Plan Stability, Baselines, and SQL Plan Management

Arup Nanda

Starwood Hotels and Resorts

(c) 2009 Arup Nanda
About Me

• Oracle DBA for 16 years and counting
• Speak at conferences, write articles, 4 books, provides trainings
• Brought up the Global Database Group at Starwood Hotels, in White Plains, NY

(c) 2009 Arup Nanda
What you will learn

• What is SQL Plan Management
• What is a Baseline
• Using baselines to stabilize the plan
• How to enable/disable usage of baselines
Meet John the DBA

• John is a DBA at Acme Corp
• Honest, hardworking, highly experienced
• But not politically savvy; doesn’t beat around the bush. straight shooter
• Let’s see some scenarios he faced in the job
Third Party Tool

• Acme bought a third party gee-whiz tool
• The performance was terrible
• *John* was asked to explain why!
• He analyzed and determined the cause: bad optimizer plans
• He suggested putting hints to fix the plans
• But no, he can’t. The source code is not accessible, remember?
• Status: still unresolved and John is to blame!
Analyzer Gone Wild

- John collects optimizer stats every day
- One day performance went south, apps timed out
- On analysis he found that the plan of those queries had changed
- The plan changed b’coz of the new stats
- *John* got blamed for the fiasco
Stale Stats

• John heard somewhere that stats should not be collected everyday
• He decided to stop collecting stats
  – did so only occasionally
• One day performance went south
• Cause: Optimizer Plan was bad
• Reason: stale stats
• He was blamed!
Database Upgrade

• John wanted to upgrade a DB from 10g to 11g
• He was asked “can you guarantee that plans will not change”
• “Of course not”, he responded. “But most likely they will not”
• Upgrade completed
• Most plans were OK; some went south.
• *John* was blamed for that
Plan Changes

• A developer complains about performance
• John checks the plan and finds a bad plan
  – a full table scan, which should have been index scan
    or may be vice versa
• He asks the developer “is the data different”?
• “No”, comes the reply. “has been the same for 4 years”.
• John has no history of the plan
• Oracle is misbehaving – was the “root cause”
• Who do you think was blamed?
Optimizer Misbehaves

- Oracle Cost Based Optimizer sometimes does not produce most optimal plan
- Difficult to debug
- Well, John takes the blame for that as well!
Stored Outlines

• For inefficient plans, John does have a solution
• Outlines make a plan for a query \textit{fixed}
  – The optimizer will pick up the fixed plan every time
• Problem:
  – Based on the bind variable value, data distribution, etc. specific plan may change
  – A fixed plan may actually be worse
The Problem

• If optimizer calculates execution plans, it may produce inefficient ones
• If you use stored outlines, a fixed plan may be as inefficient as to be noticeable
• Can you have the best of both words?
  – Have plan fixed by outlines
  – But calculate the new plan anyway for comparison and use if appropriate
• Baselines do exactly that … and more
Quick Primer on Parsing

• When a query is submitted, Oracle performs the following:
  – Determines if there is a parsed statement
  – Parses query
    • Determines the objects being accessed
    • e.g. is EMP a table or a synonym
    • Determines if the user has privs on that object
    • Calculates the optimal execution plan
  – Binds the values to the variables
  – Stores the parsed statement in library cache
Statement Versions

**SCOTT**

```
SELECT * FROM EMP
WHERE SAL>1000
```

- EMP table in SCOTT schema
- No index on SAL column
- Will generate a Full table scan

**ARUP**

```
SELECT * FROM EMP
WHERE SAL>1000
```

- EMP table in ARUP schema
- Index on SAL column
- Will probably generate an index scan

Demo: cur_test.sql
A single SQL statement may have multiple plans associated with it.
A baseline is a collection of plans for a specific SQL statement.
A new plan was generated as a result of some change, e.g. the optimizer parameters were changed. This plan is added to the baseline.
When a SQL is reparsed, the optimizer compares the plan to the list of plans in the baseline, but **not the newly generated plan** as it is not “accepted”.

(c) 2009 Arup Nanda
A plan is no longer valid, e.g. it had an index scan; but the index was later dropped. It is marked as such.
New Plan is Worse

• Baselines contain the history of plans for an SQL statement
• If there was a good plan ever, it will be there in the baseline
• So the optimizer can choose the plan with the lowest cost

Cost = 10
Plan P1

Cost = 12
Plan P2

Cost = 9
Plan P3

New plan. Cost = 15
Plan P4

Baseline

Optimizer will choose P3 even though the new plan generated was P4

(c) 2009 Arup Nanda
New Plan is the Best

• Even if the new plan is the best, it will not be immediately used.
• The DBA can later made the plan fit for consideration by “evolving” it!

<table>
<thead>
<tr>
<th>Plan</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>12</td>
</tr>
<tr>
<td>P3</td>
<td>9</td>
</tr>
<tr>
<td>P4</td>
<td>New plan. Cost = 6</td>
</tr>
</tbody>
</table>

Baseline

Optimizer will choose P3 since it is the best in the list of “accepted” plans.
SQL Statement

New Plan Generated

any other accepted plans in baseline?

yes

Choose the best accepted plan

Use this best plan, not the new plan

no

Add this plan to the SMB

Baseline this plan but set to not Accepted

(c) 2009 Arup Nanda
SQL Management Base

• A repository where the following are stored
  – Statements
  – Plan histories
  – Baselines
  – SQL profiles
• Stored in SYSAUX tablespace
Configuring SMB

To Check

```sql
select parameter_name, parameter_value
from dba_sql_management_config;
```

<table>
<thead>
<tr>
<th>PARAMETER_NAME</th>
<th>PARAMETER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACE_BUDGET_PERCENT</td>
<td>10</td>
</tr>
<tr>
<td>PLAN_RETENTION_WEEKS</td>
<td>53</td>
</tr>
</tbody>
</table>

To Change:

```sql
BEGIN
    DBMS_SPM.CONFIGURE(
        'PLAN_RETENTION_WEEKS', 100);
END;
```
### DBA_SQL_PLAN_BASELINES

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNATURE</td>
<td>Unique identifier for the SQL, a number, e.g. 10925882130361959529</td>
</tr>
<tr>
<td>SQL_HANDLE</td>
<td>Unique ID in text form, e.g. SYS_SQL_97a087e8e6034469</td>
</tr>
<tr>
<td>SQL_TEXT</td>
<td></td>
</tr>
<tr>
<td>PLAN_NAME</td>
<td>Unique plan identifier, in text, e.g. SYS_SQL_PLAN_e603446911df68d0</td>
</tr>
<tr>
<td>ENABLED</td>
<td></td>
</tr>
<tr>
<td>ACCEPTED</td>
<td>NO - Disabled</td>
</tr>
<tr>
<td>FIXED</td>
<td>YES – Enabled</td>
</tr>
<tr>
<td>AUTOPURGE</td>
<td></td>
</tr>
<tr>
<td>OPTIMIZER_COST</td>
<td>Cost when the baseline was created</td>
</tr>
</tbody>
</table>

(c) 2009 Arup Nanda
More about baselines

• Plans in baselines stay even after the SQL is flushed from the shared pool
To Check Baselines

• Enterprise Manager
• Click on Server Tab
• Click on Plan Management
• Enter a Search String for the SQL and click Go
Baselines Demo

• Setup: spm_test1
• Table:
  SQL> select status, temporary, count(1)
        2  from accounts
        3  group by status, temporary;

<table>
<thead>
<tr>
<th>STATUS</th>
<th>T</th>
<th>COUNT(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALID</td>
<td>N</td>
<td>68416</td>
</tr>
<tr>
<td>INVALID</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>VALID</td>
<td>Y</td>
<td>138</td>
</tr>
</tbody>
</table>

• Query:
  select /* SPM_TEST */ * from accounts where status = 'INVALID' and temporary = 'Y'
To check for Plans in the baseline

```sql
select SQL_HANDLE, PLAN_NAME
from dba_sql_plan_baselines
where SQL_TEXT like 'SPM_TEST%'
/
```

<table>
<thead>
<tr>
<th>SQL_HANDLE</th>
<th>PLAN_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_SQL_4602aed1563f4540</td>
<td>SYS_SQL_PLAN_563f454011df68d0</td>
</tr>
<tr>
<td>SYS_SQL_4602aed1563f4540</td>
<td>SYS_SQL_PLAN_563f454054bc8843</td>
</tr>
</tbody>
</table>

SQL Handle is the same since it’s the same SQL; but there are two plans.
Checking Plans Being Used

Execution Plan

Plan hash value: 2329019749

| Id | Operation                      | Name           | Rows  | Bytes | Cost (%CPU) | Time     |...|    82   (0)| 00:00:01 |
|----|--------------------------------|----------------|-------|-------|-------------|----------|...|---------|----------|
| 0  | SELECT STATEMENT               |                | 17139 | 1690K| 588 (1)    | 00:00:08 |
| * 1| TABLE ACCESS BY INDEX ROWID    | ACCOUNTS       | 17139 | 1690K| 588 (1)    | 00:00:08 |
| * 2| INDEX RANGE SCAN               | IN_ACCOUNTS_01 | 34278 |       | 82 (0)     | 00:00:01 |

Predicate Information (identified by operation id):

1 - filter("TEMPORARY"='Y')
2 - access("STATUS"='INVALID')

Note

- SQL plan baseline "SYS_SQL_PLAN_51f8575d04eca402" used for this statement

This shows that a SQL Plan Baseline is being used.
To See Plan Steps in Baseline

• Package DBMS_XPLAN has a new function called display_sql_plan_baseline:

```sql
select * from table (
    dbms_xplan.display_sql_plan_baseline (sql_handle=>'SYS_SQL_4602aed1563f4540', format=>>'basic note')
)
```
Demo: Adding Baselined Plans

• Demo: spm_test2

    alter session set
    optimizer_capture_sql_plan_baselines = true

    /
    ...
    execute the query at least 2 times
    alter session set
    optimizer_capture_sql_plan_baselines = false

    /

• A plan is baselined when a SQL is executed more than once
Adding more plans

• Demo: spm_test3
• Change the optimizer parameter so that a new plan is generated
  
  \texttt{alter session set optimizer_mode=first_rows}

• Capture the plans for the baseline
• The new plan is stored in baseline but not “accepted”; so it will not be used by the optimizer
Evolve a Plan

• Make a plan as acceptable (only if it is better)

    variable rep CLOB
    begin
        :rep :=
            dbms_spm.evolve_sql_plan_baseline ( 
                sql_handle => 'SYS_SQL_5a8b6da051f8575d'
                , verify => 'YES'
            );
    end;
/

• Variable REP shows the analysis.
• Demo: spm_test4.sql
Check the use of new plan

• Demo: spm_test5
  alter session set
  optimizer_use_sql_plan_baselines = false
  – Check plan
  alter session set
  optimizer_use_sql_plan_baselines = true
  – Check plan
Fixing a Plan

- A plan can be fixed by:
  ```sql
  dbms_spm.alter_sql_plan_baseline (  
    sql_handle => 'SYS_SQL_5a8b6da051f8575d',  
    plan_name => 'SYS_SQL_PLAN_51f8575d04eca402',  
    attribute_name => 'fixed',  
    attribute_value => 'YES'  
  )
  ```

- Once fixed, the plan will be given priority
- More than one plan can be fixed
- In that case optimizer chooses the best from them
- To “unfix”, use `attribute_value => 'NO'`
Capturing Baselines in Bulk

• Setting system parameter
  ```sql
  alter system set optimizer_capture_sql_plan_baselines = true
  ```

• Capture from Cursor Cache
• Capture from SQL Tuning Set (STS)
• Convert from Stored Outlines (11gR2)
Capturing from Cursor Cache

declare
	cnt number;
begin

cnt := dbms_spm.load_plans_from_cursor_cache
    (sql_id => '003vmga5rcrs4');

cnt := dbms_spm.load_plans_from_cursor_cache
    (sql_id => '005nuc1nd7u93');

cnt := dbms_spm.load_plans_from_cursor_cache
    (sql_id => '009su850aqyha');
end;
Capturing from Cursor Cache

```
declare
cnt number;
bEGIN
  cnt :=
    dbms_spm.load_plans_from_cursor_cache(
      attribute_name => 'sql_text',
      attribute_value => '%SPM_TEST%'
    );
end;
```
Capturing from STS

declare
    cnt  number;
begin
    cnt := dbms_spm.load_plans_from_sqlset(
        sqlset_owner => 'SYS',
        sqlset_name => 'TEST_STS',
        basic_filter => '%SPM_TEST%'
    );
end;
Create STS

declare
    l_task_name varchar2(2000);
beginn
    l_task_name :=
        dbms_sqltune.create_tuning_task (sql_id => '7zpphmzu2m1j6');
e nd;
/

How else can you tune a query

You can also use SQL Tuning Advisor

1. Create a tuning task
   
   ```
   variable l_task varchar2(2000)
   exec :l_task :=
   dbms_sqltune.create_tuning_task(
     sql_id => 'cbynbmssqudbx');
   ```

2. Execute the task
   
   ```
   exec dbms_sqltune.execute_tuning_task(
     task_name => :l_task)
   ```
3. Check for recommendations

```sql
select dbms_sqltune.report_tuning_task(
    :l_task, 'TEXT', 'BASIC') FROM dual;
```

4. If there is a SQL Profile, accept it

```sql
exec dbms_sqltune.accept_sql_profile(
    task_name => :l_task);
```

This will add the tuned plan as per SQL Profile to the baseline as well.

- So you can use either Evolve or STA for creating baselined plans
Use of Baselines

- Fixing Plan for Third Party Applications
- Database Upgrades
  - Both within 11g and 10g->11g
  - Capture SQLs into STS then move the STS to 11g
- Database Changes
  - Parameters, Tablespace layout, etc.
  - Fix first; then gradually unfix them
Use of SMB

- SQL Management Base is a historical repository of SQLs and associated plans
- The plan exists even though SQL is flushed out of memory
## Let’s Revisit John’s Issues

<table>
<thead>
<tr>
<th>Issue</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad plans in 3(^{rd})-party apps</td>
<td>Change opt. env.; generate new baselines and fix them</td>
</tr>
<tr>
<td>Optimizer misbehaving</td>
<td>Won’t happen since the bad plans will not be in baseline</td>
</tr>
<tr>
<td>Stats collection causing bad plans</td>
<td>But he can check them later and evolve them if good</td>
</tr>
<tr>
<td>Upgrade breaking good plans</td>
<td>Get all the plans in STS and accept all of them</td>
</tr>
<tr>
<td>Developers not aware of plan changes</td>
<td>Query the SQL Management Base</td>
</tr>
</tbody>
</table>

(c) 2009 Arup Nanda
Thank you!