupby, connectby, ...)

플랜 노드에는 자기의 해당 operation을 수행하기 위한 필요한 정보를 담고 있어서, SQL 수행 시에 이 정보들을 참고하게 된다. 이런 수행에 필요한 자세한 정보는 사용자에게 제공하지는 않고 플랜 노드 별 operation 종류, 추가 option, optimizer에서 계산한 노드의 cost 값과 cardinality 정도의 정보만 조회가 가능하다.

optimizer 단계에서 cost 계산 방식은 SQL 수행과 마찬가지로 최하단부터 cost 계산을 하면서 최상단 플랜 노드까지 cost 를 누적하는 방식으로 이루어지는데, 중간중간 그리고 최종적으로 가장 cost 가 작은 플랜을 선택하게 된다. 이렇게 최적으로 판단되어 만들어진 SQL 플랜은 library cache 에 등록되어 같은 SQL 에 대한 수행 요청 시에 재활용되어 optimizer 단계를 건너뛰고 SQL 수행을 하게 된다. 앞의 optimizer 단계를 거치는 parsing 을 hard parsing 이라 하고, optimizer 단계를 건너뛰고 library cache 에서 해당 플랜을 가져오는 parsing 을 soft parsing 이라고 부른다.

2.2. SQL Plan 확인하기

optimizer에 의해 만들어진 플랜을 확인하는 방법은 다음과 같다.

- V$SQL_PLAN 뷰를 통해 보기
- TBSQL 프로그램의 AUTOTRACE 기능을 통해 보기
- SQLTRACE 기능을 이용해서 TBPROF 프로그램으로 수행 결과와 같이 보기
- EXPLAIN PLAN 문을 통해 PLAN_TABLE 에 저장하고 보기

가) V$SQL_PLAN 뷰를 통해 보기

플랜을 확인하는 가장 기본적인 방법으로서, V$SQL_PLAN 뷰는 library cache를 직접 access하여 등록된 플랜들의 정보를 가상의 table 형태로 만들어주는 DPV(dynamic performance view)이다. 하나의 SQL 플랜에 대해 각 플랜 노드 정보가 각 row 로 표현되고, row의 각 column들은 해당 플랜 노드를 분석하는데 필요한 모든 정보를 포함하고 있다. 이 뷰의 각 컬럼의 의미는 다음과 같다.

컬럼 이름	컬럼 타입	설명
HASH_VALUE	NUMBER	library cache에서의 hash 값
SQL_ID	NUMBER	플랜의 unique ID
OPERATION	VARCHAR	플랜 노드 이름
OBJECT#	NUMBER	플랜 노드에서 access 하는 object 번호
OBJECT_OWNER	VARCHAR	플랜 노드에서 access 하는 object 를 소유하는 유저 이름
OBJECT_NAME	VARCHAR	플랜 노드에서 access 하는 object 이름
OBJECT_TYPE	VARCHAR	플랜 노드에서 access 하는 object 타입
ID	NUMBER	플랜 노드 ID
PARENT_ID	NUMBER	플랜 노드의 부모 ID
DEPTH	NUMBER	플랜 노드의 트리 깊이
POSITION	NUMBER	같은 부모의 자식들 간의 순서
SEARCH_COLUMNS	NUMBER	index 검색에서 사용되는 컬럼 개수
COST	NUMBER	optimizer에서 예측한 플랜 노드의 총 cost
CPU_COST	NUMBER	optimizer에서 예측한 플랜 노드의 CPU cost
IO_COST	NUMBER	optimizer에서 예측한 플랜 노드의 IO cost
CARDINALITY	NUMBER	optimizer에서 예측한 플랜 노드의 최종 row 수
PSTART	NUMBER	partition table/index인 경우 시작 partition 번호
PEND	NUMBER	partition table/index인 경우 끝 partition 번호
OTHERS	VARCHAR	dblink를 사용하는 플랜 노드의 경우 remote sql
ACCESS_PREDICATES	VARCHAR	join, index scan에서 사용하는 predicate 정보
FILTER_PREDICATES	VARCHAR	filter로 사용하는 predicate 정보

이 뷰는 library cache에 등록된 모든 플랜을 보여주기 때문에 원하는 SQL_ID 값을 조건절에 입력하여 보고자 하는 특정 플랜만 조회해야 한다. 다음 예제 쿼리를 수행하면서 플랜을 V$SQL_PLAN 뷰로 조회하는 방법에 대해서 설명하도록 하겠다.

```
SQL> select d.deptno, sum(sal)
                from dept d, emp e
                where d.deptno = e.deptno and e.mgr is not null
                group by d.deptno order by d.deptno;

DEPTNO    SUM(SAL)
---------- ----------
         10       3750
         20      10875
         30       9400
3 rows selected.
```

SQL text를 가지고 해당 플랜의 SQL_ID를 찾기 위해서는
다음과 같은 쿼리를 수행한다.

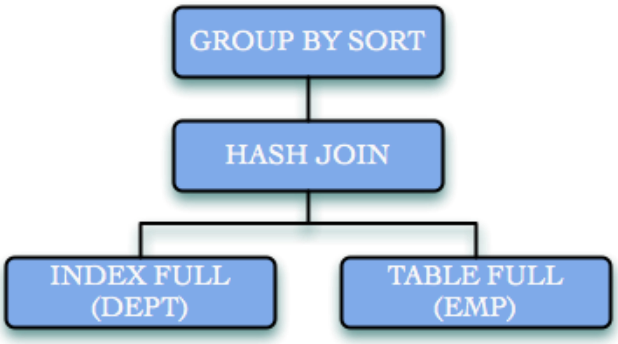
```
SQL> select sql_id from V$SQLTEXT where upper(sql_text) like '%DEPT D, EMP E%';
```

해당 플랜의 SQL_ID를 구했으면 V$SQL_PLAN에 다음과
같은 쿼리를 수행하여 조회하면 해당 SQL의 플랜 정보를
한 눈에 볼 수가 있다.

```
SELECT SUBSTRB(TO_CHAR(ID), 1, 3) || LPAD(' ', LEVEL * 2) || UPPER(OPERATION) ||
        DECODE(OBJECT_NAME, NULL, NULL, ': '||OBJECT_NAME) || ' (Cost:' || COST ||
', %%CPU:' || DECODE(COST, 0, 0, TRUNC((COST - IO_COST) / COST * 100)) ||
', Rows:' || CARDINALITY || ') ' ||
        DECODE(PSTART, '', '', '(PS:' || PSTART || ', PE:' || PEND || ')') AS "Execution Plan"
FROM (SELECT * FROM V$SQL_PLAN WHERE SQL_ID = 281)
START WITH DEPTH = 1
CONNECT BY PRIOR ID = PARENT_ID AND PRIOR SQL_ID = SQL_ID
ORDER SIBLINGS BY POSITION;

Execution Plan
----------------------------------------------------------------------------
1   GROUP BY (SORT) (Cost:23, %%CPU:0, Rows:4)
2     HASH JOIN   (Cost:24, %%CPU:0, Rows:13)
3       INDEX (FULL): PK_DEPT (Cost:1, %%CPU:0, Rows:4)
4       TABLE ACCESS (FULL): EMP (Cost:23, %%CPU:0, Rows:13)
```

해당 SQL 플랜은 다음과 같은 형태의 트리 구조를 가지고 있는 SQL이다.



나) TBSQL 프로그램의 AUTOTRACE 기능을 이용하기

V$SQL_PLAN는 많은 정보를 가지고 있지만, 위와 같이 실제로 사용하기에는 여러 단계에 필요한 정보가 있어서 많은 번거로움이 따른다. Tibero는 TBSQL이라는 인터액티브 쿼리 수행 프로그램을 제공하고 있는데, 이 프로그램에서 제공하는 많은 기능들을 이용하면 쉽게 SQL을 수행하고 결과를 확인할 수 있다. TBSQL의 AUTOTRACE 기능은 방금 수행한 SQL에 대한 플랜 정보와 수행 통계 정보를 바로 바로 보여주는 기능으로 다양한 option을 제공하고 있고 매우 편리하다.

동작 방식은 위에서 수동으로 V$SQL_PLAN 뷰로 플랜을 확인한 것과 같이 TBSQL 프로그램 내부에서 방금 수행한 SQL의 SQL_ID를 구해서 V$SQL_PLAN 뷰에 조회해서 보여주는 방식이다.

AUTOTRACE 옵션을 사용하려면 다음과 같이 TBSQL 프로그램 내에서 SET 명령어를 이용하면 된다.

```
SET AUTOT[RACE] {OFF|ON|TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]] [PLANS[TAT]]
```

보여주는 TRACE 정보는 EXPLAIN (플랜 노드 정보), STATISTICS (수행 후 전체 통계 정보), PLANSTAT (플랜 노드 별 수행 정보) 이다. 옵션이 많아서 사용하기 복잡해 보이지만, option의 조합은 다음과 같다.

SET AUTOTRACE OFF
: 수행 결과만 보여주기 (default)
SET AUTOTRACE ON
: 수행 결과와 trace 정보 (EXPLAIN, STATISTICS) 보여주기
SET AUTOTRACE ON EXPLAIN
: 수행 결과와 trace 정보 (EXPLAIN) 보여주기
SET AUTOTRACE ON STATISTICS
: 수행 결과와 trace 정보 (STATISTICS) 보여주기
SET AUTOTRACE ON PLANSTAT
: 수행 결과와 trace 정보 (PLANSTAT) 보여주기
SET AUTOTRACE ON EXPLAIN PLANSTAT
: 수행 결과와 trace 정보 (EXPLAIN, PLANSTAT) 보여주기 (세 가지 trace 정보 조합 가능)
SET AUTOTRACE TRACEONLY
: 수행은 하지만 결과 출력 없이 trace 정보 (EXPLAIN, STATISTICS)만 보여주기
SET AUTOTRACE TRACEONLY EXPLAIN
: 수행은 하지 않고 결과 출력 없이 trace 정보 (EXPLAIN)만 보여주기
SET AUTOTRACE TRACEONLY STATISTICS
: 수행은 하지만 결과 출력 없이 trace 정보 (STATISTICS)만 보여주기
SET AUTOTRACE TRACEONLY PLANSTAT
: 수행은 하지만 결과 출력 없이 trace 정보 (PLANSTAT)만 보여주기

TBSQL에서 AUTOTRACE 기능으로 출력한 화면 예는 다음과 같다. TRACE에서 보여주는 정보에 대해서는 다음 장에서 설명하도록 하겠다.

```
SQL> SET AUTOTRACE ON EXPLAIN STATISTICS PLANSTAT
SQL> select d.deptno, sum(sal)
              from dept d, emp e
              where d.deptno = e.deptno and e.mgr is not null
              group by d.deptno order by d.deptno;

DEPTNO    SUM(SAL)
---------- ----------
        10       3750
        20      10875
        30       9400
3 rows selected.

SQL ID: 528
Plan Hash Value: 3266050601


Execution Plan
--------------------------------------------------------------------------------
1   GROUP BY (SORT) (Cost:23, %CPU:0, Rows:4)
2     HASH JOIN   (Cost:24, %CPU:0, Rows:13)
3       INDEX (FULL): PK_DEPT (Cost:1, %CPU:0, Rows:4)
4       TABLE ACCESS (FULL): EMP (Cost:23, %CPU:0, Rows:13)


Predicate Information
--------------------------------------------------------------------------------
2 - access: ("D"."DEPTNO" = "E"."DEPTNO")
4 - filter: ("E"."MGR" IS NOT NULL)


     VALUE EVENT_NAME
---------- ------------------------------
         2 db block gets
        20 consistent gets
         0 physical reads
         0 redo size
         0 sorts (disk)
         1 sorts (memory)
         3 rows processed

Execution Stat
--------------------------------------------------------------------------------
1   GROUP BY (SORT) (Time:.03 ms, Rows:3, Starts:1)
2     HASH JOIN   (Time:.02 ms, Rows:13, Starts:1)
3       INDEX (FULL): PK_DEPT (Time:.01 ms, Rows:4, Starts:1)
4       TABLE ACCESS (FULL): EMP (Time:.04 ms, Rows:13, Starts:1)
```

위와 같이 하나의 SQL을 수행한 후 플랜 정보, SQL 수행 정보, 플랜 노드 별 수행 정보 등을 한 눈에 파악할 수 있기 때문에 SQL 분석에 많은 도움이 된다. TBSQL의 다른 기능들인 스풀링, 사용자 변수, 포맷팅 등을 함께 사용하면 더 편하게 분석할 수 있으므로 이 기능들을 TBSQL 매뉴얼에서 참고하기 바란다.

다) SQLTRACE 기능으로 수행 결과와 함께 보기

특정 SQL에 대한 플랜을 분석하는 것이 아니라, 실제 운영 중인 서버에서 수행되고 있는 불특정 다수의 SQL 플랜들을 분석하고 싶을 때는 SQLTRACE 기능을 이용하는 것이 가장 좋은 방법이다. 특정 시간 동안 SQLTRACE를 켜고 끄기 위해서는 다음과 같은 ALTER 문을 수행한다.

```
SQL> alter system set sql_trace=Y;
…
SQL> alter system set sql_trace=N;
```

사용할 때 주의할 점은 ALTER SYSTEM의 경우, 기존에 접속을 맺고 있는 세션에 대해서는 적용이 안 되고, ALTER SYSTEM이 수행된 이후에 새로 접속을 맺는 세션부터 해당 파라미터 변경이 적용 받는다.

SQLTRACE가 적용된 세션에서 수행된 SQL은 매 수행마다 플랜 노드 별 수행 정보를 SQL_TRACE_DEST 파라미터에 정의된 경로에 확장자 TRC 파일로 남기게 된다. 매 수행마다 파일에 수행 정보를 write 하게 되어 성능 저하가 발생하지만, 세션 간 경합 없이 각자 파일에 남기므로 일정 비율(5-10%)로만 성능 저하가 발생한다.

하지만, optimizer에서 예상한 정보와 실제 수행 정보가 동시에 남기 때문에 정확한 문제 분석이 가능하고, 실제 운영에서 수행되는 SQL 플랜 정보이므로 정기적으로 분석해보는 것이 많은 도움이 될 수 있다.

SQLTRACE를 통해 만들어진 TRC 파일은 수행 중에 최소한의 성능 피해를 주기 위한 포맷으로 정보를 남기기 때문에 바로 읽고 분석하기에는 많은 불편이 있으므로 이 정보를 같은 SQL 수행 별로 통합하거나 사용자가 보기 더 편한 포맷으로 변환해주는 툴인 TBPROF를 이용해야 한다. TBPROF 프로그램에도 많은 옵션이 있는데 사용법은 다음과 같다.

```
tbprof   tracefile outputfile [print= ] [sys= ] [sort= ]
[aggregate= ]
print=integer  (첫 n개의 SQL 수행 정보만 출력)
sys=yes|no ('SYS' 사용자로 수행된 SQL 수행 정보를 제거)
aggregate=yes|no (같은 SQL 수행 정보를 합쳐서 보여주기)
sort=options (주어진 옵션으로 정렬해서 출력. 옵션 종류는
매뉴얼 참고)
```

SQL 수행 후 TBPROF 프로그램을 이용해서 변환한 파일 내용은 다음과 같다.

```
select d.deptno, sum(sal)
                from dept d, emp e
                where d.deptno = e.deptno and e.mgr is not null
                group by d.deptno order by d.deptno;


 stage      count        cpu    elapsed    current      query       disk       rows
------------------------------------------------------------------------------
---
 parse          1       0.00       0.00          0          0          0          0
  exec          1       0.00       0.00          0          0          0          0
 fetch          1       0.00       0.00          2         14          0          3
------------------------------------------------------------------------------
---
   sum          3       0.00       0.00          2         14          0          3

    rows          execution plan
  ----------------------------------------------------------
       3          group by (sort) (et=36, cr=0, cu=0, co=23, cpu=0, ro=4)
      13            hash join   (et=26, cr=0, cu=0, co=24, cpu=0, ro=13)
       4              index (full) PK_DEPT(3456) (et=16, cr=1, cu=0, co=1, cpu=0, ro=4)
      13              table access (full) EMP(3457) (et=50, cr=0, cu=2, co=23, cpu=0, ro=13)
```

V$SQL_PLAN 뷰나 TBSQL을 통해서 보는 정보와 매우 유사하지만, 사용자 입력 뿐이 아니라 운영 중인 Tibero에서 수행 중인 모든 SQL들을 분석할 수 있는 점에서 다르다. 일반적으로 SQLTRACE 기능을 이용해서 문제의 쿼리를 찾아내고, 해당 쿼리를 TBSQL을 이용해서 수행해보면서 분석/튜닝하는 식으로 SQL 수정 작업을 진행하게 된다.

라) EXPLAIN PLAN 구문을 이용해서 플랜 정보 저장하기

이전까지 소개했던 방법들은 V$SQL_PLAN 뷰와 실제 수행할 때 가지고 있는 SQL 플랜 정보를 통해서 조회하는 방법이었지만, EXPLAIN PLAN 구문을 이용하면 분석하고자 하는 대상 SQL 플랜을 PLAN_TABLE이라는 임시 테이블에 저장하고 이 테이블에 조회함으로써 플랜 정보를 얻을 수 있다.

library cache가 큰 시스템이거나 V$SQL_PLAN 뷰 조회조차도 운영 서버에 부하를 줄 수 있는 상황에서 서버에 대한 부하를 최소화하면서 특정 SQL 플랜을 수행없이 확인하는 용도로 사용하고 싶거나, 해당 SQL의 플랜 정보를 특정 persistent 테이블에 저장하고 지속적인 분석을 하고 싶을 때 사용하면 좋다.

사용법은 다음과 같다.

```
EXPLAIN PLAN [SET STATEMENT_ID = literal] [INTO
table_name] FOR SQL STATEMENT;
 : statement_id는 저장된 테이블에서 해당 SQL 플랜을 찾는 ID로 사용된다.
 : INTO 절을 이용하면 기본 PLAN_TABLE이 아닌 다른 테이블을 지정해서 저장할 수가 있다.
```

위에서 언급한 대로 V$SQL_PLAN 에 대한 조회없이 특정 테이블에 해당 플랜을 저장하고 조회할 수 있다는 점을 제외하고는 사용법은 뷰를 이용할 때와 다르지 않다.

2.3. Understanding SQL Plan

이제까지 SQL 플랜을 볼 수 있는 방법에 대해 알아보았는데, 플랜에서 보여주는 정보들의 의미가 무엇인지 알아보도록 하겠다.

가) Execution Plan (TBSQL)

TBSQL에서 AUTOTRACE 기능을 이용하면, 다음과 같은 형식으로 조회 결과를 얻게 된다. 볼 수 있는 정보는 크게 세 가지로 분류가 된다.

플랜 노드 별 정보를 보여주는 Execution Plan 부분

플랜 노드 별로 적용된 predicate 정보를 보여주는 Predicate Information 부분

외부 database 에 대한 dblink 를 이용하는 노드에서 수행할 SQL 정보를 보여주는 Remote SQL Information 부분

```
select d.deptno, sum(sal)
                from dept d, emp@ln01 e
                where d.deptno = e.deptno and e.mgr is not null
                group by d.deptno order by d.deptno;

Execution Plan
--------------------------------------------------------------------------------
1   GROUP BY (SORT) (Cost:23, %CPU:0, Rows:4)
2     HASH JOIN   (Cost:24, %CPU:0, Rows:13)
3       INDEX (FULL): PK_DEPT (Cost:1, %CPU:0, Rows:4)
4       PROXY: LN01 (Cost:22, %CPU:0, Rows:13)

Remote SQL Information
--------------------------------------------------------------------------------
4 - SELECT QB_007."SAL",QB_007."DEPTNO" FROM "EMP" QB_007 WHERE (QB_007."MGR" IS
 NOT NULL)

Predicate Information
--------------------------------------------------------------------------------
2 - access: ("D"."DEPTNO" = "E"."DEPTNO")
4 - filter: ("E"."MGR" IS NOT NULL)
```

Execution Plan에서 볼 수 있는 내용은 다음과 같다.
operation name (option) [: object name] (total_cost, cpu_cost_percentage, cardinality)

항목	내용
operation name	해당 노드에서 수행할 operation 이름
option	operation 의 부가적인 정보를 보여준다. join 타입(semi, anti, left outer 등), sort 종류 (top-N), rowid scan 종류(local/global rowid), partition iterating 종류(empty, single, subset, all)
object name	object 에 대한 operation 의 경우 해당 object 이름을 보여준다. (table name, index name, dblink name 등)
total cost	V$SQL_PLAN 의 cost 컬럼 값으로 하위 플랜 노드 밑으로 총 누적된 cost 값을 보여준다.
cpu cost percentage	total cost 에서 cpu cost 가 차지하는 비율
cardinality	optimizer 에서 예측한 플랜 노드 수행 결과 row 수

Predicate Information 부분은 join predicate이나 index access predicate에 대해선 access란 키워드로 표시해주고, 그 외 일반 filter predicate에 대해서는 filter란 키워드로 predicate 정보를 보여준다.

Remote SQL Information 부분은 dblink를 이용해서 타 database에 SQL을 수행해서 결과를 받아와서 처리하는 플랜 노드에서 수행하는 SQL을 보여준다. Tibero 5.0에서는 dblink 에 대해서도 cost 계산을 통해 최적의 플랜을 만들어내기 때문에 dblink가 포함된 SQL을 분석하기 위해서 꼭 필요한 정보이다.

TBSQL에서 AUTOTRACE 중 PLANSTAT을 켰을 때 나오는 SQL 수행 정보는 다음과 같다.

```
Execution Stat
--------------------------------
1  GROUP BY (SORT) (Time:.03 ms, Rows:3, Starts:1)
2    HASH JOIN  (Time:.02 ms, Rows:13, Starts:1)
3      INDEX (FULL): PK_DEPT (Time:.01 ms, Rows:4, Starts:1)
4      TABLE ACCESS (FULL): EMP (Time:.04 ms,
       Rows:13, Starts:1)
```

operation name (option) [: object name] (elapsed time, output rows, recursive execution)

항목	내용
elapsed time	해당 플랜 노드 수행 시간
output rows	해당 플랜 노드를 수행하고 나온 결과 row 수
recursive execution	해당 플랜 노드가 전체 SQL 수행하면서 반복 수행된 횟수 (nested loop join, index join 의 오른쪽 플랜 노드는 여러 번 수행 가능하다)

TBSQL에서 execution 플랜 정보와 플랜 수행 정보를 동시에 보게 되면, optimizer의 예측과 실제 수행 결과를 한눈에 보게 되므로 문제 분석에 많은 도움을 준다.

나) Execution Plan (TBPROF)

TBPROF에서 보여주는 정보는 TBSQL에서 보여주는 정보와 비슷하지만, 다음과 같이 수행 결과와 optimizer에서 예측한 정보를 약간 다른 형태로 보여주고 있다.

```
rows          execution plan
  ------------------------------------------------------------
       3          group by (sort) (et=36, cr=0, cu=0, co=23, cpu=0, ro=4)
      13            hash join   (et=26, cr=0, cu=0, co=24, cpu=0, ro=13)
       4              index (full) PK_DEPT(3456) (et=16, cr=1, cu=0, co=1, cpu=0, ro=4)
      13              table access (full) EMP(3457) (et=50, cr=0, cu=2, co=23, cpu=0, ro=13)
```

rows 컬럼은 각 플랜 노드에서의 최종 결과 row 수이고, execution plan 컬럼 부분에서 보여주는 정보는 다음과 같다.

operation name (option) [: object name] (elapsed time, cr read, cu read, cost, cpu_cost, cardinality)

항목	내용
elapsed time	해당 플랜 노드 수행 시간
cr read	해당 플랜 노드를 수행하면서 읽은 CR 블럭 개수
cu read	해당 플랜 노드를 수행하면서 읽은 current 블럭 개수
cost	V$SQL_PLAN 의 cost 컬럼 값으로 플랜 노드 밑으로 총 누적된 cost 값
cpu cost	total cost 에서 cpu cost 가 차지하는 비율
cardinality	optimizer 에서 예측한 플랜 노드 수행 결과 row 수

다) Operation 종류

SQL 플랜 노드는 각 고유의 operation을 처리하기 위한 정보를 담고 있다. 이런 플랜 노드들의 조합을 통해 사용자가 수행한 SQL을 의미에 맞게 절차적으로 수행할 수가 있다. Tibero에는 SQL 99 스펙과 그 밖의 특수 기능들을 지원하기 위해 65가지 정도의 operation 노드가 존재한다. 새로운 기능이 계속 추가되면 해당 operation 수도 계속 늘어나게 된다. Tibero에 존재하는 operation들을 파악하고 있으면 수행한 SQL과 SQL 플랜을 매치시켜 해당 SQL의 구조를 파악하기가 쉬워지고, 효율적인 플랜인지 아닌지를 판단할 수가 있다.

현재 Tibero 5.0 버젼에서 제공하는 operation은 다음과 같다.

항목	내용
table scan	table 에 접근(full scan/rowid scan)
index scan	index 에 접근(full/range/unique/skip/fast full)
join	두 개의 table 에 대한 join 을 처리(hash/sort merge/nested loop/index join)
set	집합 함수 처리(union/intersect/minus/bag union)
project	컬럼을 줄이거나 늘리거나 expression 계산 처리
filter	조건문을 계산하여 결과 row 를 줄이는 작업 처리
orderby	결과 정렬
groupby	같은 그룹에 속하는 row 를 모아 그룹 함수 처리 (sort/hash)
dml	dml 처리 (insert/update/delete/merge/multi table insert)
count rownum	특정 개수의 row 만 통과시키기
connect by	계층 관계 데이타 처리
call	프로시져/함수 호출
file scan	외부 파일 접근 처리
window	analytic 함수 처리
buff	SQL 수행 중 저장된 중간 결과에 대한 접근 처리
buff transformation	SQL 수행 중 중간 결과 저장 및 접근에 대한 제어
cube	cube 기능을 위한 처리
proxy/proxy dml	외부 database 에 대한 dblink 처리
sort aggregation	aggregation 함수 처리
pe manager	parallel execution manager
pe send	parallel execution 중간 결과를 다른 slave 에 보내기 (hash/range/broad/rr/qc rand/qc order/index maintenance/partition)
pe recv	parallel execution 중간 결과를 다른 slave 에서 받기
pe block iterator	parallel 하게 table 접근을 위한 반복 수행 처리
pe index maintenance	parallel dml 에서 index 무결성 보장을 위한 처리
cache	subquery 결과 cache
inlist iterator	in predicate 반복 수행 처리
partition iterator	partition 반복 수행 처리
xml table	xmltable() 구문 처리
collector	pipeline table 처리
result cache	쿼리 결과 저장 및 cache 처리
for update	select for update 처리

Finding Bad SQL Plan

수행 시간이 긴 SQL은 원래 오래 걸리는 SQL일 수도 있지만, 서두에서 언급한 것처럼 optimizer가 플랜을 잘못 푸는 경우도 많이 존재하므로, 예상한 것과 달리 플랜이 잘못 풀리게 되어 수행이 오래 걸리는 SQL일 수 있다. 이런 SQL들을 찾아내서 플랜을 확인하고 적절한 조치를 취하는 것은 매우 중요한 일인데, Tibero에서는 이런 SQL들을 찾아내기 위한 여러 가지 방법들을 편리하게 제공하고 있다.

라) APM

Tibero APM(Automatic Performance Monitoring) 기능을 사용하면, 측정 기간 동안 Tibero에서 수행한 SQL에 대한 요약 정보를 얻을 수가 있다. APM에서 SQL이외에도 볼 수 있는 항목이 많지만, SQL Reports 부분에서 우리가 원하는 다양한 기준의 top 5 SQL에 대한 수행 정보와 SQL ID를 알아낼 수가 있다.

일단 SQL ID를 구하면 해당 SQL에 대해 SQL문(V$SQLTEXT), SQL 플랜 정보(V$SQL_PLAN), SQL 플랜 수행 정보(V$SQL_PLAN_STATISTICS)를 조회할 수 있으므로 문제가 될 만한 플랜들을 점검할 수가 있다.

SQL Reports에서 보여주는 정보는 다음과 같다.

마) Top SQL by elapsed time

SQL 수행 시간이 가장 긴 top 5 SQL

컬럼	내용
Elapsed time	SQL 총 수행 시간
Exec	SQL 수행 횟수
Elap/Exec	SQL 실행 당 수행 시간
DB Time (%)	해당 APM 측정 기간 동안 총 DB time 에 대한 수행 시간 비율
SQL ID	SQL ID

사) Top SQL by executions

SQL 수행 횟수가 가장 많은 top 5 SQL

컬럼	내용
Exec	SQL 수행 횟수
Rows processed	총 SQL 수행 결과 row 수
Rows/Exec	SQL 실행 당 평균 row 수
Elap/Exec	SQL 실행 당 수행 시간
SQL ID	SQL ID

아) Top SQL by CR blocks read count

SQL 수행 중 CR block 읽는 횟수가 가장 많은 top 5 SQL

컬럼	내용
Buffer Gets	CR block 요청 횟수
Exec	SQL 수행 횟수
Gets/Exec	SQL 실행 당 CR block 요청 횟수
Total Gets(%)	총 CR block 요청 횟수에 대한 비율
Elapsed Time	SQL 총 수행 시간
SQL ID	SQL ID

자) Top SQL by I/O

SQL 수행 중 I/O 시간이 가장 많은 top 5 SQL

컬럼	내용
Disk Read Time	disk read 시간
Exec	SQL 수행 횟수
Disk Read Time/Exec	SQL 실행 당 disk read 시간
Elapsed Time	SQL 총 수행 시간
SQL ID	SQL ID

차) Top SQL by temporary segment usage

SQL 수행 중 temporary segment 사용 시간이 가장 많은 top 5 SQL

컬럼	내용
Tmp Sgmt IO Tm	temp segment read/write 시간
Exec	SQL 수행 횟수
Tmp Sgmt IO Tm/Exec	SQL 실행 당 temp segment read/write 시간
Elapsed Time	SQL 총 수행 시간
SQL ID	SQL ID

카) Top SQL by CPU

SQL 수행 중 CPU 시간이 가장 많은 top 5 SQL

컬럼	내용
Avg. CPU Time	총 수행 시간에서 IO 시간을 제외한 시간
Exec	SQL 수행 횟수
Elapsed Time	SQL 총 수행 시간
SQL ID	SQL ID

타) Top SQL by Parse Calls

SQL 수행 중 parse 횟수가 가장 많은 top 5 SQL

컬럼	내용
Parse Calls	soft parse 요청 횟수
Exec	SQL 수행 횟수
Total Parse(%)	SQL 총 parse 횟수에 대한 비율
SQL ID	SQL ID

파) V$SESSION_LONGOPS

특정 시간 이상 수행되는 operation에 대한 정보를 조회할 수 있는 뷰이다. 특정 SQL 수행 중 한 플랜 노드가 LONGOPS_THRESHOLD_SEC 값의 초 이상 수행되면 자동으로 세션에 해당 정보를 남기게 된다. SQL 수행 이외에도 사용자가 임의의 작업에 대해서도 DBMS_APPLICATION_INFO 패키지를 이용해서 longops 정보를 남길 수가 있다. 등록된 정보는 동적으로 계속 업데이트되기 때문에 처음에 등록된 총 작업량에 비해 현재 어느 정도 진행되었는지를 알 수가 있어 앞으로 남은 작업 시간을 예상할 수가 있다.

해당 뷰의 각 컬럼 값은 다음과 같은 의미를 가진다.

컬럼 이름	컬럼 타입	설명
SESS_ID	NUMBER	session ID
SERIAL_NO	NUMBER	session serial number
USER_NAME	VARCHAR	접속 유저 이름
WTHR_ID	NUMBER	Tibero work thread ID
SQL_ID	NUMBER	해당 플랜의 SQL ID
SQL_PLAN_ID	NUMBER	해당 플랜의 SQL 플랜 ID
OPNAME	VARCHAR	opeartion 이름
TARGET	VARCHAR	operation 대상 객체 이름
SOFAR	NUMBER	지금까지 수행된 작업량
TOTALWORK	NUMBER	앞으로 수행할 총 작업량
UNITS	VARCHAR	작업의 단위
START_TIME	DATE	작업 시작 시간

다음과 같은 조회 쿼리를 통해서 현재 오래 수행되고 있는
SQL의 정보를 쉽게 파악할 수가 있다.

```
SQL> select sql_id, sql_plan_id, substr(opname, 1, 14) opname,
        to_char(sofar/totalwork*100, 99.99) || '%' progress,
        (sysdate - start_time)*24*60*60 elapsed_sec from v$session_longops;
   SQL_ID SQL_PLAN_ID OPNAME                        PROGRESS ELAPSED_SEC
------------ ------------------ ---------------------------- ------------
          215                  3 nested loops                 3.37%         1955
1 row selected.
```

위의 예에서는 215번 SQL_ID를 가지는 쿼리의 3번 플랜 노드가 nested loop join인데 현재 1955초 동안 수행되고 있고 전체 예상 작업 중 3.37% 진행 중인 쿼리 수행이 존재한다라는 것을 알 수가 있다. 아주 오래 걸리는 batch 작업을 psm을 이용해서 작성하는 경우에 DBMS_APPLICATION_INFO 패키지의 SET_SESSION_LONGOPS() 프로시져를 적절히 이용하면 위의 뷰로 해당 작업에 대해 진행 상황을 파악할 수가 있다.

이 기능과 관련된 설정 파라미터는 다음과 같다.

파라미터 이름	값 타입	설명
LONGOPS_THRESHOLD_SEC	NUMBER	이 값보다 오래 수행되는 작업을 LONGOPS에 등록
LONGOPS_KEEP_INTERVAL	NUMBER	LONGOPS 정보들을 keep하는 주기
_LONGOPS_CLEANUP_INTERVAL	NUMBER	주기적으로 오래된 LONGOPS를 지우는 시간 간격

하) SQLTRACE 이용하기

SQLTRACE 기능에 대한 소개에서 언급한 바와 같이 ALTER 구문을 이용해서 특정 시간 동안 수행되는 SQL 정보를 파일로 남길 수가 있다. 남겨진 TRC 파일로 TBPROF 프로그램을 이용할 때, 다양한 SORT 옵션을 이용하면 해당 기준으로 정렬된 trace 파일을 얻을 수가 있다. 이 파일을 분석함으로써 잘못 수행되고 있는 SQL 플랜을 찾을 수 있다.

SORT 옵션에서 FCHELA (select 쿼리 수행 시간), EXEELA (dml 수행 시간) 옵션을 이용하여 정렬하면 느린 쿼리/DML 을 찾는데 많은 도움이 된다.

3. Conclusion

SQL을 수행하면 Tibero optimizer가 통계 정보에 의한 cost 값에 따라 플랜을 수립하는데, 이렇게 만들어진 SQL 플랜을 확인하는 방법에 대해 알아보았고, 비효율적으로 만들어진 SQL 플랜을 쉽게 찾을 수 있는 방법을 소개하였다.

앞으로 Tibero에 사용자들이 SQL 플랜을 더 편리하게 관리할 수 있는 기능들을 지속적으로 추가할 예정이다. 이 문서에서 제시한 방법들은 앞으로 제공할 기능들의 가장 기본이 되는 것들이고 이 기능들을 바탕으로 계속 확장될 것이므로 이에 대한 정확한 이해를 가지는 것이 앞으로 많은 도움이 되리라 생각된다.

그리고, 향후에 이 문서에서 다루지 않은 비효율적인 SQL 플랜을 튜닝하는 방법에 대해서는 다른 문서를 통해 제공하도록 하겠다.

ⓒ Copyright TIBERO 2012. All Rights Reserved.

본 문서에서 제공하는 기술적 또는 상업적 정보에 대한 저작권은 TIBERO 에 있으며, 본 문서는 TIBERO 의 허락 없이 타인에 의해 복제 또는 다른 언어, 수단, 목적으로 변형되거나 배포될 수 없다.

본 문서는 오로지 정보의 제공만을 목적으로 하고, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 아니하며, 본 문서상의 내용은 구두로 제공되거나 법적 또는 상업적인 특정한 조건을 만족시키는 것을 보장하지는 않는다. 본 문서의 내용은 제품의 업그레이드나 수정에 따라 그 내용이 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 아니한다.

TIBERO 상표는 한국 및 기타 다른 나라에서 TIBERO 의 상표로 등록된 것이다.
기타 다른 회사명 또는 상표는 다른 권리자의 상표 혹은 서비스표일 수 있다.

문서번호 넣으실 곳