IBM Business Analytics Proven Practices: Dynamic Cubes Hardware Sizing Recommendations

Product(s): IBM Cognos Dynamic Cubes; Area of Interest: Performance

David Cushing
Software Engineer
IBM

24 June 2014
(First published 17 April 2014)

This document discusses some of the sizing variables involved with Dynamic Cubes and provides some high level estimates to help guide the user in selecting the proper resources for their environment. This document is intended for 10.2.1 FP1 and 10.2.1.1 IF3.

View more content in this series

Introduction

Purpose of Document

The purpose of this document is provide guidance to IBM employees and customers in regards to determining the minimum hardware requirements to provide a reasonable level of query performance and product stability from IBM Cognos Dynamic Cubes. After a brief description of the Dynamic Cubes architecture, the document outlines the information that is useful when determining the hardware requirements and then provides a simple means of using this information to obtain the CPU core and memory requirements for Dynamic Cubes.

The document then provides additional background description of how hardware requirements can be more accurately computed for a particular application.

With the addition of Dynamic Cubes, the Dynamic Query Mode (DQM) server becomes an in-memory database meaning additional CPU cores and memory are required over and above what is typically required for the DQM server.

The recommendations here do not account for the resources required for the Dispatcher or the accompanying Report Server processes on the same server.
Applicability

This document is intended only for IBM Cognos 10.2.1 FP1 (10.2.1.1) and 10.2.1.1 IF3 (Interim Fix 3). This document is not intended for previous versions because changes were made to Dynamic Cubes to enhance the performance capabilities. In addition, later versions may have different hardware requirements. For further assistance or clarification please contact your local IBM Services group for assistance in performing a proper requirements gathering.

Overview Of The Dynamic Cubes Architecture

Dynamic Cubes exist within the Dynamic Query Mode (DQM) server. Only a single DQM server may exist under a single Dispatcher and it services queries from all of the Report Servers that reside under the Dispatcher. All of the cubes that are configured to run on a single Dispatcher must have all of the necessary hardware resources (cores and memory) available to the DQM server for the cubes to operate efficiently.

With Dynamic Cubes, the DQM server in essence becomes an in-memory repository of data. Because of the volumes of data that the DQM server is required to store and process, additional CPU cores and memory are required to support Dynamic Cubes over and above what is typically required for a DQM server. A DQM server will make use of available cores to improve Dynamic Cube performance and will make use of available memory, though there may be practical limits to the amount of memory that can be effectively used for Dynamic Cubes. This will be discussed later in this document.

It is very important to note that the sizing recommendations here do not account for the resources required for the Dispatcher or the accompanying Report Server processes.

There are three pieces of hardware which need to be sized for Dynamic Cubes - CPU cores, memory, and hard disk space. CPU cores are required for two purposes - to support concurrent execution of user queries and to allow Dynamic Cubes to perform certain internal operations in parallel. Memory is required to contain all of the dimensional members of a cube in memory and to provide adequate space in memory to store data that is retained in its various caches. Finally, hard disk space is required for the result set cache associated with each cube.

Collecting Information About Your Application

Prior to estimating the hardware requirements for Dynamic Cubes, it would be helpful to gather the following information (listed in order of importance) or at least obtain estimates for each item below. If you need to estimate, it is usually better to over-estimate - if you under-estimate, performance could suffer.

- The total number of cubes being deployed. If possible, identify which cubes are virtual. If unsure, assume all the cubes are base cubes and none are virtual.
- If there are virtual cubes, determine if any of the cubes upon which they are based are directly accessible or not.
- For each cube, the number of named users granted access to the cube.
• For each cube, the number of members in total across the hierarchies which are significantly larger (typically by orders of magnitude) than all others.
• For each of the largest hierarchies, the number of members in each hierarchy.
• The number of measures.
• The number of hierarchies.

High Level Sizing Recommendations for Individual Cubes

To allow for the easy estimation of hardware requirements for Dynamic Cubes, guidance is provided based on small, medium, large, and extra-large configurations, each of which is distinguished by a range in which applications may fall. Graphs are also provided which may also be used to estimate requirements when an application's values fall outside of those in the guidance.

From smallest to largest, the configurations can be briefly described as follows:

• **Small** - Development environment scale or small line-of-business application. Small number of concurrent users and small data volume.
• **Medium** - Small to medium sized, enterprise-wide application.
• **Large** - Large enterprise, divisional application, accessing large data volumes.
• **Extra-large** - Enterprise-wide user access, core application, accessing very large data volumes at day, consumer, product (SKU) level of data.

Taking into account that the number of named users of an application and the amount of data within it are distinct from one another, the Table 1 provides a quick reference for the combination of small, medium, large, and extra-large user and data volume combinations. For the purposes of the guidance provided here, the assumption is that a cube contains 12 hierarchies.

**Table 1 - CPU core, memory and disk space recommendations for 10.2.1**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>CPU cores</th>
<th>Memory for 10.2.1 FP1</th>
<th>Memory for 10.2.1 IF3</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>1-4</td>
<td>1.1 GB</td>
<td>1.1 GB</td>
<td>1 GB</td>
</tr>
<tr>
<td>Medium</td>
<td>4-8</td>
<td>6.2 GB</td>
<td>5.7 GB</td>
<td>10 GB</td>
</tr>
<tr>
<td>Large</td>
<td>8-16</td>
<td>44.5 GB</td>
<td>42.3 GB</td>
<td>50 GB</td>
</tr>
<tr>
<td>Extra large</td>
<td>16-32</td>
<td>134 GB</td>
<td>130 GB</td>
<td>100 GB</td>
</tr>
</tbody>
</table>

Figures 1 and 2 are a graphical representation of the data in Table 1.
Detailed Sizing Recommendations For Dynamic Cubes

Number of CPU cores

The number of users can be expressed in three ways:

- **Named** users are all people who have access to an IBM Cognos 10 application.
- **Active** users are a subset of the Named user community who are logged on to the system and may or may not be interacting with IBM Cognos 10 system at any given time.
- **Concurrent** users are a subset of Active user community who at a given time are either submitting a request or already have a request in progress.

The standard relationship used is 100 named users equates to 10 active users which in turn equates to a single concurrent user. The recommendations that follow are based on the number of concurrent users as 1% of the number of named users. At the end of this document is a reference to the number of concurrent requests, which at their simplest are equivalent to the number of concurrent users. Depending upon the overall application workload (e.g. presence of Workspace Advanced reports and active dashboards) it is likely that the number of concurrent requests will actually increase, which will increase the overall hardware requirements for Dynamic Cubes.
As the number of users increases, so does the number of cores required to process the queries concurrently. As mentioned before, Dynamic Cubes can take advantage of additional CPU cores to perform internal operations in parallel, the result of which is that individual queries perform faster.

Table 2 outlines the suggested number of cores based on the number of named users.

**Table 2 - Number of CPU cores based on named users**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Number of named users</th>
<th>Number of CPU cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>&lt; 100</td>
<td>4</td>
</tr>
<tr>
<td>Medium</td>
<td>100 - 1000</td>
<td>4 - 8</td>
</tr>
<tr>
<td>Large</td>
<td>1000 - 5000</td>
<td>8 - 16</td>
</tr>
<tr>
<td>Extra Large</td>
<td>5000 - 10000</td>
<td>16 - 32</td>
</tr>
</tbody>
</table>

Scaling beyond 10,000 users, the general rule is to add a minimum of 1 core per 10 concurrent users above 10,000 named users. For example, if there are 25,000 named users, this means 32 + 15 = 47 ~ 48 cores.

One of the assumptions behind the recommendations above is that the volume of data processed by Dynamic Cubes itself remains relatively constant although the amount of data processed in the relational database may increase. If this assumption does not hold true, it may be prudent to add additional cores based on the size of the aggregate and data caches, allotting an additional core for each 5 GB of memory used. This is for the total memory used by the aggregate and data caches combined.

Figure 3 is a graphical representation of the data in Table 2.

**Figure 3 - IBM Cognos 10.2 named user CPU core recommendations using the data from Table 2**

---

The more cores on your machine the better. Any available cores will not be wasted - those which are not used to execute queries concurrently will be used by Dynamic Cubes to perform internal operations in parallel, which will improve individual query performance.
Memory requirements

Dynamic Cubes uses memory for storing the members of all hierarchies, storing data in the data cache and the aggregate cache and providing temporary space for the execution of MDX queries within the DQM server. In order to compute the total amount of memory required for a cube, the following computation can be used:

<member cache size> + <data cache size> + <aggregate cache size> + <temporary query space>

Member cache

Dynamic Cubes loads all of the members of a cube’s hierarchies into memory when a cube is started. The intent of this is to provide fast access to member metadata for not just the metadata browser in the various BI studios but also to the DQM server during query planning and to the SQL generation component of Dynamic Cubes, which converts member unique names into SQL filter expressions based on the relational key column values stored in the member cache. Once a cube has been started, Dynamic Cubes will only ever access the relational database for measure values that are not stored in one of its several caches.

The estimated amount of memory required to store a member in a Dynamic Cube in 10.2.1 FP1 is 700 bytes on a 64-bit Java Virtual Machine (JVM) and in 10.2.1.1 IF3 it is 550 bytes on a 64-bit JVM.

Since in most cases the largest hierarchies typically dwarf the size of all other hierarchies by orders of magnitude, the estimated size of the member cache is the sum of the number of members in these hierarchies multiplied by 700 bytes in 10.2.1 FP1 and multiplied by 550 bytes in 10.2.1.1 IF3.

The following equation can be used to compute the amount of memory required for the member cache in 10.2.1 FP1:

<# of members in largest hierarchies> * 700 bytes

The following equation can be used to compute the amount of memory required for the member cache in 10.2.1.1 IF3:

<# of members in largest hierarchies> * 550 bytes

The number of members in the largest hierarchies is outlined in the table below.

Table 3 - Number of members in largest hierarchies

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Number of members in largest hierarchies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>600,000</td>
</tr>
<tr>
<td>Medium</td>
<td>3,000,000</td>
</tr>
<tr>
<td>Large</td>
<td>15,000,000</td>
</tr>
<tr>
<td>Extra Large</td>
<td>30,000,000</td>
</tr>
</tbody>
</table>
The bytes per member number does not take into account the presence of additional member properties or multiple locales in a base cube. The following comments do not apply to members in a virtual cube:

- Each locale added to a cube adds approximately 100 bytes to each member.
- In both 10.2.1 FP1 and 10.2.1.1 IF3, each distinct attribute value, for each locale, requires approximately 200 bytes of memory.
- Members that share the same value of an attribute amortize those 200 bytes amongst themselves, effectively reducing the number of bytes per attribute value.

To compute the space required for member attributes, in each of the two largest dimensions estimate the number of distinct values for the entire set of attributes for the members of the dimension. The amount of space required for the attribute is computed as:

\[
\frac{(# \text{ of members}) \times 200 \text{ bytes} \times (\text{number of distinct attribute values})}{(\text{total number of attribute values})}
\]

The above value then needs to be multiplied by the number of locales.

**Calculating the Overall Member Cache Size**

The overall size of the member cache for a base cube, must account for the size of the members, any additional attributes, and the number of locales in a cube model. This can be expressed as:

\[
(# \text{ of members}) \times (\text{memory per member} + \text{memory per member per locale} \times (# \text{ of additional locales}) + \text{memory per member per attribute} \times \frac{(\text{number of distinct attribute values})}{(\text{total # of attribute values})} \times (# \text{ of all model locales})
\]

**Data cache background**

At first glance it would seem that the best way to estimate the amount of memory required for the data cache would be to base the requirements on the number of rows in the fact table. However, what is more important is the number of possible data points in the cube that is built upon the data warehouse and the manner in which those data points are accessed in reports and analyses.

The possible number of data points can be computed by multiplying the number of members in all of the hierarchies, including the measure hierarchy - effectively computing the size of the cross join of all the hierarchies. In any data warehouse of an appreciable size, the possible number of values present in a cube is in the billions, trillions or more. This number is entirely independent of the number of rows in the fact table, though the number of rows in the fact table does provide a good indication of the density of the cube at the leaf level of all hierarchies. The number is usually many of orders of magnitude less than the theoretical number of data points in a cube and of all of the possible data points in a cube, it is expected that users as a whole will access a small common subset, as well as individual, smaller portions of the cube that are pertinent to their individual data exploration. This is important because a Dynamic Cube only loads data on demand and all detail facts, unless explicitly required by a query, remain in the relational database.
For example, if a fact table contained 10 billion rows of fact data for 10 years of data and a query requested the annual sales totals for each year for 25 sales districts, the dynamic cube would only store 250 data points in its data cache – even though the database was required to read all data in the fact table.

It is typically only when queries filter large, lower level sets of members by measure values in the largest hierarchies of a cube that a large number of values are stored in the data cache. These sorts of queries could be posed as part of a cube’s start up trigger, allowing the intermediate result sets to be stored in the expression cache so the results are stored, even though the data in the data cache may be flushed during the course of other query processing.

From a reporting/analysis point of view, limiting the filtering of entire swaths of a leaf level of a large hierarchy can help in reducing the required size of the data cache – restricting such filter expressions by a parent level member at a higher level can make a significant impact (e.g. filtering customers within a region as opposed to all customers of an entire organization).

It is also important to keep in mind that the data cache has a fixed size and will evict earlier query results if it approaches its maximum size. Dynamic Cubes uses a heuristic to determine which query results it should evict from its cache. If data that was once retrieved is evicted, a subsequent request will cause the Dynamic Cube to re-retrieve the data from the underlying database.

Note, that over the course of a business day, users may explore data, most likely making use of data previously retrieved by other users running the same or other related reports. During that time, some data will be viewed a number of times and then not examined again – it becomes stale and is an ideal candidate for eviction. The removal of these query results from the data cache does not impact performance but does allow the data cache to provide space for subsequent queries retrieving data which has not yet been viewed.

**Computing memory requirements for the data cache**

The size of the data cache depends on the number of cells required by reports, analyses and dashboards, as well as the presence of an aggregate cache. Since it is difficult to forecast report/analysis behavior, the data cache size is estimated based on the size of the dimensional space, which is an estimate based on the size of the member cache. As the size of the dimensional space grows, so will the size of the data cache.

Note that each cell stored in a Dynamic Cube will consume 20 bytes in a 64-bit JVM. In addition, once the data cache consumes 90% of its allotted memory, Dynamic Cubes will evict the least accessed portions of the cache so the entire amount of space allotted to a data cache is not used except for short intervals. As a result, approximately each gigabyte of Dynamic Cube’s data cache will store 45-50 million cells (values).

In the calculation below, note that the presence of an aggregate cache reduces the amount of space required for the data cache, as the data cache needs to contain fewer values. The aggregate cache will be discussed in the next section. As well, each user is likely to retrieve some data which is specific to the report they are executing, even if they are possibly running the same
report (e.g. prompt values are different). In addition, every 5,000 named users require an additional 1 GB of memory. This is known as the **user factor** and is calculated as follows,

\[
\text{user factor} = (\# \text{ of named users} \times 200K)
\]

If an aggregate cache is present then in 10.2.1 FP1 the minimum data cache size is:

The greater of 
\((15\% \times \text{member cache size} + \text{user factor})\) or 
\((28\% \times \text{member cache size} + \text{user factor}) - (\text{size of the aggregate cache})\)

In 10.2.1.1 IF3 the minimum data cache size if an aggregate cache is present is:

The greater of 
\((19\% \times \text{member cache size} + \text{user factor})\) or 
\((36\% \times \text{member cache size} + \text{user factor}) - (\text{size of the aggregate cache})\)

If an aggregate cache isn't present then in 10.2.1 FP1 the minimum data cache size is:

\((28\% \times \text{member cache size}) + \text{user factor}\)

In 10.2.1.1 IF3 the minimum data cache size is:

\((36\% \times \text{member cache size}) + \text{user factor}\)

Note that this is a minimum requirement - allotting more memory for the data cache will improve performance as the system gets used. The aggregate cache is a case where "more is not necessarily better". The more aggregates that are defined mean more objects in the system. Give the amount of memory needed to hold the aggregates necessary for your workload, and be wary going beyond that number even if you have the memory space defined by the limits above. In most cases, it is the time available in which to load the in-memory aggregates and the time required to construct or update the supporting in-database summary tables that is the limiting factor to the size of the aggregate cache.

**Computing the size of the aggregate cache**

The calculations for the size of the aggregate cache are based on the size of a unilingual member cache with no additional member attributes. The aggregate cache contains a collection of measure values at the intersection of different combinations of levels from various hierarchies in a cube. The aggregate cache can satisfy many data requests without the need to retrieve values from the database, which reduces the amount of data that needs to be stored in the data cache. This is especially true since the aggregate cache's contents are based on user workload - the contents are tuned to maximize the use of the data in the aggregate cache. As a result, when an aggregate cache is present, it can make sense to reduce the size of the data cache.

The contents of the aggregate cache are intended to provide quick access to values computed at higher levels of aggregation in a cube. Though the amount of values can grow as a cube's overall size increases, the amount of data required in the aggregate cache is not necessarily linearly correlated to the increase in cube's size. As a result, a sliding scale is used to compute the minimum size of the aggregate cache relative to the size of the member cache.
Table 4 - Aggregate cache estimates based on member cache

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Aggregate cache size as % of member cache in 10.2.1 FP1</th>
<th>Aggregate cache size as % of member cache in 10.2.1.1 IF3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>100%</td>
<td>127%</td>
</tr>
<tr>
<td>Medium</td>
<td>85%</td>
<td>109%</td>
</tr>
<tr>
<td>Large</td>
<td>45%</td>
<td>57%</td>
</tr>
<tr>
<td>Extra Large</td>
<td>45%</td>
<td>57%</td>
</tr>
</tbody>
</table>

Temporary query space

When computing memory requirements for the temporary query space, it is necessary to take into account the amount of memory required on average as well as the occasional query that evaluates large volumes of data (e.g. choosing the top 10 customers in a year's sales) - this is captured in Table 5 below as Peak Query Memory.

Each concurrent query executed against a Dynamic Cube requires space for the construction of intermediate result sets for use within the DQM MDX engine. In general, there is an average per query overhead that needs to be accounted for as well as space allotted for one or more queries which are atypical and may retrieve large volumes of data, requiring large amounts of memory to transfer the values within the engine.

On a server which hosts multiple cubes, the assumption is that a single user with access to multiple cubes executes a query on only one of those cubes at a time. Consequently, the amount of memory required for temporary query space is computed for each logical group of concurrent users and the cubes to which they have access. These individual values are added together to obtain the amount of memory required on the server to support concurrent queries across all the groups of users and the cubes to which they have access.

The calculation to compute the memory required for query processing for a group of users and the cubes to which they have access is as follows:

Concurrent query usage size = [size of max query memory usage] + [average memory usage per query] * (# of concurrent users - 1)

Note that the average memory usage per query value does not include the size of the maximum query.

In 10.2.1 FP1, average memory usage per query is approximately 4.5% of the size of the largest member cache amongst the cubes to which the group of users have access. In 10.2.1.1 IF3, average memory usage per query is approximately 5.7% of the size of the largest member cache amongst the cubes to which the group of users have access.

In 10.2.1 FP1, size of max query memory usage is approximately 45% of the size of this member cache. In 10.2.1.1 IF3, size of max query memory usage is approximately 57% of the size of this member cache. The guidance is to account for at least one maximum usage query, though a more thorough assessment of the entire application workload by a Cognos field representative may
indicate that there is more than one such concurrent query, in which case the computation above should be adjusted accordingly to allot enough memory for these types of queries to execute concurrently.

The overall amount of memory required for temporary query space can be calculated as follows:

\[
\text{Memory required} = (\text{Concurrent query usage size for group 1}) + (\text{Concurrent query usage for group 2}) + \ldots + (\text{Concurrent query usage for group N})
\]

**Miscellaneous adjustments to the memory requirements**

If it is expected that the member cache will be refreshed, then it is necessary to double the amount of memory required for the member cache - Dynamic Cubes builds the new member cache in the background while the current member cache is being used. As a consequence, for a short time both member caches are present in memory.

If a cube exists solely for use within a virtual cube and it is not accessible from any reporting packages, it may not be necessary to assign a data cache to the cube since data will be cached in the virtual cube which is being accessed. Since the MDX engine is not invoked for intermediate cubes which are not accessed directly by users, there is no need to account for the peak user query. Some memory should still be allotted to account for space required to transfer values within the cube, so the amount allotted for average queries can be used to account for this.

If a virtual cube is built to combine historic and recent data into a single cube, it may make sense to assign a data cache to the base cube as this will ensure fast query performance when the recent data cube is updated.

It is important to note that when a server supports virtual cubes, any cube (base or virtual) which is not accessible via a package has no need for a data cache - it is only important to cache data for the cubes which are accessible by users. Also, when estimating the size of cubes, note that virtual cubes do not have aggregate caches.

**Hard Disk Recommendations for Dynamic Cubes**

The result set cache retains a copy of the result of MDX queries executed against a Dynamic Cube on disk in a binary format. Entries in the cache are identified by the MDX query and the combination of security views of the user who executed the query. This information is stored in a small, in-memory index that is used to quickly search for cache entries. The result of queries are only added to the cache if they exceed a predefined, minimum query execution time to ensure the cache is not populated with the result of queries which are already fast to execute. This value is configurable in the administration console on a per cube basis.

The benefits of the result set cache are most obvious when a group of users are executing a common set of managed reports that have a limited number of prompts. The more opportunities there are for differences in a report's specification, the less often the result set cache will be used. Ad hoc analysis is unlikely to make much use of the result set cache, except in cases where a user drills up, in which case a previous analysis is re-executed and can make use of the result set cache.
cache. Consequently, the retention of an MDX result set in the cache does not necessarily benefit subsequent users, simply because there are limited chances that the exact same query will be executed in the future.

The size of the result set cache is tied to the number of active users - the more active users there are, the larger the cache needs to be to retain MDX query result sets. Result sets are typically very small, approximately 10K to 20K in size. The average size is conservatively estimated at 50K per result set. Each user is allotted 200 entries in the cache, which equates to 10 MB of disk space per active user or 100K per named user.

**Table 5 - Disk space estimates**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Estimated disk space requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>1 GB</td>
</tr>
<tr>
<td>Medium</td>
<td>10 GB</td>
</tr>
<tr>
<td>Large</td>
<td>50 GB</td>
</tr>
<tr>
<td>Extra Large</td>
<td>100 GB</td>
</tr>
</tbody>
</table>

A result set cache is only required for cubes which are directly accessible by users, that is there is at least one published package which includes the cube as a data source.

- Final base JVM heap size was computed as
  
  \[ \text{Final base JVM heap size} = \text{Member cache size} + \text{aggregate cache size} + \text{data cache size} + \text{average query memory per user} \times (\# \text{concurrent users} - 1) + \text{peak query memory} \]
  
  \[ = 3 \text{ GB} + 1.8 \text{ GB} + 500 \text{ MB} + 90 \text{ MB} \times (10 - 1) + 900 \text{ MB} \]
  
  \[ = 7 \text{ GB} \]

**Recommended Hardware Sizing for Various Scenarios 10.2.1 FP1**

Table 6 provides the details behind the initial, high level sizing table earlier in this document. The calculations used to compute the values in the table are those described above. This is strictly meant as an example of how these values can be computed. The **Final Base JVM Heap Size** column represents the amount of memory required for a Dynamic Cube. The table below contains the values discussed earlier in this document as a single source reference.

**Table 6 - All Sizing Estimates from Document for 10.2.1 FP1**

<table>
<thead>
<tr>
<th>Category</th>
<th>Number of Named Users</th>
<th>Number of Members in Two Largest Dimensions</th>
<th>Member Cache Size</th>
<th>Aggregate Cache Size</th>
<th>Data Cache Size</th>
<th>Average Query Memory per User</th>
<th>Peak Query Memory</th>
<th>Final Base JVM Heap Size</th>
<th>JVM Heap (member cache refresh support)</th>
<th>Hard Disk Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>100</td>
<td>600,000</td>
<td>420 MB</td>
<td>420 MB</td>
<td>83 MB</td>
<td>19 MB</td>
<td>190 MB</td>
<td>1.1 GB</td>
<td>1.5 GB</td>
<td>1 GB</td>
</tr>
<tr>
<td>Medium</td>
<td>1000</td>
<td>3,000,000</td>
<td>2.1 GB</td>
<td>1.8 GB</td>
<td>515 MB</td>
<td>95 MB</td>
<td>950 MB</td>
<td>6.2 GB</td>
<td>8.3 GB</td>
<td>10 GB</td>
</tr>
<tr>
<td>Large</td>
<td>5000</td>
<td>15,000,000</td>
<td>10.5 GB</td>
<td>4.7 GB</td>
<td>470 MB</td>
<td>4.7 GB</td>
<td>44.5 GB</td>
<td>55 GB</td>
<td>50 GB</td>
<td>50 GB</td>
</tr>
<tr>
<td>X Large</td>
<td>10000</td>
<td>30,000,000</td>
<td>21 GB</td>
<td>9.5 GB</td>
<td>950 MB</td>
<td>9.5 GB</td>
<td>134 GB</td>
<td>155 GB</td>
<td>100 GB</td>
<td></td>
</tr>
</tbody>
</table>
In the case of the medium configuration, the numbers were computed as follows:

- The size of the member cache was computed as
  \[= \text{# of members in largest hierarchies} \times 700 \text{ bytes}\]
  \[= 3,000,000 \times 700 \text{ bytes}\]
  \[= 2.1 \text{ GB}\]
- The size of the aggregate cache was computed as
  \[= \text{size of member cache} \times \text{medium percentage}\]
  \[= 2.1 \text{ GB} \times 85\%\]
  \[= 1.79 \text{ GB}\]
- The size of the data cache was computed as
  \[= \text{greater of two values:}\]
  1. Member cache size \(\times 15\% + \text{user factor}\)
     \[= 2.1 \text{ GB} \times 15\% + 200 \text{ KB} \times 1000\]
     \[= 315 \text{ MB} + 200 \text{ MB}\]
     \[= 515 \text{ MB}\]
  2. Member cache size \(\times 28\% + \text{user factor} - \text{size of aggregate cache}\)
     \[= 2.1 \text{ GB} \times 28\% + 200 \text{ KB} \times 1000 - 1.8 \text{ GB}\]
     \[-1 \text{ GB}\]
Minimum data cache size is 515 MB
- Average query memory per user was computed as
  \[= 2.1 \text{ GB} \times 4.5\%\]
  \[= 95 \text{ MB}\]
- Peak query memory was computed as
  \[= 2.1 \text{ GB} \times 45\%\]
  \[= 950 \text{ MB}\]
- Final base JVM heap size was computed as
  \[= \text{Member cache size} + \text{aggregate cache size} + \text{data cache size} + \text{average query memory per user} \times (\# \text{concurrent users} - 1) + \text{peak query memory}\]
  \[= 2.1 \text{ GB} + 1.8 \text{ GB} + 515 \text{ MB} + 95 \text{ MB} \times (10 - 1) + 950 \text{ MB}\]
  \[= 6.2 \text{ GB}\]

**Recommended Hardware Sizing for Various Scenarios 10.2.1.1 IF3**

Table 6 provides the details behind the initial, high level sizing table earlier in this document. The calculations used to compute the values in the table are those described above. This is strictly meant as an example of how these values can be computed. The **Final Base JVM Heap Size** column represents the amount of memory required for a Dynamic Cube. The table below contains the values discussed earlier in this document as a single source reference.

**Table 7 - All Sizing Estimates from Document for 10.2.1 FP1**

<table>
<thead>
<tr>
<th>Category</th>
<th>Number of Named Users</th>
<th>Number of Members in Two</th>
<th>Member Cache Size</th>
<th>Aggregate Cache Size</th>
<th>Data Cache Size</th>
<th>Average Query Memory per User</th>
<th>Peak Query Memory</th>
<th>Final Base JVM Heap Size</th>
<th>JVM Heap (member cache)</th>
<th>Hard Disk Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Largest Dimensions</td>
<td>refresh support)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>100</td>
<td>1.1 GB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>1000</td>
<td>5.7 GB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large</td>
<td>5000</td>
<td>42.3 GB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X Large</td>
<td>10000</td>
<td>130 GB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the case of the medium configuration, the numbers were computed as follows:

- The size of the member cache was computed as
  \[ \text{size} = \text{number of members in largest hierarchies} \times 700 \text{ bytes} \]
  \[ = 3,000,000 \times 550 \text{ bytes} \]
  \[ = 1.65 \text{ GB} \]

- The size of the aggregate cache was computed as
  \[ \text{size} = \text{member cache size} \times \text{medium percentage} \]
  \[ = 1.65 \text{ GB} \times 109\% \]
  \[ = 1.79 \text{ GB done to here} \]

- The size of the data cache was computed as
  \[ \text{size} = \text{greater of two values:} \]
  \[ 1. \text{ member cache size} \times 19\% + \text{user factor} \]
  \[ = 1.65 \text{ GB} \times 19\% + 200 \text{ KB} \times 1000 \]
  \[ = 314 \text{ MB} + 200 \text{ MB} \]
  \[ = 514 \text{ MB} \]

  \[ 2. \text{ member cache size} \times 36\% + \text{user factor} - \text{size of aggregate cache} \]
  \[ = 1.65 \text{ GB} \times 36\% + 200 \text{ KB} \times 1000 - 1.79 \text{ GB} \]
  \[ = 590 \text{ MB} + 200 \text{ MB} - 1.79 \text{ GB} \]
  \[ = -1 \text{ GB} \]

  Minimum data cache size is 515 MB

- Average query memory per user was computed as
  \[ \text{size} = 1.65 \text{ GB} \times 5.7\% \]
  \[ = 95 \text{ MB} \]

- Peak query memory was computed as
  \[ \text{size} = 1.79 \text{ GB} \times 57\% \]
  \[ = 950 \text{ MB} \]

- Final base JVM heap size was computed as
  \[ \text{size} = \text{member cache size} + \text{aggregate cache size} + \text{data cache size} + \text{average query memory per user} \times (\# \text{ concurrent users} - 1) + \text{peak query memory} \]
  \[ = 1.65 \text{ GB} + 1.8 \text{ GB} + 515 \text{ MB} + 95 \text{ MB} \times (10 - 1) + 950 \text{ MB} \]
  \[ = 5.7 \text{ GB} \]

## Advanced Topics

### Multiple cubes and cube refresh

The amount of memory required for cube refresh on a server with multiple cubes can be the size of the largest member cache amongst the set of cubes. If the cubes are refreshed one at a time,
additional memory is only required to retain a second member cache in memory at a time. The size of this additional space is that of the largest member cache amongst the set of cubes.

**Multiple cubes**

If there are multiple cubes hosted on a server, it is important to identify the groups of cubes to which sets of users are provided access. The sizing guidelines from earlier in this document for each of these groups of cubes and the individual CPU core requirements for the groups are added together for the overall CPU core requirement for an individual server. For example:

- If a server hosts 4 cubes and all of the 1000 named users have access to all of these cubes, the CPU core requirement is 4 - 8 cores.
- If a server hosts 4 cubes and 100 named users have access to two of the cubes and 1000 have access to the other two cubes, the CPU core requirement is 8 - 12 cores.

**Multiple servers**

If a cube is deployed across multiple Dispatchers, the assumption is that the concurrent user load is distributed evenly across all of the Dispatchers (servers). Accordingly, the number of cores required on each machine is at least the number of estimated cores divided by the number of servers across which a cube is deployed. A cube requires a minimum of 1 core to be assigned to it (remembering that more is better).

Similarly, the amount of temporary query space is also affected. The number of concurrent users is divided by the number of servers and then the temporary query space calculated for each node individually. The important thing to note is that each server must account for at least one “peak” query.

**Using concurrent requests to estimate hardware requirements**

As mentioned earlier in the document, a more accurate means of estimating a cube’s workload, and hence its hardware requirements, is to estimate the concurrent request workload, as opposed to the number of concurrent users since a single user, depending upon their workload, may invoke multiple concurrent requests against a cube. The following table provides a range for the number of concurrent requests in the different configurations described earlier.

**Table 8 – Number of concurrent requests**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Number of Concurrent Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>1-10</td>
</tr>
<tr>
<td>Medium</td>
<td>10-40</td>
</tr>
<tr>
<td>Large</td>
<td>40-80</td>
</tr>
<tr>
<td>Extra Large</td>
<td>80-160</td>
</tr>
</tbody>
</table>

The same adjustments described above can also be applied to the number of concurrent requests for a cube (or group of cubes). Once the number of concurrent requests has been estimated, it is then possible to use that information to estimate the size of the data cache and the amount of temporary query space required for a cube or group of cubes.
Data cache based on concurrent requests

The calculation for the size of the data cache based on concurrent requests is a slight modification of the original calculation earlier in this document. The user factor is now computed as:

\[
\text{User factor} = \# \text{ of concurrent requests} \times 20 \text{ MB}
\]

Temporary query space based on concurrent requests

The calculation for temporary query space based on concurrent requests is relatively the same as the computation earlier in the document, except that instead of the number of concurrent users, the number of concurrent requests is used.

Shared Dimension Members

In the 10.2.1.1 IF3 release it is possible to share the members of a dimension between cubes on the same server. The benefit of this feature is that the members of large dimensions only need to be loaded into memory once, reducing both the time required to load multiple cubes (the members of shared dimensions are only loaded once) and the amount of memory required for the member cache of a set of cubes.

When computing the amount of memory required for individual cubes, the guidelines in this document should be followed. However, when multiple cubes are deployed on a single server and those cubes shared one or more dimensions, then the overall memory requirements should be adjusted by eliminating the space for each shared dimension except for a single instance of each shared dimension.
Resources

- For more information on Dynamic Cubes see the IBM Cognos Dynamic Cubes Redbook at http://www.redbooks.ibm.com/abstracts/sg248064.html?Open
About the author

David Cushing

David Cushing is a Software Engineer at IBM Cognos with over 24 years of experience. He has been responsible for the development of data access technologies in a range of products including Cognos Impromptu, Cognos ReportNet, Cognos 8 and Cognos 10, as well as IBM Cubing Services and most recently, IBM Cognos Dynamic Cubes.

© Copyright IBM Corporation 2014

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)