Oracle Spatial User Conference

May 23, 2012
Ronald Reagan Building and International Trade Center
Washington, DC  USA
Siva Ravada
Director of Development
New Performance Enhancements in Oracle Spatial
Program Agenda

- Introduction
- Faster Operators
- Improved GeoRaster operations
- Router and Geocoder enhancements
- Data Management Improvements
- Discussion
What are Performance Enhancements?

- Better utilization of DB resources and new features
  - Securefiles, compression
- Improved and faster Algorithms for solving problems
  - Spatial operators
- Tools to improve business workflows
  - Transportable tablespaces
- Hardware related enhancements
  - Exadata
Securefiles and SDO_GEOMETRY

- The SDO_ORDINATES in the geometry can be stored as securefiles lob

```
CREATE TABLE MAP_WORLD_SEC (
    COUNTRY VARCHAR2(50), CAPITAL VARCHAR2(30),
    GEOMETRY MDSYS.SDO_GEOMETRY
) VARRAY "GEOMETRY"."SDO_ELEM_INFO" STORE AS SECUREFILE LOB VARRAY "GEOMETRY"."SDO_ORDINATES" STORE AS SECUREFILE LOB
```

- Several options for securefiles storage
  - SECUREFILE LOB (CACHE)
  - SECUREFILE LOB (CACHE COMPRESS)
  - SECUREFILE LOB (CACHE COMPRESS HIGH)
Securefiles and SDO_GEOMETRY

- Test with a simple SQL operation
  - `select sum(sdo_util.getnumvertices(geometry)) from map_world_XXX`
- Read Performance for different options

<table>
<thead>
<tr>
<th>Lob Type</th>
<th>Read Cost</th>
<th>Storage Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Lob</td>
<td>00:00:01.66</td>
<td>35651584</td>
</tr>
<tr>
<td>SEC File CACHE</td>
<td>00:00:01.61</td>
<td>39452672</td>
</tr>
<tr>
<td>SEC File CACHE COMPRESSSS</td>
<td>00:00:02.22</td>
<td>26673152</td>
</tr>
<tr>
<td>SEC File CACHE COMP HIGH</td>
<td>00:00:02.19</td>
<td>25427968</td>
</tr>
</tbody>
</table>
How to get the most out of Securefiles for SDO_GEOMETRY

- VARRAY’s have 3 types of storage based on the size and storage options
  - Always store as a lob (always stored in lob segments)
  - Store small varrays (less than 4K) inline and store bigger varrays as out of line lob segments
- When the VARRAY storage is specified explicitly, it always stores it as a lob, even for smaller varrays
  - Specify ENABLE STORAGE IN ROW to make sure inline storage is enabled for smaller VARRAYs
- Set this parameter in init.ora
  - db_block_checking=false
**Securefiles and GeoRaster**

- Used the dataset of 50 DOQs to conduct the tests
- The results show very good performance improvement of SecureFiles over BasicFiles:
  - load: 1.16 times faster
  - mosaic: 2.5 times faster
  - pyramiding: 3.3 times faster
  - copy: 4.15 times faster
  - changeFormatCopy: 4.23 times faster
  - subset: 3.4 times faster
Redo Log Enhancements

- **Problem:** Excessive redo generated for DML operations on tables with Spatial indexes
  - MDRT tables store the index nodes aslobs and these lobs tend to generate lot of redo
- **Several techniques are used to reduce the redo in 11.2.0.3**
  - Reduce the updates to the index nodes
  - Delayed write of the index nodes
  - Better bulk insert/delete algorithm to reduce the number of modified nodes
Redo Log Enhancements

• Improvements with delayed write: 380K rows
  – Normal Insert  1,509,711,160  07:34.09  00:06.26
  – Delayed Write  1,246,742,644  06:36.33  00:05.47

• New redo algorithms
  – New Code : 2,577,576,572 blocks; Elapsed: 00:12:24.92
  – Old Code : 1,034,054,352 blocks; Elapsed: 00:07:02.97

• Real World proof points
  – I/O reduced by about 90% for index insert/delete operations
  – Throughput increased by 50% for DML operations
Redo Log Enhancements

• Improvements with delayed write: 380K rows
  – Normal Insert  1,509,711,160  07:34.09  00:06.26
  – Delayed Write  1,246,742,644  06:36.33  00:05.47

• New redo algorithms
  – New Code : 2,577,576,572 blocks; Elapsed: 00:12:24.92
  – Old Code   : 1,034,054,352 blocks; Elapsed: 00:07:02.97

• Real World proof points
  – I/O reduced by about 90% for index insert/delete operations
  – Throughput increased by 50% for DML operations
Query Processing Enhancements

- Better use of R-tree indexing technology to improve the performance of all operators (in memory R-tree)
- Improve both Geodetic and non-Geodetic operations
- Interior tile optimizations in 10.2 is based in quad-tree tiles
  - This does not extend well to Geodetic data
- This new approach addresses 4 areas
  - High performance
  - Self-tuning $\Rightarrow$ no user tuning
  - Small footprint $\Rightarrow$ high scalability for enterprise users
  - Handle Earth’s curved surface, poles, equator, international date line
Use in memory R-tree in two steps

- MBR (A) is from a R-tree index, either leaf node, or non-leaf node: 1) MBR (A) intersects polygon? 2) MBR (A) is inside polygon?
Query Processing Enhancements

- Data Sets
  - 1: An energy company: 39,000 locations in USA, a query polygon 4 times as big as Texas, 197,146 line segments
  - 2: A transportation system: 3 million locations and 30 regions around world as query polygons: e.g. “Canada/Mountain”, “American/Honolulu”, “Germany”. average: 59,000 line segments, “Norway” 343,395 line segments
  - 3: US Business Area data set: 10 million locations
    - 50 US states, average: 1,755 line segments, “Alaska” 18,603 line segments;
    - 1061 local regions, average: 520 line segments, “Long Island”: 6,915 line segments
Query Processing Enhancements: ANYINTERACT

<table>
<thead>
<tr>
<th>ANYINTERACT Operator (geodetic)</th>
<th>Data Set 1</th>
<th>Data Set 2</th>
<th>Data Set 3</th>
<th>Data Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2</td>
<td>204</td>
<td>20883</td>
<td>2559</td>
<td>11685</td>
</tr>
<tr>
<td>11.2.0.3</td>
<td>1.4</td>
<td>88</td>
<td>52</td>
<td>81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ANYINTERACT Operator (cartesian)</th>
<th>Data Set 1</th>
<th>Data Set 2</th>
<th>Data Set 3</th>
<th>Data Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2</td>
<td>71</td>
<td>7778</td>
<td>935</td>
<td>2158</td>
</tr>
<tr>
<td>11.2.0.3</td>
<td>2.44</td>
<td>190</td>
<td>58</td>
<td>64</td>
</tr>
</tbody>
</table>
Query Processing Enhancements: INSIDE/CONTAINS

<table>
<thead>
<tr>
<th>INSIDE Operator (geodetic)</th>
<th>Data Set A</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2</td>
<td>22 minutes</td>
</tr>
<tr>
<td>11.2.0.3</td>
<td>1 minute</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INSIDE Operator (cartesian)</th>
<th>Data Set A</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2</td>
<td>12 minutes</td>
</tr>
<tr>
<td>11.2.0.3</td>
<td>30 seconds</td>
</tr>
</tbody>
</table>

Data set A: Table 1 has 260,520 geometries. Table 2 has 12,795 geometries.

Both tables have polygon geometries.

Experiment: Nested Loop join: 12,795 calls to the operator.
Query Processing Enhancements: SDO_WITHIN_DISTANCE

<table>
<thead>
<tr>
<th>WITHIN_DISTANCE Operator (geodetic)</th>
<th>Data Set with</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2</td>
<td>2hr 28 minutes</td>
</tr>
<tr>
<td>11.2.0.3</td>
<td>4 minute</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WITHIN_DISTANCE Operator (cartesian)</th>
<th>Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2</td>
<td>21 minutes</td>
</tr>
<tr>
<td>11.2.0.3</td>
<td>3 minutes</td>
</tr>
</tbody>
</table>

Data set A: Table 1 has 260,520 geometries. Table 2 has 12,795 geometries. Both tables have polygon geometries.

Experiment: Nested Loop join: 12,795 calls to the operator.
GeoRaster Load Performance
ETL tool – the GDAL GeoRaster Driver

- GDAL is the best open source geospatial ETL tool/API for raster data
  - It now natively supports importing and exporting many formats, to/from SDO_GEORASTER, including GeoTIFF, JPEG2000, ECW, NITF, HDF, NetCDF, ERDAS IMG, USGS DEM, SPOT, and more.
  - GDAL is written in C++ and has very good performance
  - It provides C/C++, Java, Python API for accessing GeoRaster
  - It provides many tools. Two of the important ones are:
    - `gdal_translate` – utility to translate raster formats to/from GeoRaster objects
    - `gdalinfo` – utility to view information about a raster, such as a GeoRaster object
Loading a 800 MB GeoTiff
- File is compressed LZW
- Uncompressed size is 5GB
- 4 band image

Loading using GDAL with default blocking (256x256x4) takes … 4 hours!

Loading with a blocking of (256x256x1) takes 4 minutes!

Most of the time is CPU: used by gdal_translate
Understanding Image Structure

- Check the structure of the input TIFF file:

```
> gdalinfo 129-286.tif

Driver: GTiff/GeoTIFF
Files: D:\Files\Data\129-286.tif
Size is 42252, 31198
...
Band 1 Block=42252x32 Type=Byte, ColorInterp=Red
Band 2 Block=42252x32 Type=Byte, ColorInterp=Green
Band 3 Block=42252x32 Type=Byte, ColorInterp=Blue
Band 4 Block=42252x32 Type=Byte, ColorInterp=Undefined
```

- Image is blocked in very wide and very narrow strips
- Very costly for the GeoTIFF driver to collect the pixels to fill our blocks!
GDAL and Memory

- GDAL uses little memory by default
- Set larger caches using GDAL_CACHEMAX option

```
gdal_translate -of georaster sf1.tif --config GDAL_CACHEMAX 1024
georaster:scott/tiger@orcl111,us_rasters,georaster
-co blockxsize=512 -co blockysize=512 -co blockbsize=3
-co interleave=bip -co srid=26943
-co "insert=values (1, 'sf1.tif', 'Aerial photo San Francisco 1',
      sdo_geor.init('us_rasters_rdt_01', 1),null,null)"
```

- Loading the 800 MB Tiff with a blocking of (256x256x4) now completes in less than 3 minutes!
Loading and Compression

• Option 1
  – Load and compress, then generate pyramid

• Option 2
  – Load, then generate pyramid, then compress the result

• The second option is more efficient
Q&A