

# **File-Based vs Block-Based Caching**

A NEVEX White Paper

## **Abstract**

This white paper outlines the differences between a block-based and a file-based approach to disk caching to improve I/O performance. Block-based caches run at a lower level in the operating system and work directly with block storage devices to provide high speed performance, whereas a file-based solution operates in the file stack of the operating system at a higher level. This paper will highlight the strengths and weaknesses of each approach, and demonstrates that a file-based solution often meets or exceeds the raw performance of a block-based solution and allows for a much richer and more powerful set of controls on an intelligent caching policy, essential for delivering targeted application acceleration.

Gene Amdur

NEVEX Virtual Technologies Inc.

October 3, 2011

## Background

NEVEX CacheWorks is one of a number of server-level caching solutions available in a new market that utilizes a local high-performance Flash media device to cache active data for the purpose of addressing the challenges of storage I/O performance and latency. CacheWorks offers a file-based caching solution whereas competitive products are block-based.

For those not familiar with the concept of caching, it's simply the copying of the most current data to the fastest access location, and is analogous to how we ourselves deal with data. We remember the information we need now, write down what we might need later, and file away information not needed for some time. The analogy goes further in that re-reading our notes automatically “caches” the data back into our memory.

Performance of a caching solution is directly related to the ratio of information that can be served from the cache (a cache hit<sup>1</sup>) versus the information that has to be looked up (a cache miss<sup>2</sup>).

Caching is prevalent in computing and networking; CPUs, disks, memory, Web and DNS servers, databases, and just about everything else has built-in caching schemes to improve performance. Application-specific caches allow for context awareness and cache in ways that are most advantageous to how the application operates. Databases cache tables, rows, and/or query results. Web servers cache hot pages and transactional results. Context awareness allows these caches to increase the ratio of cache hits, increasing performance.

Faster storage technologies like Flash media have opened up new, promising areas for caching. In particular, with faster and larger storage devices like PCIe Flash cards and SSDs, backend storage caching not only becomes possible, it's practical.

At first, it seemed natural to cache data in the same format as it is stored on disk. This direct mapping approach is the core of block-based caching schemes. An alternate approach, implemented in the file system stack, is layered on top of the block structure of the disk. This file-based approach leaves the underlying block structure of the disk behind and caches files instead.

Both block and file-based approaches have advantages and disadvantages, which are compared and contrasted in the following sections.

---

<sup>1</sup> When requested data is contained in (and returned from) the cache

<sup>2</sup> When requested data is not in the cache, and therefore must be retrieved from its original storage location

## **Block-Based Caching**

Magnetic information is accessed on physical disks by read/write heads operating on a set of spinning platters. The disks are prepared first by using a low-level format that initializes the blocks on the disk for use, then a high-level format that creates the file system structure needed by the operating system.

The low level format sets a basic block size, typically 512 bytes long, and the high level format organizes the file system on top of that 512 byte structure. A block caching scheme then looks at caching the 512 byte segments as they appear on disk. It is possible to cache larger segments that are composed of multiple 512 byte segments as well but this approach would also typically reflect the actual structure on the disk.

### Performance

Block caching, depending on how it's implemented, can be very fast because all of the caching happens in the lowest levels of the operating system kernel<sup>3</sup>. All that is needed is to intercept the disk accesses and decide if they are going to be cached or not, or served from cache or backend storage. This interception can take place just before the device drivers in the kernel. Note that while the caching system itself can be operating system independent, device drivers would still be needed for each supported operating system.

Operating systems that have robust block access drivers like Linux and other Unix derivatives would be able to drive very good performance from a block caching system.

### Context Awareness

Block solutions cache data in the same format that it is stored on disk. They operate beneath the file system level and identify data based on the low-level disk volume on which it is stored. To define what data is to be managed by the cache, you target specific LUNs<sup>4</sup> (effectively the disk volumes themselves) and any active blocks from those LUNs are cached.

The advantage of this is that it is relatively management light on the side of the caching solution; you need only define the LUNs to be cached. The disadvantage is that you can not apply a sense of priority to the cached data (in order to increase cache hits) beyond the management heavy process of storing important or application-specific data on specific volumes.

To increase context awareness in a block-based scheme, some current solutions allow

---

<sup>3</sup> The main component of the operating system, and the bridge between applications and the data processing done at the hardware level

<sup>4</sup> Logical Unit Number - A number used to identify a device accessed by storage protocols which supports read/write operations (usually a logical disk). The term is often also used to refer to the disk itself

you to configure a “read near”<sup>5</sup> mode, which reads into the cache blocks believed to be needed in the near future. On a nicely sequenced storage volume you can assume initially read blocks represent the start of files and the next blocks will represent further portions of the same files and therefore are likely to be shortly requested. In some cases you will read into the start of the next file, but the read ahead benefit should outweigh the cons of caching some potentially inactive data.

Unfortunately, this benefit degrades as disks become fragmented. Reading nearby blocks can then result in reading blocks from unrelated files and/or free space, degrading the utilization of the available cache size. There is, however, a large amount of ongoing research<sup>6</sup> into improving the concept of predictive caching, including better predictive algorithms and state-based inspection of the I/O data streams. As yet, there is no definitive answer on the subject.

### Extensibility

A block-based caching scheme provides a natural fit with block-based devices such as direct-attached disk or SAN<sup>7</sup>. However, it is more challenging to extend the paradigm to non-block devices such as a NAS<sup>8</sup> or a file-based cloud storage gateway. In both those cases, files coming over the network do not necessarily have an underlying block scheme associated with them by the time they arrive at the application server. It is not obvious how a block-based system can cache these files to improve performance when accessing these types of data over the network.

One key advantage of the block-based approach is that it is relatively operating system independent. Since the cache is acting on the basic structure of the disk, the high level file structure is irrelevant to the cache system. They are not completely operating system independent, as there is still device driver work to be done for support of each platform. It does, however, make porting to a new operating system relatively easy as you need only hook into the low level device driver on the new target platform, and (testing aside) it should essentially just work.

A further advantage is that although much of the accessed data are files, not everything need be. A block-based system ignores the structure of the data and can cache raw data as easily as file data. This can be useful for applications or equipment that read/write large amounts of data and support using raw disk partitions as storage devices for added performance. Block-based caching systems can extend to this raw data without problem.

---

<sup>5</sup> Sometimes referred to as a “Read Around” cache mode, but this conflicts with the other use of that term to mean a read that avoids (goes around) the cache

<sup>6</sup> A Google search of the phrase *block cache schemes* will return with a wide range of hits from university dissertations and journals articles

<sup>7</sup> Storage Area Network - Framework used to attach remote computer storage devices to servers. Storage devices appear as if they were attached locally to the operating system

<sup>8</sup> Network-Attached Storage - File-level data storage that is directly linked to a storage area network or other network

## File-Based Caching

File-based caching is exactly what it sounds like; the caching of assembled files instead of the constituent blocks. The cache solution must integrate at a higher level in the operating system kernel in order to gain access to the file context.

A file-based caching approach can have a large performance implication. Files can be extremely large and thus natively caching them could be very time consuming. When accessing a hundred gigabyte file, only a small part of that file might ultimately be accessed by that process in that session and thus caching the whole file would be a waste of time and resources. NEVEX CacheWorks side-steps this problem by caching only the *active portions* of each file, utilizing “sparse files”<sup>9</sup> in the cache, and populating the file as parts are accessed. This looks more like a block caching scheme, except it also retains the advantage of the file context when making caching decisions.

Another issue that must be considered is that to act on file context it is sometimes necessary for the file-based caching system to come out of the kernel into user space<sup>10</sup> to perform some of its operations. This is a time expensive operation and if not handled correctly can reduce performance significantly.

On the positive side, most operating systems already have a file cache that they maintain in memory. This cache is relatively small in comparison to the size of Flash devices but is significantly faster. When operating at the file level, it is possible to manage this memory cache and incorporate it into the caching policies. In another white paper, *The Vision of Multi-Level Caching*, NEVEX discusses the mechanism whereby it extends the Windows Server memory cache using Flash, and how its software seamlessly handles the promotion of cached files into DRAM memory. This method provides a dramatic performance boost to the server.

A further, very significant performance advantage is achieved by focusing the cache on the specific files that support a high value application or process. This is derived from the application and data context awareness available by operating at the file level, and is discussed further in the next section.

## Context Awareness

The context awareness that is provided by being at the file level is where there is a significant performance win to be had over block caching schemes. The targeting of files and data that supports only specific applications eliminates collateral damage that would otherwise result from the eviction of important data from the cache due to the access of large, non-critical files. This requires some management overhead to define the business critical applications and/or data, but allows the solution to *only cache files belonging to specific applications or needs*. This selective caching results in a much higher cache hit ratio, and hence shows some very large performance gains.

Further intelligent caching decisions can be made that are unavailable at the block level.

---

<sup>9</sup> A type of file that utilizes file system space more efficiently by only storing the necessary blocks of a file, with the unneeded pieces represented by "empty" space

<sup>10</sup> The memory area in the Operating System where user mode applications work

For example, a file-based caching system could implement an algorithm which says *if the file is less than one megabyte in size then cache the whole file, otherwise cache the amount being read, plus an additional megabyte up to the end of the file*. Unlike the block case, the file view ensures the cache system always reads bytes from the correct file, never goes beyond the end of the file, and never accidentally pulls in bytes from another file.

Providing further benefit, the cache system can see and act upon other interesting key parameters, including what user and/or process is accessing the file. This gives the solution another level of control over cache operations.

Policies can be specified that govern which users or programs receive the benefit of the cache. For example, *cache only files used by the database system*, or, *only cache files used by the finance group during month-end* are reasonable types of policies that a file-based system could manage. This type of intelligent caching completely sets file-based caching apart from block caching technology.

### Extensibility

As mentioned previously, a file-based caching system requires tight integration with the operating system itself. This tight integration allows the cache system to make use of kernel and user space operations to manage the cache. Selective cache decisions based on context awareness and integration with the system memory cache are examples. This also means that the cache solution is tightly bound to each supported operating system platform, and porting requires much more effort than what would be needed for a block-based system.

Further, it is not obvious how a storage device that does not have a file system mapped onto it can be supported using a file caching system. Most block storage devices are used for file storage, so this is usually not an issue. However, as mentioned above, there are cases where file systems are not used, such as some database deployments or equipment systems that store data on raw partitions. These would need to be ported to a file system to be supported by the cache.

The converse is scenarios where the data access has no underlying block structure, for example data accessed over the network from a NAS or cloud storage solution. These types of storage systems present data as files, and would be obvious legitimate targets for a file-based caching system with little or no change.

Finally, due to the tight integration with the operating system, as new storage hardware is released it can be seamlessly supported by the cache system. For example, although we've only discussed caching to a Flash device, should a faster and/or less expensive storage media become available then the caching system would be able to cache to it as soon as the device was supported by the operating system.

## **Conclusion**

In sum, block-based caching systems are fast low-level context-starved caching schemes, which work virtually independently of the file systems and operating systems whose data they are caching. File-based systems allow for context-aware decisions to increase the cache hit ratio for additional performance, but have reduced portability due to the tight integration with the operating system.

After carefully evaluating both schemes, NEVEX decided strongly that a file-based approach provided clear and necessary customer benefits. NEVEX CacheWorks only caches active portions of files, addressing a file-based performance concern, while retaining the significant advantages of being at the file level. The ability to selectively optimize the cache for specific applications and data, and the benefits gained from integrating with the Windows Server memory cache allow for significant performance gains. The capability to support non-block data sources such as a NAS or cloud storage system is another key benefit.

NEVEX provides an industry unique file-based intelligent cache solution for Windows Server. NEVEX CacheWorks is designed to provide selective optimized caching that guarantees storage performance gains targeted directly to the server applications that really need it.