May 2012
Oracle Spatial User Conference
May 23, 2012
Oracle Spatial User Conference
Ronald Reagan Building and International Trade Center
Washington, DC USA
Daniel Geringer
Senior Software Development Manager
Oracle’s Spatial Technologies
Spatial Performance
Techniques and Examples
For Massive Spatial
Computations and Ingest
Program Agenda

• Exadata
• Parallel Query and Spatial
• Spatial data EHCC compression
• Massive spatial data ingest
• Q&A
• Code examples in presentation appendix
Oracle Exadata Database Machine
Engineered System
What Is the Oracle Exadata Database Machine?

- Oracle SUN hardware uniquely engineered to work together with Oracle database software

- Key features:
  - Database Grid – Up to 128 Intel cores connected by 40 Gb/second InfiniBand fabric, for massive parallel query processing.
  - Raw Disk – Up to 336 TB of uncompressed storage (high performance or high capacity)
  - Memory – Up to 2 TB
  - Exadata Hybrid Columnar Compression (EHCC) – Query and archive modes available. 3x to 30x compression.
  - Storage Servers – Up to 14 storage servers (168 Intel cores) that can perform massive parallel smart scans. Smart scans offloads SQL predicate filtering to the raw data blocks. Results in much less data transferred, and dramatically improved performance.
  - Storage flash cache – Up to 5.3 TB with I/O resource management
Exadata Database Machine Configurations

• X2-2 - configured as Quarter, Half or Full racks
  – X2-2 Quarter Rack – 24 database cores
  – X2-2 Half Rack – 48 database cores
  – X2-2 Full Rack – 96 database cores
• X2-8 – configured as a full rack
  – 128 database cores
Parallel Query and Spatial
With Spatial Operators and Partitioned Tables
Parallel Query And Spatial Operators

Partitioned Tables

- If a spatial operator’s query window spans multiple partitions, partitions are spatially searched in parallel.
- True for all spatial operators
Parallel Query And Spatial Operators

Partitioned Tables

• Example:
  – A re-insurance company maintains portfolios for hundreds of insurance companies.
  – Which companies will be affected by the projected path of hurricane Ike.
  – 36 million rows, 64 partitions, each with about 571,000 rows.
    • 50 seconds serial
    • 1.28 second parallel on \( \frac{1}{2} \) rack Exadata database machine (39 times faster)
Parallel Query and Spatial
With Spatial Operators
Parallel Query And Spatial Operators

- Spatial operators can parallelize when multiple candidates feed the second argument
Parallel Query and Spatial
US Rail Application
Parallel Query And Spatial Operators

US Rail Application

• Requirement
  – GPS locations for each train collected throughout the day
  – Each location has other attributes (time, speed, and more)
  – GPS locations have a degree of error, so they don’t always fall on a track.
  – Bulk nearest neighbor queries to find closest track, and project reported train positions onto tracks

• This information is used for:
  – Tracking trains
  – Analysis for maintenance, ensure engineers are within parameters, etc…
Parallel Query And Spatial Operators

What we tested

- 45,158,800 GPS train positions.
- For each train position:
  - Find the closest track to the train (with SDO_NN)
  - Then calculate the position on the track closest to the train
ALTER SESSION FORCE PARALLEL ddl PARALLEL 72;
ALTER SESSION FORCE PARALLEL query PARALLEL 72;

CREATE TABLE results NOLOGGING
SELECT /*+ ordered  index (b tracks_spatial_idx) */
    a.locomotive_id,
    sdo_lrs.find_measure (b.track_geom, a.locomotive_pos)
FROM locomotives a,
    tracks b
WHERE sdo_nn (b.track_geom, a.locomotive_pos,
    'sdo_num_res=1') = 'TRUE';
Parallel Query And Spatial Operators

Exadata Results

- On Exadata Half RAC:
  - 34.75 hours serially vs. 41.1 minutes in parallel
  - 48 database cores - 47x faster
- On Exadata X2-2 Full Rack
  - 96 database cores – about 94x faster
- X2-8 (128 cores) even faster
Parallel Query and Spatial
Government Sponsored Enterprise
Validation Of Home Appraisals
Validation Of Home Appraisals

Exadata Results

• Validate home appraisals for a Government Sponsored Enterprise (GSE)

• Requirement - Find all the parcels touching parcels to validate appraisals

• Processed 2,018,429 parcels
  – Exadata ½ RAC:
    • Serially – 38.25 minutes
    • Parallel - 48 cores (45x faster) - 50 seconds
  – Exadata X2-2 Full RAC (96 cores) about 90x faster
  – Exadata X2-8 (128 cores) even faster
Parallel Query and Spatial
Parallel Enabled Pipeline Table Functions
Parallel Enabled Pipeline Table Function

• Parallelize a function that’s called a massive amount of times.
  – Batch geocoding (sdo_gcdr.geocode_address)
  – Batch reverse geocoding (sdo_gcdr.reverse_geocode)
  – Batch Digital Elevation Model (DEM) raster lookups (sdo_geor.get_cell_value)

• Pipeline Table Function returns a table of results
  – Table of geocodes
  – Table of reverse geocodes
  – Table of elevation values
Parallel Enabled Pipeline Table Function

How does it work?

• Define cursor that selects the input to the batch process:
  – SELECT customer_addresses
    FROM address_table
    WHERE state = 'NY';
  – SELECT vehicle_location
    FROM vehicles
    WHERE sdo_anyinteract (location,:region) = 'TRUE'

• Parallel Query distributes the batch process over a specified number of database cores.
Parallel Enabled Pipeline Table Function

Exadata ¼ RAC has 2 nodes, each with 12 cores (24 cores total)

```
ALTER SESSION FORCE PARALLEL DML PARALLEL 24;
ALTER SESSION FORCE PARALLEL QUERY PARALLEL 24;
-- To balance parallel slaves across nodes.
ALTER SESSION SET "_parallel_load_balancing"=false;

CREATE TABLE results (id NUMBER, geom SDO_GEOMETRY);

INSERT /*+ append */ INTO results
   SELECT id,
         sdo_geometry (2001,8307,sdo_point_type(longitude,latitude,null),null,null) geom
   FROM table(geocode_pipelined(CURSOR(SELECT id, streetname, city,
                              state, zip, housenum
                              FROM ascii_addrs) )));

COMMIT;
```
ALTER SESSION FORCE PARALLEL DML PARALLEL 48;
ALTER SESSION FORCE PARALLEL QUERY PARALLEL 48;
ALTER SESSION SET "_parallel_load_balancing"=true; (or don’t specify, true default)

CREATE TABLE results (id NUMBER, geom SDO_GEOMETRY);

INSERT /*+ append */ INTO results
SELECT
  id,
  sdo_geometry (2001,8307,sdo_point_type(longitude,latitude,null),null,null) geom
FROM table(geocode_pipelined)(CURSOR(SELECT id, streetname, city,
  state, zip, housenum
  FROM ascii_addrs)
));

COMMIT;
Parallel Pipelined Table Function

Exadata Results

- Exadata ¼ RAC Has 24 Cores
- Batch geocoding – 1365/second
- Batch reverse geocoding – 3388/second
- Batch DEM get_cell_value raster lookups – 8951/seconds
Parallel Enabled Pipeline Table Function

Batch comes from spatial operator search - Example

```
ALTER SESSION FORCE PARALLEL DML PARALLEL 48;
ALTER SESSION FORCE PARALLEL QUERY PARALLEL 48;
-- no_parallel hint in top part of query because top part does not need parallel slaves.

INSERT /*+ append */ into results
WITH temp AS

  (SELECT /*+ ordered no_parallel(a) no_parallel (b) materialize */
   b.location_id id, b.geom.sdo_point.x x, b.geom.sdo_point.y y
  FROM regions a,
     locations b
  WHERE sdo_anyinteract(b.geom,a.geom) = 'TRUE AND a.id = 1)

SELECT id, value
FROM table(geor_utils.get_cell_value (raster_id =>2,
  CURSOR(SELECT id,x,y FROM temp)));
```

EHCC and Spatial
Exadata Hybrid Columnar Compression and Spatial
EHCC and Spatial

• Point, Line and Polygon geometries can all benefit from EHCC

• Lines and Polygons, they must be stored inline (less than 4K in size).

• Options include:
  – COMPRESS FOR QUERY LOW
  – COMPRESS FOR QUERY HIGH
  – COMPRESS FOR ARCHIVE LOW
  – COMPRESS FOR ARCHIVE HIGH
EHCC and Spatial

- Two ways to compress:
  - Create Table As Select
  - Direct Path Inserts

1. Create Table As Select
   
   CREATE TABLE edges_compressed
   COMPRESS FOR QUERY LOW
   NOLOGGING AS SELECT * FROM edges;
EHCC and Spatial

2. Direct Path Inserts (full code example in presentation appendix)

-- PL/SQL Example with append_values hint.
DECLARE
  id_tab      ID_TAB_TYPE;
  edge_tab    GEOM_TAB_TYPE;
BEGIN
  -- Population of id_tab and edge_tab shown in presentation appendix
  FORALL i IN edge_tab.first .. edge_tab.last
    INSERT /*+ append_values */ INTO edge_ql VALUES (id_tab(i), edge_tab(i));
  COMMIT;
### EHCC and Spatial

#### Point data

<table>
<thead>
<tr>
<th>Within_distance Query</th>
<th>Uncompressed</th>
<th>Query Low 3.5x</th>
<th>Query High 5.3x</th>
<th>Archive High 6.8x</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 mile (323 rows)</td>
<td>0.02 sec</td>
<td>0.02 sec (0.0x)</td>
<td>0.06 sec (3.0x)</td>
<td>1.36 sec (68x)</td>
</tr>
<tr>
<td>5 mile (1251 rows)</td>
<td>0.03 sec</td>
<td>0.06 sec (2.0x)</td>
<td>0.12 sec (4.0x)</td>
<td>2.77 sec (92x)</td>
</tr>
<tr>
<td>10 mile (31466 rows)</td>
<td>0.29 sec</td>
<td>0.61 sec (2.1x)</td>
<td>1.3 sec (4.5x)</td>
<td>32.11 sec (110x)</td>
</tr>
</tbody>
</table>
**EHCC and Spatial**

**Line / Polygon data**

<table>
<thead>
<tr>
<th>Within_distance Query (lines)</th>
<th>Uncompressed</th>
<th>Query Low 3.5x</th>
<th>Query High 5.3x</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 mile (809 rows)</td>
<td>0.25 sec</td>
<td>0.75 sec (3.0x)</td>
<td>2.35 sec (9.4x)</td>
</tr>
<tr>
<td>5 mile (3079 rows)</td>
<td>0.65 sec</td>
<td>2.02 sec (3.1x)</td>
<td>6.00 sec (9.2x)</td>
</tr>
<tr>
<td>10 mile (80771 rows)</td>
<td>2.93 sec</td>
<td>8.69 sec (2.9x)</td>
<td>26.0 sec (8.97x)</td>
</tr>
</tbody>
</table>
EHCC and INSERT /*+ append_values */

- Currently, INSERT /*+ append_values */ can EHCC compress points, lines and polygons
- Lines and Polygons must be less than 4K (stored inline)
- INSERT /*+ append_values */ does not compress if the column contains a spatial index.
  - The lifting of this restriction is under investigation.
Massive Spatial Ingests
Massive Spatial Data Ingest
With Spatial Indexes

- Two types of requirements:
  - **Staging Table** - Window of time to make the spatial data available.
    - Use a staging table with no indexes for initial load.
    - Create spatial and non spatial indexes in parallel
    - Exchange staging table into an existing partitioned table.
  - **No Staging Table** - Data must be available immediately for spatial queries. No time for a staging table.

- Different strategies for the two types of requirements.
- Both requirements require data modeling considerations (discussed in upcoming slides).
Massive Spatial Ingests
With A Staging Table
Massive Spatial Data Ingest
With A Staging Table

• Major benefits are performance and manageability of very large data sets

• Customer example:
  – Requirement:
    • Ingest and maintain 2 days of weather data online
    • 270,000 samples every 30 seconds (Oracle Spatial can do even more…)
  – Implemented with:
    • 30 second partitions (5760 partitions over 2 days)
    • New partitions rolled on, older partitions rolled off
Massive Spatial Data Ingest
With A Staging Table (continued)

- Load data into a staging table with no indexes (called new_weather_data)
- Parallel create index (spatial index too) on staging table
- Partition P1 is an empty leading partition
- Update partitioned table with new weather data in a fraction of a second.
- No need to maintain an index on INSERT !!!!

```
ALTER TABLE weather_data_part EXCHANGE PARTITION p1
WITH TABLE new_weather_data
INCLUDING INDEXES
WITHOUT VALIDATION;
```
Massive Spatial Ingests
Without A Staging Table
Massive Spatial Data Ingest
Without A Staging Table

- Spatial data must be inserted and committed immediately
- No time for a staging table.
- If possible, don’t commit after every insert.
- If possible, perform batch inserts.
- Set sdo_commit_interval=15000 when you create the spatial index.
  - PL/SQL - FORALL BULK INSERTS (example in appendix)
  - Java - JDBC UPDATE BATCHING
  - C – Array inserts
Massive Spatial Data Ingest
Without A Staging Table (continued)

• Very Common Misconception:

"Increasing number of connections in a pool and number of threads performing inserts will increase spatial data insert throughput”

– **Only true if** no connections in pool write to the same spatial index.
– For high ingest rates, even two connections writing to the same spatial index can adversely affect spatial index ingest performance.
Massive Spatial Data Ingest
Without A Staging Table (continued)

• Quick fix… try streaming all writes to one connection in the pool.
  – One connection for writes
  – Many connections for reads

• To really maximize spatial ingest throughput:
  – Use multiple connections in a connection pool
  – Dedicate each connection to one spatial index.
    – This eliminates all spatial index contention.

• Strategy continued on next slide.
Massive Spatial Data Ingest

• Example is hourly, but could be monthly or quarterly
• Multiple threads perform batch inserts, each with a dedicated process id

```
CREATE TABLE composite_example
  ( t              timestamp
  , process_id     number
  , geom           sdo_geometry
  , hour_partition as (substr(t,1,12))
  PARTITION BY RANGE (hour_partition, process_id)
  (PARTITION DAY1_H5_1  VALUES LESS THAN ('30-NOV-10 05', 2 ),
   PARTITION DAY1_H5_2  VALUES LESS THAN ('30-NOV-10 05', 3 ),
   PARTITION DAY1_H5_3  VALUES LESS THAN ('30-NOV-10 05', 4 ),
   PARTITION DAY1_H6_1  VALUES LESS THAN ('30-NOV-10 06', 2 ),
   PARTITION DAY1_H6_2  VALUES LESS THAN ('30-NOV-10 06', 3 ),
   PARTITION DAY1_H6_3  VALUES LESS THAN ('30-NOV-10 06', 4 ),
   PARTITION REST     VALUES LESS THAN (MAXVALUE,MAXVALUE));
```
Massive Spatial Data Ingest
Quarter RAC Exadata - Results

- Inserts per second with spatial index enabled
  - Parallel Degree - how many threads performing bulk inserts in parallel
  - For 1 million row partition, 15K batches, 66,123.57 inserts per second

<table>
<thead>
<tr>
<th>Parallel Degree</th>
<th>Inserts/Sec (100k/15k)</th>
<th>Inserts/Sec (1m/15k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,901.92</td>
<td>3,701.10</td>
</tr>
<tr>
<td>2</td>
<td>7,646.74</td>
<td>7,352.68</td>
</tr>
<tr>
<td>8</td>
<td>30,757.95</td>
<td>28,870.05</td>
</tr>
<tr>
<td>12</td>
<td>44,843.68</td>
<td>41,557.92</td>
</tr>
<tr>
<td>8x2</td>
<td>59,547.36</td>
<td>50,237.07</td>
</tr>
<tr>
<td>12x2</td>
<td>84,309.38</td>
<td>66,123.57</td>
</tr>
</tbody>
</table>
Massive Spatial Data Ingest
Quarter RAC Exadata - Results

Inserts Per Second

Number Of Cores

- (100k/15k)
- (1m/15k)
Q&A
Parallel Pipelined Geocoder

Code Example

Appendix
CREATE OR REPLACE TYPE sdo_geo_addr_table_type AS TABLE OF sdo_geo_addr;
/

CREATE OR REPLACE FUNCTION geocode_pipelined (source_table_cursor IN SYS_REFCURSOR)
RETURN sdo_geo_addr_table_type DETERMINISTIC PIPELINED PARALLEL_ENABLE (PARTITION source_table_cursor BY ANY) IS

  TYPE num_array_type IS TABLE OF NUMBER;
  TYPE varchar_array_type IS TABLE OF VARCHAR2(1000);
id_table                 NUM_ARRAY_TYPE;
streetname_table  VARCHAR_ARRAY_TYPE;
city_table              VARCHAR_ARRAY_TYPE;
state_table            VARCHAR_ARRAY_TYPE;
zip_table               VARCHAR_ARRAY_TYPE;
housenum_table   VARCHAR_ARRAY_TYPE;
geo_addr               SDO_GEO_ADDR;

begin
  LOOP
    FETCH source_table_cursor
    BULK COLLECT INTO id_table, streetname_table, city_table, state_table, zip_table, housenum_table LIMIT 100;
    
    -- Exit when no rows were fetched.
    EXIT WHEN id_table.count = 0;
FOR i IN id_table.first .. id_table.last LOOP
    geo_addr := SDO_GCDR.GEOCODE_ADDR('NAVTEQ_SF',
        SDO_GEO_ADDR(0, null, null,
            streetname_table(i), NULL, NULL,
            city_table(i), NULL,
            state_table(i), 'US',
            zip_table(i), NULL, NULL, NULL,
            housenum_table(i), NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            'Default',NULL,NULL));
    geo_addr.id := id_table(i);
    PIPE ROW (geo_addr);
END LOOP;
END LOOP;
CLOSE source_table_cursor;
RETURN;
end;
/

-- See “Parallel Enabled Pipeline Table Function” slide earlier in presentation that -- demonstrates how to call this geocoding.
Parallel Pipelined Reverse Geocoder
Code Example
Appendix
CREATE OR REPLACE TYPE sdo_geo_addr_table_type AS TABLE OF sdo_geo_addr;
/

CREATE OR REPLACE function reverse_geocode_pipelined (source_table_cursor IN SYS_REFCURSOR)
    RETURN sdo_geo_addr_table_type DETERMINISTIC PIPELINED PARALLEL_ENABLE
    (PARTITION source_table_cursor BY ANY) IS

    TYPE num_array_type IS TABLE OF NUMBER;
id_table                NUM_ARRAY_TYPE;
geometry_table   SDO_GEOMETRY_ARRAY;
geo_addr             SDO_GEO_ADDR;
begin                SDO_GEO_ADDR;
  LOOP
    FETCH source_table_cursor
    BULK COLLECT INTO id_table, geometry_table
    LIMIT 100;

-- Exit when no rows were fetched.
EXIT WHEN id_table.count = 0;
FOR i IN id_table.first .. id_table.last LOOP
    geo_addr := SDO_GCDR.REVERSE_GEOCODE('NAVTEQ_SF', geometry_table(i), 'US');
    geo_addr.id := id_table(i);
    PIPE ROW (geo_addr);
END LOOP;
END LOOP;

CLOSE source_table_cursor;
RETURN;
end;
/

-- See “Parallel Enabled Pipeline Table Function” slide earlier in presentation that
-- demonstrates how to call a similar function for geocoding.
EHCC Line Geometry Code Example
Appendix
-- Create an empty table, compressed for query low
CREATE TABLE edge_ql (edge_id NUMBER, geometry SDO_GEOMETRY)
    COMPRESS FOR QUERY LOW;

DECLARE
    TYPE ID_TAB_TYPE        IS TABLE OF number;
    TYPE GEOM_TAB_TYPE IS TABLE OF sdo_geometry;

    id_tab       ID_TAB_TYPE;
    edge_tab GEOM_TAB_TYPE;

    CURSOR get_edge_data_cur IS SELECT edge_id, geometry FROM edge;
BEGIN
-- This example reads data from an uncompressed table, 
-- and writes to a compressed table
OPEN get_edge_data_cur;
LOOP
  FETCH get_edge_data_cur BULK COLLECT INTO id_tab, edge_tab LIMIT 1000;
  EXIT WHEN edge_tab.count = 0;
  FORALL i IN edge_tab.first .. edge_tab.last
    INSERT /*+ append_values */ INTO edge_ql VALUES (id_tab(i), edge_tab(i));
  COMMIT;
  EXIT WHEN get_edge_data_cur%NOTFOUND;
END LOOP;
CLOSE get_edge_data_cur;

END;
/

-- Size uncompressed
SELECT sum(bytes)/1024/1024
FROM user_extents
WHERE segment_name = 'EDGE';

-- Size compressed with COMPRESS FOR QUERY HIGH
SELECT sum(bytes)/1024/1024
FROM user_extents
WHERE segment_name = 'EDGE_QH';