

Re: High Avg. Disk Queue Length – Memory Mapped File

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2006-09/msg00047.htm>

- *From:* "anton bassov" <soviet_bloke@xxxxxxxxxxx>
 - *Date:* 5 Sep 2006 19:11:04 –0700
-

Hi mate

I think that, instead of mapping/unmapping portions of the file, you should just specify `FILE_FLAG_NO_BUFFERING` in `CreateFile()` call, and write data to the file with `WriteFile()`. The way I understood it, you do it like

- Map a region of file
- Update mapping
- Flush view
- Unmap mapped region
- Restart at A

When you do it this way, you become dependent on Memory Manager, which operates synchronously. It is well-known fact that, in case of huge writes, synchronous nature of Memory Manager's operations may give you significant overhead. This is why it makes sense to use `FILE_FLAG_NO_BUFFERING` in case of huge writes – by doing so, you disable the system-level cache for the file, so that you become independent on Memory Manager's synchronous operations. As a result, you may improve the speed of writing to the file dramatically

In your case, you have to map-flush-unmap all the time (i.e you depend on the synchronous nature of Memory Manager's operations), and the amount of data to be flushed to the disk is not that small. Have you got any more questions why you get overhead???? I believe that if you manage to avoid dealing with Memory Manager behind the scenes by specifying `FILE_FLAG_NO_BUFFERING` flag and writing data with `WriteFile()`, you may get a significant improvement

Anton Bassov

gsudeesh@xxxxxxxxxxx wrote:

Hi Pavel,

I am very thankful for the first answer. I am not aware of this. I

Re: High Avg. Disk Queue Length – Memory Mapped File

think I am experiencing the above said problem.

I will explain my design again. I have an hardware that transfers data to a non paged pool. The data is transferred as series of frames. I shall decide the size of frame and set it to the hardware. The hardware has a control register for starting and stopping the transfer of data. When I start the data transfer, hardware transfers the data to the non paged pool frame by frame. After transferring a frame, the hardware generates an interrupt. When my application gets the interrupt, it copies the data from the non-paged pool to a memory mapped file, the file located in location C:\Temp. This goes on till stop is set to the hardware.

Besides my application, there are other applications that need the data from the hardware to do some processing. Hence we are using the memory mapped file for sharing data instead of virtual Alloc. The applications that need the data shall read from memory mapped file. To allow all applications to read the data in a unique way, we have created a library that shall manage the memory mapped file. Using this library my application shall write to the file and other applications shall read from the file.

Since the operations take place in real time, wrapping does not have any significance to other applications and also wrapping is managed by the library.

The library maps and unmaps the file by 48MB. Hence there shall 11Mapping and Unmapping shall occur to fill the entire file.

I hope I am more clear to You.

Also, I observed one interesting fact. I tested the application on two machines. One with Serial ATA Hard disk and another with Parallel ATA Hard disk. In the machine with the Serial ATA Hard disk, the average disk queue length is mostly 0. But at some durations, it goes high and comes back to zero after a little time. The system response is fine. In the case of Parallel ATA Hard disk, the average disk queue length is not zero it is around .2028(In performance monitor at a scale 100, it is at 20). After some time, it goes high and comes back to zero after some time, which is greater than Serial ATA. Do you find anything interesting here?

Also in the Parallel ATA Hard disk there are two Hard disks. With Logical Drive C, D and E on one physical disk and F on second hard disk. When I put my files on F drive(F:\Temp), the average disk queue length is high. But the system response is fine and ok. This is not the case, when I put the files in C:\Temp. At this time the average disk queue length is high and system response is very slow. Infact it is very difficult to work.

What can be the reason?

Re: High Avg. Disk Queue Length – Memory Mapped File

Thanks In Advance,
Sudeesh G

Pavel Lebedinsky [MSFT] wrote:

When you unmap a region the pages that were unmapped are moved to the modified page list. When this list grows beyond a certain size the mapped writer thread in the memory manager will wake up and start flushing pages to disk. If you repeatedly map and unmap portions of a large file you can generate a constant stream of writes, and this is probably what you're seeing.

I'm not sure I understand your overall design – do you want the data you write to the mapped file to be persisted on disk? Wouldn't this data get overwritten when the file wraps around? If you don't need the data to be persisted then perhaps you could use regular buffers (allocated on the heap or using VirtualAlloc) instead of a mapped view?

--

This posting is provided "AS IS" with no warranties, and confers no rights.

<gsudeesh@xxxxxxxxxx> wrote:

The data transferred by hardware is around 500KB. The file size is 500MB. The file mapping and unmapping size is 48 MB. It takes around 30 seconds to fill the file. My program does not stop on reaching the end of file. It does not stop until explicitly requested. It starts from the beginning after reaching the end, that is the file is wrapped. I am creating the file using the following function:

```
::CreateFile( wsFilePath.c_str(),  
GENERIC_READ | GENERIC_WRITE,  
FILE_SHARE_READ |  
FILE_SHARE_WRITE,  
0, // No security attributes  
CREATE_ALWAYS,  
FILE_ATTRIBUTE_NORMAL,  
0 ); // No template file
```

Re: High Avg. Disk Queue Length – Memory Mapped File

If I use FILE_FLAG_NO_BUFFERING, then I have to do any special mechanism with mapped pointer ?

Sudeesh G.

Pavel Lebedinsky [MSFT] wrote:

Did you mean 500 MB or 500 KB? You said below that each chunk was 48 MB. How long does it take to go over the entire 528 MB file? What happens when you reach the end of the file – does your program stop, or does it start over at the beginning? If it stops, how long does it take for the system performance to recover?

What flags do you pass to CreateFile? Does it help if you add FILE_FLAG_NO_BUFFERING?

--

This posting is provided "AS IS" with no warranties, and confers no rights.

"Sudeesh" wrote:

:

I am using Windows XP.
The memory mapped file is backed by user page file located in the directory C:\Temp. The file size is 528Mb. I have 2GB RAM.

Regarding the map and unmap operations, it depends on the size the data being transferred by the hardware. The size may be in the range of 500KB.

What OS
are you

Re: High Avg. Disk Queue Length – Memory Mapped File

using? Is
the memory
mapped file
backed by
pagefile or a
user file?
How much
RAM do
you have
and how
many
unmap
operations
happen
before you
notice the
problem?

I
need
a
solution
for
avoiding
high
Avg.
Disk
Queue
length.
I
am
programming
an
hardware
that
transfers
raw
data
to
a
non-paged
pool.
The
size
of
the
non-paged
pool
is
32Mb.

Re: High Avg. Disk Queue Length – Memory Mapped File

My application moves the data in the non-paged pool to a buffer in real time. The size of the buffer is 528Mb. My application cannot map the entire 528Mb. So, I am mapping and unmapping the buffer at sizes of 48Mb. The problem is that when I run my application, the

Re: High Avg. Disk Queue Length – Memory Mapped File

machine
becomes
slow.
It
does
not
occur
immediately.
It
starts
to
happen
after
some
time.
Using
the
performance

monitor,
I
checked
the
reason
for
this
scenario.
I
found
that
when
the
Avg.
Disk
Queue
Length
becomes
high,
the
system
becomes
slow.