Sybase Cache Manager

Software Gems Pty Ltd
Derek Ignatius Asirvadem
V2.4
26 Jun 21

1 Introduction

Sybase ASE Cache Management is mature, and highly configurable; conveying its behaviour in a single document is not a simple task.

Assuming robust monitoring, every performance enhancement endeavour has the following characteristics, in order of presentation:

a. The currently visible performance bottleneck is identified, and contemplated
b. An enhancement to eliminate or substantially reduce it is determined, and implemented
c. The effect of the enhancement is monitored, and observed:
   c.1. If it eliminates the bottleneck:
   The next performance bottleneck is now visible; it was not visible before (a) had been addressed; return to (a) for the next cycle
   c.2 If it does not eliminate the current bottleneck, but it substantially reduces it:
   The enhancement is correct, but incomplete: return to (b)
   c.3. If it does not substantially reduce it:
   The enhancement is incorrect; reverse it; return to (b).

Sybase ASE Cache Management is the result of a series of performance enhancements, over two and a half decades; it constitutes a progression of improvements on a set of configurable resources, each of which has separate features, and all of which perform together. It is therefore best understood in the context of that chronology, rather than by attempting to understand the behaviour of specific components or parameters or features in isolation.

Discussing one feature or another without the context of the operation of the whole, as configured, is meaningless.

To that end, this document is a chronology; in order of appearance of the enhancements in that progression.

Overview

- A Data Cache consists of:
  - when created, **one** PageSize Buffer Pool
  - if additionally configured, Large I/O Buffer Pools, one each for each multiple of PageSize
- When a server is installed, several Data Caches, based on content and usage, are configured. In fact, all the Data Caches must be configured according to an overall plan. Our ***High Performance Configuration*** includes a full, server-specifc Cache configuration.
  - This paper defines the behaviour of any cache; overall Data Cache configuration is not included (paying customers only).

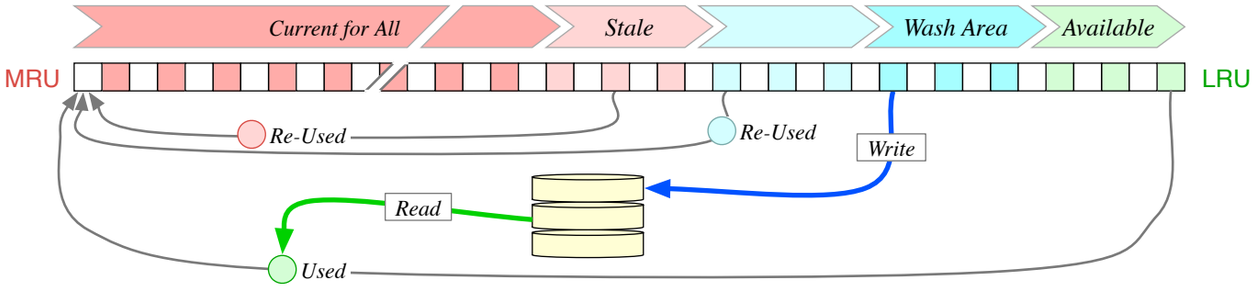
- Only the PageSize and one (the largest) I/O Large I/O Buffer pool are used; any additional Buffer Pools will be dead space, wasted, unused memory.
- There is one exception, irrelevant as exceptions are, because it does not mitigate the above statement: in an unconfigured server (where all objects use the default data cache), if the Transaction Log buffer size is larger than the PageSize, and a Buffer Pool for that size has been configured, it will be used.

- To begin with, we will look at a simple overview [this will be expanded as the document progresses]
- Each Buffer Pool is maintained as a chain, a linked list of buffers, in order from ***M**ost **R**ecently **U**sed*, to ***L**east **R**ecently **U**sed*.
  - for the purpose of understanding the operation of the Caches and Buffer Pools, it is best to visualise this as a *chain*
  - there is, therefore an MRU *end* and an LRU *end* to the chain

- To maintain its position in the chain, a buffer is not physically moved in memory; instead, the linkage (previous; next pointers) of the affected buffers in the chain are modified.
- To locate buffers, an efficient hash mechanism is used; the chain is not navigated.

- Due to activity and turnover within the Pool, buffers 'move' from the MRU end to the LRU end, making the MRU *warm* and the LRU *cool*
  - this is known as *ageing*

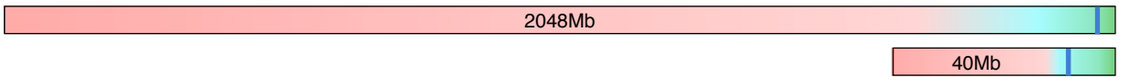
- The diagrams provide a progression, they show a single Data Cache, and initially, the PageSize Buffer Pool.
  - a buffer that is changed whilst in the cache is known as *dirty*; one that is not changed is *clean*
  - dirty buffers are coloured; clean buffers are white



- The **Status** of the buffers in the chain can be classified by age, as follows:
  - Fresh buffers are always taken from the LRU end (that is the purpose of the LRU classification)
  - Freshly Used and Re-Used buffers remain in the MRU end
  - dirty buffers are written to disk, before they reach the LRU end [more later]; the buffer is clean when the write completes
  - if a buffer reaches the LRU end, and it is clean, it is avsilable

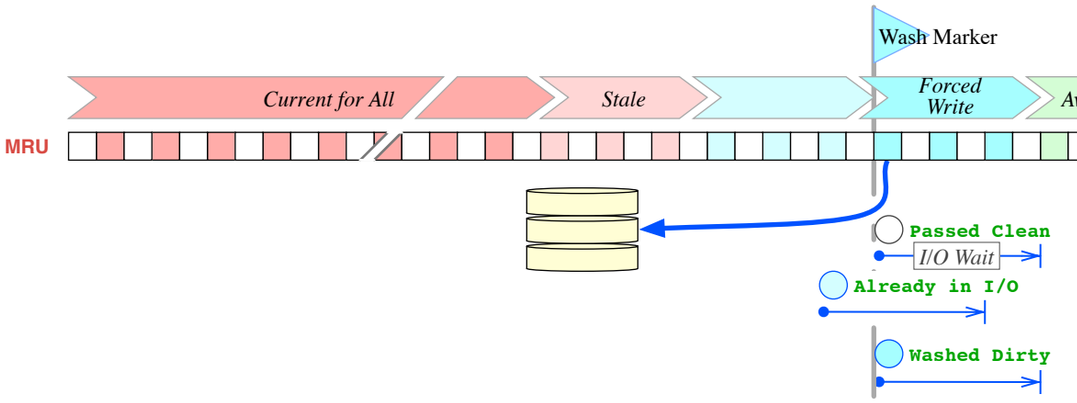
2 Wash Area

- The default Wash Size is 20% of the Pool Size, up to a maximum of 60MB; of course, the default can be overridden.
- The diagonal break in the diagrams indicates that the vast majority of buffers (minimum 80%) precede the Wash Marker; the Wash Area is emphasised for discussion purposes
- After configuration (ie. not default), the Wash Area is not proportionate to the Buffer Pool Size, it depends on the activity in the Wash Area; whether a Large I/O Pool exists; etc [more later].



Delayed Write

- Disk writes are *delayed* until necessary: this allows a buffer to be re-used several times, without writing it each time
  - Delayed Writes are implemented by the use of a demarcated area for writing buffers or *washing dirty* buffers, which is known as the *Wash Area*. It begins at the *Wash Marker*.
- When buffers reach the Wash Marker, they are in one of three states
  (`sysmon` terms are presented in `Courier Green`; Configuration parameters are presented as `Courier Blue`):
  - `Passed Clean` either unmodified, or modified; written to disk; write completed
  - `Already in I/O` the buffer was modified, a disk write has been initiated, but it is not complete
  - `Washed Dirty` the buffer was modified, but a disk write has not been initiated; a disk write is initiated.



There is a delay between I/O initiation and completion; the wait depends on the Device type; Disk I/O Structures; raw disk speed; and how well the I/O subsystem is tuned

Disk Write

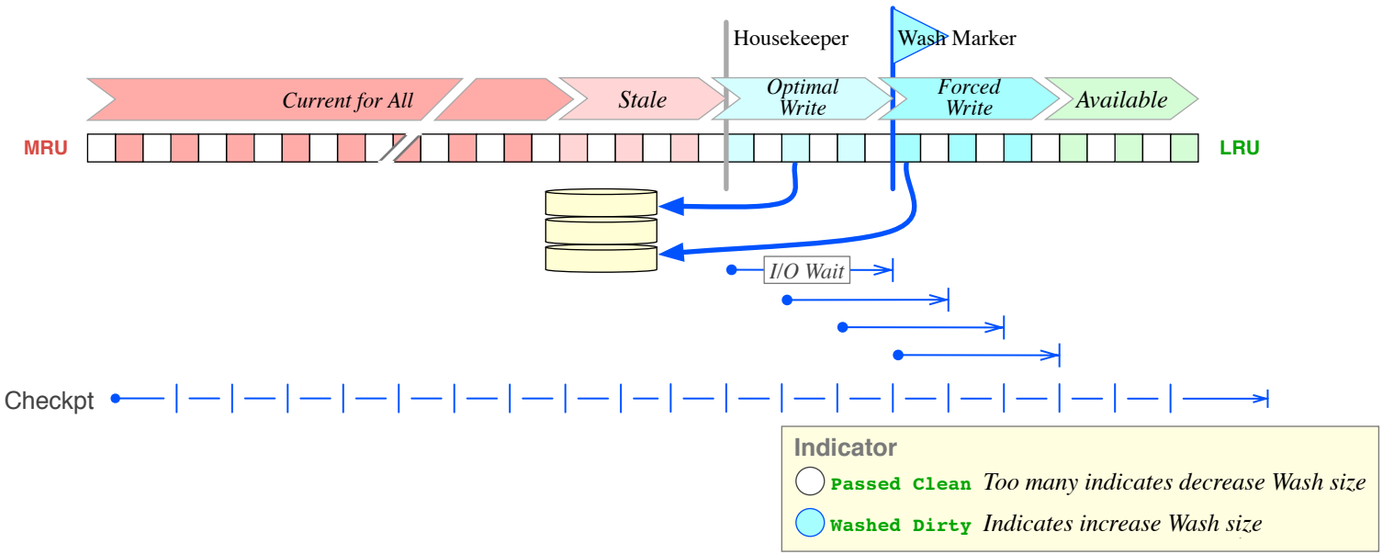
Disk Writes take place when any of the following events occur:

- The `CHECKPOINT SLEEP` task writes *all* dirty buffers to disk; it executes in intervals based on **`recovery interval`** (ancient).
- The `CHECKPOINT` command writes *all* dirty buffers belonging to the affected database to disk.
- The `HOUSEKEEPER WASH` task writes dirty buffers to disk during idle CPU cycles, ie. at non-contentious moments. The frequency is based on `housekeeper free write percent`. Whether some or all dirty buffers are written depends on those two variables.

  - This can be disabled on a Cache basis (all Buffer Pools in the Cache) via `HK IGNORE CACHE`. Unfortunately this cannot be affected via `sp_cacheconfig`; it must be affected by editing the configuration file directly.

- The Transaction Log is a **Write-Ahead Log**: Log records are written immediately upon `COMMIT` [for simplicity. noting that additional conditions apply], but generally not index or data pages; they are allowed to be re-used as above. However, if certain pages (eg. Index Non-Leaf or OAM) are changed, they must be written immediately. Therefore `COMMIT` also writes data pages, but only a few.
- A dirty buffer crosses the Wash Marker

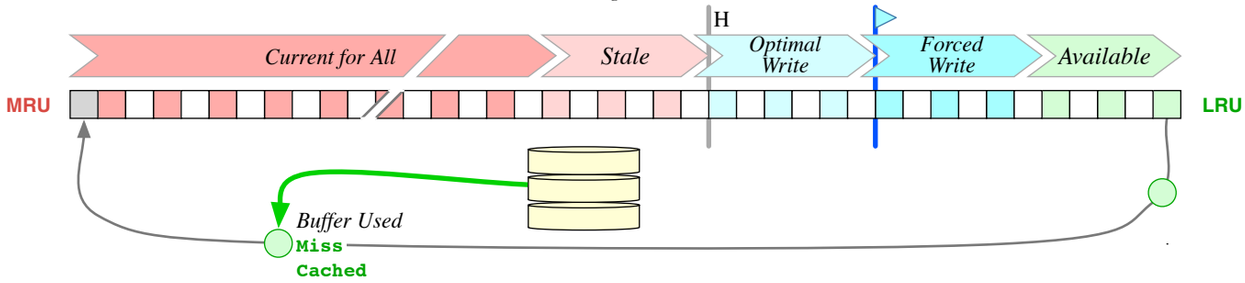
  - After entering the Wash Area, writes may still not be initiated: if they exceed limits; or there are not enough `disk i/o structures` available.



3 Normal Strategy

Buffer Use

- Whenever a buffer is required, it is always taken from the LRU end: this is an **`LRU Buffer Grab`**.
- Any access to a buffer indicates a Logical I/O; the **`Miss`** indicates a Physical I/O
- The buffer is used, which increments the **`Cached`** count; the page is read into it; and it is 'placed' at the MRU end
- This is the default for **Random Reads**; it is also known as *Page I/O*

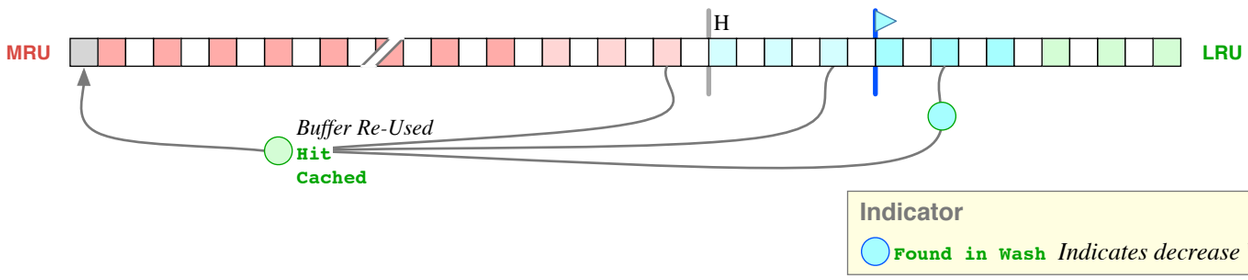


- The Normal strategy, unfortunately, and contrarily, is called **LRU Buffer Replacement Strategy**.
  - *It should have been called* **Normal**, *or* **MRU**, *or* **Cached**, *because it maintains the warmth of the Buffer Pool by prioritising buffers that are MRU over those that are LRU, and the buffer is 'placed' at the MRU end*
  - This is confirmed in the **`sysmon`** name, which identifies it correctly, but has to bracket the incorrect name to clarify the confusion:
    `Cache Strategy/Cached (LRU) Buffers`

> **Warning**: The manuals are particularly bad re Cache Behaviour, and sometimes state exactly the *opposite* of the correct information.

Buffer Re-Use

- If a buffer that is requested (Logical I/O) is already resident in the Buffer Pool, it is re-used; it is simply 'moved' to the MRU end.
- This is a **`Hit`**; a Physical I/O is avoided
- If the buffer was found in the Wash Area, the **`Found in Wash`** counter is incremented
  - This means that a disk write was initiated, therefore indicating that the Wash Area is too large.



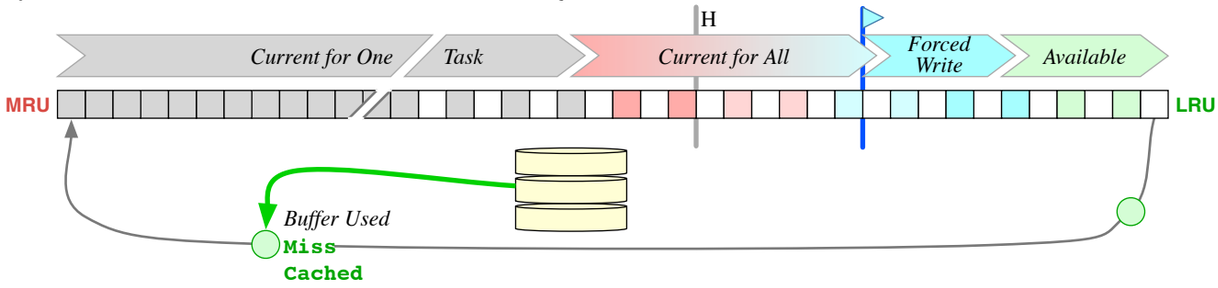
Indicator

`Found in Wash` *Indicates decrease Wash size*

4 Fetch & Discard Strategy

Data or Index Scan

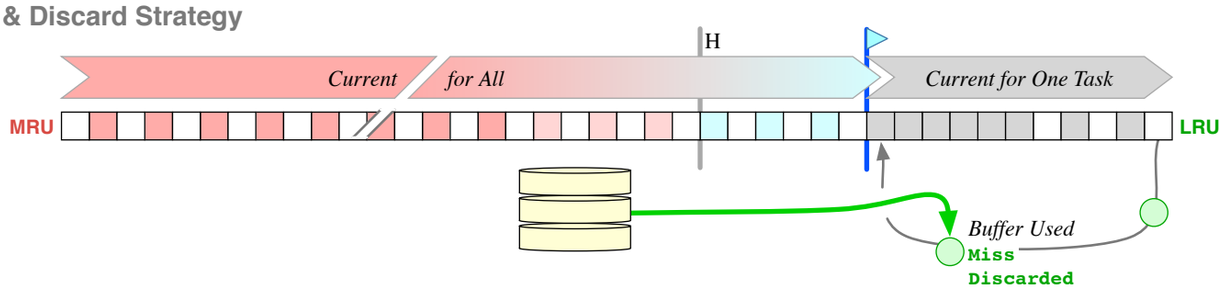
- The Normal strategy works extremely well, except for one case: where a large number of pages are read into the Buffer Pool for one task (eg. "Table Scan" [1] or Range Scan [2] or Index Scan [3]), the buffers would all be replaced, and the currency or warmth of the Buffer Pool would be lost. The Buffer Pool would be iced over. Buffers would be written prematurely. This is especially problematic if the number of buffers being read is a large proportion of the Buffer Pool.
- Grey denotes buffers that would have been read for the subject task.



- *Sybase ASE* does not do that, the diagram above is for explanatory purposes only.

- For cases where a large number of pages are being read for one task (and also upon request), there are two available options:
  - *Sybase ASE* uses the Large I/O Pool, if one exists (chapter 6)
  - if not, it uses an alternate caching strategy in the PageSize Pool (below).
- We will examine the two options in the chronological order of their advent, as they are a progression. The alternate strategy was available from the outset, 4.9.2; the Large I/O Pools came with 11.0.

Fetch & Discard Strategy



- When this strategy is invoked, as always, an available buffer is taken from the LRU end; the page is read into it; but it is 'placed' at the Wash Marker, thereby *discarding* it; rather than at the MRU end, which would *cache* it.
- Unfortunately, and contrarily, this strategy is called **MRU Buffer Replacement Strategy**
  - *It should have been called* **LRU**, *because it maintains the warmth of the Buffer Pool, and the buffers remain at the LRU end*
  - **Fetch & Discard** *is an even better name, because that reflects exactly what it does*
  - This is confirmed in the `sysmon` name, which identifies it correctly, but has to bracket the incorrect name to clarify the confusion:
    `Cache Strategy/Discarded (MRU) Buffers`

Warning: The manuals are particularly bad re the Cache Manager, and sometimes state exactly the *opposite* of the correct information.

- Fetch & Discard is chosen by the Optimiser when the number of Buffers required is more than half the Buffer Pool size.
- The goal is the same in both strategies: to maintain the currency or warmth of the Buffer Pool for all tasks. These buffers are required for one task, they are unlikely to be used by any other task
- In effect, the task re-uses the buffers that it has released, especially if the number of buffers it requires is greater than the Wash Size
- This strategy has the additional effect of *further* delaying disk writes, because modified buffers (which precede the Wash Marker) are prevented from entering the Wash Area.
- In this case (Fetch & Discard in the PageSize pool, where there is no Large I/O Pool) the Wash Area is used for two disparate purposes, and care must be taken in order to avoid conflict: it defines the point at which dirty buffers are written; and it denotes the number of buffers available for Fetch & Discard activity.
  - If Fetch & Discard Strategy is used frequently, the Wash Size needs to be increased in order to supply enough Buffers. However, that interferes with the need to keep the Wash Size small in order to delay writes.

---

1 A table does not exist as such in the physical realm, a table is a logical concept, it is a collection of one or more physical **DataStructures**. Therefore there is no such thing as a *Table Scan*, more precisely it is a *Data Scan* (compare with *Index Scan*), the term refers to the operation of scanning the DataStructure that contains the data rows:
- APL: the Leaf level of the Clustered Index or the Heap, via the PageChain
- DOL: the Heap, via OAM method.

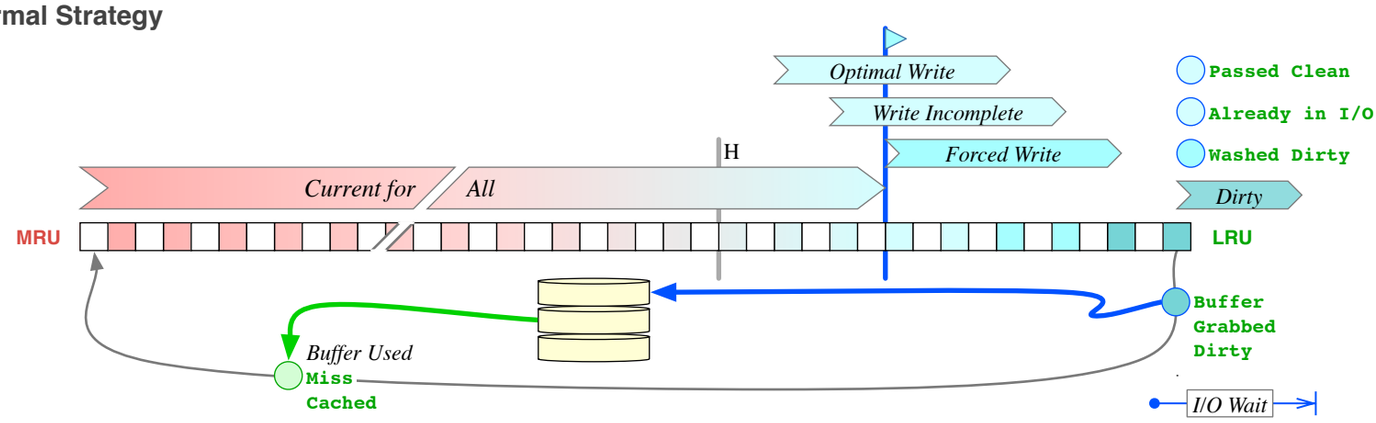
Note that there is no Clustered Index for a DOL table, it is a **Non-Clustered Index** plus a couple of parameters. The correct term is **Placement Index**, and it has none of the features of a **Clustered Index**. In command syntax, to address either the Heap or the Placement Index of a DOL table, *Sybase* forces the false term to be used.

2 A Range Scan is possible only for a Clustered Index. It reads the Leaf level of the CI via the PageChain, starting at a specific location.

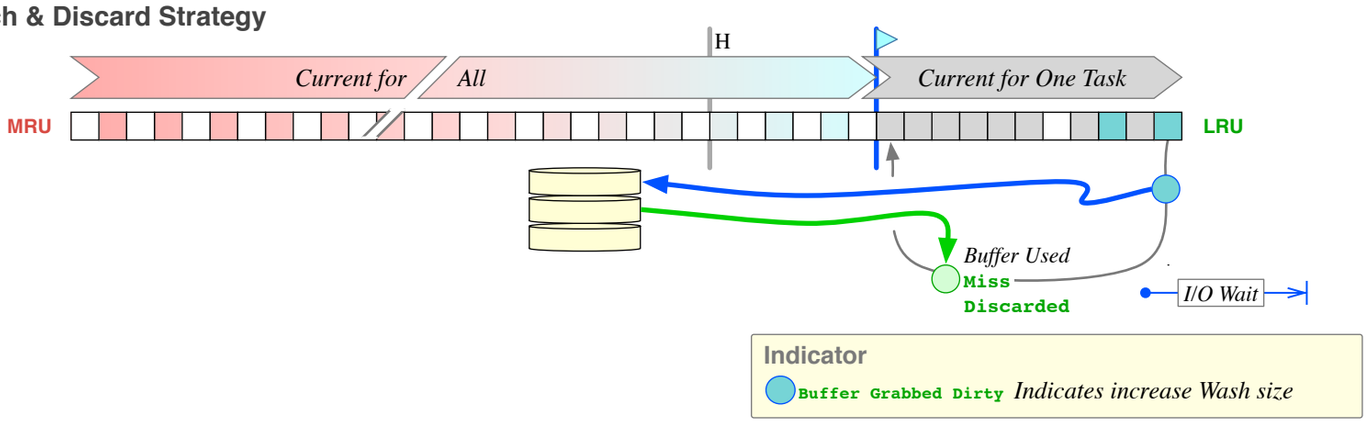
3 An *Index Scan* reads the Leaf level pages of a Non-Clustered Index, which contain the index entries, via the PageChain. Reading the Non-Leaf level uses Page I/O.

- This issue occurs in the PageSize Pool only, when the Cache does not have a Large I/O Pool.
  - A different flavour of it occurs in Large I/O Pools.
- When the Buffer Pool is overused, or the Wash Size is too small for the throughput, or the I/O subsystem is saturated, modified buffers do not get written to disk before reaching the LRU end, the Least Recently Used buffer, which is required for the task, is dirty (*not* **`Washed Dirty`**, which is write-initiated at the Wash Marker; *not* **`Already in I/O`**, which is write-initiated and incomplete at the Wash Marker).
- This is a **`Buffer Grabbed Dirty`**, it causes the task seeking a buffer to initiate the disk write, and to wait for its completion.
- It is known as *Buffer Starvation*. It has a substantial performance impact, as it *stalls* the task.
- It can occur with either Cache strategy, the effect is the same.

Normal Strategy



Fetch & Discard Strategy



Wash Size Tuning

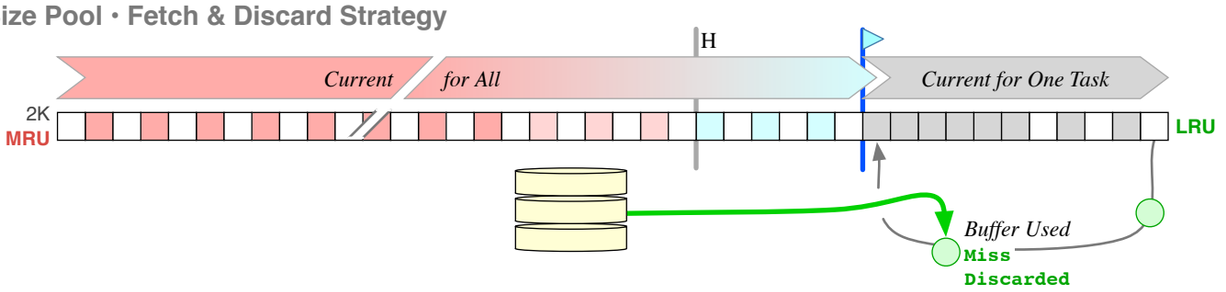
- This is a critical tuning issue for Buffer Pools. The goal is to strike a balance, such that dirty buffer writes are completed and enough buffers are available at the LRU End for use, *and* that buffers are not written too early (which would lead to premature, and therefore undesirable, writes).
- Optimally, enough buffers are written and clean *before* they cross the Wash Marker; and few buffers are written within the Wash Area
- It is best to view the Wash Area as a *forced write* area, the last ditch, to ensure dirty buffers are written before reaching the LRU
  - Several monitoring metrics are provided to assist the administrator.
  - The defaults are perfectly good starting points for Caches and Pools, however, as Caches and Pools are determined and added, the Wash sizes must be addressed, especially in smaller Buffer Pools.
  - The optimal size is dependent mostly on the speed of host I/O subsystem, and the actual Read/Write activity; not on the Cache or Pool size.
- For Buffer Pools that are not used for Writes, the Wash Size is irrelevant (for Writes), it remains relevant for Fetch & Discard activity in that Buffer Pool).
  - It should be set correctly to catch any Writes that may occur [more later], as well as for Fetch & Discard.

- **Asynchronous Pre-Fetch** refers to the mechanism; **Large I/O** refers to the resources it uses. Both need to be tuned correctly.
- When invoked, Asynchronous Pre-Fetch reads a large number of Pages in a single request, from an entire Extent (8 Pages) up to an AllocationUnit I/O (31 Extents or 248 Pages) may be read. Each buffer transfer operation is 8 Pages, thus it is also known as *Extent I/O*.
- Where the Cache does not have a Large I/O Pool, the Fetch & Discard Strategy is used in the PageSize Pool. Where a Large I/O Pool has been configured, it is used instead.
- ASE uses the Buffer Pool with the largest PageSize to which the **DataStructure** is bound. Only the largest Large I/O Pool within a Cache is used, any others are ignored (dead space, remove them).
  - (There is one exception: in an *unconfigured* server, where the Transaction Log is not bound to a named cache, and then only, the LogIOSize Pool in the default data cache is used; in all other cases only the PageSize Pool and the largest Large I/O Pool are used).

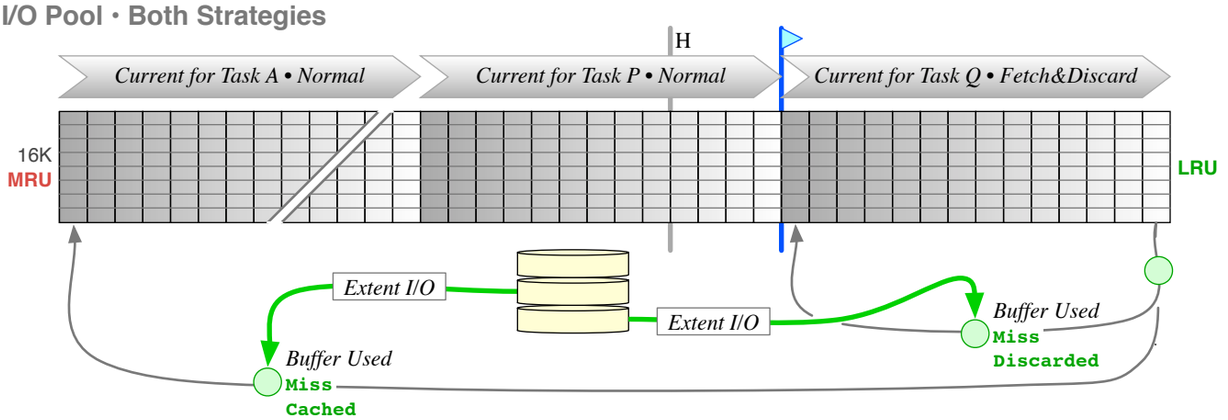
Large I/O Read

- Asynchronous Pre-Fetch & Large I/O are invoked for Reads for:
  - "Table Scans" [1]
  - Range Scans [2]
  - `BCP OUT` (same as `SELECT`)
  - `DBCC` (Clustered Index [1] or Heap operations)
  - `UPDATE STATISTICS` (all flavours; index or data)
  - Non-clustered Index Leaf scans [3]
- When a Large I/O buffer (eg. one extent) is requested, if *any* of the Pages in it are resident in the PageSize Pool, the request for those pages (ie. the one extent) is rejected, and reported as: **`Large I/Os Denied due to/Pages Requested Reside in Another Buffer Pool`** and the remaining (non-resident) pages in such extents must be read using Page I/O.
  - This is always the case for the first extent in each AllocationUnit: due to the AllocationPage, the remaining pages require Page I/O. That is why 31 Extents, not 32, is the maximum Extents that can be read for an AllocationUnit; the total reads are 31 + 7 = 38.
- Note that the diagram is semantic (size is not proportionate): the ratio of PageSize Buffer Pool to the Large I/O Pool is (eg) 80:20%.
  - For diagrammatic clarity, tasks in the Large I/O Pool are shown consuming buffers consecutively; in practice they are mixed.

PageSize Pool • Fetch & Discard Strategy



Large I/O Pool • Both Strategies



- This effectively transfers the Fetch & Discard activity from the Wash Area of the PageSize Pool (upper diagram), to the Large I/O Pool (lower diagram); the Wash Area of the PageSize Pool is thus relieved of Fetch & Discard activity. This is the primary purpose of Large I/O.
  - *Do not look for Fetch & Discard activity in the* PageSize Pool *of Caches that have a Large I/O Pool.*
  - In effect, Fetch & Discard strategy in the PageSize Pool is roughly equivalent to Normal strategy in the Large I/O Pool, except that the buffers are (eg) 8 times the size.
  - Fetch & Discard strategy in the Large I/O Pool is invoked only if the pages required ≥ 50% Buffer Pool size
  - Fetch & Discard in the PageSize Pool vs the Large I/O Pool, mean quite different things, and the former no longer happens.
- The Wash Area in the PageSize Pool can now be optimised for writes only, without regard to Fetch & Discard requirements.
- The Wash Area in the Large I/O Pool may be set to accommodate the determined Normal::Fetch & Discard ratio, eg. 50-50%.
- As illustrated, note that although the warm-MRU through to cool-LRU applies, it is not quite the same for Large I/O Pools.

`Pages Cached`
`Pages Used`

- *For Large I/O Pools,* `sysmon` *keeps track of additional counters on a Page basis (ie. as requested by APF).*
- **`Pages Cached`** *is the I/O size ÷ PageSize.*
- *System and Monitoring tables contain detail re Buffers, and I/Os, on a Pool basis.*
- *Unfortunately, Buffers are counted at the Cache, not Pool, level only.*

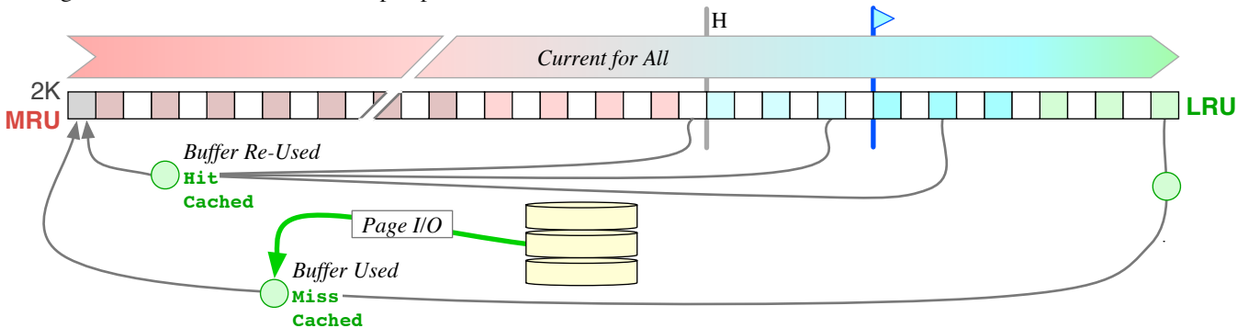
6.1 Large I-O Write

The above concerns Reads, the cache strategies regard Reads; we will now consider Writes from the Large I/O Pool.

- Asynchronous Pre-Fetch & Large I/O are invoked for Writes for the following commands (refer previous page when considering the related source table). Note that such is rare, especially on Production servers:
  - `SELECT INTO` (target table)
  - `DBCC` (table write operations) except `DBCC CHECKSTORAGE`
  - `CREATE INDEX`
  - `BCP IN`
- Therefore the Large I/O Pool size, and the Wash Area size, should be determined on the basis of Reads. The Wash Size would be 20%, and it could be as high as 50%; that will be completely adequate for rare Writes as well.
  - For the duration of maintenance operations, if the Wash Size is not large, the experienced Administrator working on a Production server changes it to 50%.
- `DBCC CHECKSTORAGE` uses the `dbccdb` database; being a nightly occurrence, it is not rare; the Wash Area size in the Large I/O Pool in the applicable cache should be set accordingly.

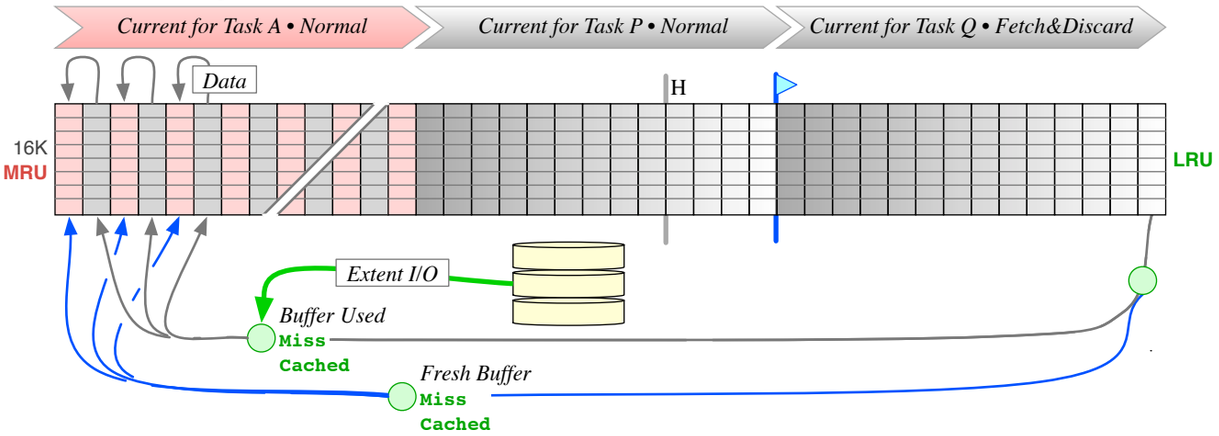
PageSize Pool

- The PageSize Buffer Pool is shown for perspective and reference.



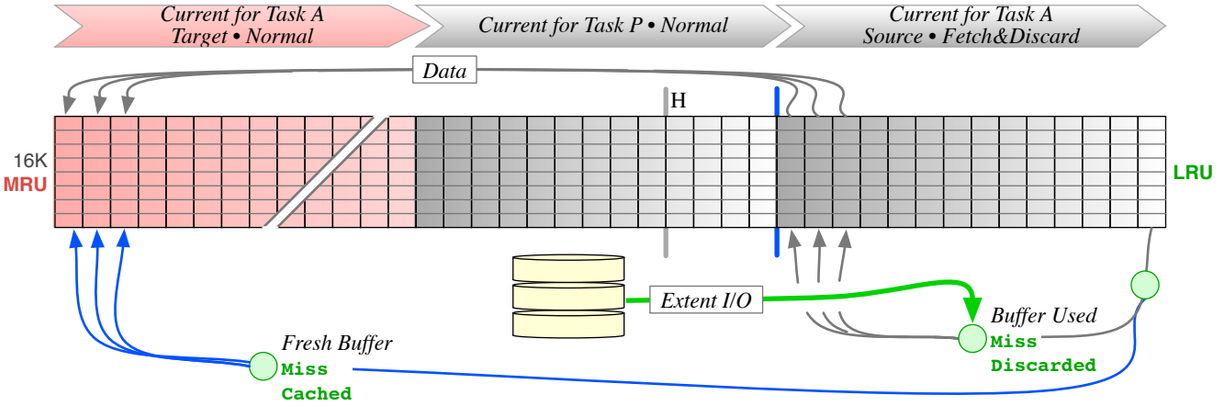
Large I/O Pool · Normal Strategy

- This depicts Task A using Normal strategy and Large I/O for the source, and acquiring new buffers for the target (eg. **`CREATE INDEX`**). Of course, the source may well be the PageSize Pool; it may not be a **`Miss`**; etc. The dirty buffers will be written when they reach the Wash Marker, or when the command completes (not shown).

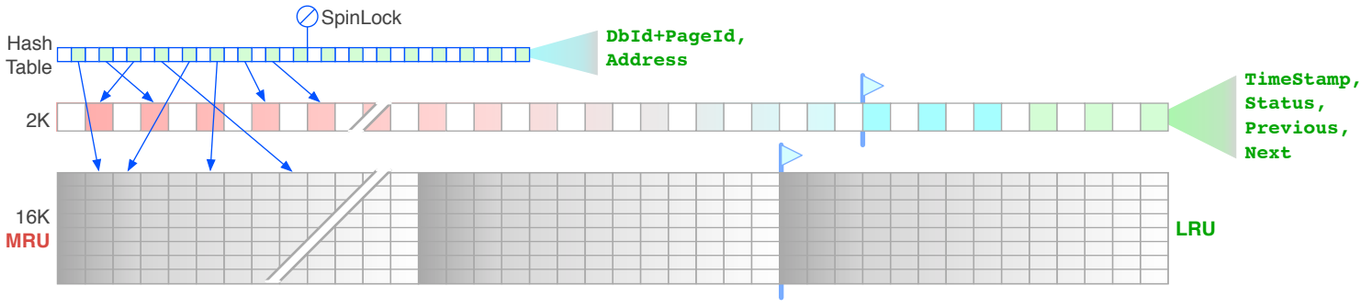


Large I/O Pool · Fetch & Discard Strategy

- This depicts Task A using Fetch & Discard strategy for the source, and acquiring new buffers for the target (eg. **`SELECT INTO`**), both in the Large I/O Pool. The dirty buffers will be written when they reach the Wash Marker, or when the command completes (not shown).

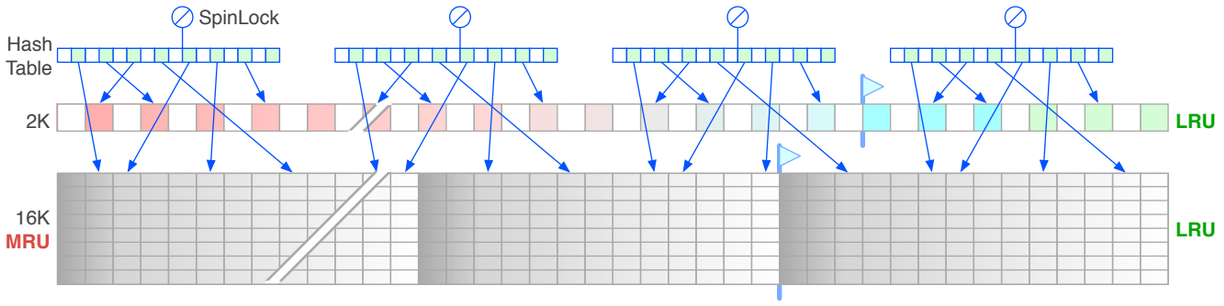


- In each Cache, an efficient HashTable is used to locate the buffers (in all the Buffer Pools in the cache)
  - Each HashTable entry identifies the database and page id of the buffer (aka MASS)
  - The TimeStamp, the linkage (previous & next pointers), and buffer Status are located in the buffer, not the HashTable
- The Hash Table is protected by a single SpinLock, which engines contend for (by spinning).
- A change to the buffer linkage ('movement' in the MRU-LRU chain, under `STRICT` maintenance) or the buffer content (a physical Read or fresh buffer for a new page), requires a change to the HashTable, and therefore requires acquisition of the SpinLock.



Symmetric Multi-Processing

- When multiple engines are used, the SpinLock becomes a point of contention. This is relieved by dividing the Cache into partitions (or, heaven forbid, *cachelets*), each with its own HashTable.
  - This effectively allows several engines to update the Cache, concurrently.
  - This arrangement is different to the normal SpinLock Ratio type of configiuration parameter.
- The number of partitions is set via: `sp_cacheconfig cache_name, "cache_partition = n"` where *n* is a power of two, and (if it is set) it must be at least the `number of engines` (engine threads in 15.7).
- It can also be set for all caches, via `sp_configure "global cache partition number", n`. However, this is not advised, because small caches and Dedicated caches should not be partitioned. All Cache configuration items, including partitioning, should be set on the basis of cache type, size and usage; therefore there cannot be a valid overall value for any of them.
- The cache above has been divided into four partitions, below; it is otherwise unchanged.



Strict vs Relaxed

- `STRICT` and `RELAXED` refer to the maintenance of the Hash Table and the MRU-LRU buffer chain (they are not "replacement policies").
- All the above defines `STRICT` maintenance, it is the default.
- `RELAXED` means:
  - the linkages in the MRU-LRU chain are not maintained; the Hash Table requires less work; therefore a small amount of cache management overhead is avoided

> The manuals show a circular buffer chain with a wedge-shaped Wash Area that keeps shifting. That does not depict the action accurately, nor is there enough logic to understand it, therefore I will not reproduce it.

  - the `HOUSEKEEPER WASH` task is not invoked (`HK IGNORE` is implied); dirty buffers are not written during idle CPU cycles.
  - `RELAXED` is advised *only* if the cache is Dedicated and sized correctly (to entirely contain the intended DataStructures, eg. a typical Log cache).
  - In large caches, the non-maintenance of the MRU-LRU chain actually makes it slower.

Example

For a full-blown example, and for full appreciation, the examination of server level monitor metrics is best. Here is a server monitoring report comparing 2 days (24 hours) plus a delta column, before 7 after correction of a SAN configuration error (the Devices were moved from a poorly configured SAN/Logical Volume to a appropriately configured one). No changes were made to the server configuration, or the databases, or the DataStructures. Inspect the following sections:

- Kernel/Context Switch
- Cache Manager
- Cache Total
  - Each Cache & the relevant Devices, eg. `Cache[log_cache]` & `Disk[Log][4]`
- Asynch Pre-Fetch

- When Asynch Pre-Fetch is invoked, Fetch & Discard strategy will be used if the buffers required are ≥ 50% of the Buffer Pool size, and Normal strategy will be used otherwise.
- When Asynch Pre-Fetch & Large I/O are configured properly, it is quite possible to saturate the I/O subsystem. While that is a goal, it is not desirable to do so to the detriment of other users. To that end, any single invocation of Asynch Pre-Fetch in any Buffer Pool Level, can be limited at the server level, and at the Buffer pool level. The default is 10 percent. The server level is set via:

  ```
  sp_configure "global asynch prefetch limit", percent
  ```
- At the Buffer Pool level it can be set (overriding the server setting) via:

  ```
  sp_poolconfig cache_name, "io_size", "local asynch prefetch limit = percent"
  ```
- For **Large I/O Buffer Pools**, it should be set substantially higher than 10%, noting the actual use. The Large I/O Pool that `DBCC CHECKSTORAGE` is bound to, benefits from 50%.
- For the **PageSize Buffer Pool** that *does* have a Large I/O Pool, the use of the Large I/O Buffer Pool is desired (refer Fragmentation below). Where the DataStructures are badly fragmented, the Large I/O Pool cannot be used, and thus the PageSize Pool will be used.
  - In this case, while 10% is a valid default, a value based on the actual size of the Pool should be determined.
  - The Large I/O Pool can be removed (it can be re-implemented if and when the DataStructures have been de-fragmented).
- For the **PageSize Buffer Pool** that *does not* have a Large I/O Pool, a value based on the actual size of the Pool should be determined, noting that:

  a. Asynch Pre-Fetch translates to Fetch & Discard

  b. the Wash Area size defines *both* the point of disk Writes *and* the Fetch & Discard buffer area.

Under-Utilisation of Asynch Pre-Fetch & Large I/O

- The main reason Asynch Pre-Fetch is not invoked, and Large I/O Pools are not used, is **Fragmentation** of the **DataStructures**.
- There are three distinct Levels of Fragmentation.
  - Anything that is done to prevent Level I Fragmentation in the first place, has the greatest benefit, and it reduces the frequency of DataStructure rebuild operations.
  - Failing that, in the second place, rebuild operations at Level II are demanded. This returns the use of Asynch Pre-Fetch and Large I/O, and therefore great speed to the DataStructures.
  - Level III is a bonus form of Fragmentation that is bestowed on God-forsaken DOL Heaps. Genuine Relational databases have nothing to worry about, but in abject DOL data heaps, weekly, if not daily, Level III de-fragmentation operations are required.

Note

Caveat

There is a fair amount of incorrect or misleading information posted on the web; some of it by *Sybase* "evangelists". On this subject, information to the effect that **the Wash Area is divided by the number of partitions**, and that that is a relevant consideration, has been posted. That is a misleading half-truth. From the perspective that the HashTable locates and protects a partition of the buffers in the Cache, well, yes, the buffers in the Wash Area may be *perceived* to be divided as well. But that is not relevant to the operation, which is:

- Neither the Wash Area nor the remainder is divided, the buffers in the Buffer Pool remain in a single chain, a single volume of buffers per Buffer Pool; the Wash Size is a Pool level, not a partition level, setting.
- When `HOUSEKEEPER WASH` task (or any over-arching disk writer task, for that matter) is invoked, for each Buffer Pool, if any dirty buffers have entered the Wash Area, disk Writes are commenced; access to he HashTable and the SpinLock is not required (the linkage is not changed; it is not a new buffer).
- If and when a buffer is changed, the SpinLock for the applicable HashTable is acquired.
- Whether the Wash Area *appears* to be divided by the number of partitions, from the fourth moon of Jupiter, is not relevant.
- *It is the usual bombardment of irrelevant technical trivia, impressive for the quantity and the volume.*

Tempdb Cache

This cache specifically suffers the heaviest usage on any server, and it is a large proportion of Writes; therefore it requires:

- the highest number of partitions, to support the highest level of concurrency, not the lowest.
- Further, it should have `HK IGNORE` set, therefore the Wash Size is a non-issue; the entire discussion does not apply.
- `STRICT` maintenance is recommended over `RELAXED`, because retaining the speed of the MRU-LRU chain is very important.

MASS

- From 12.5 onwards, *Sybase* appears to be implementing a long term, generic, memory management architecture. These generic memory address spaces are called **M**emory **A**ccess **S**pace **S**egments. When the term MASS is used in the context of Caches and Pools, it is simply a buffer in any Buffer Pool, a multiple of PageSize. A buffer occupies a MASS.
- The English in the `MonSysWaits` and `MonProcessWaits` lookup tables is horrible; we provide a corrected and consistent version.
- As always, ever since the concept of a *buffer* was invented 60 years ago, a buffer is a contiguous block of memory, which is read into, or written from, as a single unit, using a single I/O operation.