OAC Essbase Hybrid Block Storage Option Performance Tuning

Oracle Analytics Cloud (OAC) Essbase has a new Hybrid Calculation Engine that improves the overall performance of Essbase Applications and reduces development time. This whitepaper highlights new considerations for developers to build Hybrid Block Storage Option (BSO) Applications for optimal performance during retrieval.

WHITE PAPER / JUNE 11, 2018

MIKE LARIMER, ORACLE & RICK SAWA, KROGER

Table of Contents

PERFORMANCE TUNING ON OAC ESSBASE

INTRODUCTION

One of the good things about the new Hybrid Block Storage Option (BSO) is that many of the concepts that we learned over the years about tuning a BSO cube still apply. Knowing what a block is and why it is important is just as valuable today in Hybrid BSO as it was 25 years ago when BSO was introduced. There are new considerations, however, which are the subject of this white paper.

For example, performance optimization requires determining whether to perform sparse dynamic calculations and aggregations instead of storing data. The data block must be defined carefully to ensure that it meets all processing requirements. Formulas must be scrutinized to determine whether they can be dynamically calculated or whether they are better stored and procedurally calculated. Dimension ordering still impacts things like calculation order, and it can also be used to optimize parallel operations and reduce index contention. Calculator cache usage also depends on the order of sparse dimensions.

The default cache settings for OAC Essbase have been increased to 100 MB for both the index and data caches. In high concurrency environments that include parallel calculations as well as multi-user query environments, the proper index cache setting to hold the entire index in memory will prove beneficial to performance. This is because by keeping the full index in memory, requests to the index take less time, reducing contention for index pages.

With Hybrid BSO, it is important to understand that the layout of a report affects query performance. The reason is that the hybrid query engine processes symmetric reports and when it encounters an asymmetric report, OAC Essbase must break that report down into multiple symmetric queries.

There are new logs to help OAC Essbase developers debug calculation flow and query performance. Also, OAC Essbase has added functionality within Smart View to help developers understand and modify the solve order.

WHITE PAPER ORGANIZATION

This paper has been divided into four sections: Cube Design, Caches, Report Layout and Diagnostics. There is an appendix which contains case studies to help you visualize and understand the impact of the topics being discussed.

CUBE DESIGN

Dense and Sparse Dimensions

BACKGROUND

The BSO storage structure is correctly conceived as an array (or matrix). In BSO, we talk about intersection points and cells. We talk about data blocks as opposed to data rows. BSO OAC Essbase essentially converts business elements to members of an array and stores floating point numbers at intersection points defined by one member from each dimension in the array.

Multidimensional arrays are inherently empty, or sparse, structures: the number of intersection points in an array that contain data is significantly fewer than the total number of intersections defined by the array. Let’s walk through a trivial example.

We can easily conceive a company that sells every product in every region of operation. This very primitive ‘business model’ contains the two dimensions of product and geography. In terms of sales, they completely populated with data: Every product has sold in every region.

When we start adding dimensionality to the business model to provide more analytic value, the concept of sparseness becomes manifest. If we track sales by the day, we suspect that we probably don’t sell every product, in every region, on every day. If we want to track sales for every customer by the hour, it becomes easier to see that no customer purchases every product, in every region, every hour of every day. The point here is that an array will contain an intersection, or cell, for every possible combination even when there won’t be anything to store there.

The size of an array in OAC Essbase gets very large, very quickly. For example, consider a hypothetical matrix with the following dimensionality:

Array (172, 21, 27, 32, 209, 32765)

This array contains six dimensions that define 172 * 21 * 27 * 32 * 209 * 32765 = 21,370,660,375,680 potential points of intersection. How do software applications represent this matrix? The solution the developers of OAC Essbase came up with was to modularize the matrix by mapping it as index and data block storage structures, and introduce the concepts of dense and sparse dimensions.

Dense dimensions comprise the data block and are materialized within ess*pag files. Sparse dimensions comprise the index and are materialized in ess*ind files. The database designer determines which dimensions are tagged either as dense or as sparse. Dense dimensions create a unit of storage called the block. The size of the block in the above example is 21*172 = 3612 cells. In OAC Essbase, each cell consumes eight bytes, so this data block consumes 28,896 bytes. The dimensions that are tagged sparse comprise the index. Index entries are unique in OAC Essbase, and each entry points to a single data block.

Configured in this way, the larger array has been modularized into the smaller and more manageable units of storage (block and index) that a computer is able to move in and out of memory with efficiency. An important corollary is that where no index entries exist, no data blocks are created. This is how BSO OAC Essbase squeezes sparseness out of the model.

MATRIX MODELLING

In matrix modelling, the goal of defining the block is to make the dense matrix match the over-arching requirements of the application. For example, if the highest priority of the application is to produce the fastest queries where the matrix is defined as Time across the columns and Accounts in the rows, then it stands to reason that the dense dimensions should include the Time dimension and the Accounts dimension. And if the highest priority of the application is to perform an allocation from one department to hundreds of departments as fast as possible, then it would stand to reason that the dense dimensions should include the Department dimension.

This type of modeling is a great starting point for the definition of the block as it is a simple and fast way to get a reasonably tuned block. One drawback to this type of modelling is that many times the combination of the desired dense dimensions will produce a block that is too large. And it is rarely the case that an OAC Essbase cube has only a single performance objective. A configuration that optimizes one objective (e.g. the Department allocation) often conflicts with the configuration to optimize another objective (e.g. daily outline updates of the Department dimension).

TECHNICAL MODELLING

In technical modelling, the goal of defining the block is to produce the fewest number of the densest blocks. This requires understanding the density of the data on a per dimension basis. It also requires understanding how many blocks will be generated on a per dimension basis.

One way to get the necessary statistics for technical modeling is to systematically set each dimension as the only dense dimension, load the data and then observe the number of blocks created for each dimension. The ODL log records the total number of cells that were created during the data load process. The density for a dimension can then be calculated by this formula:

Density = (Total Cells with data / Total Blocks) / Total Possible Cells in a block

Technical modelling uses a formula to produce a value which will rank the dense dimensional combinations based on their actual density and the number of blocks created. It gives a higher weight to the combination creating the fewest number of blocks. The higher the value the better.

Value = Total Cells with data / (Total Blocks^2 * Total Possible Cells in a block)

BLOCK (AND SIZE)

The block size in bytes is computed by multiplying the total number of stored members in each dense dimension by each other, and multiplying that result by 8. The size of the block will have an impact on performance. A very large block will have too much sparsity which will make it inefficient to process at query time because more missing values will be “touched” during the block scan.

For example, suppose that you have a 1G data block and a query that requests all cells. If only 1K of cells are not #Missing (i.e. contain data) then OAC Essbase still must scan all (1G of) cells. But if the block would be made smaller and split into more blocks – completely empty blocks would be omitted and many fewer “misses” would happen during the data scan. And it is also possible that a large block read into memory will have to reside on multiple memory pages, and the number of switches between these pages comes with additional I/O overhead. On the other hand, a very small block can lead to too many blocks being swapped in and out of memory.

The optimal block size depends upon the application requirements. There are no strict rules on block size but a typical block size range is between 2 KB and 200 KB. There are customer Production cubes out there that have a block size many times larger than 200 KB. Whether this block size will remain efficient in OAC Essbase will require verification.

DATA SECURITY (BLOCK LOCKING)

The OAC Essbase locking system is based on block locking, and which blocks are locked depends upon the sparse dimensions being retrieved. Those dimensions defined as dense will have all members locked when a lock is required for any one member. For example, consider a hypothetical cube with Time (dense) and Location (sparse) dimensions. A lock on January when the Time dimension is dense will result in all months being locked for that location since they are all contained within the block being locked, but updates can still happen for individual Locations. Now consider the configuration where Time is sparse and Location is dense. If locking for update is required for only one location, all locations would be unavailable for update by other users/processes requesting update lock status for the same Time. Here, the Location dimension would not be a good candidate for being tagged dense.

Whereas the complexity of locking becomes more granular as dimensionality increases, the principle remains the same.

VOLATILITY (RESTRUCTURING)

A restructure of the database is required when the outline is modified. This modification can be in the form of loading an alias table, reorganizing a hierarchical relationship, adding a new member, etc.

There are two types of restructures that can occur depending on the event that caused the restructure. A minimal restructure is performed when a change to the outline does not cause the size of the block to be altered. This restructure is very fast and does not require a back-up of the database. A complete restructure is performed when the block size has been altered either by adding/deleting a member of a dense dimension or the dense/sparse settings have been modified.

During a complete restructure creates a new version of the index and data files by ***rebuilding every*** data block. Depending on the size of the database, the time it takes to complete this task can be considerable. Understanding the dynamic nature of each dimension in the application, and setting dimensions with frequently changing members as sparse, can save a lot of time restructuring databases.

ATTRIBUTES

Nothing has changed here. Only a sparse dimension can have an Attribute dimension associated with it. Therefore, any dimension that is designated as a base dimension for attributes cannot be declared dense.

FORMULAS

The number and type of formula assigned to members in the outline might dictate whether to make that dimension dense. Typically, the Account type dimension has many formulas that can be calculated at query time, and this makes that dimension a good candidate for being dense. If a formula is considered “intense” because it requires a lot of data during execution, this too makes its dimension a good candidate for being dense.

An easy methodology for selecting dense dimensions based on formulas is to try to ensure that the data required by formulas are located within a single block. This is the fastest type of calculation in OAC Essbase. Less optimal configurations occur when the data required for the calculation require multiple blocks to be accessed.

Dimension Storage

DYNAMIC

In OAC Essbase there are three types of calculations: hierarchical aggregations, member formulas and procedural calculation scripts. The value proposition of the Hybrid technology is to reduce the amount of data that is generated and stored on disk by relying on the robust nature of the Hybrid Aggregation calculation engine. This is achieved by dynamically calculating as many of the hierarchical aggregations and member formulas as is practical. The overall effectiveness of this approach is determined by the number of data cells existing at the leaf level. For example, if a query requires the query engine to retrieve millions of leaf-level cells, it is expected that this will not be a sub-second query. There is simply too much data to retrieve and calculate for that query performance expectation.

STORING TO REDUCE QUERY INTENSITY

When the number of cells required to satisfy a report becomes too large, query performance becomes unacceptable. The number of cells required is ultimately determined by the number of hierarchies/members in the outline, or by a member formula which requires a large amount of data to compute.

The technique for reducing the number of cells required at query time is to calculate and store the data only for specific narrow slices of the cube. This is very similar to the concept of “aggregate views” in an ASO database. Intense member formulas can be easily set to be stored by OAC Essbase, and a calculation script can be written to execute the member formulas on demand. Similarly, whole or partial dimensions can be calculated and stored that will reduce the number of dynamically calculated cells required to satisfy the upper level queries.

Dimension Ordering

QUERY SOLVE ORDER

Solve order is important in multi-dimension databases to ensure accurate results. Solve order can also impact the performance of the calculation script or query. In OAC Essbase, solve order can be applied per dimension or per member. The minimum solve order is 0 and the maximum solve order is 127. The higher the solve order setting, the lower in the order the member is calculated. Put another way, a formula with a solve order of 1 will be solved ***before*** a member with a solve order of 2. The following chart provides an overview of solve ordering settings:

Table 1: Default Solve Order Settings

DIMENSION TYPE	DEFAULT SOLVE ORDER VALUE

Stored members	0
Sparse dimension	10
Dense dimension – Account	30
Dense dimension – Time	40
Dense dimension – Account	50
Dense dimension – Two Pass Account	60
Dense dimension – Two Pass Time	70
Two Pass	80
Attribute dimension	90

DEFAULT CALCULATION ORDER REDEFINED

The order of the dimensions in the outline will also determine the solve order for those dimensions not marked as Account, Time or Two Pass. Sparse dimensions are calculated in outline order from top to bottom. Although two Pass is supported in OAC Essbase, it is a best practice recommendation to explicitly set the Solve Order instead of using Two Pass.

FORMULAS

The order of operation for formulas can be managed at the dimension level or at the individual member level. There are two general considerations for formulas. First, the order of operations can impact the correctness of the result. And prices, rates and ratios do not aggregate.

The following chart summarizes best practices:

Table 2: Best Practices for Formula

TYPE OF FORMULA	BEST PRACTICE
Any formula that can be calculated after an aggregation such as a ratio.	Dynamically calculate with a high solve order.
Any formula which must be computed before aggregation.	If performance is slow, consider making it a stored member and use a calculation script.
Any formula that requires data to be retrieved from many blocks such as a rolling forecast.	If performance is slow, consider making it a stored member and use a calculation script.

The second consideration is more performance related. In OAC Essbase, it is fastest to let the OAC Essbase kernel perform hierarchical aggregations first, and then let the hybrid engine perform other calculations dynamically. This can be achieved either by ordering those dimensions which contain formulas to calculate last for both the dense and sparse dimensions, or by setting the precise solve order for each formula.

This is the recommended dimension ordering in OAC Essbase:

1. Dense Dimensions (Dynamic non-formula)

2. Dense Dimensions (Dynamic formula)

Enhanced Hybrid Engine

The value proposition of the Hybrid technology is to reduce the amount of data that is generated and stored on disk by relying on the robust nature of the Hybrid Aggregation calculation engine.

3. Sparse Dimension (Calculated and Stored)

4. Sparse Dimension (Dynamic non-formula)

5. Sparse Dimension (Dynamic formula)

6. Sparse Dimension (Task dimension for Parallelism)

Case Study #4 in the Appendix shows an example of a properly re-designed cube that includes this recommended ordering.

CONCURRENCY

OAC Essbase has a high dependency on the index when reading and writing data. Index entries to all blocks are stored on individual index page, and the location of index entries to these pages are strictly dependent on the order of sparse dimensions.

The last sparse dimension will have its index entries farthest apart in the index, and therefore are most likely to be located on different index pages. In high concurrency situations, such as performing multiple parallel calculations, it is crucial that the operations do not ask for the same index page at the same time, or there will be index page contention, and performance will degrade. It is a best practice to declare the “task” dimension(s) for parallel calculations explicitly as the last sparse dimension(s).

EXPORT LAYOUT

One thing to keep in mind when ordering dense dimensions is that the layout of a system export file will change depending on the dense dimension order. If the column dimension happens to be very small, you can end up with an export file layout that produces many rows with very few values on each row. This will slow the export process and inflate the size of the export file(s).

Here is the logic OAC Essbase follows in choosing the system export layout:

Start looking at the first dense dimension size compared to the 2nd dense dimension.

1. If the 2nd dimension size is less than 100 and it has more members than last chosen column dimension then choose it as the new column dimension

2. If the 2nd dimension is shorter than 1st dense dimension then choose it as shorter dimension

If after the execution of the loop we find a column dimension in step 1, that dimension is taken as the final chosen column dimension otherwise the shortest dense dimension found in 2 is finally chosen as the dense dimension.

Essbase will not select the first dense dimension to be in the column of the export file unless all other dense dimensions have more than 100 stored members.

Note: There are alternatives to using the system export such as the DATAEXPORT calc script command and the MDX Export command. With these alternative export methods, the layout of the export file is defined regardless of the dimension outline order.

CACHES

Index Cache

OAC Essbase stores information about each block in an index. The location of the index entries in the index are determined by the order of the sparse dimensions. The first sparse dimension's index entries will be clustered together. Subsequent sparse dimension index entries will be stored further and further apart within the index.

The index has its own cache and can have a minimum value of 1 MB. The default value for this cache in OAC Essbase is set to 100 MB.

In memory, the index cache is made up of index pages that are 8 KB in size. The purpose of the index cache is to minimize the wait time when a new index page is required to complete the operation.

The index is persisted on disk in files with the extension ess*ind.

The index gets locked at the index page level whenever it needs to be updated. This is an exclusive lock and all other requests must wait until the update operation has completed. It is very important to ensure that any contention on the index is minimized. Index contention can occur during concurrent calculations such as when using CALCPARALLEL or FIXPARALLEL.

Data Cache

OAC Essbase stores the matrix of data values in a collection of blocks defined by the dense dimensions.

The data blocks have their own data cache which has a minimum value of 3 MB. The default value for this cache in OAC Essbase is set to 100 MB. The purpose of the data cache is to minimize the wait time when new blocks are required to complete an operation. The data is persisted on disk in files with the .pag extension.

Calculator Cache

Under the right circumstances, the calculator cache can be very beneficial to the performance of calculations that involve aggregations. It stores a bitmap of the index for the uppermost sparse dimensions that fit into the size of the defined calculator cache. There are several types of calc cache configurations that OAC Essbase can establish, all based on the sparse dimensionality of each cube individually.

The calculator cache is essentially an area of memory that can be set aside by OAC Essbase to enhance sparse calculations that create data blocks. It is not a cache in the sense of a stash of index or data pages. It is better understood as a roadmap that tracks children and parents in support of block creation. By default, the CALCCACHE is on, and OAC Essbase will use either 200,000 bytes or whatever value is established using the CALCCACHEDEFAULT configuration setting, though there are reasons to disable it.

One reason is the amount of available RAM. OAC Essbase will establish one calculator cache for every thread involved in the aggregation process. A parallel aggregation involving 16 CPUs, for example, will spawn 16 calculator cache instances. And if the configuration sets the calculator cache to the maximum 200 MB, then 3.2 GB of memory will be consumed for the calc cache alone for this one calculation. In an environment where multiple parallel calculations are occurring, as in planning type cube environments, the amount of memory being allocated can easily consume available memory. The calculator cache can be detrimental at times, specifically due to the overhead of having to create the bitmap over and over for every task in the parallel processing for every thread in the process.

In an OAC Essbase environment where blocks tend to be updated rather created, the setting is of nominal use. Under the right circumstances, aggregations can benefit from a CALC CACHE. And developers have the option to toggle the CALC CACHE on and off (as well as the degree of parallelism) procedurally in complex calc script scenarios.

In OAC Essbase, the customer modified CFG file is set on an application basis and can be modified using the OAC Essbase UI. It is stored in a file called appName.cfg. The essbase.cfg file still exists for global settings. It is controlled by Oracle and should not be modified by the customer.

The three values (HIGH, DEFAULT and LOW) for CALCCACHE are set in the appName.cfg file. The specific value used during a given calculation can be controlled through the SET CACHE setting in the calc script. The default setting is for OAC Essbase to use the CALC CACHE but OAC Essbase really should be configured to use this cache based on the specific contents of the calculation script and cube outline.

It is also true that in complex FIX operations OAC Essbase will turn the CALC CACHE off. Developers can force its use by issuing the SET CACHE ALL command in the script. Conversely, because the default is for OAC Essbase to use the CALC CACHE, you can force it off by issuing the SET CACHE OFF command.

CALCPARALLEL, CALCTASKDIMS

The manual setting of CALCTASKDIMS has been deprecated in OAC Essbase. This means that it can remain transparent to the developer that OAC Essbase can select different CALCTASKDIMS during a calculation when the CALC CACHE is ON than when it is set OFF. This situation is compounded by the fact that OAC Essbase can be expected to behave (i.e. perform) very differently depending on the order of sparse dimensions. Testing has shown that performance is better when favoring parallelism than it is when favoring the use of the CALC CACHE. Regardless, optimal calc script performance will be achieved by systematically testing to determine the correct outline order in combination with the use, or not, of the CALC CACHE. Remember that the range of useful calculator cache values depends on cube metadata as well as on available memory resources. Performance results are cube dependent.

REPORT LAYOUT

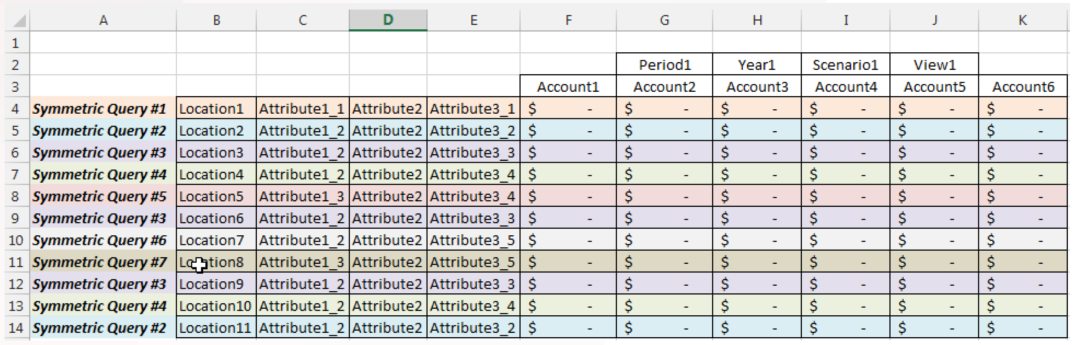
Symmetric vs Asymmetric

An asymmetric report is one where the cross dimensional layout of the members being queried changes in either the rows or the columns. The Hybrid query engine processes only symmetric grids so if it encounters an asymmetric query, it automatically breaks that asymmetric grid down into multiple symmetric grids. These multiple symmetric grids are then processed one at a time and then returned to the client in the original asymmetric form.

Query Odometers

In the OAC Essbase logs, there is a new term called an odometer which represents the number of queries the query engine had to execute to complete the query. A single symmetric query has a query odometer value of 1. The odometer value of an asymmetric query will represent the total number of symmetric queries the query engine had to populate to complete the asymmetric query. The odometer value of a query can be viewed by using either the LongQueryTimeThreshold or QUERY_TRACE application configuration settings (see below in Diagnostics).

The higher the query odometer value, the greater the impact the layout of the report will have on query performance due to the serial nature of asymmetric report processing. In other words, each query in the odometer is run one after the other.



	A	B	C	D	E	F	G	H	I	J	K
1											
2							Period1	Year1	Scenario1	View1	
3						Account1	Account2	Account3	Account4	Account5	Account6
4	***Symmetric Query #1***	Location1	Attribute1_1	Attribute2	Attribute3_1	$ -	$ -	$ -	$ -	$ -	$ -
5	***Symmetric Query #2***	Location2	Attribute1_2	Attribute2	Attribute3_2	$ -	$ -	$ -	$ -	$ -	$ -
6	***Symmetric Query #3***	Location3	Attribute1_2	Attribute2	Attribute3_3	$ -	$ -	$ -	$ -	$ -	$ -
7	***Symmetric Query #4***	Location4	Attribute1_2	Attribute2	Attribute3_4	$ -	$ -	$ -	$ -	$ -	$ -
8	***Symmetric Query #5***	Location5	Attribute1_3	Attribute2	Attribute3_4	$ -	$ -	$ -	$ -	$ -	$ -
9	***Symmetric Query #3***	Location6	Attribute1_2	Attribute2	Attribute3_3	$ -	$ -	$ -	$ -	$ -	$ -
10	***Symmetric Query #6***	Location7	Attribute1_2	Attribute2	Attribute3_5	$ -	$ -	$ -	$ -	$ -	$ -
11	***Symmetric Query #7***	Location8	Attribute1_3	Attribute2	Attribute3_5	$ -	$ -	$ -	$ -	$ -	$ -
12	***Symmetric Query #3***	Location9	Attribute1_2	Attribute2	Attribute3_3	$ -	$ -	$ -	$ -	$ -	$ -
13	***Symmetric Query #4***	Location10	Attribute1_2	Attribute2	Attribute3_4	$ -	$ -	$ -	$ -	$ -	$ -
14	***Symmetric Query #2***	Location11	Attribute1_2	Attribute2	Attribute3_2	$ -	$ -	$ -	$ -	$ -	$ -

Figure 1. Serial nature of asymmetric report processing.

One example of an asymmetric query layout that might not seem obvious is the use of attribute members within the rows. In the example below, the query odometer is seven since there are seven individual symmetric queries which can be constructed from this single asymmetric grid:

Table 3: Individual Symmetric Queries in single Asymmetric Grid

	OUTER ROW DIMENSION	INNER ROW DIMENSIONS
Symmetric Query #1	Location1	Attribute1_1->Attribute2-> Attribute3_1
Symmetric Query #2	Location2 Location11	Attribute1_2->Attribute2-> Attribute3_2
Symmetric Query #3	Location3 Location6 Location9	Attribute1_2->Attribute2-> Attribute3_3
Symmetric Query #4	Location4 Location10	Attribute1_2->Attribute2-> Attribute3_3
Symmetric Query #5	Location5	Attribute1_3->Attribute2-> Attribute3_4
Symmetric Query #6	Location7	Attribute1_2->Attribute2-> Attribute3_5
Symmetric Query #7	Location8	Attribute1_3->Attribute2-> Attribute3_5

DIAGNOSTICS

App Configuration Parameters

Table 4: Application Configuration Parameters

PARAMETER	DESCRIPTION
LongQueryTimeThreshol d	Specify the time limit, in seconds, for queries. Any query that runs longer than this limit is considered slow and will trigger alert messages.

LongCalcTimeThreshold	Specify the time limit, in seconds, for top level calc commands. Any top level calc command, which runs longer than this limit, is considered slow and will trigger alert messages.
QUERYTRACE	Specify the value to determine the contents of the tracing log for this application. A value of -1 will provide all details.
CALCTRACE	Specify TRUE|FALSE to determine if calc tracing will be enabled for this application.

ODL Logs

CALC

For calc diagnostics, there is a log entry which can be activated by a CFG property called LongCalcTimeThreshold. The value to use is dependent on the application and whether you want to trigger this alert. As was stated earlier, the customer modified CFG file is now on an application basis and can be modified using the OAC Essbase UI.

Alert messages contain the following information:

- User name
- Calc script name
- Corresponding line numbers [x – y]
- The first 256 characters of the command's text.
- Number of blocks read, write, and create.
- Execution time of the command

There is also useful information in the default ODL logs for calculations such as the number of blocks existing after the calculation at both the leaf level and total level.

```
[2018-03-30T12:41:50.351-07:00] [_K_Baseline] [NOTIFICATION:16] [REQ-449] [REQ] [ecid: 1522433698545,0] [tid: 140437732837120] [DBNAME: SOS200] [REQ_ID: 6538802943238340626] [REQ_INFO: Calculate] [REQ_STATE: finished] [EXEC_TIME: 1110141] [LEVEL0_BLOCKS: 24510297] [TOTAL_BLOCKS: 53129960]
```

QUERY

For query diagnostics, there is a log entry which can be activated by an appname.cfg configuration called LongQueryTimeThreshold. The value to use is something small so that it triggers the log entry. I use the value of .01.

There are many important pieces of information contained in this log entry. The number of blocks read and the number of output cells of the query gives an indication of the amount of data being requested. The number of formulas executed is an indication of the formula intensity on the query. The number of odometers is an indication of the asymmetric layout of the query.

```
[2018-04-01T18:02:14.893-07:00] [_K_All] [NOTIFICATION:16] [QX-70] [QX] [ecid: 1522622817955,0] [tid:
140114092427008] [DBNAME: SOS200] [EXEC_TIME: 2700] [QUERY_USER: admin] [reportType: Grid]
[nHash: 2908443362] [sts: 0] [nOdometers: 618] [sumOfOdomSizes: 42495] [sumOfOdomVolumes: 90675]
[bSuppressMissing: F] [bCellStatusMDX: F] [bMeaningless: F] [bSuppressInvalid: T] [ullBlocksRead:
86216] [ullFormulaExec: 0] [ullFormulaMissing: 0] [ullFormulaExecOOT: 0] [bAsoCacheFull: F]
[nMicrocubes: 0] [nRetrClusters: 0] [nRetrTuples: 0] [nOutCellsAll: 90714] [nOutCellsNonMi: 85771]
[tScanAgg: 630] Report Text…
```

Query Trace Logs

Using the QUERYTRACE configuration parameter sets a query calculation flow trace to be run and the results to be printed out to a file.

With the parameter QUERYTRACE -1, the query tracing output file includes:

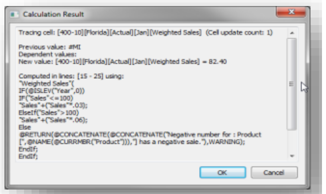
- The input query
- An expanded query odometer
- General information about query calculation units
- A list of formulas and aggregations
- An ordered list of all output cells that are calculated or aggregated during the query, according to solve order

The query tracing output file, query_trace.txt, is written to the database files location. It is extremely detailed and therefore can be very large (not shown). A summary query tracing file is also created called query_summary.txt (shown below). It contains a subset of the important statistics contained in the query trace file.

```
************************
Fri Feb 9 12:07:54 2018

Number of odometers retrieved: 1
Odometers sizes: 81
Odometers volumes: 720.000000
Number of blocks read: 8037
Number of cells acquired after scan/agg (for calculations and/or output): 1701001
Number of formula cache windows processed : 69535
Total number of dynamic cells processed: 38398
Number of aggregated cells : 311860
Number of formula calculation cells : 526
Number of calculations returned #Missing : 30631
Number of skipped executions: 0
Number of non-missing output cells produced by this query : 537
-------------------- Units processing times: -------------------------
Stored data scan and kernel aggregation total processing time: 0.150
Calculation units total processing time: 4.980
Total query processing time: 5.150 sec
Max odometer processing time: 5.150 sec for odometer No.1

 ************************ Hints for this query optimizations ************************
More than 30 percent of formulas calculations returned #Missing values.
 It might be caused by top down formulas.

 ************************************************
```

Calc Trace Logs

Using the CALCTRACE configuration parameter you can analyze member formula processing, and refine your calculation scripts. Calculation tracing enables you to access logged information about a calculation after the calculation script successfully executes.



```
Tracing cell: [400-10][Florida][Actual][Jan][Weighted Sales]  (Cell update count: 1)

Previous value: #MI
Dependent values:
New value: [400-10][Florida][Actual][Jan][Weighted Sales] = 82.40

Computed in lines: [15 - 25] using:
"Weighted Sales"(
IF(@ISLEV("Year",0))
IF("Sales"<=100)
"Sales"+("Sales"*.03);
ElseIf("Sales">100)
"Sales"+("Sales"*.06);
Else
@RETURN(@CONCATENATE(@CONCATENATE("Negative number for : Product
[",@NAME(@CURRMBR("Product"))),"] has a negative sale."),WARNING);
EndIf;
EndIf;
```

Tracing a calculation does not change anything about calculation behavior. If a calculation is launched in Smart View, and the connected server has calculation tracing enabled by an administrator, Smart View displays a pop-up dialog box containing details after the calculation finishes. The calculation tracing information can be pasted from the pop-up dialog into a text editor. The same information is written into calc_trace.txt, located in the database files directory on the cloud service.

The calculation tracing information can help you debug calculation script execution, in case the results of the calculation are not what you expected. To enable calculation tracing, the administrator must first turn on the CALCTRACE application configuration parameter. Then, Smart View users can select data cells to trace.

Outside of Smart View, you can also use the SET TRACE calculation command in calculation scripts to select data cells to trace. SET TRACE enables you to trace multiple data cells. Additionally, you can trace sections of calculation scripts by using a combination of SET TRACE mbrList and SET TRACE OFF. However, to use SET TRACE command, you must execute the calculation script outside of Smart View, using Cube Designer or the Jobs page of the cloud service.

Calculation tracing is not supported on applications with scenario management enabled.

The calculation tracing log provides the following insights about how the calculation worked, on the cell that was traced:

- Whether a traced cell was calculated several times.
- The value of the cell before calculation.
- For each calculation occurrence, dependent values and new values are shown.
- The final value of the traced cell after all calculation completes.
- The lines of the calculation script which are responsible for the updated values.

FRAGMENTATION

Background

The process of updating data has the potential to cause cube fragmentation. And fragmentation can happen whenever a block needs to be updated.

When OAC Essbase updates a block during a data load or a calculation, it does so first in memory, and eventually the block is passed to the OS file system cache to be persisted within the data file(s). If the block being written back to the PAG file no longer fits in the same location from which it was originally read, (because it contains more data), the block is written to a new location within the PAG file and the index is updated to reflect the new location of the block. The old block location is abandoned, and when a block location is abandoned, the PAG file becomes fragmented. As the PAG file becomes fragmented, the IND file also becomes fragmented for the same reasons.

The higher the fragmentation in the cube, the slower the overall performance of operations.

Causes

POORLY SORTED DATA LOAD

The block is defined by the dense dimensions. The process of how a block is brought into memory and written to disk makes it critical that the data being bulk loaded has been sorted so that all the data for each block gets loaded at once. This requires that the data file be sorted by the sparse dimensions. This way the block is brought only one time into memory, and after the data has been loaded into the block, it is ready to be written to the PAG file.

The point is that if the data load file is not properly sorted, the block will be updated (read and written to disk) many times. Because the data in the block is constantly changing, it might have to be written to a new location in the PAG file, causing fragmentation.

It is also possible to cause a kind of block deadlock situation during unsorted data loads. This happens when a block that has been sent by OAC Essbase to the OS to be written is simultaneously requested to be brought back in to be updated again by the OAC Essbase data load process. The I/O overhead required to manage blocks that OAC Essbase wants to update that are sitting in the file system cache waiting to be written, greatly reduces data load performance on top of potentially fragmenting the cube.

POORLY FOCUSED USE OF CREATEBLOCK IN CALC SCRIPT

A new function was introduced to allow for special cases when a block needs to be explicitly declared before any data is written to the block. One example is when a parent block needs to be created so that the @SUM function can then be used to populate it with the data of the children. This function should be used with extreme caution as it can lead to empty blocks in a database. The best practice when using CREATEBLOCK is to use the CLEARBLOCK EMPTY command after all processing has completed.

Block Margin

In OAC Essbase, the concept of a block margin was introduced to address the issue of PAG file fragmentation. The basic concept is that when a block is written to disk, the size of the block will be padded by a certain percentage to accommodate future growth. When an updated block is written to storage, it will ideally fit into its original location. This strategy effectively eliminates fragmentation but is dependent upon the Administrator allocating the block margin accurately.

The margin percent is controlled by a CFG setting called INPLACEDATAWRITEMARGINPERCENT and the default value is 20. This means that a block will always be allotted 20 percent more space on disk to allow for block growth. As was stated earlier, the customer modified CFG file is now on an application basis and can be modified using the OAC Essbase UI.

Defragmenting

Typical best practice is to determine a baseline size for the index and page files of an OAC Essbase cube. When the sizes begin to grow without explanation, it is a good indication that these files have become fragmented.

There is a MAXL command for defragmenting a cube:

```
alter database 'appName'.'cubeName' force restructure;
```

Note: The cube is unavailable for read or write during the restructure defragmentation process.

APPENDIX

Case study #1

METHODOLOGY

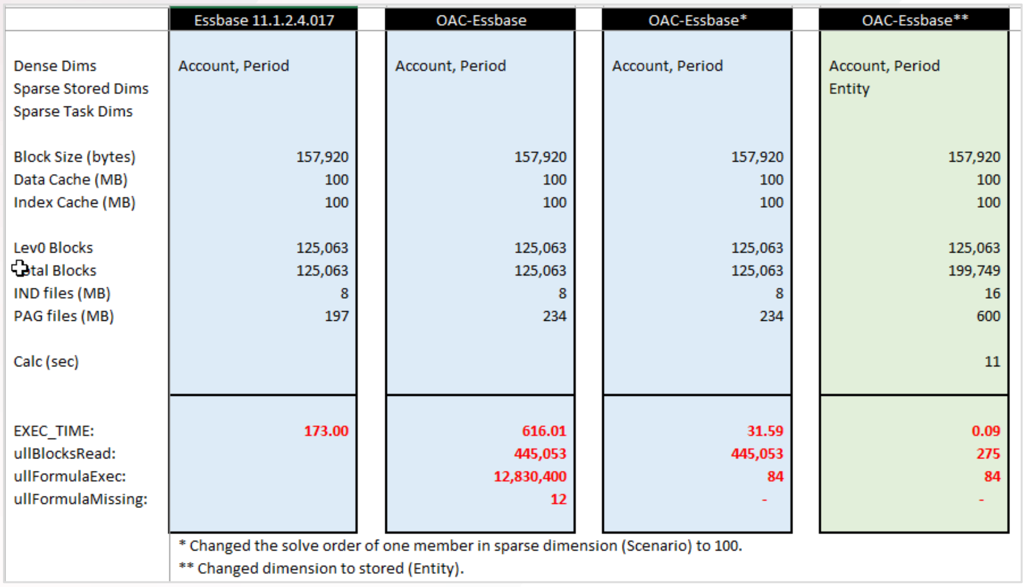
1. Export outline into DBX workbook because it is very easy to analyze the outline in this format.
2. Convert upper level member's storage property to dynamic calc in all dimensions using either Cube Designer or OAC Essbase web UI.
3. Run query and analyze ODL log.
4. Correct for too many formula executions during the query.
5. Run query and analyze ODL log.
6. Correct for too much data required by the query.

SUMMARY

In this case study, there are two issues effecting query performance when taking an on-premise BSO application and shifting it to Hybrid BSO in OAC Essbase. Before starting the conversion process, the application configuration parameter called "LongQueryTimeThreshold" was set to .01 to trigger the query statistics in the ODL log. These query statistics are important in understanding the intensity of the query. Then the Hybrid conversion process converted all upper level members of all dimensions into dynamic members. This can be done using either the Cube Designer Hierarchy Viewer or the web UI outline editor. The data was then loaded and a representative query was run and the query statistics noted.

The first issue found was that there was one member in a sparse dimension which contains a formula. Sparse member formulas can have a dramatic impact on query performance if the order of the calculation is not dealt with properly. In this case, the member is in the third of six sparse dimensions and the formula is a variance type formula. The best practice is to calculate this type of sparse formula after the sparse aggregations which either requires a high solve order on the member or the dimension containing the formula member get moved after the aggregating sparse dimensions.

After correcting for the member formula, the query time was still not acceptable. The combination of the size of the block and the number of blocks required by the query indicates that the number of blocks needs to be reduced. The technique for reducing the number of blocks is to calculate and store a selected group of blocks. The simplest method of this is to calculate and store one dimension.



	Essbase 11.1.2.4.017	OAC-Essbase	OAC-Essbase*	OAC-Essbase**
Dense Dims	Account, Period	Account, Period	Account, Period	Account, Period
Sparse Stored Dims				Entity
Sparse Task Dims				
Block Size (bytes)	157,920	157,920	157,920	157,920
Data Cache (MB)	100	100	100	100
Index Cache (MB)	100	100	100	100
Lev0 Blocks	125,063	125,063	125,063	125,063
Total Blocks	125,063	125,063	125,063	199,749
IND files (MB)	8	8	8	16
PAG files (MB)	197	234	234	600
Calc (sec)				11
EXEC_TIME:	173.00	616.01	31.59	0.09
ullBlocksRead:		445,053	445,053	275
ullFormulaExec:		12,830,400	84	84
ullFormulaMissing:		12	-	-

* Changed the solve order of one member in sparse dimension (Scenario) to 100.
** Changed dimension to stored (Entity).

Figure 2. Case Study #1 Observations

Case Study #2

METHODOLOGY

1. Export outline into DBX workbook because it is very easy to analyze the outline in this format.
2. If parallel script is running poorly due to index contention, modify the index cache to contain the entire index files (*.ind) OR move the task dimension to the bottom of the outline.

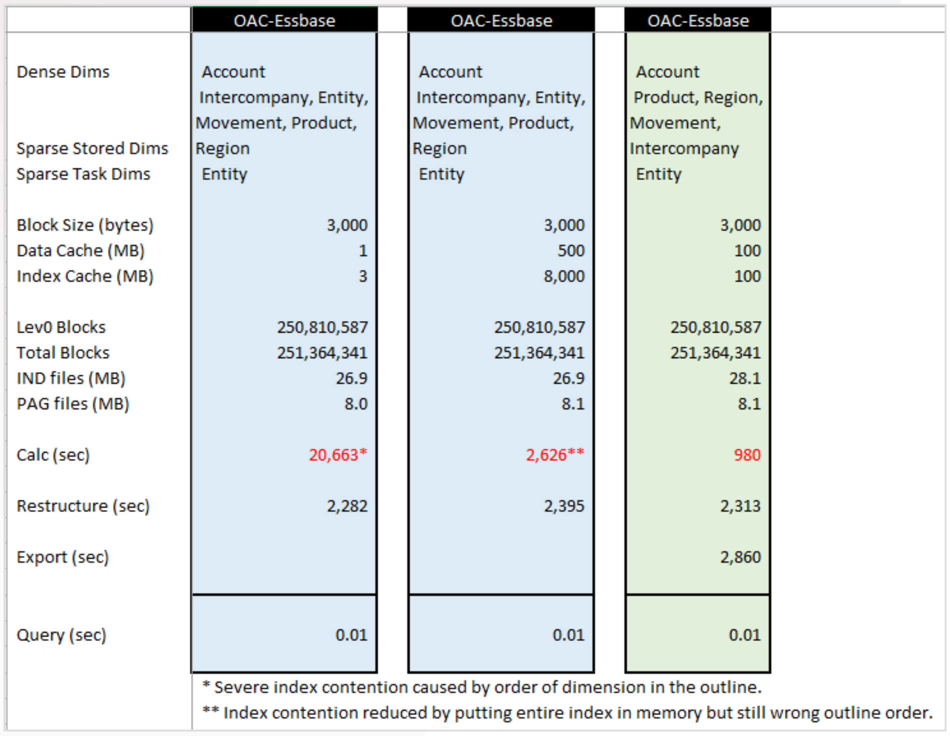
SUMMARY

This case study involves a poorly performing calculation script that was utilizing the FIXPARALLEL capabilities of the calculator. The issue was that the multi-threaded nature of parallel calculations led to contention on the OAC Essbase index and it was causing threads to remain idle until the index became free.

There are two ways to address this index contention. The first way is to increase the index cache so that the entire index can fit into memory. While this does not eliminate the chance of index contention, the index operations are faster and the thread wait time reduced.

The second way to virtually eliminate index contention is to locate the “task” dimension of the FIXPARALLEL command at the bottom of the outline. The index is made up of 8KB pages. The pages contain location information for each block in the database (sparse combinations). The lower in the outline order you place the “task” dimension, the less chance that the index entries for the task members will exist on the same index page and therefore, there will be no index contention during processing.

Note: With proper outline sorting, the index cache does not need to be increased.



	OAC-Essbase	OAC-Essbase	OAC-Essbase
Dense Dims	Account	Account	Account
Sparse Stored Dims	Intercompany, Entity, Movement, Product, Region	Intercompany, Entity, Movement, Product, Region	Product, Region, Movement, Intercompany
Sparse Task Dims	Entity	Entity	Entity
Block Size (bytes)	3,000	3,000	3,000
Data Cache (MB)	1	500	100
Index Cache (MB)	3	8,000	100
Lev0 Blocks	250,810,587	250,810,587	250,810,587
Total Blocks	251,364,341	251,364,341	251,364,341
IND files (MB)	26.9	26.9	28.1
PAG files (MB)	8.0	8.1	8.1
Calc (sec)	20,663*	2,626**	980
Restructure (sec)	2,282	2,395	2,313
Export (sec)			2,860
Query (sec)	0.01	0.01	0.01

* Severe index contention caused by order of dimension in the outline.
** Index contention reduced by putting entire index in memory but still wrong outline order.

Figure 3. Case Study #2 Observations

Case Study #3

METHODOLOGY

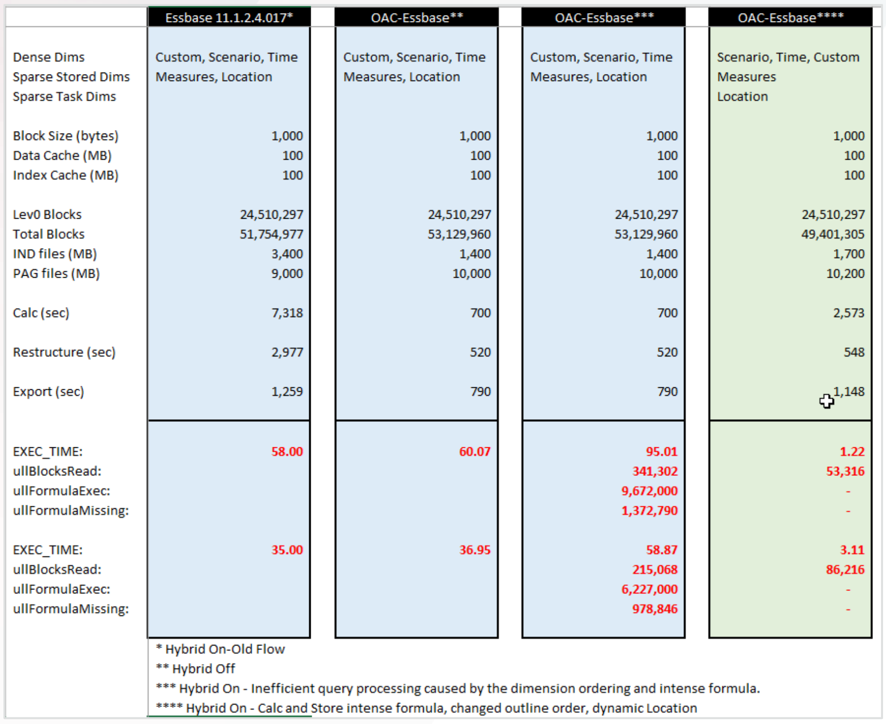
1. Convert upper level member's storage property to dynamic calc in all dimensions using either Cube Designer or OAC Essbase web UI.
2. Run query and analyze ODL log.
3. Correct for the intense formula executions during the query.
4. Run query and analyze ODL log.
5. Correct for too much data required by the query.

SUMMARY

In this case study, the first dense dimension called "Custom" contained multiple formulas. When a dense dimension contains formulas, those formulas need to be calculated after any aggregations to improve performance. One of the formulas computed a rolling 13-month total which pulled data from multiple dimensions. This intense formula was causing millions of formula calculations during the query and the performance was unacceptable.

In addition to identifying formulas, the dimensions were segregated into types and the order of the dimensions adjusted accordingly. The Location dimension was converted to dynamic upper level calculations and it was used as the task dimension during parallel calculation.

The combination of storing a few key areas of the cube together with the new outline ordering brought the query performance to an acceptable level.



	Essbase 11.1.2.4.017*	OAC-Essbase**	OAC-Essbase***	OAC-Essbase****
Dense Dims	Custom, Scenario, Time	Custom, Scenario, Time	Custom, Scenario, Time	Scenario, Time, Custom
Sparse Stored Dims	Measures, Location	Measures, Location	Measures, Location	Measures
Sparse Task Dims				Location
Block Size (bytes)	1,000	1,000	1,000	1,000
Data Cache (MB)	100	100	100	100
Index Cache (MB)	100	100	100	100
Lev0 Blocks	24,510,297	24,510,297	24,510,297	24,510,297
Total Blocks	51,754,977	53,129,960	53,129,960	49,401,305
IND files (MB)	3,400	1,400	1,400	1,700
PAG files (MB)	9,000	10,000	10,000	10,200
Calc (sec)	7,318	700	700	2,573
Restructure (sec)	2,977	520	520	548
Export (sec)	1,259	790	790	1,148
EXEC_TIME:	58.00	60.07	95.01	1.22
ullBlocksRead:			341,302	53,316
ullFormulaExec:			9,672,000	-
ullFormulaMissing:			1,372,790	-
EXEC_TIME:	35.00	36.95	58.87	3.11
ullBlocksRead:			215,068	86,216
ullFormulaExec:			6,227,000	-
ullFormulaMissing:			978,846	-

* Hybrid On-Old Flow
** Hybrid Off
*** Hybrid On - Inefficient query processing caused by the dimension ordering and intense formula.
**** Hybrid On - Calc and Store intense formula, changed outline order, dynamic Location

Figure 4. Case Study #3 Observations

Case Study #4

METHODOLOGY

1. Examine the allocation calc script logic and determine a new cube shape was required.
2. Convert upper level member's storage property to dynamic calc in all dimensions using either Cube Designer or OAC Essbase web UI.
3. Run query and analyze ODL log.
4. Moved the stored sparse dimensions to top of the sparse list to get them included in the calc cache.
5. Run query and analyze ODL log.
6. Moved the sparse dimensions to be after the dynamic aggregated sparse dimensions to utilize the OAC Essbase kernel for the aggregation processing.

SUMMARY

In this case study, the requirement was to improve the performance of an allocation process for an on-premise BSO cube. The cube was re-designed to be much smaller and faster in all aspects of cube operations.

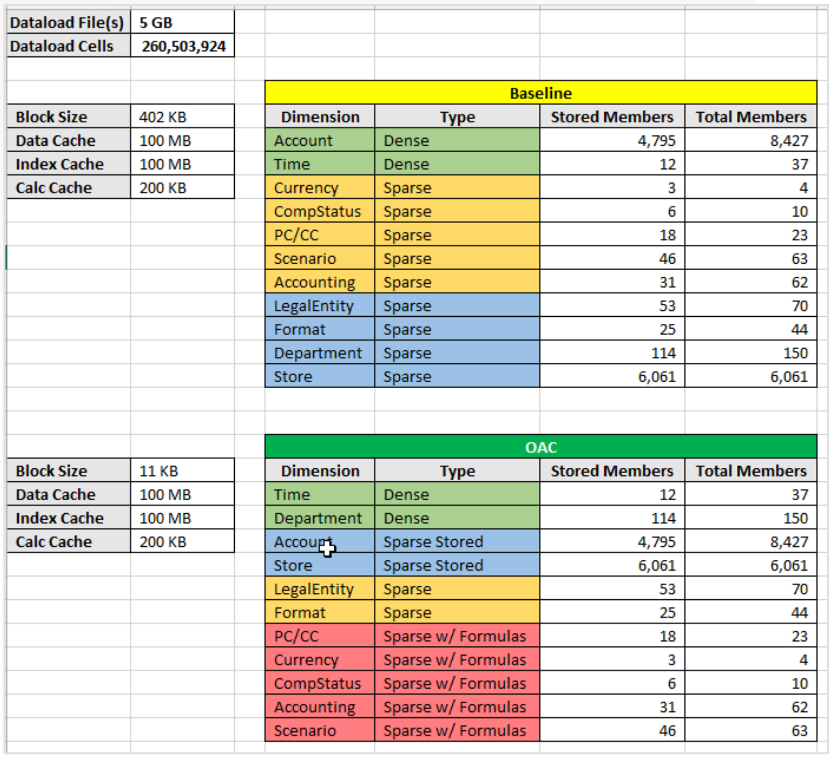
Changes to note:

- The block was resized.
- The outline was re-ordered.

- Some of the dimensions were made dynamic.

Results to note:

- The PAG and IND files got smaller.
- The query time, the restructure time and the export time improved.



Dataload File(s)	**5 GB**
Dataload Cells	**260,503,924**

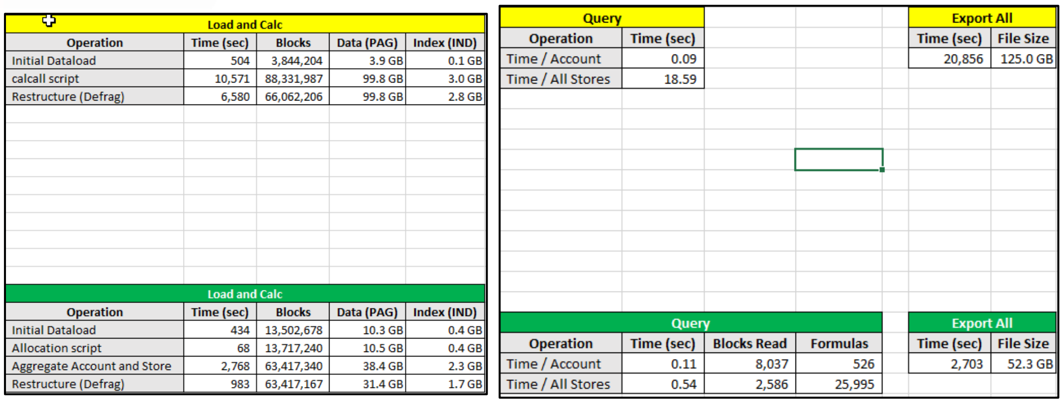
Block Size	402 KB
Data Cache	100 MB
Index Cache	100 MB
Calc Cache	200 KB

Baseline			
Dimension	**Type**	**Stored Members**	**Total Members**
Account	Dense	4,795	8,427
Time	Dense	12	37
Currency	Sparse	3	4
CompStatus	Sparse	6	10
PC/CC	Sparse	18	23
Scenario	Sparse	46	63
Accounting	Sparse	31	62
LegalEntity	Sparse	53	70
Format	Sparse	25	44
Department	Sparse	114	150
Store	Sparse	6,061	6,061

Block Size	11 KB
Data Cache	100 MB
Index Cache	100 MB
Calc Cache	200 KB

OAC			
Dimension	**Type**	**Stored Members**	**Total Members**
Time	Dense	12	37
Department	Dense	114	150
Account	Sparse Stored	4,795	8,427
Store	Sparse Stored	6,061	6,061
LegalEntity	Sparse	53	70
Format	Sparse	25	44
PC/CC	Sparse w/ Formulas	18	23
Currency	Sparse w/ Formulas	3	4
CompStatus	Sparse w/ Formulas	6	10
Accounting	Sparse w/ Formulas	31	62
Scenario	Sparse w/ Formulas	46	63

Figure 5. Case Study #3 Change in Block Size



Load and Calc				
Operation	**Time (sec)**	**Blocks**	**Data (PAG)**	**Index (IND)**
Initial Dataload	504	3,844,204	3.9 GB	0.1 GB
calcall script	10,571	88,331,987	99.8 GB	3.0 GB
Restructure (Defrag)	6,580	66,062,206	99.8 GB	2.8 GB

Query	
Operation	**Time (sec)**
Time / Account	0.09
Time / All Stores	18.59

Export All	
Time (sec)	**File Size**
20,856	125.0 GB

Load and Calc				
Operation	**Time (sec)**	**Blocks**	**Data (PAG)**	**Index (IND)**
Initial Dataload	434	13,502,678	10.3 GB	0.4 GB
Allocation script	68	13,717,240	10.5 GB	0.4 GB
Aggregate Account and Store	2,768	63,417,340	38.4 GB	2.3 GB
Restructure (Defrag)	983	63,417,167	31.4 GB	1.7 GB

Query			
Operation	**Time (sec)**	**Blocks Read**	**Formulas**
Time / Account	0.11	8,037	526
Time / All Stores	0.54	2,586	25,995

Export All	
Time (sec)	**File Size**
2,703	52.3 GB

Figure 6. Case Study #3 Observations

ORACLE CORPORATION

Worldwide Headquarters
500 Oracle Parkway, Redwood Shores, CA 94065 USA

Worldwide Inquiries

TELE	+ 1.650.506.7000	+ 1.800.ORACLE1
FAX	+ 1.650.506.7200	

oracle.com

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com and cloud.oracle.com. Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com/oracle facebook.com/oracle twitter.com/oracle

Integrated Cloud Applications & Platform Services

Oracle is committed to developing practices and products that help protect the environment

ORACLE