Solving Performance Bottlenecks in Servers: A Deep Dive

INTRODUCTION

Data has become the new digital gold, delivering competitive advantage to those that have the technology to get more out of it faster, and the tools to derive its significance faster. Data is at the core of almost everything we touch, build, or experience. From personalized customer experiences, to predictive analytics fed by devices and sensors at the edge, and new business models enabled by Big Data, AI and machine learning, it's the data – and the insights delivered by that data - that differentiates the companies that succeed.

IT is feeling the pressure as heavy workload demands, increasing amounts of data, and legacy code converge to overwhelm the performance of their missions-critical apps and threaten the business' ability to compete effectively. But how do you overcome these obstacles without expensive alterations to your infrastructure or your apps?

DataCore offers a unique plug-and-play software alternative.

This technical whitepaper reveals how DataCore eliminates sluggish I/O performance in the host with MaxParallel™ software - leveraging the full potential of your CPUs and memory to yield more responsive applications and optimal use of your resources.

SPEED IS EVERYTHING

With a spotlight on delivering faster business insights, improving performance is a key focus for developers, business analysts, and IT admins/managers alike. There are many different optimization strategies to consider – spanning hardware and operating system (OS) to applications and databases - with costs and benefits associated with each. For example, buying newer, faster hardware will likely improve performance, but comes with long procurement cycles, and high costs. And throwing the latest and most expensive hardware at a problem isn't sustainable for most businesses. Another option is to redesign your application to reduce the I/Os required to read and write your data. Or you can re-architect the apps themselves to manipulate how I/O is handled. Both strategies require a high degree of expertise and are time-consuming and complex. As well, any time you manipulate your apps and databases, you introduce risk and the potential for costly disruptions to your business.

A key area that impacts I/O performance in the host is the operating system itself. One of the basic roles of the operating system is to manage the various I/O devices, such as your disc drives. For example, when an application requests data, the OS sends the I/O request to the physical storage device, then returns the response back to the application. But how that I/O request is handled by the operating system has a huge impact on application performance. When handled poorly, it can defeat performance gains expected from the latest technologies. The following section describes some of the challenges in how the OS handles I/O requests.

APPLICATION THREADS AND THEIR IMPACT ON I/O

An application thread is a set of programmed instructions which defines a path of execution within a process. A process that is parallelized has multiple threads which execute independently but share OS resources like executable code and the values of variables. Applications that need to perform multiple concurrent operations or service large volumes of user requests, like web servers, can greatly benefit from multi-threaded processes.

On multicore systems, multiple threads can execute at the same time, with several cores running separate threads simultaneously, theoretically increasing the throughput of the system.

The challenge with multi-threaded processes is how the operating system satisfies their I/O requests and the negative impact on application performance.

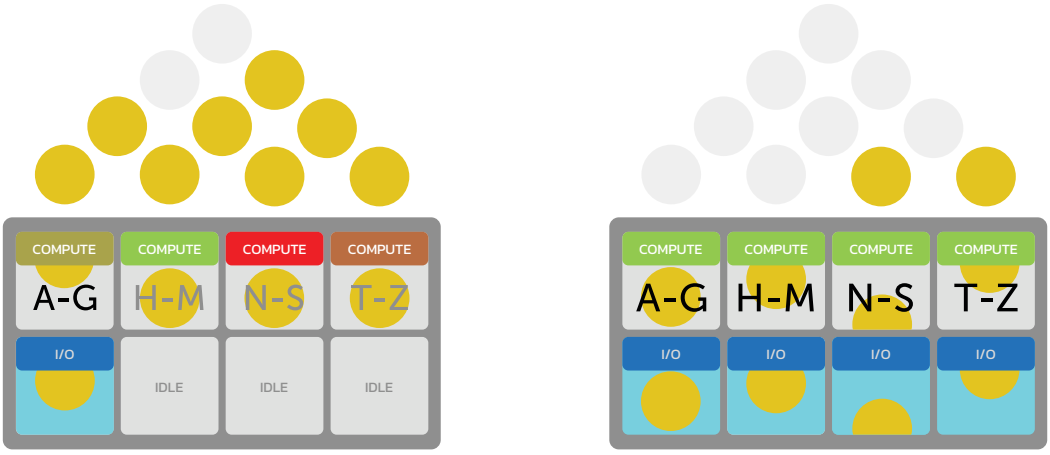


Diagram 1: Serial vs. parallel approaches to satisfying I/O requests

In the example on the left, you see how serial I/O processing in the OS causes contention and delays when multiple application threads queue I/O requests at roughly the same time. On the right is the parallel I/O approach used by MaxParallel™ software to ensure fast, concurrent access to data. In the section below, we'll dive in deeper for a greater understanding of where and how efficiencies can be applied to remove contention and latency, thereby significantly increasing responsiveness and throughput.

CHALLENGES WITH I/O REQUEST HANDLING

In this section, we look at some of the different approaches that operating systems service I/O requests from application threads.

Synchronous vs. Asynchronous

Multi-tasking with asynchronous I/O is commonly used when applications are expected to be waiting for data to be returned by the storage device. The requests are queued in order for other applications to take advantage of the processors in the meantime. In contrast, synchronous I/O puts the thread of execution in an idle state until the I/O request is completed – without releasing the processing resources. While there are different reasons for utilizing one technique over the other, one of the primary reasons is the difference in processing speed between the CPU and the storage device.

Asynchronous I/O

In the past, CPUs and RAM in hosts were much faster than storage (typically spinning discs), making it inefficient for the CPU to wait for storage to complete I/Os. To avoid these inefficiencies, I/O requests were handled asynchronously, freeing the CPU to do something else until alerted that the disc I/O completed.

To see how this works, consider the following diagram.

1. An asynchronous request is made by the application thread.
2. The disc is notified of the request and the thread goes into a wait state. The CPU is reassigned to process other threads.
3. When the data has been retrieved by the disc, an interrupt service routine (ISR) is executed.

4. A deferred procedure call (DPC) is then executed, giving priority to the interrupt handler and deferring less important tasks until later.

5. Finally, the asynchronous procedure call (APC) completes the disc I/O.

The cost of the context switches between applications vying for the CPUs is relatively small when compared to the time it takes for the disc to process the I/O.

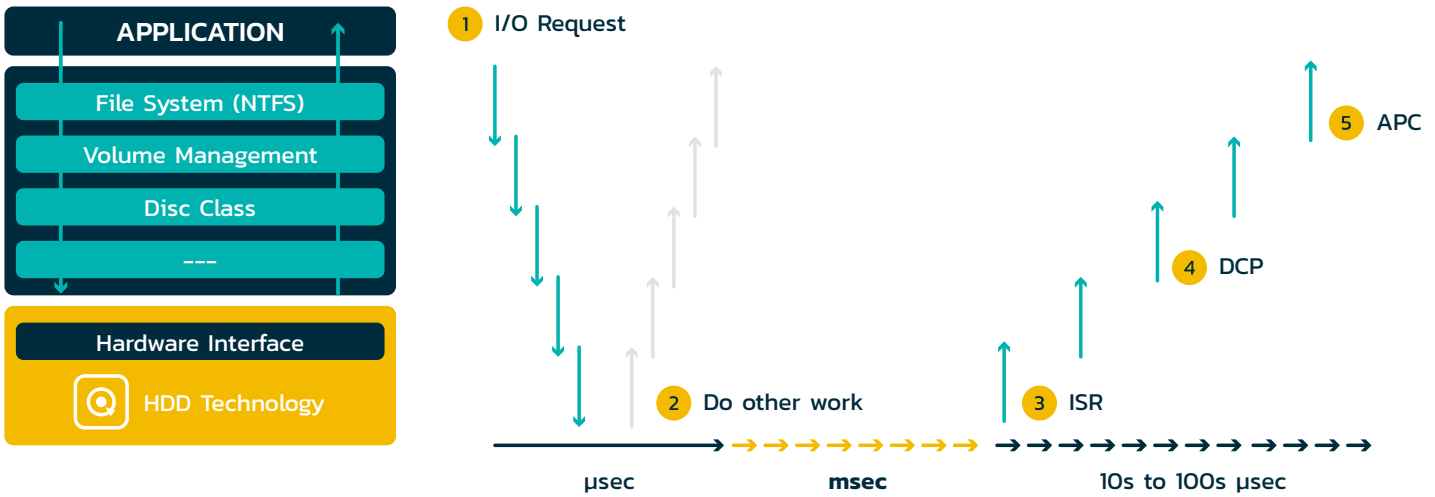


Diagram 2: Asynchronous I/O with spinning disc technology

Solid State Drives (SSDs) cause us to rethink this approach. While still orders of magnitude slower than the CPU, the response times of the latest SSDs have reduced latency significantly. As you can see in diagram 3, data is now being returned from disc so fast that in an asynchronous model, the CPU is constantly getting interrupted, adding a lot of unnecessary overhead. Now add multiple cores and multiple threads of execution to this picture and the problem just gets compounded. It's like the difference between a conversation between 2 people in a room and a conference call with 50 people on the line, vying for attention and talking over each other.

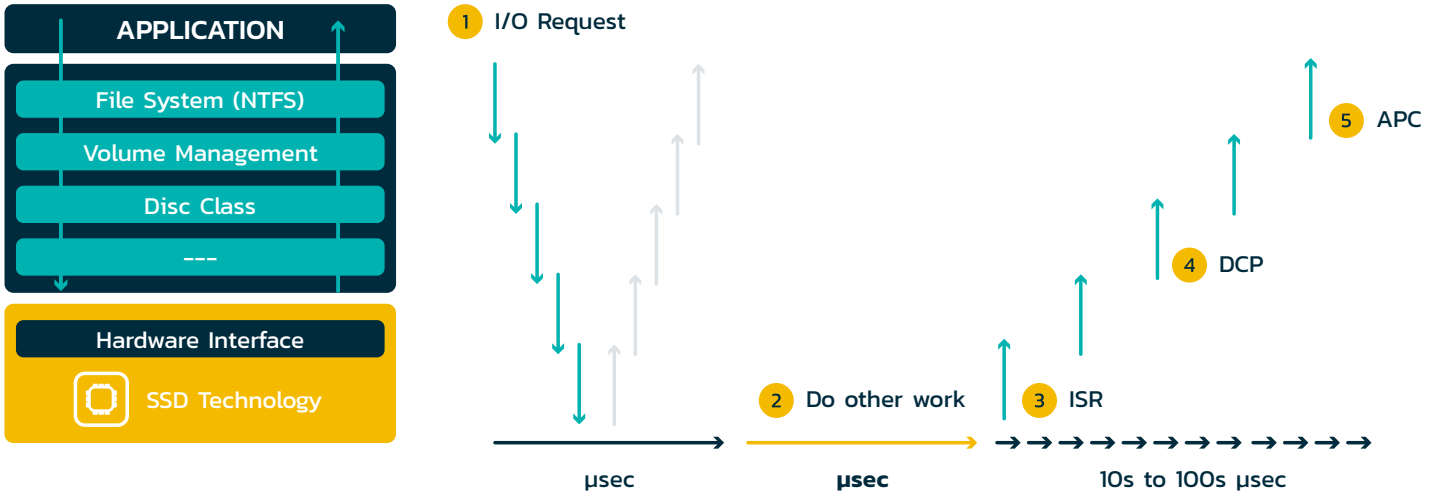


Diagram 3: Asynchronous I/O with low-latency solid state drives

Synchronous I/O

As shown in diagram 4, with synchronous I/O, the application thread spins actively until the I/O request is completed. This polling state effectively blocks it from performing other tasks, but it also eliminates the now significant overhead of interrupt handling and context switching. When using fast SSD technology on multicore systems, real-time synchronous processing techniques make more sense – especially when it comes to latency-sensitive workloads that demand quicker attention.

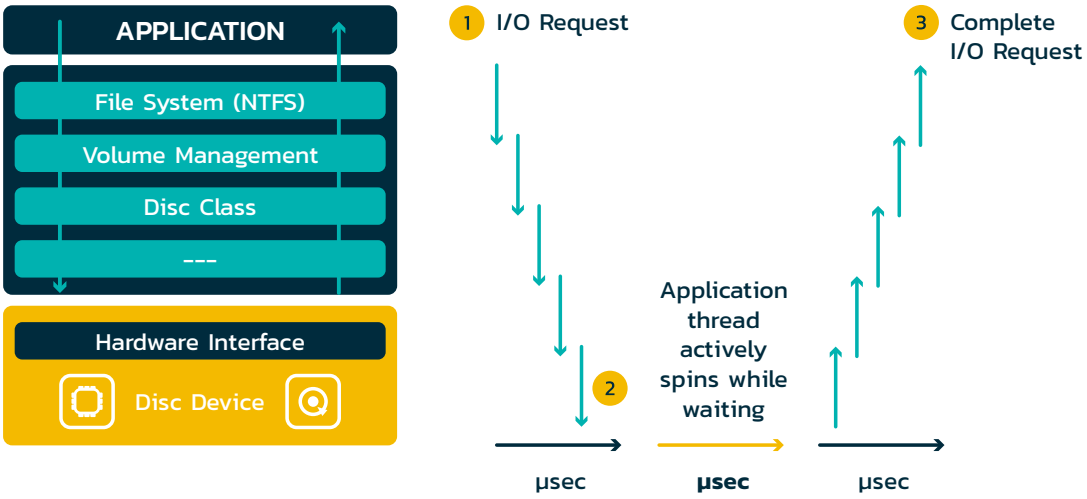


Diagram 4: Synchronous I/O

1. A synchronous request is made by the application thread.
2. The SSD is notified of the request and the thread spins actively while waiting.
3. When the data has been retrieved by the disc, the thread continues processing.

Now that we understand how I/O requests from an application thread are processed in both synchronous and asynchronous approaches, let's look at the impact that multiple application threads have on I/O performance.

Multiple application threads

Multiple application threads execute concurrently and take better advantage of multicore systems. For example, an application can execute longer-running tasks in the background while still remaining responsive to user input versus a serial approach which would freeze the application while waiting for the task to complete.

In general, multi-threaded applications deliver better system utilization because threads that need to retrieve data from a slower storage medium such as a spinning disc (internal or external) do not hold up those threads whose data resides in local cache. That said, the more threads there are, the more points of contention can occur between the threads and the hardware devices if not handled properly.

Cache line contention

Another problem that can occur with multiple threads is cache line bounce due to cache line contention. To understand the concept of cache line bounce, let's take a deeper dive into the concept of cache lines.

CPUs move data between memory and cache in blocks of 64 bytes called cache lines which contain the copied data and a tag that denotes the memory location. The cache line improves performance by enabling the processor to retrieve data from the fast L1 caches in the CPU versus going to slower off chip RAM. When the processor needs to access a memory location – say, to fetch a long word into a core register or store

a register into a memory location in the L1 cache line - it first checks to see if that memory location exists in a cache line of the core's L1 cache. If it does, it reads or updates the contents in the cache line. If it doesn't, it has to retrieve the corresponding cache line from RAM into the core's L1 cache before executing the operation.

Since threads within the same process share the same address space, there is always the possibility that multiple threads – each on a different core with its own L1 cache - may end up trying to update a common memory location. Multicore systems implement special cache coherency protocols to coordinate such access to common cache lines.

The first core to update a memory location locks the cache line so that no other core can update its own cache line until the lock has been released. The first core then flushes the updated cache line to RAM. Now all other cores with that cache line are notified that they no longer have the latest copy in their L1 cache. If they subsequently need to access any memory location in that cache line, they must refetch it from RAM in order to get the most up-to-date values.

Now imagine a 72 core system with 72 threads of execution all attempting to read or update the same location in memory. The resulting fetching and flushing of cache lines is referred to as cache line bounce and can significantly impact application performance.

Spinlocks

For more complex shared data structures, programmers need higher level abstractions to manage access to shared data, beyond the primitives in the cache coherency protocols. Operating systems offer programmers higher-level synchronization tools such as spinlocks to coordinate each core's access to a shared data structure. Here also, system performance can degrade considerably with highly-contentious spinlocks. Since spinlocks have memory locations associated with them, they can affect operating system scheduling and interrupt processing. The result can be severely degraded system performance.

In multicore systems with fast media such as SSD, it is important to program the system software to avoid these issues. Without careful consideration, your multicore system with multiple SSD devices may behave like a single processor system with only a few SSD devices.

UNDERSTANDING THE I/O STACK

When looking to optimize storage performance, it's important to understand the I/O stack and what takes place at each layer. This provides a basis of understanding where it's possible to bypass old legacy code and implement optimizations at a higher level in the stack.

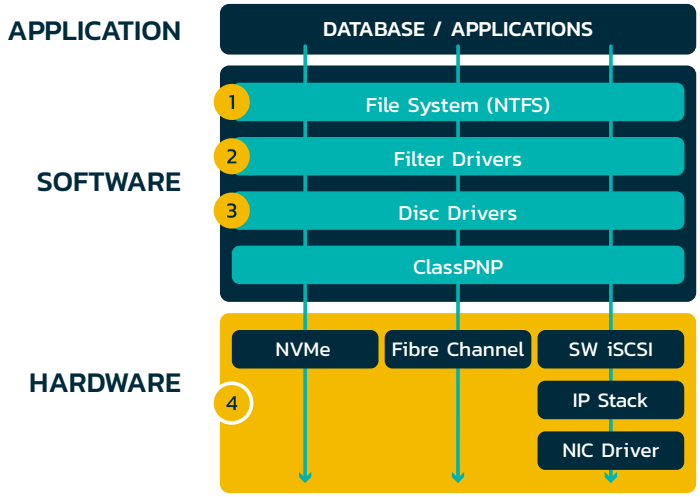


Diagram 5: The I/O stack

1. The file system translates the I/O requests into calls to other drivers below it in the I/O stack

2. Filter drivers are software that are inserted into the I/O Stack and are used to alter the behaviour of an I/O request for a device or class of devices. Upper level filters sit above the primary device driver and are typically used to deliver value-added features or functionality to the device. Multiple filter drivers can be layered on top of each other but will add additional processing that slows the system down

3. The disc drivers interact with hardware components such as storage devices to carry out the specific actions requested by the operating system.

4. At the hardware level, you have different disc devices that have different interfaces.

INTRODUCING MAXPARALLEL: ADDRESSING I/O INEFFICIENCIES IN THE OPERATING SYSTEM

Now that we've explored some of the challenges that exist in the operating system, let's look at how MaxParallel software

addresses them with a simple plug-and-play enhancement to your existing environment.

A new filter driver above all of the rest

MaxParallel software deploys as an intelligent filter driver that sits above the other disc drivers and performs a series of I/O optimizations based on the type of application I/O request. The goal is to optimize I/O locally (on the host) and avoid the negative performance impacts inherent in the original I/O code.

As depicted in Diagram 5, upper level filters reside just below the file system and can be used to enhance I/O processing without requiring changes to apps. This is very important because the programming model is preserved, thus avoiding any impact to existing code – much of which may be mission-critical to the business. By installing MaxParallel as an upper level filter driver on the disk class, all app threads will first go to the new filter driver before the disc driver and MaxParallel has the opportunity to process the I/O directly without serializing or queuing up the request and having to go out the disc device.

Some of the ways that MaxParallel optimizes I/O performance include:

1. Leveraging the filter driver's local memory store to reduce backend roundtrips

By allocating local cache in RAM and proactively bringing in complementary blocks of data, many of the read/write requests can be satisfied immediately without queueing up for physical disc I/O. Why is this important? Well, accessing data in local RAM is about 100x faster than flash, so satisfying requests via local cache provides significant performance improvements. This also avoids traversing lower-level legacy code which may not be optimized for multi-core systems, causing many spinlocks or increasing cache line bounce.

2. Using I/O polling (for Synchronous I/O) rather than Asynchronous I/O and Interrupt Handlers

I/O polling eliminates the overhead and long code paths associated with I/O completion routines and deferred procedure calls. Studies now show that with faster storage media like NVMe, polling for completion of an I/O request outperforms traditional interrupt-driven asynchronous I/O. By intercepting the I/O request via MaxParallel's filter driver, many of the I/O requests can be satisfied either via local cache or handled synchronously utilizing I/O polling for faster response.

3. Avoiding cache line bounce

MaxParallel avoids sharing cache lines between non-related I/Os and uses 'lock free' techniques. This significantly reduces cache line bounce.

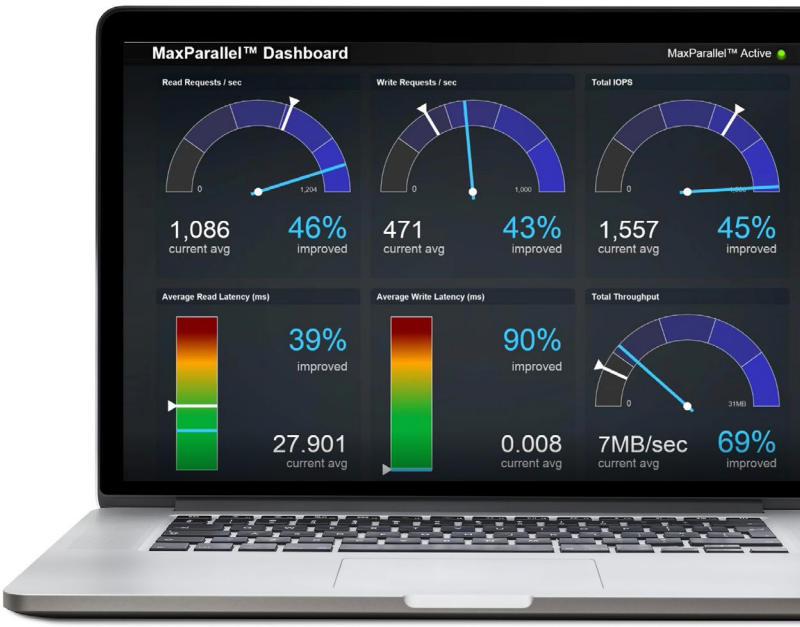
4. Avoiding global IO queues

Global IO queues serialize I/O requests and are not efficient when there are multiple storage devices. In the scenario where one disc has a lot of I/O requests and another has very little, the one disc will always be waiting on the other. MaxParallel avoids global I/O queues by creating multiple queues so that disc requests can be satisfied in parallel.

FASTER RESPONSE FROM MAXPARALLEL SOFTWARE

While there are many different approaches to optimizing application performance, DataCore's unique plug-and-play software solution avoids the heavy lifting, risk, and expense associated with coding changes, rearchitecting apps, and hardware upgrades and delivers rapid time to value. Moreover, the benefits can be easily measured and monitored with the MaxParallel Dashboard.

Diagram 6: The MaxParallel™ Dashboard



When optimizing I/O performance, there are several key performance indicators that reflect the health of your system and should be monitored before and after any optimizations take place.

- Disk Reads/sec is the rate of read operations on all disks. It is the read component of Total IOPS. Higher values represent an increase in work performed.
- Disk Writes/sec is the rate of write operations on all disks. It is the write component of Total IOPS. Higher values represent an increase in work performed.
- Total IOPS, or Input/Output Operations per Second, is the combined rate of read and write operations on all disks. Higher values represent an increase in work performed.
- Average Read Latency is the average time, in milliseconds, of a read of data measured across all disks. Lower values are better.
- Average Write Latency is the average time, in milliseconds, of a write of data measured across all disks. Lower values are better.
- Total Throughput is the rate that bytes are transferred to or from disks during I/O operations, measured across all disks. Values may be in KB/sec, MB/s or GB/sec. Higher values represent an increase in work performed.

By measuring these Key Performance Indicators (KPIs) before and after enabling the MaxParallel software you can easily see the improvements gained. The MaxParallel Dashboard above (diagram 6) demonstrates a significant acceleration in I/O performance, including lower latency and more productive use of the discs – without changing a line of application code or swapping out hardware.

SUMMARY & WRAP-UP

With the move to real-time, speed is everything. Fundamentally, faster data collection and analysis enables companies to compete more effectively. With the right data at your fingertips, decisions are made faster, creating richer customer experiences, more reliable products, and more productive employees. But just putting the best apps and infrastructure in place isn't enough. A Maserati can't drive fast if it's stuck in traffic. Same goes for your data. If your data is bogged down by poor coding practices or long legacy code stacks in the OS, it won't deliver the performance your business needs to stay ahead. MaxParallel software delivers optimizations that enable true parallel I/O processing, creating 'express lanes' that service requests quickly and independently, significantly improving responsiveness and productivity.

The best way to see the impact of MaxParallel is to try it yourself with a free 30-day trial.

Visit **datacore.com/maxparallel** today.

For additional information, please visit datacore.com or email info@datacore.com

DATACORE