EBOOK

SSD Caching

How to Boost Storage I/O Speed and Application Performance by 10X Through Content Locality Caching

VeloBit
Faster Storage Made Easy

Share this ebook

Start

INTRODUCTION

Content locality caching, pioneered by Prof. Qing Yang, Co-Founder and CTO of VeloBit, is a breakthrough in enterprise storage. Content locality adds a new dimension in caching that can boost application speed and storage I/O by 10x or more. The content locality caching algorithm prioritizes data blocks based on the "popularity" of their contents. When combined with line-speed delta compression, content locality caching optimizes the performance of solid state disk (SSD) and overcomes two of the main limitations of SSD – write speed and durability.

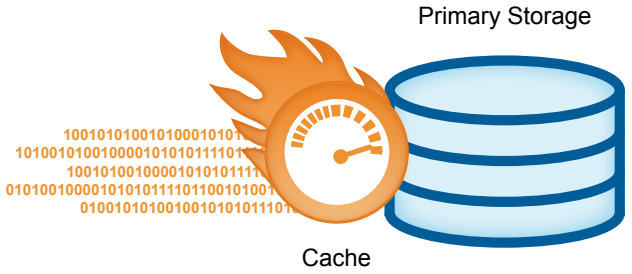
This paper provides an overview of content locality caching and illustrates how it can be successfully applied to make best use of SSD and to accelerate storage I/O and application speed.

CONTENTS

WHY CACHE APPLICATION DATA?

Caches are used to speed up access to data. A cache is a memory component placed in front of a data storage, processing, or networking device to transparently store data so that future data requests can be served faster. As data passes through a cache on its way to or from the storage/processing/networking device, some of the data is selectively stored in the cache. When an application or process later accesses data stored in the cache (a cache hit), that request can be served faster from the cache than from the slower device. The more requests that can be served from cache, the faster is the overall system performance.

There is a trade-off in cache cost and performance. Larger caches yield a higher cache hit rate and therefore better performance. Unfortunately, the hardware used for cache is generally more expensive than the hardware used for the storage, processing, or networking device. As a result, cache design is a tradeoff between size and performance. The best caching algorithms yield a higher hit rate for a given cache size.



The three most common types of caches are: CPU cache, used to speed up the processor; storage cache, intended to accelerate storage I/O; and web/network cache, designed to improve responsiveness of web applications. This paper focuses on storage cache.

WHAT APPLICATIONS BENEFIT FROM CACHING?

Storage device performance is measured by the speed at which the device executes sequential and random operations. Sequential operations access contiguous locations on the storage device and are generally associated with large data transfer sizes. Random operations access non-contiguous locations on the storage device and are generally associated with small data transfer sizes, e.g., 4 KB. Applications that require a high number of random reads and writes (e.g. database applications) are typically **latency** or **I/O** bound, and applications that require streaming of large files (e.g. backup applications or video streaming) are typically **throughput** bound.

Storage caches accelerate applications when storage speed is a bottleneck, and when reducing storage latency and/or increasing storage **IOPS** will improve application speed. Storage latency is reduced when cache hits (i.e., when requested data is found in cache) speed access to data.

Specific examples of applications which can be accelerated by caching include:

- Financial analysis, research, simulation, or modeling applications
- Databases, such as SQL Server, MySQL, Oracle, PosgreSQL, Cassandra, etc.
- Enterprise applications such as enterprise search, MS Exchange, SharePoint
- E-commerce and social networking applications

A storage cache can also be a benefit when reducing storage latency or increasing storage IOPS will reduce the amount of hardware required to support a given workload. Specific examples include:

- Distributed applications such as Hadoop
- Virtualization, such as VMware deployments

In a storage system, **latency**, a synonym for *delay* or *access time*, is an expression of how much time it takes for a read or write operation to complete.

The **throughput** of a storage system is the average rate of data transmission. The throughput is usually measured in bytes per second.

IOPS (Input/Output Operations Per Second) is a common performance measurement used to benchmark computer storage devices like hard disk drives (HDD), solid state drives (SSD), and storage area networks (SAN). IOPS is inversely proportional to latency. The lower the latency, the higher the resulting IOPS.

An application is **I/O bound** if I/O speed is the bottleneck, i.e. if increasing IOPS will increase application performance.

An application is **throughput bound** if throughput is the bottleneck, i.e. if increasing throughput will increase application performance.

An application is **CPU bound**, if processor compute speed (rather than storage performance) is the bottleneck.

COMMON TYPES OF CACHING

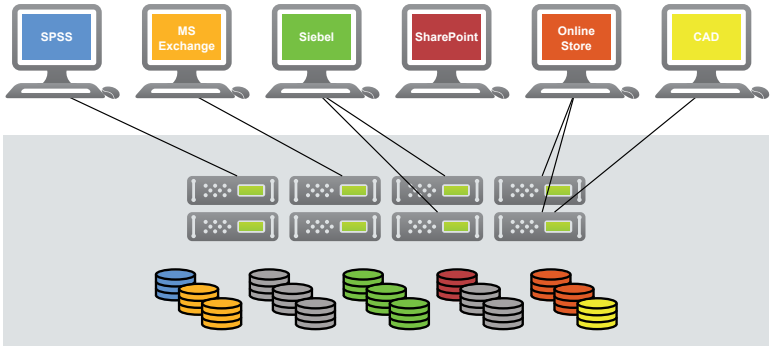
Since a cache's size is typically only a fraction of the size of the overall data set, a cache usually holds only a subset of an application's data. A cache is most effective when the data it holds is likely to be accessed in the future. The goal of cache algorithm design is to somehow "predict" which data will be accessed so that an optimal subset of the data is held in cache. The better the ability to predict future data access, the higher the cache hit rate and the better the application performance.

Caches have proven themselves in many areas of computing because most computer applications have predictable patterns for accessing data. The two most well-known principles that enable caching are temporal locality and spatial locality.

Data exhibits **temporal locality** if a data element (data block, file, field, column, attribute, etc.) that has been used at one point in time is used again in the near future. Algorithms based on temporal locality classify data into "hot" and "cold" based on the recency and/or frequency of use. "Hot" data is stored in cache.

Temporal Locality

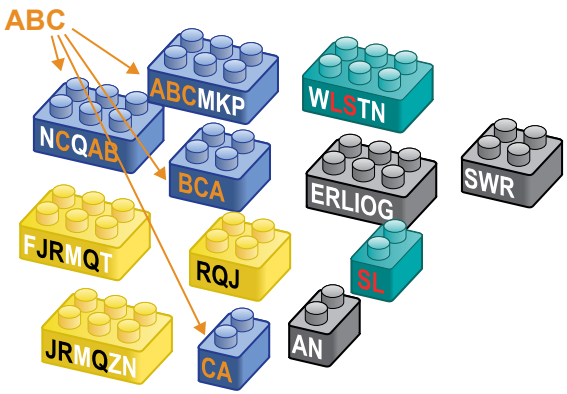
Spatial locality is a property of the data storage location. Data blocks that are stored next to each other in the storage medium exhibit spatial locality. In a hard disk drive (HDD) based storage system, search time is typically longer than data transmission time. Therefore for an HDD system, a cache relying on spatial locality improves IOPS by pre-loading multiple data blocks (the requested data and spatially contiguous blocks) whenever data is requested by the application.



Spatial Locality

Content locality caching is a new caching technique that has the potential to increase application speed and storage IOPS by 10x. Data exhibits content locality if blocks in a data set share similar or even the same content. Computer applications generate data in regular patterns, so there is a high degree of similarity, i.e. content locality, in the data blocks generated by most applications. This similarity can be leveraged to improve cache performance.

A cache leveraging content locality scans data blocks and assesses the "popularity" of the contents of each block. "Popular" data blocks are defined as blocks whose contents are often shared by other blocks. Popular data blocks are stored in cache and served much faster compared to fetching them from disk.



Content Locality

Let's contrast the three caching methods:

- A cache with temporal locality stores data blocks based on how **recently and/or how frequency they were used.**
- A cache with spatial locality stores data blocks based on their **physical storage location.**
- A cache with content locality stores data blocks based on the **popularity of their contents.**

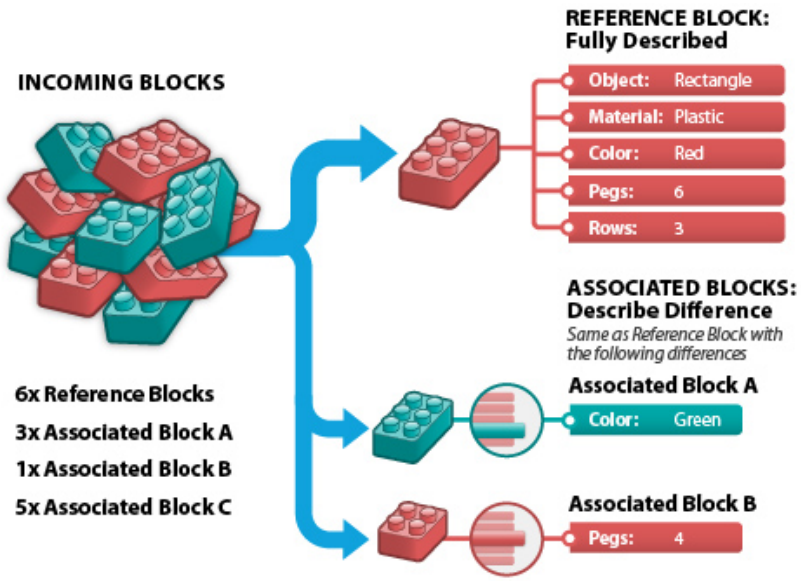
HOW CONTENT LOCALITY CACHING WORKS

The cache scans incoming data blocks at line speed and computes the "popularity" of each block. "Popularity" is a function of a) how recently the block was used, b) how frequently the block is used, c) how similar the block's contents are to other blocks.

There are two types of blocks stored in cache: reference blocks and associated blocks. Popular blocks are designated as *"reference blocks".* Blocks that are sufficiently similar to reference blocks are defined as *"associated blocks".* The VeloBit content locality cache represents associated blocks in a compressed form which describes a block using a pointer to a reference block and a *"delta"* that describes the differences between the two blocks. The combined delta and pointer entity is much smaller than the original associated block, yielding *"delta compression".* The compression is lossless, and enables the cache to hold much more data—increasing the cache hit rate, and boosting performance.

Associated blocks are decompressed when they leave the cache. Because all operations are performed at line speed and all data written to and read from the cache is in original (uncompressed) form, the cache is transparent to both the application and the primary storage device.

When the application requests data and a requested data block is in cache, a *reference block* is delivered directly and an *associated block* is re-constructed at line speed. Reconstructing an *associated block* from its compressed form consumes CPU



cycles. However, since CPU speed is much faster than storage disk I/O speed, *associated blocks* are reconstructed and delivered much faster than if they were fetched from disk. In addition, the VeloBit content locality cache algorithm is computationally efficient, so the CPU load is minimal.

The content locality caching algorithm prioritizes blocks based on the popularity of their contents. There is a strong correlation between the popularity of the contents of a block and the probability that the block will be accessed in the future. Additionally, because data is stored in compressed form, the cache appears much bigger than its physical size. The combination of caching popular data and compressing the data that is cached, increases cache hit rates. Higher cache hit rates produce a better application performance.

WHY CONTENT LOCALITY CACHING OUTPERFORMS OTHER CACHES ON SSD

Solid State Disk (SSD) Technology overview

Solid state disk (SSD) based on flash memory has emerged as a popular storage medium. Because SSD uses semi-conductor chips as the storage medium, it provides great advantages over mechanical disk in random read speed, power consumption, size, and shock resistance. However, SSD has limitations in write performance (read speed is much higher than write speed) and durability because of the physical properties of flash.

Let's explore the design of flash-based SSD so that we better understand what's causing these limitations. A typical NAND-gate array flash memory chip consists of a number of blocks, each of which contains a number of pages (e.g. a block with 64/128 pages of 4KB/8KB each). In NAND flash, blocks are the smallest erasable units whereas pages are the smallest programmable (i.e., writable) units. When a write operation is performed, the

SSD needs to first find a free page for the write. If there is no free page available, an erase operation is necessary to make free pages. A read operation usually takes a few or tens of microseconds, whereas a write takes hundreds of microseconds and an erase operation takes 1.5 to 3 milliseconds. Thus, the performance of an SSD is “asymmetric”—writes are much slower than reads.

The lifetime of flash is limited by the number of erase operations that can be performed on a block. Typically, a block can be erased only 10k times (for multi level cell: MLC) or 100k times (for single level cell: SLC) before it fails. After that, the block becomes bad.

After an SSD is initially filled with data, unused (stale) pages must be erased before new data is written. However, an SSD block often includes a combination of pages with current data and pages with stale data. Since a block is the smallest erasable unit in an SSD, it is not possible to erase just the pages with stale data without also erasing the pages with current data. Therefore, before a block is erased, all pages with current data must be copied into another block. This process is called *garbage collection*. Writing a single page to SSD can result in many writes operation if garbage collection is required to create space. This *“write amplification”* is a major cause of SSD wear and causes SSD performance to degrade as the device fills.

If the system were able to separate static data (data that does not change) from dynamic data (data that changes often), it would greatly reduce write amplification. Furthermore, if data is optimized to maximize reads and minimize random writes to SSD, both performance *and* reliability would be improved. Content locality caching is designed specifically to maximize cache hits and minimize writes to improve SSD performance and reliability.

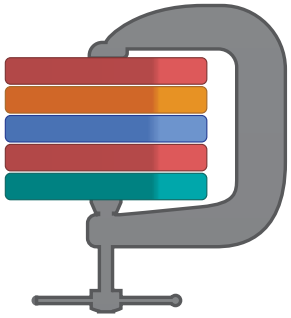
How content locality caching optimizes data for SSD

Content locality caching optimizes data for SSD in three major ways:

- **Compress Data**

  The contents of computer data objects (files, images, databases, etc.) include a number of similar and often repeating data blocks. A typical cache stores data at the object level and at this level there are often many redundant data blocks. Storing redundant data blocks is inefficient and writing redundant data to SSD causes unnecessary SSD write-wear—reducing the reliability and life-span of the SSD. Redundant data blocks also take unnecessary space—shrinking the cache, reducing the likelihood of a cache hit, and therefore lowering performance.

  A cache that leverages content locality eliminates redundancies and efficiently compresses content at the block level. As a result, the cache can contain a larger volume of unique content, which increases the cache hit rate and application performance. Additionally, a cache that employs content locality minimizes SSD write wear by eliminating unnecessary writes of redundant blocks.



- **Compress Writes to SSD**

  Write amplification, caused by SSD garbage collection, slows SSD performance and causes SSD wear. Content locality caching aids garbage collection and reduces write amplification. As described previously, a content locality cache stores *associated blocks* in compressed form. Compression makes associated blocks smaller, which is a benefit when an SSD is used as part of the cache. Compressed associated blocks are easier to scatter and write in existing open page slots. This results in fewer data re-organization tasks on the SSD and, consequently, fewer erase operations.

- **Optimize Data For SSD**
  SSD reads and writes are asymmetric: read operations are much faster than write operations; a write operation often is preceded by an erase operation; and the lifetime of SSD is limited by the number of erase operations performed. So, both SSD performance and SSD reliability are improved if data is organized in such a way that the cache closely resembles a “read-only” cache.

  The content locality caching algorithm structures data based on the popularity of the contents of data blocks. While popularity can change over time as more data is written by the application, changes occur gradually. The result is that “popular” *reference blocks* remain popular for a long time. Therefore, reference blocks are usually written once but read many times (as they are used to describe *associate blocks*). The result is that, as a whole, there are many more read operations than erase operations in a content locality cache. Alternative caching algorithms require a much higher number of write and erase operations. Consequently, content locality caching achieves a much higher level of both SSD performance and reliability.

HOW CONTENT LOCALITY CACHING DIFFERS FROM DE-DUPLICATION

Content locality caching and the delta compression it employs are sometimes confused with data deduplication. While there are similarities, content locality caching is fundamentally different from deduplication. Deduplication is a subset of the operations performed by the VeloBit content locality caching algorithm.

Data deduplication is a data compression technique for eliminating redundant data. In a dedupe system, unique chunks of data are identified and stored during an analysis process. As additional data chunks are analyzed, they are compared to the stored chunks to look for *identical* matches. When a match occurs, the redundant chunk is replaced with a pointer to the stored chunk. Some deduplication algorithms work at the data block level (similar to content locality caching). Deduplication algorithms are used to reduce storage or network bandwidth consumption. In comparison to content locality caching, data deduplication is computationally intensive, and is not suitable for use in caching.

Like deduplication, content locality caching also eliminates redundancies between identical blocks. In addition, it can compress an associated block that is similar to, but not identical to, a reference block. The fundamental difference between content locality caching and deduplication are the cache functions, beyond compression, that benefit application performance. These functions include: prioritizing blocks based on their popularity; making optimal decisions on which blocks tostore in, or evict from, the cache; and managing reads and writes to SSD. The final difference between content locality caching and data deduplication is computational efficiency. VeloBit's content locality caching is extremely computationally efficient.

In VeloBit's current implementation of content locality caching, data is compressed only in cache (where compression boosts performance and optimizes SSD). Data is written to the primary storage device in its original decompressed form. The use of delta compression for reducing consumption of primary storage is a topic for the future.

VeloBit combines temporal, spatial, and content locality caching and line-speed delta compression to deliver the industry's best performance

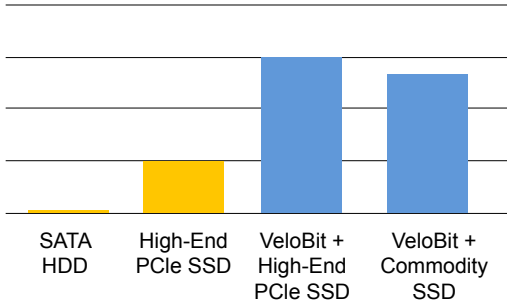
VeloBit is the first solution to simultaneously leverage all three localities of reference concepts (temporal, spatial, and content locality) to create the world's fastest and most efficient cache. In addition, VeloBit software employs a number of other proprietary features, such as flash drive optimization, horizontal architecture, and automatic configuration, to enhance performance while reducing cost and minimizing disruption and complexity.

Learn more about VeloBit

Free Trial

See a Demo

How Content Locality Optimizes Data For SSD	Benefit
Data in cache is compressed	• Enlarges the cache and SSD, increasing the cache hit rate and boosting performance • Reduces the required SSD size, lowering cost
Writes to SSD are compressed	• Reduces the write load on the SSD. This lowers cost, reduces wear, and boosts performance
Optimize data for SSD	• Boosts performance • Lowers cost



Storage Performance

SUMMARY

Storage caches are used to reduce latency and increase IOPS, improving application performance and/or reducing the amount of hardware required to support a given workload. Existing caching methods today exploit only temporal locality and spatial locality. A cache with temporal locality stores data blocks based on how recently and/or how frequency they were used. A cache with spatial locality stores data blocks based on their physical storage location.

Content locality caching is a new caching mechanism that prioritizes data blocks based on the popularity of their contents. Popular data blocks are stored in cache as read-only blocks. Related (associated) blocks are stored in cache in a compressed form. Compared to traditional caching based on temporal and spatial locality, content locality caching provides a leap forward in terms of cache efficiency. This is especially true for SSD-based solutions because read-operations with SSDs are fast, but write operations are slow and wearing. The net effect of content locality cache is that applications run much faster than traditional systems.

VeloBit is the industry's first solution that leverages content locality caching to improve performance by 10x while reducing cost and minimizing disruption and complexity. VeloBit is a software only solution that seamlessly installs, automatically optimizes data for SSD, and is transparent to applications and primary storage.

Learn more about VeloBit

Free Trial See a Demo