Does Exadata Need Performance Tuning?

Jože Senegačnik, Oracle ACE Director, Member of OakTable
DbProf d.o.o.
Ljubljana, Slovenia

Keywords
Exadata, Cost Based Optimization, Statistical Optimizer, Physical I/O, direct path read

Introduction
With new engineered systems produced by Oracle like EXA* there were also a lot of rumors coming out which may be sometimes true, sometimes completely wrong and sometimes could be partly true. So the rumor I heard was that on Exadata box one should drop all indexes as Exadata does not need indexes, actually it runs much better without them. True or false? The reader will get the answer later on in this paper.
The other rumor which I heard is that Exadata does not need performance tuning. For this one I was completely sure at the moment of hearing it that it is completely wrong. So let us discuss the answer to this one at the end of this paper.
But let us first start with some funny stuff – comparing my laptop with Exadata box. Why so silly comparison as everyone knows that Exadata will beat my laptop in any manner. True? Let us see.
Here is a simple test which I am using probably since version 7 of Oracle database for testing the speed of CPU and memory access on different hardware platforms.

- My laptop

```
SQL> with a as (select /*+ materialize */ 1 from dual connect by level <= 14)
  2  select count(*) from a,a,a,a,a,a,a;

  COUNT(*)
----------
 105413504

Elapsed: 00:00:04.77
```

- Exadata

```
SQL> with a as (select /*+ materialize */ 1 from dual connect by level <= 14)
  2  select count(*) from a,a,a,a,a,a,a;

  COUNT(*)
----------
 105413504

Elapsed: 00:00:04.48
```

Figure 1: Laptop and Exadata Comparison

The above test performs practically no physical I/O, so all Exadata features will be excluded. Also on any other box this test will show only how fast the CPU and memory access are. Running the same test on non-virtualized system and virtualized system will show how much of the performance is lost due to virtualization (you might be surprised when seeing such results).
As the reader can see the difference is actually about 0.3 seconds what is really very small. If we would run the test several times more the difference could be even smaller. So obviously the database

running on Exadata has no significant better performance than the database of the same version on my laptop.

So where is hidden the secret for power of Exadata?

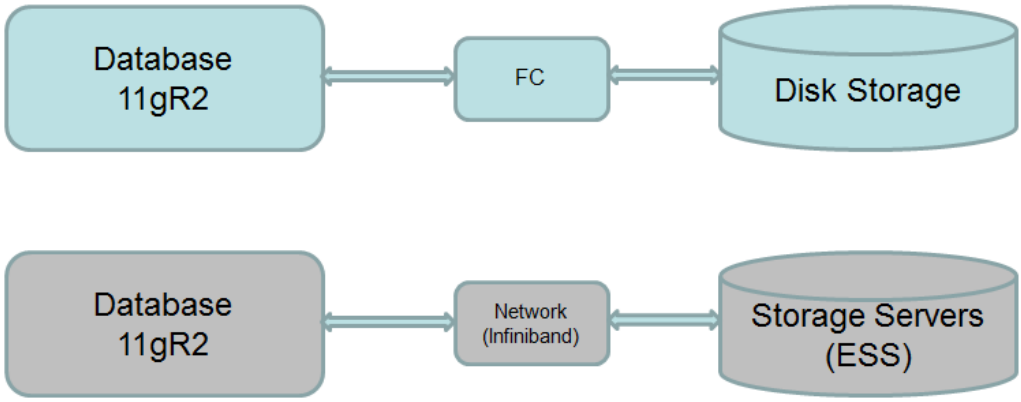

Figure 2: Different Configuration in Exadata

From Figure 2 we can see that the secret power lies in extremely fast communication between storage cells (storage servers) and database engine and on the disk storage side we have some storage cell software which perform some of the read operations very very fast and not only that, it can partially process the data as well.

So armed with this knowledge we can see that the benefits are mainly coming from the reduced time required for read, reduced amount of data transferred (because of pre-processed data) and reduced time to transfer data between the database and storage cells.

But the crucial question is: are those special Exadata features always used, can we force their usage, how we really know what is going on when the query is executed.

Here is list of the most important features which are performed by a storage cell and are part of so called smart scan:

- Column projection
- Predicate filtering
- Storage indexes
- Simple Joins
- Function Offloading
- Smart Flash Cache
- Virtual Column Evaluation
- Hybrid Columnar Decompression
- Decryption

The purpose of "**Column Projection**" is to return as the result only the referenced columns in the select list therefore the result set is already partially processed. Actually not only the columns from the select list but also the columns used in join operations.

```
SQL> select empno, ename from employees;
```

However, a smart scan operation still needs to read all table data but only the listed columns are returned by a smart scan operation into the session's PGA.

The "**Predicate Filtering**" operation returns only those rows which match the filter. So the result set might be significantly reduced. However, a smart scan operation still needs to read all table data.

The "**Storage Indexes**" are special kind of in-memory indexes which are built on the fly by a cell server and are built on columns used in the WHERE clause. Storage indexes are used for fast determination in which parts of the table data is present / not present. Storage indexes can significantly reduce the amount of data read from disk. As they are built as min/max value for 1MB extents they are highly dependent on the data distribution.

Smart Scan Prerequisites
Let us start with some of the most important facts – with the requirements for the smart scan operation.

In order that a smart scan operation is performed the following must be true:
- Full Table Scan / Fast Full Index Scan operation must be requested (in execution plan)
- Direct Path Read operation must be performed

The direct path read operation is actaullay a physical read in which the blocks that are read are passed to the PGA and completely bypass the SGA (buffer cache) and therefore no blocks are read from the buffer cache.

The runtime engine decides if the direct path read operation will be used. In Oracle 11g there have been changes in the heuristics which chooses between direct path reads or reads through the buffer cache (classical full table scan operation) for serial table scans. Since 11g, this decision to read via direct path or through cache is based on the size of the table, buffer cache size and various other stats. Important to notice is that direct path reads are faster than scattered reads and have less impact on other processes because they avoid latches.

The runtime engine will preform direct path read in these cases:
- Parallel Queries
- For all full table / fast full index scan operations if we set hidden parameter "**_serial_direct_read**" = TRUE
- Since Oracle 11g when the query is performed on a "big" table. The "size" of the table is determined by:
    - How many table blocks are cached in the buffer cache at the execution time
    - Beyond certain percentage a table may not qualify for a smart scan
    - Storage server decide whether a smart scan will be performed, not the CBO

If we carefully look at the above constraints we can conclude that the bigger the buffer cache is, the smaller your tables will appear. So a smaller buffer cache means more smart scans!

The direct path read decision is influenced by
- Type of read (FTS or FFIS)
- Size of segment (> 5 * _small_table_threshold)
- Number of blocks cached (~ 50%)

For details about **direct path read** wait event, please, see MOS note 793845.1.

Now as we know more about the smart scan operation we can discuss the rumor about dropping indexes on Exadata. The answer is that for OLTP environment you will probably still need indexes for index access of a single row (PK) or relatively small number of rows when the access via index is cheaper. Although Exadata is fast, repetitive full scans on relatively big tables can bring even Exadata platform to the knees. When index access is in question one should check the amount of work performed by Smart Scans in comparison with the amount of work performed by index access. Dropping indexes, especially those that are never used, is beneficial as they are slowing down DML operations (index maintenance costs – insert, update, delete operations).

Was A Smart Scan done or better were Exadata features used?

The most important question for anybody using Exadata is if the secrete sauce – the smart scan – was performed for a particular statement. In order to determine this we need to check the SQL statement level statistics gathered in (G)V$SQL view. The column IO_CELL_OFFLOAD_ELIGIBLE_BYTES stores the statistics about possible offload, actually the amount of IO saved by Smart Scan. If the value of this column is bigger than 0 then the smart scan was performed.

Here is the SQL statement which can be used to check this:

```
select last_load_time, sql_id, child_number,
          decode (IO_CELL_OFFLOAD_ELIGIBLE_BYTES,0,'No','Yes') Offloadable
from v$sql
where sql_text like '%&sql_text%'
order by last_load_time desc;
```

The smart scan operation can also use the storage indexes for further performance improvement. In 11g the session level statistics named ' cell physical IO bytes saved by storage index ' is available to determine the amount of reduced physical reads due to storage index usage.

```
select s.name, m.value
from v$statname s, v$mystat m
where name like (' cell physical IO bytes saved by storage index ')
and s.statistic# = m.statistic#
```

One should be aware of the fact that this statistics is cumulative and will be increased whenever storage indexes were used. So we have to obtain the value of this statistic before and after the execution of SQL statement in order to get the right value.

Here is a more complete SQL statement used:

```
SQL> select decode(name,
  2  'cell physical IO bytes saved by storage index', 'SI Savings',
  3  'cell physical IO interconnect bytes returned by smart scan', 'Smart Scan')
         as stat_name,
  4  value/1024/1024 as value
  5  from v$mystat s, v$statname n
  6  where s.statistic# = n.statistic#
  7  and n.name in ('cell physical IO bytes saved by storage index',
  8   'cell physical IO interconnect bytes returned by smart scan');

STAT_NAME       VALUE
---------- ----------
SI Savings          0
Smart Scan 1055.22408

Elapsed: 00:00:00.00
```

The percentage of I/O saved can be calculated with the following statement (source Kerry Osborne):

```
select sql_id,
       decode(io_cell_offload_eligible_bytes,0,'no','yes') offloaded,
       decode(io_cell_offload_eligible_bytes,0,0,
       100*
       (io_cell_offload_eligible_bytes/io_interconnect_bytes)/
       io_cell_offload_eligible_bytes) as  "io_saved"
from v$sql
where sql_text like '%&sql_text%';
```

Predicate filtering
One of very important features of the smart scan is also so called "predicate filtering". The storage server evaluates the query predicates and thus only rows, which pass the predicates are returned from storage server to the PGA. In case that we don't achieve smart scan all required rows (blocks) are read by database and predicates are applied later on. In case of smart scan the result is already at least partially processed, less memory needed in PGA (as result is smaller) and therefore less work is performed at the database level what all improves the performance.

Offloadable functions
Some of the built-in SQL functions can be offloaded, some of them not. The list of offloadable functions can be found by querying **v$sqlfn_metadata** view (result from 11gR2).

```
SQL> select OFFLOADABLE, count(*)
   2 from v$sqlfn_metadata group by OFFLOADABLE;

OFF   COUNT(*)
--- ----------
NO          530
YES         393
```

When the function is not offloadable the data is returned to the SGA as in traditional systems
Functions used in where clause are executed for every single row (at partucular step of the execution plan), so offloading is very beneficial to filter out as many rows as possible already during the smart scan operation.

Some more details about Storage Indexes
The purpose of storage indexes is to eleiminate I/O operations. Actually this is the only Exadata feature which really eliminates physical I/O.

Storage indexes are built automatically by the cell servers (11g) for a maximum of 8 columns per table. Unfortunately there is no documented way to alter or even more important to tune them.

Internally the storage index stores the min and max column values for disk storage units (1MB by default). During the query the storage units, which don't contain the requested values, are skipped. Storage indexes can thus boost the performance if the column value(s) is/are present only in relatively small number of storage units. As they are highly dependent on data distribution, they are very effective on sorted or partitioned data.
Automatically maintained storage indexes are problematic because they are created on the fly and may or may not be present so the response time for a particular query can significantly fluctuate.

The requirements for storage index usage are: must be a smart scan operation, "where clause" with at least one predicate with a simple comparison operator (=, <, >, BETWEEN, IS (not) NULL, etc.). They can be used in multi-column predicates, joins, parallel queries, partitions.

However storage indexes are not used: for not equals (!=) operators, on LOB's / CLOBs type columns, on column encryption, wildcards, where clauses with subqueries, clustered tables, index-organized tables (IOT).

Conclusion

Exadata was built in order to boost the performance of some of the database operations which could be very time consuming. On large databases the most time consuming operation is physical read which is on Exadata offloaded to storage cells. However, in order to achieve offloadable operation, some prerequisites must be met otherwise the storage cell server is acting like "normal" disk array performing full table scan operation and returning data to the buffer cache.

As Exadata is fast we can't conclude from the timing information if some or all features of Exadata were actually used. So for every single SQL statement we should check if we were really using those wonderful features. In our presentation we learned how to check if the query was really offloaded to the storage cells and if a storage index was used.

In case that the features were not used we have to find out why the query was not offloaded. The size of the buffer cache and the size of the table might play a big role in the process of determining if smart scan will be used. As the "direct path read" operation is required for smart scan we can even force its usage by setting the parameter "**_serial_direct_read**" = TRUE.

As with any other SQL statement execution the prepared execution plan might be sub-optimal and therefore we need to tune it. Just the fact that the smart scan operations boost the performance of the query doesn't mean that we don't need to tune the queries on Exadata platform. So tuning is still needed on Exadata platform.

Kontaktadresse:
Jože Snegačnik
DbProf d.o.o.
Smrjene 153
SI-1291 Škofljica
Slovenia

Telefon: +386 41 72 44 61
E-Mail joze.senegacnik@dbprof.com
Internet: www.dbprof.com