Performance tuning an Oracle Spatial process: experience at a customer site

Simon Greener
SpatialDB Advisor
Business Need

- Old Dual CPU Solaris hardware
- “Wireframe” (350,000 lines) land parcel fabric
- Radius Topology structured 157,374 faces from “wireframe”
  - Thus there are 157,374 potential land parcels (topological faces)
- Requirements
  - Generate actual land parcel polygons from faces by intersection with parcel “point” (centroid) layer (156,543) and parcel data table
  - Do it “as fast as” possible on this old hardware
It is possible to have multiple points inside a parcel but only one has the actual parcel details associated with it.
Methodology

- Use multiple sqlplus sessions
  - How?

- Avoid session conflicts
  - How?

- Achieve fast processing and insertion
  - How?
Approach

- Multiple sqlplus sessions
  
  ```
  SET Commit_Interval=2000
  SET Process_Max=3
  SET parcel_table=SP_PARCEL
  ECHO Create SP_PARCEL records from Faces with centroid processing via a number of parallel sessions
  FOR /L %%i IN (1,1,%Process_Max%) DO @start "Face %%i %uname%.%parcel_table%" /D"%CD%" /i /min /realtime sqlplus -s %uname%/%upass%@%dbcon%
  @create_sp_parcel.sql %mflfd_name% %%%i %Process_Max% %Commit_Interval% %parcel_table% %uname%
  ```

- Easy!
Avoid Conflict?

- **Simple stratification algorithm:**

```sql
v_sql := 'SELECT ID, GEOM_VERSION ';
v_sql := v_sql || ' FROM LSL_FACE$'||v_mfld_id;
v_sql := v_sql || ' WHERE ID NOT IN (SELECT UNIVERSE_FACE_ID FROM USER_LSL_MANIFOLD_METADATA WHERE manifold_id = '||v_mfld_id||')';
OPEN faceCursor FOR v_sql;
LOOP
  FETCH faceCursor INTO v_id, v_geom_vn;
  EXIT WHEN faceCursor%NOTFOUND;
  -- Is this row for me?
  IF ( MOD( faceCursor%ROWCOUNT - 1, &&max_processes. ) + 1 = &&process_id. ) THEN
    THEN
```
### Row Stratification - Visually

![Data Table Example]

<table>
<thead>
<tr>
<th>GID</th>
<th>TAG X</th>
<th>TAG Y</th>
<th>GEOMETRY</th>
<th>PRCL TYPE</th>
<th>LSTATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>588666 460699.8475... 56463307.711090...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>588667 460707.3491... 56463256.281887...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>36871 439167.6059... 56475554.035103...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>36872 458948.9774... 5645701.710220...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>36873 433965.2711... 5644963.762625...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>36856 447974.4669... 56470158.296694...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>36857 446759.1895... 56468571.423166...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>36905 447999.8575... 56470260.028922...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>36897 454264.8342... 56471014.9667904</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>36898 447997.6712... 56470090.4635929</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>36899 447798.1696... 56470330.258925...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>36900 449771.1806... 56470996.581411...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>36901 451708.0861... 56440498.645756...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>36902 446959.6953... 56468446.057426...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>36904 449531.0015... 56471924.5229294</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>36895 460806.4356... 56452439.611480...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>36896 468828.9964... 56472833.942067...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>36886 449492.1578... 56471784.437184...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>36887 448013.7556... 56470189.706746...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>36888 451997.8139... 56439931.535267...</td>
<td>oracle.sql.STRU... P</td>
<td>R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Process 1**
- **Process 2**
- **Process Max (3)**
Fast Processing

- Bottlenecks – Identify them!
- Hints – Use Them!
  - How?
  - Explain Plan
    - Huh?
      - Learn it!
- Batch or Array Inserts
  - The only way!
(Timing) Bottlenecks

- **Find them by recording timing:**
  - Eg Create `Sdo_Geometry` from RT topological face:

```sql
v_GetFaceStartTime := dbms_utility.get_time;
v_geom := lsl_topo_util.face_get_geometry
           (v_mfld_id, v_id, v_geom_vn);
v_GetFaceTotalTime := v_GetFaceTotalTime +
                       (dbms_utility.get_time -
                        v_GetFaceStartTime);
```
v_SdoInsideStartTime := dbms_utility.get_time;
SELECT /*+ ORDERED FIRST_ROWS(2) */
    pt.gid,  
    pt.geometry.sdo_point.x,  
    pt.geometry.sdo_point.y
INTO v_spi, v_Tag_X, v_Tag_Y
FROM sp_parcel_pt pt,  
    sp_parcels_bw pl
WHERE SDO_INSIDE (pt.geometry, v_geom)  
    = 'TRUE'
    AND pl.gid = pt.gid;

v_SdoInsideTotalTime := v_SdoInsideTotalTime +  
    ( dbms_utility.get_time -  
    v_SdoInsideStartTime );
Hints and Spatial

- Hints can be used for Oracle Spatial too!
  - What Oracle says is “Best-Practice” is to use the ORDERED hint when the query window (geometry-2) in an operator (SDO_INSIDE) comes from a table, then use the ordered hint, putting the table geometry-2 comes from first in the from clause.

  -- Now, is this a face that is really representing a parcel?
  -- If it is, then:
  -- 1. It should have a point in it
  -- 2. And it should be THE point that represents the master centroid

  BEGIN
  -- Use a pl/sql block to capture no row, too many row EXCEPTIONS.
  v_SdoInsideStartTime := dbms_utility.get_time;
  SELECT /*+ ORDERED */
    pt.gid, pt.geometry.sdo_point.x, pt.geometry.sdo_point.y
  INTO v_spi, v_Tag_X, v_Tag_Y
  FROM sp_parcel_pt pt,
       sp_parcel_bw pl
  WHERE SDO_INSIDE (pt.geometry, v_geom) = 'TRUE'
    AND pl.gid = pt.gid;

- But it doesn’t appear fast enough: Why?
Explaining the query

SQL> explain plan
2 set statement_id = 'FACE QUERY'
3 for
4 SELECT /*+ ORDERED */
5 pt.gid,
6 pt.geometry.sdo_point.x,
7 pt.geometry.sdo_point.y
8 FROM sp_parcel_pt pt,
9 sp_parcels_bw pl
10 WHERE SDO_INSIDE (pt.geometry, 
11 lsl_topo_util.face_get_geometry (2, 45, 2))
12 = 'TRUE'
13 AND pl.gid = pt.gid;

Explained.

Short cut for purpose of testing in Explain Plan
SQL> select * from table(dbms_xplan.display('PLAN_TABLE','FACE QUERY','TYPICAL'));

PLAN_TABLE_OUTPUT

Plan hash value: 714054537

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1566</td>
<td>186K</td>
<td>87 (5)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>* 1</td>
<td>HASH JOIN</td>
<td></td>
<td>1566</td>
<td>186K</td>
<td>87 (5)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>SP_PARCEL_PT</td>
<td>1566</td>
<td>177K</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 3</td>
<td>DOMAIN INDEX</td>
<td>SP_PARCEL_PT_$X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INDEX FAST FULL SCAN</td>
<td>SP_PARCELS_BW_GID</td>
<td>155K</td>
<td>909K</td>
<td>80 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

**Interpret the results**

Good?

Err... No

Definitely Not!
Change the query - FIRST_ROWS(n)

- **Query** should return 1 row if correct. If 2 rows then wrong answer.
- **So, let’s try the FIRST_ROWS(m) (or FIRST_ROWS_n) hint**
  - What’s that?
  - From the documentation:

  "FIRST_ROWS(n)"

This hint instructs Oracle to optimize an individual SQL statement with a goal of best response time to return the first n number of rows, where n equals any positive integer. The hint uses a cost-based approach for the SQL statement, regardless of the presence of statistic."
FIRST_ROWS(2) in action

Our business rule says that while a parcel may have multiple points inside it, only ONE point (pl.gid = pt.gid) holds the right information.

- That is our query should return 1 row if correct any more rows then we have the wrong answer.
- So, let’s optimise for a few rows via the FIRST_ROWS(n) hint:

```sql
BEGIN
  -- Now, is this a face that is really representing a parcel?
  -- If it is, then:
  -- 1. It should have a point in it
  -- 2. And it should be THE point that represents the master centroid
  v_SdoInsideStartTime := dbms_utility.get_time;
  SELECT /*+ ORDERED FIRST_ROWS(2) */
    pt.gid, pt.geometry.sdo_point.x, pt.geometry.sdo_point.y
  INTO v_spi, v_Tag_X, v_Tag_Y
  FROM sp_parcel_pt pt,
       sp_parcels_bw pl
  WHERE SDO_INSIDE (pt.geometry, v_geom) = 'TRUE'
    AND pl.gid = pt.gid;
  ...
Explain Plan (First_Rows(2))

```
SQL> select * from table(dbms_xplan.display('PLAN_TABLE','FACE QUERY','TYPICAL'));
PLAN_TABLE_OUTPUT
---
Plan hash value: 4094247311

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>2</td>
<td>244</td>
<td>4</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>2</td>
<td>244</td>
<td>4</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>SP_PARCEL_PT</td>
<td>2</td>
<td>232</td>
<td>2</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 3</td>
<td>DOMAIN INDEX</td>
<td>SP_PARCEL_PT_$X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 4</td>
<td>INDEX RANGE SCAN</td>
<td>SP_PARCELS_BW_GID</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
```

Yes... much better

Now that’s more like it!
Array or Batch inserts

- **Single INSERT statements are costly**
- **PL/SQL makes “ARRAY” inserts easy via:**
  - FORALL statement
- **Approach**
  - Cache results of parcel building in array
  - Execute single FORALL statement for the cached set at a set interval
Declare Array

TYPE parcel_t IS TABLE
  OF sp_parcel%ROWTYPE
  INDEX BY PLS_INTEGER;

v_parcels parcel_t;
v_parcel s_parcel%ROWTYPE;

(Note: sp_parcel has an Sdo_Geometry column – fully supported in PL/SQL)
Fill the array with results of search

-- Fill our record with data

```sql
SELECT munseq_gid.NEXTVAL
    INTO v_parcel.gid
FROM DUAL;
```

```plaintext
v_parcel.tag_x := v_tag_x;  -- From SELECT
v_parcel.tag_y := v_tag_y;
v_parcel.spi := v_spi;
v_parcel.geometry := v_geom;
```

-- Assign record to array

```plaintext
v_counter := v_counter + 1;
v_parcel(v_counter) := v_parcel;
```

Keep count of rows in cache
Batch insert

- When cache (varray of record) has reached parameter-driven limit – batch insert the data in the cache into the target table (sp_parcel):

```sql
IF ( MOD(v_counter,v_iterations) = 0 ) THEN
  v_BatchStartTime := dbms_utility.get_time;
  FORALL i IN 1..v_counter
    SAVE EXCEPTIONS
    INSERT INTO &&sp_parcel_table_name.
      VALUES v_parcels(i);
  COMMIT;
  v_BatchTotalTime := v_BatchTotalTime +
    ( dbms_utility.get_time -
      v_BatchStartTime );
  v_parcel_count := v_parcel_count + v_counter;
  v_counter := 0;
END IF;
```
Pinning Tables and RTree Indexes

- As a final performance improvement:
  - The `sp_parcel_pt` table and its RTree index, and
  - The `sp_parcel_bw` table, plus;
  - The Radius Topology topology tables,
- ... were all “pinned” into memory.
- Examples:

```sql
ALTER TABLE sp_parcel_pt STORAGE(BUFFER_POOL KEEP);
DECLARE
  v_nl_table_name varchar2(30);
BEGIN
  SELECT sdo_nl_index_table
    INTO v_nl_table_name
    FROM user_sdo_index_metadata
    WHERE sdo_index_name = 'SP_PARCEL_PT_$X';
  EXECUTE IMMEDIATE 'ALTER TABLE ' || v_nl_table_name || ' STORAGE(BUFFER_POOL KEEP)';
END;
/```
Result

Report (sample)

Faces Processed: 7869
Parcels from Faces: 7586
Faces with no centroids: 249
Faces with multiple centroids: 34
Total Execution Time (secs): 448

Face_Get_Geom Time (secs): 229.3
_Sdo_Inside Query Time (secs): 142.3
Batch_Insert Time (secs): 3.84

Processing speed (Faces/sec): 17.6

Face_Get_Geom (Avg): .0291
_Sdo_Inside (Avg): .0181
Batch_Insert (Avg): .00768
Thanks

My thanks go to:

- Barwon Water (for the problem)
- Open Spatial Pty Ltd (for the chance to work at their site)
- Daniel Abugov (for QA’ing the talk so well)
- Steven Feuerstein – PL/SQL Guru (for his books that helped me out with some tricky array processing of records).