OWB Tuning

Methods to tune OWB mappings and flows that leverage your existing Oracle tuning expertise!

Need - Common Situation

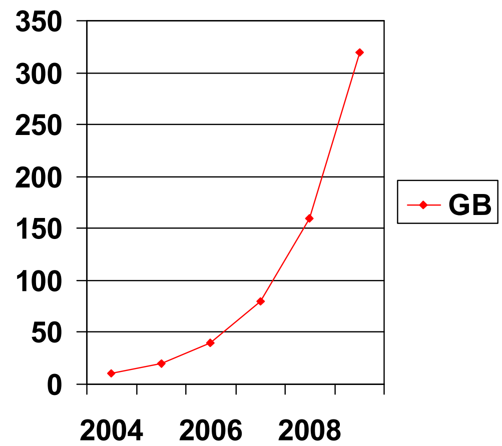
- System loads 100k records daily with no apparent issues (2 hour load time)
- 110k records = 4 hour load time
- 120k records = 12 hour load time
- 130k records = **MAJOR PROBLEM**

Most professionals need little convincing that building well tuned OWB code is of benefit to the enterprise. Administrators and DW Developers often face circumstances like the one enumerated above.

In low volume situations “tuned” OWB code is less critical because the system is not adversely affected by inefficient application algorithms and storage structures. If an organization is not already involved in an ongoing practice of tuning, the point at which it becomes necessary to address the performance of OWB code is when it affects the service level.

Need - General

- DW deals with some of the largest volumes of data in enterprise
- Data volume is growing, always.
- Efficient and effective code is not just desirable, it is *crucial*



The typical hockey puck of data volume growth is documented well by industry associations, analysts, and our own experiences within our enterprises. The growth of data is very real and it is likely that the volume for your data warehouse will be 4 times what it is today 3 years from now (18mo data volume growth average, analyst)

The data warehouse has to face these issues head on. OLTP systems can limit windows, offload data, etc. OLTP systems typically only have to deal with the "current" growth. OLTP systems have to be able to handle volume = 4 times today, 3 years from now. However, the DW environment has to deal with the increased volume cumulatively which compounds the issue. In other words, you have to pay attention and expect the hockey puck.

In a DW environment, tuning is not just a good practice, it is crucial to meet the actual requirements of projects (daily loads completing before start of business).

What is OWB Tuning?

- The process of
  - Measuring
  - Analyzing
  - Diagnosing
  - Revising
- OWB solutions to
  - Decrease loading times
  - Increase efficiency

bayon technologies

Measuring : We'll measure clock time for mappings, throughput figures, algorithm growth.

Analyzing : We'll learn how generate diagnostic information, piece it together, and leverage our existing knowledge of Oracle tuning.

Diagnosing : We'll learn how to use all our available information to determine a root issue, or possible solutions to **decrease loading times or increase efficiency**.

Revising : We'll learn how to navigate the challenging task of tuning code we can't adjust directly. OWB code is generated, so we'll learn how to do our best to tune PL/SQL that we have only indirect control over.

OWB Tuning Basics

- Goal = Tuned PL/SQL / Data Access
- OWB Tuning Methods provide:
  - Monitoring, Measurements, and Diagnostic information
- OWB Tuning Methods augment existing Oracle tuning methods and expertise
  - OWB Tuning = (Oracle tuning)++

Ultimately the goal is to entice OWB to produce tuned Oracle PL/SQL and data structures. This is our only method to improve our DW performance outside of purchasing additional hardware/software (and even then it might not improve).

OWB Tuning methods described in this module are meant to only be a “portion” of the overall tuning practice. There are literally hundreds of books by just as many authors that put forth methods on how to tune Oracle, Data Structures, SQL, and PL/SQL. Oracle tuning as a subject warrants an entire course in and of itself, and will not be covered here today.

OWB Tuning methods are methods to provide common diagnostic information so that one can leverage their existing Oracle tuning expertise and knowledge.

Two General "Methods"

1. OWB Runtime Data Analysis
   - Macro level, elapsed time, throughput, growth of time, etc.
2. Mapping/Process Flow Analysis
   - Tracing specific mappings/process flows, explain plans, use of idxes, etc.
     - *Mark Rittman/Jon Mead with SolStonePlus developed the fundamentals of this method!*

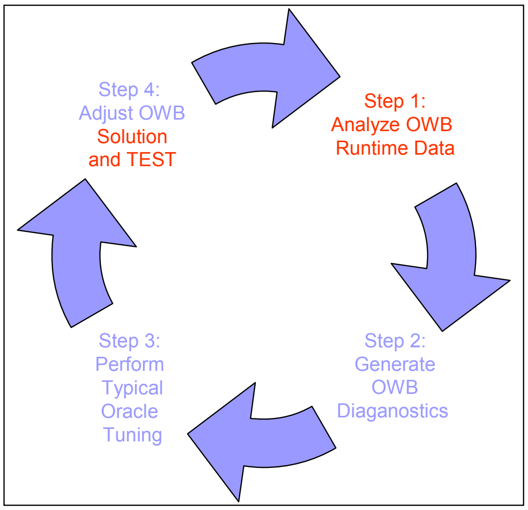
Method 1: OWB Runtime Data Analysis

This involves examination in greater detail the runtime audit information provided by the OWB runtime engine. The web based interface provides only per-run reporting, and there is currently not any Oracle provided interface for analyzing the data in the Runtime Repository. We'll examine some of the data available as Oracle views, and we'll understand what information is available to us. It provides performance data on a macro level, and does not provide detailed diagnostic information (an explain plan for instance).

Method 2: Mapping/Process Flow Analysis

We'll learn how to set our mappings and process flows up to generate very detailed information about how Oracle is processing our logic, and an entire wealth of information used for common Oracle tuning practices. This includes being able to "explain plan," receiving information on wait events, etc. We'll learn how to implement these diagnostics in our mappings and process flows along with how to collect and assemble them from our Oracle server.

Overall Methodology



- Step 1: Determine candidates for tuning
- Step 2 : Generate diagnostic information on problem some mappings/flows
- Step 3 : Use diagnostic information as part of Oracle tuning
- Step 4 : Adjust OWB mapping and measure improvement
- RED = OWB Runtime Data Analysis
- BLUE = Mapping/Process Flow Analysis

OWB Data Analysis

- Based on “Wall Clock” analysis
- Sample of actual runs
- Relatively fine level of detail
- Unobtrusive

bayon technologies

OWB Data analysis is the starting point for OWB tuning. The OWB runtime repository keeps track of a significant amount of runtime data that can provide information on throughput, volume/processing time, processing over time (degradation), etc. The OWB runtime repository provides public views that can be accessed using SQLPlus so you can write whatever reports you require.

Wall Clock analysis means that nearly all the information we’ll be able to obtain is based on actual time, or more precisely elapsed time. It does not express any metrics such as CPU time, wait time, I/O wait time, etc. We’re receiving information on the actual time mapping x takes (1000 seconds for 20k records).

It’s based on actual runs of the mapping under real runtime conditions. It’s a true sample of the actual performance of your OWB code, and should definitely be used to document effectiveness. Unlike the Mapping/Flow tuning method, the OWB data analysis has zero impact on your application and is entirely unobtrusive. You can do OWB Data Analysis on your development, test, and PRODUCTION systems.

OWB Data Analysis GOALS

- Establish baseline
- Monitor ongoing performance
- TUNING GOALS
  - Identify Tuning Candidates
  - Quantify ROI for tuning work
- Measure improvement

Knowing what you are beginning with is important; if you don't actually know how long a mapping is taking then you'll have no way of knowing if you've made any improvements. Monitoring the ongoing performance through the views provides a benefit to administrators so they can understand in greater detail how long their loads are taking. For instance, an operator will almost always know the total load time (DW load kicks off at 1am, done by 7am). Seems like there's some significant room for improvement since the load is taking 6 hours; however, perhaps most of that time is spent retrieving over a slow DBLink one large table over the WAN? This kind of insight into the granular performance is quite necessary and useful.

Our goals with tuning in mind are: identify potential candidates for tuning and quantify the benefit of tuning work.

Identifying tuning candidates involves identifying mappings/flows that are taking large amounts of time, or are showing a pattern of performance degradation. If you run 100 mappings, but 85% of your load time is spent (on average) in just 3 mappings you should consider tuning those mappings. If mapping x is taking 4 times as long today with twice as much data it is showing that this particular mapping is not scaling linearly with data volume and should be considered a candidate for tuning.

Quantifying the ROI for tuning is important to ensure a positive return on time and resources employed. Improving a mapping with an average load time of 1 minute improves by 50% has limited value (30 seconds probably won't be noticed). Improving a mapping with an average load time 2 hours by 20% provides a significant better investment in time and resources (24 minute improvement).

Lastly OWB data analysis can help you measure what actual performance improvements you delivered. This is both useful in development systems (make a change, run the mapping, sample the new method) and also in production (new mapping saves warehouse on average 28 minutes). Good for job security and making the business case for performance tuning.

Views Look Like?

- ALL_RT_AUDIT_EXECUTIONS
  - execution_audit_id
  - execution_name
  - object_name
  - elapse_time
  - updated_on
- ALL_RT_AUDIT_MAP_RUNS
  - execution_audit_id
  - map_name
  - start_time
  - elapse_time
  - number_records_inserted

A full description of the views and data is available in the "**Warehouse Builder Public Views"** Appendix in the OWB users guide.

http://download-west.oracle.com/docs/html/B12146_01/d_pub.htm#sthref3826

Tuning Candidates

```
select
    OBJECT_NAME name,
    trunc(min(elapse_time)) min,
    trunc(max(elapse_time)) max,
    trunc(sum(elapse_time)) sum,
    trunc(avg(elapse_time)) avg,
    count(elapse_time) count
    from all_rt_audit_executions
    where 1=1
    AND task_type = 'PLSQL'
    AND created_on >=
    to_date('12/10/2004',
    'MM/DD/YYYY')
    group by OBJECT_NAME
    order by avg desc;
```

NAME	MIN	MAX	SUM	AVG
map1	0	31988	165372	20671
map2	5494	68905	135111	19301
map3	1672	3509	20542	2567
map4	316	3511	14502	1812
map5	1018	2170	13089	1636
map6	436	1353	6784	848
map7	478	1272	6476	809
map8	309	2243	5351	668

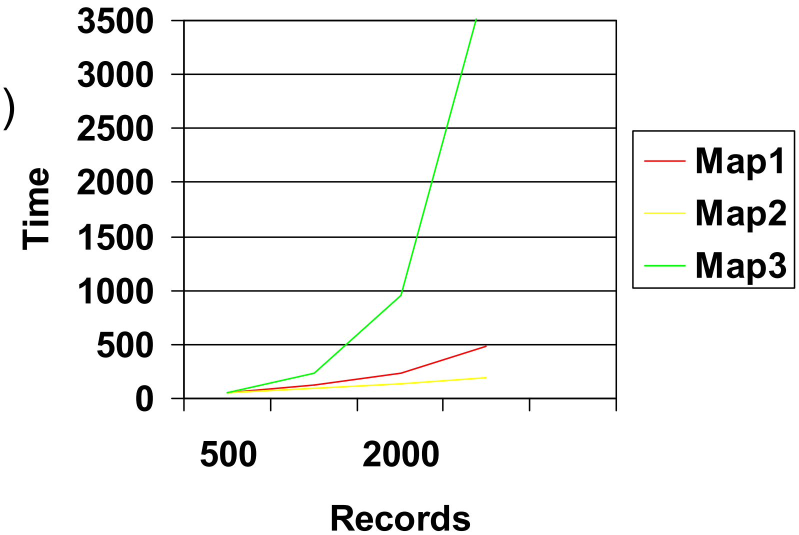
bayon technologies

An example report that shows the top candidates for tuning based on their average amount of elapsed time. Notice that map1 and map2 require nearly 10x as much time to complete as any other mappings. Map1 and map2 are prime candidates to tune.

You also have an idea of what kind of performance gain you can hope to expect. If you improve the avg time of map1 by 10% you'll save yourself approximately 20671 * .9 / 60 = **approx 35min** per execution.

Monitoring Performance

- Throughput (time / record)
- Scalability



bayon technologies

We can determine the throughput of our mapping, determining how many records per second the mapping can process. This is useful to determine during testing, if the hardware is sufficient to handle the estimated load.

Plotting mapping performance by data volume and elapsed time allows one to determine how well the particular mapping will scale. Consider the above dataset:

Map1 appears to scale linearly with volume (twice as much data twice as much time).

Map2 appears to gain efficiency with volume (we should all be so lucky) even though more volume means additional elapsed time.

Map3 forms an exponential curve, and should be considered a candidate for tuning.

An increase in data volume poses a critical risk to mappings that exhibit behavior like map3. Measuring and determining a mapping's scalability "pattern", ahead of actually seeing the huge spikes in behavior will help prevent "fire fighting" drills of tuning. This also allows for the ongoing observation and proactive improvement of OWB mappings.

OWB Tuning : Exercise 1