Dynamic Dataset Expansion

Fred White
Adager Corporation
Sun Valley, Idaho 83353-3000 • USA
http://www.adager.com

Introduction

A TurboIMAGE (or, for short, IMAGE) dataset is enabled for dynamic expansion by DBSCHEMA at RootFile creation time if the capacity specification for the dataset includes a maximum capacity, an initial capacity and an increment. For convenience, I will use *DDX (**detail** dynamic expansion)* and *MDX (**master** dynamic expansion)*.

When the database is created by DBUTIL, the EOF (end-of-file) and disc space allocated to a dynamic dataset is determined by the initial capacity. The file's LIMIT is determined by the maximum capacity. The increment is unknown to the File System but IMAGE maintains it in the RootFile along with the initial, current and maximum capacities.

You may use Adager to enable (or disable) an existing dataset for dynamic expansion (i.e., to change the various IMAGE dataset capacity values and their corresponding file-system LIMIT and EOF settings).

The term *dynamic* refers to the fact that, whenever DBPUT determines that a dataset requires additional capacity, DBPUT can perform an online expansion of that dataset (bumping the file's EOF to a higher value and updating the database's RootFile and the dataset's user label to reflect this increase in current capacity).

DDX for details

When a DBPUT occurs on a dynamic detail that is "full" (relative to its current capacity), DBPUT checks whether the current capacity is less than the maximum capacity. If "yes," DBPUT asks the file system to allocate additional disc space to accommodate the specified increment of capacity. (If the current capacity is equal to the maximum capacity, DBPUT returns a *dataset full* error code.)

If the disc Space Manager is able to fulfill the request, the newly acquired space is zeroed out, the EOF is increased in keeping with the increased size of the space allocated to the dataset, DBPUT updates the RootFile and the dataset's user label to match the new current capacity and then utilizes the newly acquired space to complete the DBPUT. Otherwise, DBPUT returns a *dataset full* error code.

Benefits of DDX for details

You may create your detail datasets with smaller initial capacities with little fear of encountering a *dataset full* condition. This minimizes your initial disc space requirements while still permitting your detail datasets to grow (or not) on an *as needed* basis.

Over time, some details may expand significantly and others little, if at all. DDX may help you minimizes disc space usage and backup times.

You may never need to shut down your application simply to increase the capacity of a detail dataset.

Adding entries to masters

Before discussing master datasets that have been enabled for dynamic expansion, let's understand how IMAGE goes about adding an entry to an ordinary master (i.e., a non-MDX master).

DBPUT uses the master's capacity and the new entry's *key* value to calculate the primary address of the new entry. DBPUT accesses the dataset block containing the primary address.

If the primary address location is empty, the new entry is placed there.

If the primary location is occupied by a secondary (of some other synonym chain), a search is made (see below) for an empty location and the secondary is moved (thus becoming a *migrating secondary*) to that location and the new entry is placed in its primary location.

If the primary location is occupied by a primary entry, DBPUT verifies that the new entry's key value doesn't match that of the primary entry and then traverses the synonym chain (if any) verifying that the new entry's key value doesn't match the key value of any of the synonyms (IMAGE does not allow duplicate key values for master entries). A search is then made (see below) for an empty location and the new entry is placed there and attached to the end of its synonym chain.

Searching for an empty location

DBPUT first checks the current block. If an empty location is not found in the current block, DBPUT cyclically searches successive blocks until an empty location is found.

The disc access and wall time to perform this search is usually quite small and significantly degrades performance only when the dataset is large and ***(a)*** through the mis-use of non-hashing keys (or the use of hashing key values whose primary addresses are not evenly distributed) one or more long clusters of occupied locations exist (even when the master is not nearly full) or ***(b)*** despite good distribution of entries the dataset becomes so full (>95%?) that sheer overcrowding results in long clusters which some searches are compelled to span.

Minimizing the performance degradation of long searches

If non-hashing keys have been mis-used, in some cases changing the capacity can result in all entries becoming primaries so that searching is never needed and the performance of DBPUT, DBFIND and directed DBGET is optimal. If your case doesn't lend itself to that solution, your other option is to convert the keys to hashing keys with Adager, edit your applications software to allow for the new data types, and recompile.

If your search performance is bad with hashing keys, your only option for regaining performance is to change capacity (increasing it if the master is nearly full).

These approaches may (or may not) be expensive, depending on your circumstances, but the resulting performance gains may be worth the price. The only way to find out is to experiment.

The two storage areas of MDX masters

Dynamic masters have a *primary* storage area (which ***must*** contain all of the primaries and ***may*** contain secondaries) and a *secondaries-only* storage area (initially non-existent).

At dataset creation time, only the disc space for the primary area is allocated and its size is determined by the value of the initial capacity.

Disc space is allocated for the secondaries-only area only when the master is dynamically expanded.

Additional expansions enlarge the secondaries-only area but the size of the primary area never changes.

Adding entries to MDX masters

DBPUT uses the initial capacity and the new entry's key value to calculate the primary address of the new entry.

It then accesses the IMAGE block corresponding to the primary address. If the primary address location is empty, the new

entry is placed there. If the primary location is occupied by a secondary (of some other synonym chain), a search is made for an empty location and the secondary is moved (thus becoming a migrating secondary) to that empty location and the new entry is placed in its primary location.

If the primary location is occupied by a primary entry, DBPUT verifies that the new entry's key value doesn't match that of the primary entry and then traverses the synonym chain (if any) verifying that the new key value doesn't match the key value of any of the synonyms (IMAGE does not allow duplicate key values for master entries). A search is made for an empty location and the new entry is placed there and attached to the end of its synonym chain.

Searching for an empty location in an MDX master

There are two possibilities, depending on the location of the current block.

*If the current block is within the **primary** area*

DBPUT checks the current block. If an empty location is not found in the current block, DBPUT cyclically searches successive blocks until an empty location is found or the cyclical searching becomes excessive, in which case DBPUT obtains an empty location from the secondaries-only area.

If the secondaries-only area is full or doesn't exist, DBPUT performs a dynamic expansion before obtaining the empty location.

DBPUT employs two thresholds to determine whether the search time is excessive. One is a count of the maximum number of IMAGE blocks to be examined. The other is a percent of the initial capacity.

A search time is considered to be excessive if either of these thresholds is reached without finding an empty location.

*If the current block is in the **secondaries-only** area*

DBPUT obtains an empty location directly from the secondaries-only area (whose space management is identical to that of detail datasets).

If the delete chain head is non-zero, its value provides the address of the empty location. Otherwise, the secondaries-only area's *HighWater Mark* (HWM) is incremented by one to provide the address of the empty location.

Benefits of MDX for masters

You can eliminate performance degradation caused by long searches for empty locations. You may never need to shut down your application to increase the capacity of a master dataset.

Of course, if your master employs hashing keys, you can obtain better performance (except for serial DBGET) right from the start (and in the long run) by specifying an initial capacity that provides at least as much capacity as you can afford and about 10% more than you may ever need. The problem with this solution is that you may not know how much capacity you need.

By providing a large primary area, you minimize the likelihood of synonyms and (with the exception of serial reads and backups) maximize performance.

By providing a small primary area you maximize synonyms and minimize performance.

Your motivation for using MDX masters should not be to conserve disc space. If you wish to conserve on disc space usage, focus your efforts on detail datasets, not on master datasets. If you do a good job with your detail datasets, you can be more generous with disc space for your master datasets.

If you specify MDX for a master, stipulate a modest increment just in case you have underestimated your needs (and solely to protect against search-time performance degradation).

The primary address calculation for *integer* (i.e., non-hashing) keys is predictable and such keys should never be used unless you know exactly what you're doing. See my paper, *The Use and Abuse of Non-hashing Keys,* in Adager's web site: *http://www.adager.com/TechnicalPapers.html*

However, if you have inherited a master with integer keys and your performance falls off a cliff, either ***(a)*** someone has mistakenly changed its capacity from the one carefully chosen by the designer to a capacity that destroyed the performance or ***(b)*** integer keys shouldn't have been used in the first place.

Shortcomings and Pitfalls

Jumbo datasets (i.e., exceeding 4 gigabytes in size) cannot be enabled for dynamic expansion.

Fortunately, IMAGE version C.10.05 introduced LargeFile datasets, which allow dynamic expansion and can be as large as 128 gigabytes. Unfortunately, a database can have either jumbo datasets or LargeFile datasets, but not both. Fortunately, you can use Adager to convert a database that has jumbo datasets into a database that supports LargeFile datasets. "Enable LargeFiles" is the Adager command that takes care of the mechanics, including the physical conversion of jumbo datasets (if any) into LargeFile datasets.

If your increments are too small and expansion occurs frequently, your disc space becomes fragmented over time.

If your increments are unnecessarily large, at expansion time the disc Space Manager may not be able to allocate the requested disc space.

To avoid these last two problems, continue to monitor and anticipate your system-wide disc space needs.

When you specify dynamic capacity expansion, don't request an unnecessarily large increment. A smaller increment has a better chance of being available at expansion time.

Furthermore, the expansion operation (typically a hiccup in your application's performance) can become an annoying pause when the increment is so large that it takes a long time for the disc Space Manager to allocate the space and for the space to be initialized.

Epilogue

The purpose of this paper was not to provide you with hard-and-fast rules on the use of DDX/MDX but rather to provide you with a basic understanding of how dynamic dataset expansion works and why you might want to use it so that you can make informed decisions when applying the concept to your databases.

Fred White (Senior Research Scientist at Adager from 1981 until his retirement in 2001) and Jonathan Bale worked together at Hewlett-Packard as leaders of the original IMAGE/3000 development team.