

Windows Kernel Internals

Cache Manager

David B. Probert, Ph.D.

Windows Kernel Development

Microsoft Corporation

What is the Cache Manager?

- Set of kernel-mode routines and asynchronous worker routines that form the interface between filesystems and the memory manager for Windows NT

Cache Manager Functionality

- Access methods for pages of file data on opened files
- Automatic asynchronous read ahead
- Automatic asynchronous write behind (lazy write)
- Supports “Fast I/O” – IRP bypass

Who Uses the Cache Manager?

- Disk File Systems (NTFS, FAT, CDFS, UDFS)
- Windows File Server(s)
- Windows Redirector
- Registry (as of Windows XP)

What Can Be Cached?

- User data streams
- File system metadata
 - directories
 - transaction logs
 - NTFS MFT
 - synthetic structures – FAT's Virtual Volume File
- ... anything that can be represented as a stream of bytes

Virtual Block vs Logical Block Cache

- More traditional approach is logical block cache (SmartDrive):
 - File+FileOffset translated by file system into one or more partition offsets
 - Each partition offset translated by cache manager to cache address
- In a virtual block cache:
 - File+FileOffset translated by cache manager to cache address, its memory mapping!

Advantages of Virtual Block Cache integrated with VM

- Single memory manager
 - the actual “cache” manager!
- Data cache is dynamically sized – just another working set
- Cache coherency with user mapped files is free

How Does It Work?

- Mapped stream model integrated with memory management
- Cached streams are mapped with fixed-size views (256KB)
- Pages are faulted into memory via MM
- Pages may be modified in memory and written back
- MM manages global memory policy

Cache Addresses

- MM allocated kernel VA range (512MB +)
 - common: 0xc1000000 – 0xe0000000
- Visible in all kernel-mode contexts
- Member of the System Cache working set (includes paged pool and code)
 - this is what Task Manager shows you!
 - not just the “file” cache, though it does frequently dominate
- Competes for physical memory
- “Owned” by Cache Manager

Datastructures

- File Object
 - FsContext – per physical stream context
 - FsContext2 – per user handle stream context, not all streams have handle context (metadata)
 - SectionObjectPointers – the point of “single instancing”
 - DataSection – exists if the stream has had a mapped section created (for use by Cc or user)
 - SharedCacheMap – exists if the stream has been set up for the cache manager
 - ImageSection – exists for executables
 - PrivateCacheMap – per handle Cc context (readahead) that also serves as reference from this file object to the shared cache map

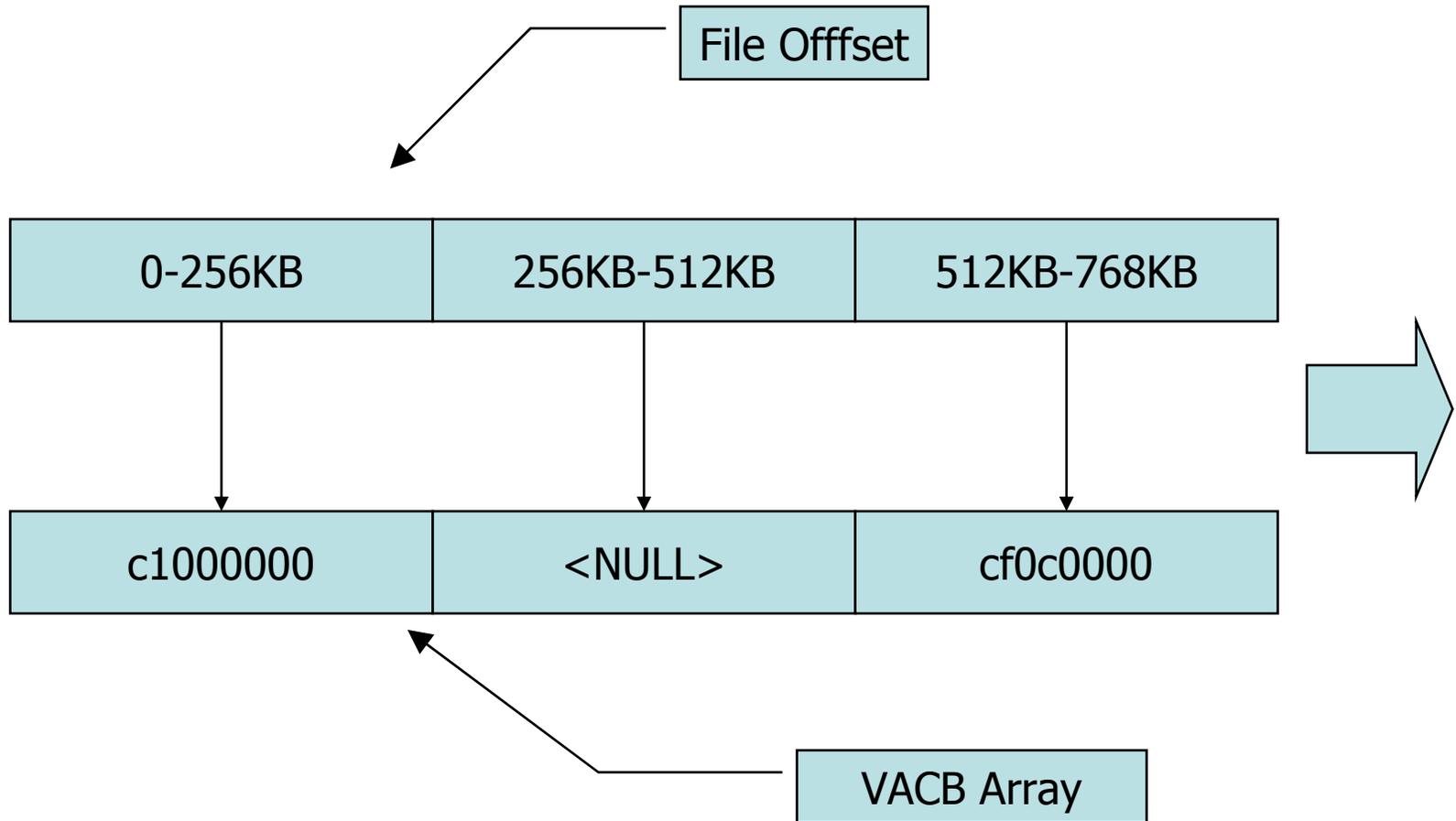
Single Instancing & Metadata

- Although filesystems represent metadata as streams, they are not exported to user mode
- Directories require a level of indirection to escape single instancing exposing the data
- Filesystems create a second internal “stream” fileobject
 - user’s fileobject has NULL members in its Section Object Pointers
 - stream fileobjects have no FsContext2 (user handle context)
- All metadata streams are built like this (MFTs, FATs, etc.)
- FsContext2 == NULL plays an important role in how Cc treats these streams, which we’ll discuss later.

View Management

- A Shared Cache Map has an array of View Access Control Block (VACB) pointers which record the base cache address of each view
 - promoted to a sparse form for files > 32MB
- Access interfaces map File+FileOffset to a cache address
- Taking a view miss results in a new mapping, possibly unmapping an unreferenced view in another file (views are recycled LRU)
- Since a view is fixed size, mapping across a view is impossible – Cc returns one address
- Fixed size means no fragmentation ...

View Mapping



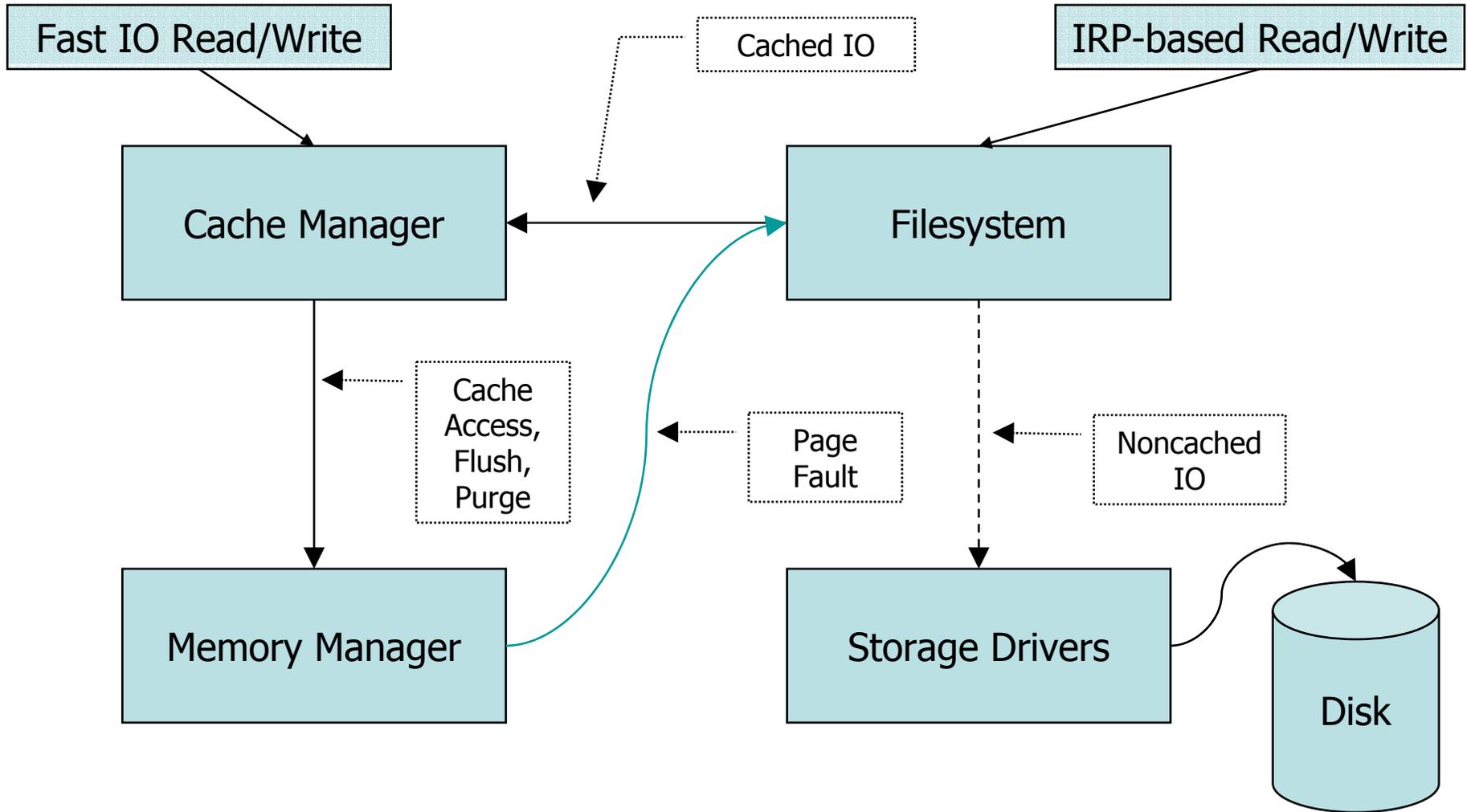
Interface Summary

- File objects start out unadorned
- CcInitializeCacheMap to initiate caching via Cc on a file object
 - setup the Shared/Private Cache Map & Mm if necessary
- Access methods (Copy, Mdl, Mapping/Pinning)
- Maintenance Functions
- CcUninitializeCacheMap to terminate caching on a file object
 - teardown S/P Cache Maps
 - Mm lives on. *Its data section is the cache!*

The Cache Manager Doesn't Stand Alone

- Cc is an extension of either Mm or the FS depending how you look at it
- Cc is intimately tied into the filesystem model
- Understanding Cc means we have to take a slight detour to mention some concepts filesystem folks think are interesting. Raise your hand if you're a filesystem person :-)

The Big Block Diagram



The Slight Filesystem Digression

- Three basic types of IO on NT: cached, noncached and “paging”
- Paging IO is simply IO generated by Mm – flushing or faulting
 - the data section implies the file is big enough
 - can never extend a file
- A filesystem will re-enter itself on the same callstack as Mm dispatches cache pagefaults
- This makes things exciting! (ERESOURCEs)

The Three File Sizes

- FileSize – how big the file looks to the user
 - 1 byte, 102 bytes, 1040592 bytes
- AllocationSize – how much backing store is allocated on the volume
 - multiple of cluster size, which is $2^n * \text{sector size}$
 - ... a more practical definition shortly
- ValidDataLength – how much of the file has been written by the user in cache, zeros seen beyond (some OS use sparse allocation)
- ValidDataLength \leq FileSize \leq AllocationSize

Valid Data Length

- The Win32 model expects full allocation of files (STATUS_DISK_FULL is uncool)
- Writing zeroes is expensive, but users tend to write files front to back
- Windows FS keep track of this as a high-water mark
- If the user reads beyond VDL, we may be able to get clever and not bother the filesystem at all.
- If a user writes beyond VDL, zeroing of a “hole” may be required
- Never SetEndOfFile and write at the end if you can help it!

How to get data *into* the cache

Fast IO – Who Needs an FS?

- Fast IO paths short circuit the IO to a common FsRtl routine or filesystem-provided call
- This is just memory mapped IO, synchronizing with the FS for ...
- Extending FileSize up to AllocationSize!
 - VDL zeroing means the cache data is already good
 - Hint set in fileobject so FS will update directory
- Extending ValidDataLength up to FileSize

Regular Cached IO

- Filesystems also implement a cached path
- Basically the same logic as the Fast IO path (or vice versa, depending)
- Reuses the same Cc functions
- Why not use Fast IO all the time?
 - file locks
 - oplocks
 - extending files (and so forth)

Copy Method

- Used for user cached IO, both fast and IRP based
- CcCopyRead maps and copies a mapped cache byte range into a buffer
- CcCopyWrite copies a buffer into a mapped cache byte range and marks the range for writing
- “Fast” versions of each are really the same code, but only taking 32bit fileoffsets
 - NT used to run on 386s! :-)

Mdl (DMA) Method

- Used by network transport layers
- CcMdlRead returns an Mdl describing specified byte range
- CcMdlReadComplete frees the Mdl
- CcPrepareMdlWrite returns an Mdl describing specified byte range (may contain “smart” zeros with respect to VDL)
- CcMdlWriteComplete frees the Mdl and marks range for writing

Pagefault Cluster Hints

- Taking a pagefault can result in Mm opportunistically bringing surrounding pages in (up 7/15 depending)
- Since Cc takes pagefaults on streams, but knows a lot about which pages are useful, Mm provides a hinting mechanism in the TLS
 - `MmSetPageFaultReadAhead()`
- Not exposed to usermode ...

Readahead

- CcScheduleReadAhead detects patterns on a handle and schedules readahead into the next suspected ranges
 - Regular motion, backwards and forwards, with gaps
 - Private Cache Map contains the per-handle info
 - Called by CcCopyRead and CcMdlRead
- Readahead granularity (64KB) controls the scheduling trigger points and length
 - Small IOs – don't want readahead every 4KB
 - Large IOs – ya get what ya need (up to 8MB, thanks to Jim Gray)
- CcPerformReadAhead maps and touch-faults pages in a Cc worker thread, will use the new Mm prefetch APIs in a future release

Unmap Behind

- Recall how views are managed (misses)
- On view miss, Cc will unmap two views behind the current (missed) view before mapping
- Unmapped valid pages go to the standby list in LRU order and can be soft-faulted. In practice, this is where much of the actual cache is as of Windows 2000.
- Unmap behind logic is default due to large file read/write operations causing huge swings in working set. Mm's working set trim falls down at the speed a disk can produce pages, Cc must help.

Cache Hints

- Cache hints affect both read ahead and unmap behind
- Two flags specifiable at Win32 CreateFile()
- `FILE_FLAG_SEQUENTIAL_SCAN`
 - doubles readahead unit on handle, unmaps to the *front* of the standby list (MRU order) if all handles are `SEQUENTIAL`
- `FILE_FLAG_RANDOM_ACCESS`
 - turns off readahead on handle, turns off unmap behind logic if any handle is `RANDOM`
- Unfortunately, there is no way to split the effect

Write Throttling

- Avoids out of memory problems by delaying writes to the cache
 - Filling memory faster than writeback speed is not useful, we may as well run into it sooner
- Throttle limit is twofold
 - CcDirtyPageThreshold – dynamic, but ~1500 on all current machines (small, but see above)
 - MmAvailablePages & pagefile page backlog
- CcCanIWrite sees if write is ok, optionally blocking, also serving as the restart test
- CcDeferWrite sets up for callback when write should be allowed (async case)
- !defwrites debugger extension triages and shows the state of the throttle

Writing Cached Data

- There are three basic sets of threads involved, only one of which is Cc's
 - Mm's modified page writer
 - the paging file
 - Mm's mapped page writer
 - almost anything else
 - Cc's lazy writer pool
 - executing in the kernel critical work queue
 - writes data produced through Cc interfaces

The Lazy Writer

- Name is misleading, its really *delayed*
- All files with dirty data have been queued onto CcDirtySharedCacheMapList
- Work queueing – CcLazyWriteScan()
 - Once per second, queues work to arrive at writing 1/8th of dirty data given current dirty and production rates
 - Fairness considerations are interesting
- CcLazyWriterCursor rotated around the list, pointing at the next file to operate on (fairness)
 - 16th pass rule for user and metadata streams
- Work issuing – CcWriteBehind()
 - Uses a special mode of CcFlushCache() which flushes front to back (HotSpots – fairness again)

Write Throttling

- Avoids out of memory problems by delaying writes to the cache
 - Filling memory faster than writeback speed is not useful, we may as well run into it sooner
- Throttle limit is twofold
 - CcDirtyPageThreshold – dynamic, but ~1500 on all current machines (small, but see above)
 - MmAvailablePages & pagefile page backlog
- CcCanWrite sees if write is ok, optionally blocking, also serving as the restart test
- CcDeferWrite sets up for callback when write should be allowed (async case)
- !defwrites debugger extension triages and shows the state of the throttle

Valid Data Length Calls

- Cache Manager knows highest offset successfully written to disk – via the lazy writer
- File system is informed by special FileEndOfFileInformation call after each write which extends/maintains VDL
- FS which persist VDL to disk (NTFS) push that down here
- FS use it as a hint to update directory entries (recall Fast IO extension, one among several)
- CcFlushCache() flushing front to back is important so we move VDL on disk as soon as possible.

Letting the Filesystem Into The Cache

- Two distinct access interfaces
 - Map – given File+FileOffset, return a cache address
 - Pin – same, but acquires synchronization – this is a range lock on the stream
 - Lazy writer acquires synchronization, allowing it to serialize metadata production with metadata writing
- Pinning also allows setting of a log sequence number (LSN) on the update, for transactional FS
 - FS receives an LSN callback from the lazy writer prior to range flush

Remember FsContext2?

- Synchronization on Pin interfaces requires that Cc be the writer of the data
- Mm provides a method to turn off the mapped page writer for a stream, `MmDisableModifiedWriteOfSection()`
 - confusing name, I know (modified writer is not involved)
- Serves as the trigger for Cc to perform synchronization on write

Mapping/Pinning Method

- CcMapData to map byte range for read access
- CcPinRead to map byte range for read/write access
- CcPreparePinWrite
- CcPinMappedData
- CcSetDirtyPinnedData
- CcUnpinData

BCBs and Lies Thereof

- Mapping and Pinning interfaces return opaque Buffer Control Block (BCB) pointers
- Unpin receives BCBs to indicate regions
- BCBs for Map interfaces are usually VACB pointers
- BCBs for Pin interfaces are pointers to a real BCB structure in Cc, which references a VACB for the cache address

Basic Maintenance Functions

- CcSetFileSizes
 - used by FS to tell Cc/Mm when it changes file sizes
 - updates VDL goal for the callback
 - extends data sections, purges on truncate
- CcFlushCache
- CcPurgeCacheSection
- CcZeroData
 - used during VDL zeroing for MDL hack
- ... and a few others, of course

Cc Grab Bag

- Every component has them, lets take a short tour of some of the more interesting ones
- This is the R-rated portion of today's presentation :-)

We Can't Use That 16GB

- It takes on the order of 2-3KB pool (minimum) to cache a single stream once the handle has been closed
- Cc structures are torn down at handle close
- File object, Filesystem contexts (FCB, CCB, MCB) Mm section structures, PTEs, etc. remain, and add up.
- If cache is dominated by small files, pool limits dominate ability to cache. Mm must trim files from the cache if pool fills.
- This dramatically limits the effective cache size on large machines in reasonable scenarios
- Motivates thinking about a large-machine cache at the volume level (block cache!), much to our chagrin – lower overhead per page of data

Single Instancing is Security Context Insensitive

- What is single instancing and why does it matter?
- Recall the datastructures – Mm and Cc need a fileobject to reference. They choose the first fileobject seen for the file.
- Terminal Server creates an unanticipated case – multiple security contexts – a lot more often
- If multiple security contexts reference a file, page faults and flushes still only occur in the context of that first fileobject
- User logs out or SMB connection backing user's fileobject is torn down – oops!

Single Instancing is Security Context Insensitive

- Windows XP SMB RDR must break single instancing down to the security context level to solve.
- This means that each TS session will have a unique data section, and thus cache, for each file that otherwise could be shared between sessions.
- Non optimal. An optimal solution will require a lot of work that isn't well understood yet.

Mapping Is An Expense

- Requires Mm to pick up spinlocks
 - easy case – MP scale problem
- Cc MDL functions have to map part of the cache to build MDLs (and then unmap)
- Mm may provide an API which does not need a virtual address
 - recall comment about readahead and mapping

Views Are Large

- Mm provides views at a fixed size, 256KB
- Many files are smaller, some are larger
- If we had a pool of views at 64KB, we may avoid view misses under heavy load
- Mediating factors
 - Once misses start, they'll happen as fast with small as large
 - Can't be used for large files or mapping cost will skyrocket
 - As a result, benefit depends on file size mix
 - Overallocation of small views would eat into VA for large views
- An area to be investigated

Flushes Are Synchronous

- Mm only has a synchronous flush API
- An asynchronous paging IO flush would require FS rework as well
- There are only so many critical worker threads, and so many workers Cc can schedule
- As a result, effective bandwidth of the mapped and lazy writers is limited

Cache Manager Summary

Virtual block cache for files not logical block cache for disks

Memory manager is the ACTUAL cache manager

Cache Manager context integrated into FileObjects

Cache Manager manages views on files in kernel virtual address space

I/O has special fast path for cached accesses

The Lazy Writer periodically flushes dirty data to disk

Filesystems need two interfaces to CC: map and pin

Discussion