DBMS _PARALLEL _EXECUTE

How to get PL/SQL to run in parallel

DOAG Conference 2012

Jan Ott

Senior Consultant

Trivadis AG

November 20th 2012

Trivadis facts & figures

11 Trivadis locations with more than 650 employees

Financially independent and sustainably profitable

Key figures 2011

- Revenue CHF 104 / EUR 84 Mio.
- Services for more than 800 clients in over 1,900 projects
- Over 200 Service Level Agreements
- More than 4,000 training participants
- Research and development budget: CHF 5.0 / EUR 4 Mio.

AGENDA

1. Introduction
2. Task
3. Prerequisite
4. Create a Task
5. Create the Chunks
6. Run - SQL
7. Other Chunking Methods
8. Run – PL/SQL
9. Observe and intervene
10. Practical Examples
11. The Bathtub Curve
12. Conclusion

Introduction

Parallelism in PL/SQL and SQL

- PL/SQL
  - Not parallel
  - Only one process
- SQL
  - Parallel but only under certain circumstances
- Solution DBMS_PARALLEL_EXECUTE
  - Run parts of PL/SQL process in parallel

Task

- To Do
  - List of ID's – T_LIST
  - For each ID run procedure P1
  - The calls for P1 are independent of each other
- Solution – without DBMS_PARALLEL_EXECUTE

```
SQL > BEGIN
        FOR i IN (SELECT id FROM t_list)
        LOOP
          p1(i.id);
        END LOOP;
      END;
      /
```

- Result
  - PL/SQL block runs for several hours
  - Only 1 processor is used

Prerequisite

- Needed Grants – as SYSTEM
  - grant execute on DBMS_PARALLEL_EXECUTE
  - grant execute on DBMS_SQL
  - grant create job

```
SQL> GRANT CREATE JOB TO scott;

Grant succeeded.
```

- Data Dictionary View – as user

```
SQL> SELECT * FROM user_sys_privs;

USERNAME                       PRIVILEGE                                ADM
------------------------------ ---------------------------------------- ---
SCOTT                          CREATE JOB                               NO
```

Create a Task

To work with DBMS_PARALLEL_EXECUTE a Task is needed

- Create the Task

```
SQL> BEGIN
  2    dbms_parallel_execute.create_task
  3      (task_name => 'work_through_list'
  4      ,comment   => 'work through the list'
  5      );
  6  END;
  7  /

PL/SQL procedure successfully completed.
```

- Data Dictionary View

```
SQL> SELECT task_name, chunk_type, status, task_comment
  2    FROM user_parallel_execute_tasks;

TASK_NAME            CHUNK_TYPE   STATUS              TASK_COMMENT
-------------------- ------------ ------------------- -------------------------
work_through_list    UNDELARED    CREATED             work through the list
```

Create Chunks (1)

How to split the task into chunks can be done in 3 ways

- CREATE_CHUNKS_BY_NUMBER_COL
- CREATE_CHUNKS_BY_ROWID
- CREATE_CHUNKS_BY_SQL

Create Chunks by number (1)

- Create the chunks

```
SQL> BEGIN
  2    dbms_parallel_execute.create_chunks_by_number_col
  3        (task_name     => 'work_through_list'
  4        ,table_owner   => 'SCOTT'
  5        ,table_name    => 'T_LIST'
  6        ,table_column  => 'ID'
  7        ,chunk_size    => 100
  8        );
  9  END;
 10  /
```

- Data Dictionary View – Task updated

```
SQL> SELECT task_name, chunk_type, status, table_name, number_column
  2    FROM user_parallel_execute_tasks;

TASK_NAME            CHUNK_TYPE   STATUS     TABLE_NAME      NUMBER_COL
-------------------- ------------ ---------- --------------- ----------
work_through_list    NUMBER_RANGE CHUNKED    T_LIST          ID
```

Create a Chunks by number (2)

- Data Dictionary View – The Chunks

```
SQL> SELECT chunk_id, task_name, status, start_id, end_id
  2    FROM user_parallel_execute_chunks
  3   ORDER BY start_id;

  CHUNK_ID TASK_NAME                     STATUS       START_ID     END_ID
---------- ------------------------------ ---------- ---------- ----------
       591 work_through_list              UNASSIGNED          1        100
       592 work_through_list              UNASSIGNED        101        200
       593 work_through_list              UNASSIGNED        201        300
       594 work_through_list              UNASSIGNED        301        400
       595 work_through_list              UNASSIGNED        401        500
       596 work_through_list              UNASSIGNED        501        600
       597 work_through_list              UNASSIGNED        601        700
       598 work_through_list              UNASSIGNED        701        800
       599 work_through_list              UNASSIGNED        801        900
       600 work_through_list              UNASSIGNED        901       1000
```

  - CHUNK_ID 591 has START_ID 1 and END_ID100
  - ID's numbers 90 to 99 do not exist
  - Only 90 entries in this chunk
- Conclusion – Chunks by number are based on the NUMBER not the count.

Run the Tasks (1)

We control how

- Run the Chunks

```
SQL> BEGIN
  2    dbms_parallel_execute.run_task
  3      (task_name      => 'work_through_list'
  4      ,sql_stmt       => 'INSERT INTO t_result(id, ts)'
  5                      || '  SELECT id, sysdate'
  6                      || '    FROM t_list'
  7                      || '   WHERE id BETWEEN :start_id AND :end_id'
  8      ,language_flag  => dbms_sql.native
  9      ,parallel_level => 5
 10      );
 11  END;
 12  /

PL/SQL procedure successfully completed.
```

- Statement – sql_stmt
  - :start_id and :end_id must be in the PL/SQL code or SQL
- Parallel – parallel_level

Please keep in mind that the execution of the code waits at the line of the call to RUN till the chunks are worked off.

Run the Tasks (2)

- Data Dictionary View – Task finished

```
SQL> SELECT task_name, chunk_type, status, job_prefix
  2       , SUBSTR(sql_stmt,1,8) AS sql_stmt, parallel_level
  3    FROM user_parallel_execute_tasks;

TASK_NAME          CHUNK_TYPE   STATUS     JOB_PREFIX SQL_STMT PARALLEL_LEVEL
------------------ ------------ ---------- ---------- -------- --------------
work_through_list  NUMBER_RANGE FINISHED   TASK$_224  INSERT I              5
```

- Data Dictionary View – Chunks processed

```
SQL> SELECT chunk_id, status, start_id, end_id, job_name, start_ts, end_ts
  2    FROM user_parallel_execute_chunks
  3   WHERE task_name = 'work_through_list';

  CHUNK_ID STATUS       START_ID     END_ID JOB_NAME    START_TS  END_TS
---------- ---------- ---------- ---------- ----------- --------- ---------
       876 PROCESSED         201        300 TASK$_224_1 08.11.12  08.11.12
                                              20:08:57, 20:08:57,
                                              252000    345000

       877 PROCESSED         301        400 TASK$_224_1 08.11.12  08.11.12
                                              20:08:57, 20:08:57,
                                              345000    345000

       878 PROCESSED         401        500 TASK$_224_1 08.11.12  08.11.12
                                              20:08:57, 20:08:57,
                                              345000    345000
```

Other Chunking Methods – by ROWID (1)

- Create the chunks

```
SQL> BEGIN
  2    dbms_parallel_execute.create_chunks_by_rowid
  3      (task_name     => 'work_through_list'
  4      ,table_owner   => USER
  5      ,table_name    => 'T_LIST'
  6      ,by_row        => TRUE -- it seems not to have a influence
  7      ,chunk_size    => 100
  8      );
  9  END;
 10  /
```

- Data Dictionary View – Task updated

```
SQL> SELECT task_name, chunk_type, status, table_name, number_column
  2    FROM user_parallel_execute_tasks;

TASK_NAME            CHUNK_TYPE   STATUS     TABLE_NAME      NUMBER_COL
-------------------- ------------ ---------- --------------- ----------
work_through_list    ROWID_RANGE  CHUNKED    T_LIST
```

Other Chunking Methods – by ROWID (2)

- Only two Chunks

```
SQL> SELECT chunk_id, task_name, status, start_rowid, end_rowid
  2    FROM user_parallel_execute_chunks
  3   ORDER BY start_id;

  CHUNK_ID TASK_NAME            STATUS     START_ROWID         END_ROWID
---------- -------------------- ---------- ------------------ ------------------

       602 work_through_list    UNASSIGNED AAASZiAAEAAAAJ4AAA AAASZiAAEAAAAJ/CcP

       601 work_through_list    UNASSIGNED AAASZiAAEAAAAJwAAA AAASZiAAEAAAAJ3CcP
```

- Data Dictionary View – Task updated

```
SQL> SELECT task_name, chunk_type, status, table_name, number_column
  2    FROM user_parallel_execute_tasks;

TASK_NAME            CHUNK_TYPE   STATUS     TABLE_NAME      NUMBER_COL
-------------------- ------------ ---------- --------------- ----------
work_through_list    ROWID_RANGE  CHUNKED    T_LIST
```

- The number of rows is influenced by the number of extends

Other Chunking Methods – by SQL (1)

- Create the Chunks

```
SQL> BEGIN
  2    dbms_parallel_execute.create_chunks_by_sql
  3      (task_name      => 'work_through_list'
  4      ,sql_stmt       => 'SELECT MIN(id) AS start_id'    || CHR(10)
  5                      || '      , MAX(id) AS end_id'     || CHR(10)
  6                      || '  FROM t_list'                 || CHR(10)
  7                      || ' GROUP BY  TRUNC(id /100)'
  8      ,by_rowid       => FALSE
  9      );
 10  END;
 11  /
```

- Data Dictionary View

```
SQL> SELECT task_name, chunk_type, status, table_name, number_column
  2    FROM user_parallel_execute_tasks;

TASK_NAME            CHUNK_TYPE   STATUS     TABLE_NAME      NUMBER_COL
-------------------- ------------ ---------- --------------- ----------
work_through_list    NUMBER_RANGE CHUNKED
```

Run – PL_SQL

- Run

```
SQL> BEGIN
  2    dbms_parallel_execute.run_task
  3      (task_name       => 'work_through_list'
  4      ,sql_stmt        => 'BEGIN'
  5                       || '   p1(in_start_id => :start_id' || CHR(10)
  6                       || '     ,in_end_id   => :end_id'   || CHR(10)
  7                       || '     );'                        || CHR(10)
  8                       || 'END;'
  9      ,language_flag   => dbms_sql.native
 10      ,parallel_level => 10
 11      );
 12  END;
 13  /
```

- Procedure

```
SQL> CREATE OR REPLACE PROCEDURE p1
  2    (in_start_id IN INTEGER
  3    ,in_end_id   IN INTEGER
  4    )
  5  IS
  6  BEGIN
  7    FOR i IN in_start_id..in_end_id
  8    LOOP
  9      INSERT INTO t_result
 10          (id, ts)
 11        VALUES
 12          (i, sysdate);
 13    END LOOP;
 14  END p1;
 15  /
```

Observe and intervene

- Procedures for maintenance
  - TASK_STATUS

```
SQL>
SQL> SET SERVEROUTPUT ON
SQL>
SQL> DECLARE
  2    v_task_status NUMBER;
  3    v_string VARCHAR2(30);
  4  BEGIN
  5    v_task_status := dbms_parallel_execute.task_status(task_name => 'work_through_list');
  6
  7    CASE v_task_status
  8      WHEN 1 THEN v_string := 'the status is: ' || v_task_status || ' - CREATED';
  9      WHEN 2 THEN v_string := 'the status is: ' || v_task_status || ' - CHUNKING';
 10      WHEN 3 THEN v_string := 'the status is: ' || v_task_status || ' - CHUNKING_FAILED';
 11      WHEN 4 THEN v_string := 'the status is: ' || v_task_status || ' - CHUNKED';
 12      WHEN 5 THEN v_string := 'the status is: ' || v_task_status || ' - PROCESSING';
 13      WHEN 6 THEN v_string := 'the status is: ' || v_task_status || ' - FINISHED';
 14      WHEN 7 THEN v_string := 'the status is: ' || v_task_status || ' - FINISHED_WITH_ERROR';
 15      WHEN 8 THEN v_string := 'the status is: ' || v_task_status || ' - CRASHED';
 16      ELSE        v_string := 'the status is: ' || v_task_status || ' - unknown';
 17    END CASE;
 18
 19    dbms_output.put_line(v_string);
 20  END;
 21  /
the status is: 4 - CHUNKED

PL/SQL procedure successfully completed.
```

Observe and intervene (2)

- STOP_TASK

```
SQL> BEGIN
  2    dbms_parallel_execute.resume_task(task_name => 'work_through_list');
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

- RESUME_TASK

```
SQL> BEGIN
  2    dbms_parallel_execute.stop_task(task_name => 'work_through_list');
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

this does hold this session until ended.

- DROP_TASK

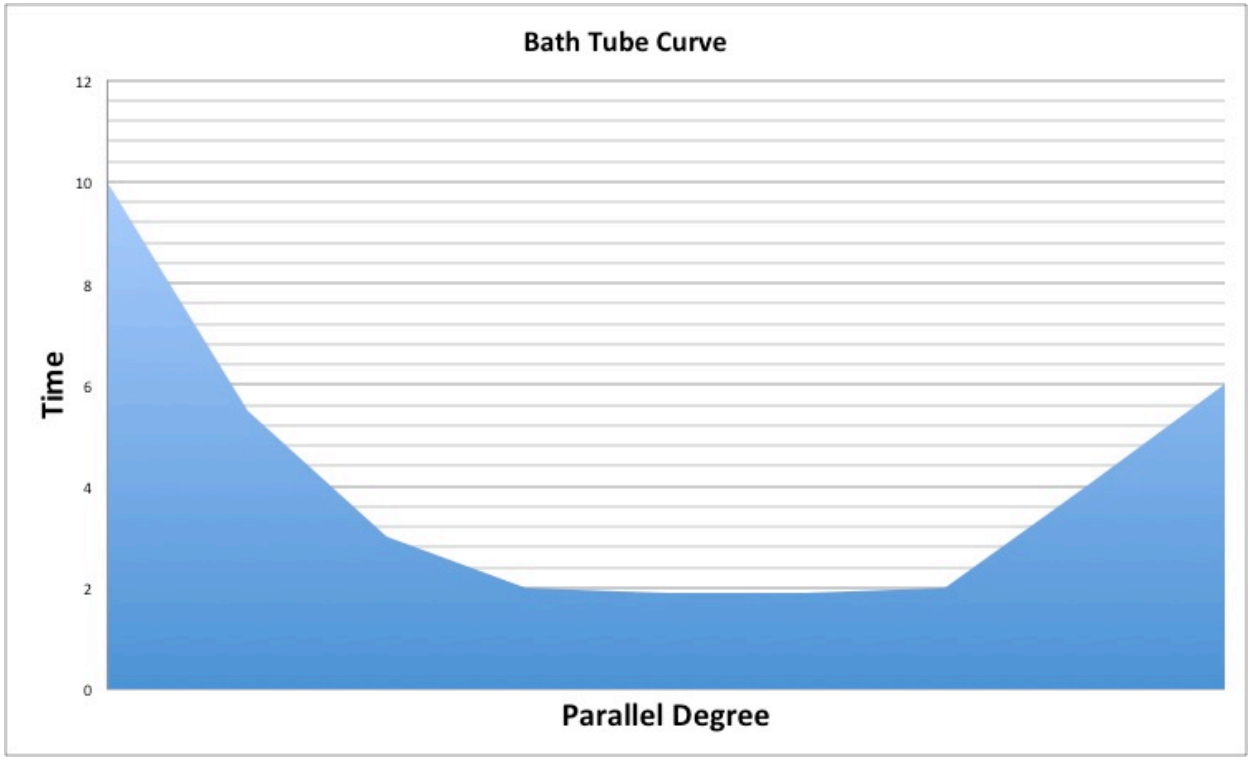
```
SQL>
SQL> BEGIN
  2     dbms_parallel_execute.drop_task
  3      (task_name => 'work_through_list'
  4      );
  5  END;
  6  /

PL/SQL procedure successfully completed.
```

Practical Examples

- Export/Import vs. DBMS_PARALLEL_EXECUTE
  - 4 Node Cluster – 64 cores – 512 GB RAM
  - Transfer 4 Tables
  - Tables contain BLOB's
  - Time
    - Exp/Imp – 8 hours
    - PL/SQL – 3 hours

The Bathtub Curve

- Increase Parallel Degree – Decrease Runtime



- Find best chunk size

Conclusion - Thoughts

- Has its limitations
  - Chunking must be possible
  - Parts must be independent of each other
- The time for reading the “chunk” table
- Job creation overhead – choose chunk size accordingly
- Number of cores/processes – choose parallelism accordingly
- Job service – possibility to split the load onto available nodes
- If usable - Valuable Option
- Going one step further
  - Run things in parallel – patch parts that can be run in parallel
  - System waits automatically till all parts are through

Interesting Links

- Oracle 11g R2 Documentation
  http://docs.oracle.com/cd/E14072_01/appdev.112/e10577/d_parallel_ex.htm#CHDBECBG
- Example by Steven Feuerstein
  http://www.oracle.com/technetwork/issue-archive/2010/10-may/o30plsql-086044.html

THANK YOU.

VISIT US AT TRIVADIS-STAND:

Floor 3, No. 304

Trivadis AG

Jan Ott

Europa-Strasse 5
CH-8152 Glattbrugg (Zürich)

Tel. +41-44-808 70 20
Fax +41-44-808 70 21

info@trivadis.com
www.trivadis.com

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN