How to Refresh a Materialized View in Parallel

Materialized views provide performance benefits to data warehouse applications. Some OLTP applications also benefit from materialized views involving non-volatile data. Oracle provides flexible ways to refresh materialized views – you can refresh them full or incremental; you can refresh them on demand or at the commit time in the source table. When the size of the materialized view grows, one needs to explore ways to perform the refresh faster. One of the ways to expedite the refresh is to use parallel execution.

Refreshing a Materialized View

The attributes related to refreshing a materialized view are specified at the creation time of the materialized view (or later through the ALTER MATERIALIZED VIEW statement). The two most important attributes that impact the refresh time are:

- Refresh FAST or COMPLETE
- Refresh ON COMMIT or ON DEMAND

A FAST refresh means an incremental refresh, which indicates that the materialized view will be refreshed based on only the data changes that have occurred in the master table(s). A COMPLETE refresh indicates that the materialized view will be refreshed by re-executing the query defining the materialized view.

The ON COMMIT refresh indicates that the materialized view will be refreshed whenever the data in the master table changes. The ON DEMAND refresh indicates that the materialized view will be refreshed on demand by explicitly executing one of the REFRESH procedures in the DBMS_MVIEW package.

In this article, we will consider the ON DEMAND COMPLETE refresh of a materialized view. However, the concepts discussed here will be applicable to all refresh methods. In this article, the materialized view will be based on the following table:

<table>
<thead>
<tr>
<th>DESC</th>
<th>SALES_HISTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>-----------------</td>
</tr>
<tr>
<td>PART_ID</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>STORE_ID</td>
<td>-----------------</td>
</tr>
<tr>
<td>SALE_DATE</td>
<td>-------------</td>
</tr>
<tr>
<td>QUANTITY</td>
<td>-----------------</td>
</tr>
</tbody>
</table>

We will use the following materialized view to illustrate various parallel refresh mechanisms.

```
CREATE MATERIALIZED VIEW MV_PART_SALES
AS
SELECT PART_ID, SALE_DATE, SUM(QUANTITY)
FROM SALES_HISTORY
GROUP BY PART_ID, SALE_DATE;
```
Parallel Execution
Parallel execution enables multiple processes to work simultaneously to refresh the materialized view, resulting in speeding up the refresh process. Before you enable parallel execution in your database, you should ensure that you have enough resources (CPU and Memory) to run multiple processes in parallel. Once you decide to use parallel execution, you should set the initialization parameter PARALLEL_AUTOMATIC_TUNING to TRUE. This enables Oracle to perform automatic tuning of the parallel execution environment.

Now that you have enabled parallel execution in the database, you can employ it while refreshing the materialized view. In the following sections, we will discuss various ways to invoke parallel refresh.

- The PARALLELISM parameter of the DBMS_MVIEW.REFRESH procedure
- PARALLEL attribute of the materialized view
- PARALLEL attribute of the master table(s)
- PARALLEL hint in the query defining the materialized view

The PARALLELISM Parameter of the DBMS_MVIEW.REFRESH Procedure
The REFRESH procedure of the supplied package DBMS_MVIEW can be used to refresh a materialized view. Among other parameters, this procedure takes one parameter PARALLELISM. You can invoke this procedure as:

EXECUTE DBMS_MVIEW.REFRESH(LIST=>'MV_PART_SALES',PARALLELISM=>4);

At the outset, it appears that the PARALLELISM parameter will invoke a parallel refresh of the materialized view. However, it doesn’t. The dynamic performance views V$PX_PROCESS and V$PX_SESSION provide information on the parallel execution processes and the sessions using parallel execution respectively. When the above refresh process is running, if we query these views, we will see that there are no parallel execution processes in use.

PARALLEL Attribute of the Materialized View
Now let’s create the materialized view with the PARALLEL attribute, and investigate the refresh behavior.

CREATE MATERIALIZED VIEW MV_PART_SALES
PARALLEL 4
AS
SELECT PART_ID, SALE_DATE, SUM(QUANTITY)
FROM SALES_HISTORY
GROUP BY PART_ID, SALE_DATE;

EXECUTE DBMS_MVIEW.REFRESH(LIST=>'MV_PART_SALES',PARALLELISM=>4);
EXECUTE DBMS_MVIEW.REFRESH(LIST=>'MV_PART_SALES');

When the materialized view is created with the PARALLEL clause, the creation process is parallelized, but the refresh process is not. Whether or not you specify the PARALLELISM parameter in the REFRESH clause, it doesn’t matter. The refresh will be executed in serial.

PARALLEL Attribute of the Master Table
Now let’s examine the impact of the PARALLEL attribute of the master table, instead of the materialized view. We will alter the master table to set the PARALLEL attribute to 4, and then create the materialized view without a PARALLEL clause.
ALTER TABLE SALES_HISTORY PARALLEL (DEGREE 4);

DROP MATERIALIZED VIEW MV_PART_SALES;

CREATE MATERIALIZED VIEW MV_PART_SALES
AS
SELECT PART_ID, SALE_DATE, SUM(QUANTITY)
FROM SALES_HISTORY
GROUP BY PART_ID, SALE_DATE;

EXECUTE DBMS_MVIEW.REFRESH(LIST=>'MV_PART_SALES', PARALLELISM=>4);
EXECUTE DBMS_MVIEW.REFRESH(LIST=>'MV_PART_SALES');

When the master table of the materialized view has the PARALLEL attribute set to > 1, then the creation as well as the refresh processes will be parallelized. Whether or not you specify the PARALLELISM parameter in the REFRESH clause, it doesn’t matter.

PARALLEL Hint in the Query Defining the Materialized View
Now let’s examine the refresh behavior by putting a parallel hint in the materialized view definition.

CREATE MATERIALIZED VIEW MV_PART_SALES
AS
SELECT /*+ PARALLEL(SALES_HISTORY, 4) */
    PART_ID, SALE_DATE, SUM(QUANTITY)
FROM SALES_HISTORY
GROUP BY PART_ID, SALE_DATE;

EXECUTE DBMS_MVIEW.REFRESH(LIST=>'MV_PART_SALES', PARALLELISM=>4);
EXECUTE DBMS_MVIEW.REFRESH(LIST=>'MV_PART_SALES');

When the materialized view is created with a PARALLEL hint, then the creation as well as the refresh processes will be parallelized. Whether or not you specify the PARALLELISM parameter in the REFRESH clause, it doesn’t matter.

Conclusion
In this article we discussed various ways to refresh a materialized view in parallel. The important thing to note here is that the PARALLELISM parameter of the REFRESH procedure doesn’t help in invoking parallel refresh of a materialized view. You can invoke parallelism while creating a materialized view by specifying a PARALLEL clause in the create materialized view statement, or by using a PARALLEL hint in the create materialized view statement, or by specifying the PARALLEL attribute for the master table(s). To invoke parallelism while refreshing a materialized view you have one less option. You can do it either by using a PARALLEL hint in the create materialized view statement, or by specifying the PARALLEL attribute for the master table(s).
Sanjay Mishra has more than 12 years of industry experience, and has extensively worked in the areas of database architecture, database management, performance tuning, scalability, ETL, backup / recovery, parallel server and parallel execution. He has coauthored 3 Oracle books published by O'Reilly & Associates (Mastering Oracle SQL, Oracle SQL Loader: The Definitive Guide, Oracle Parallel Processing). He has published articles in the Oracle Magazine, SELECT Journal and dbazine.com. Presently, he works as the lead database architect at Dallas based i2 Technologies. Sanjay can be reached at smishra_tech@yahoo.com.