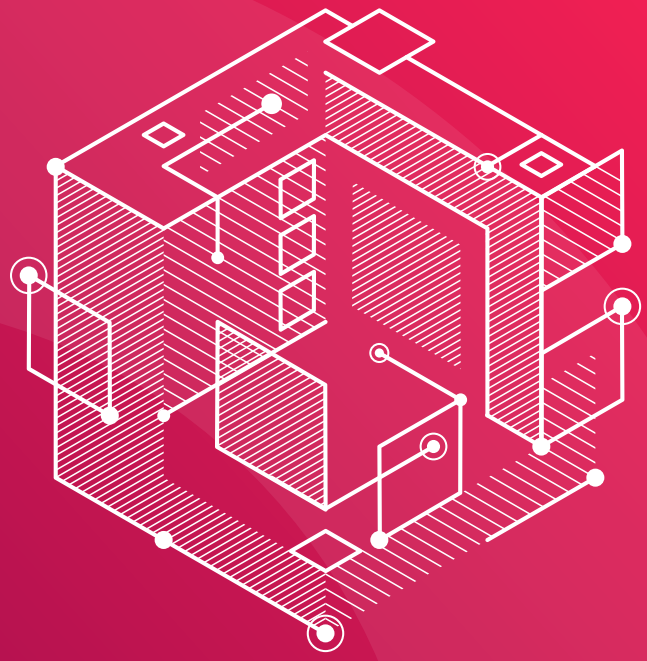
Caching

TECHNICAL CAPABILITY SHEET

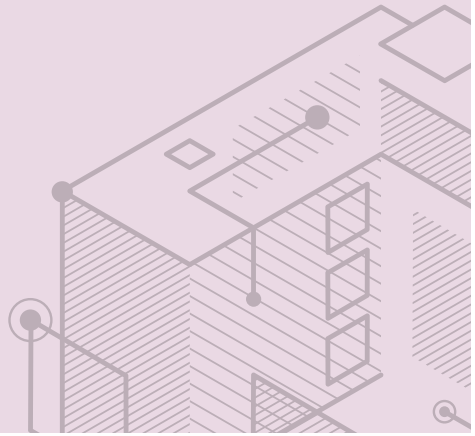


open DRIVES

Capability Definition

Caching is the process of leveraging both RAM memory (which is incredibly fast) as well as NVMe storage (also fast but not like RAM) to write data in the instance that the user calls for it. Providing data to end users and applications is much faster when pulled from cache rather than accessing the storage media (especially spinning disks). Through the ZFS file and volume manager, Atlas Core oversees the different layers of cache memory and what needs to be written to (and remain) at each layer. Ultimately, it runs smart logic to determine what type of data from disk needs to be cached, in which layer of cache the information should be stored, and how long it should remain in cache based on the capacity of cache space at any given time.

Note: Caching is not the same for every OpenDrives storage system. The Ultimate series, for example, already uses NVMe as its data drives, therefore only uses flash memory for its cache. Therefore, be cognizant of the fact that caching will behave slightly differently—and with different results—depending on the series and configuration of the OpenDrives storage system in question.



Outcome & Value

The outcome of caching (also referred to as inline caching or intelligent caching) is enhanced performance through quicker data retrieval (from faster cache) and accelerated responsiveness as a result. Because of the anticipatory intelligent nature of determining information to cache that hasn't even been requested yet, Atlas Core can significantly speed up workflows and data requests. This enhanced performance also provides a much better end-user experience because delays in getting to data are minimal to none. Faster performance equates to higher user adoption and quicker workflow execution, with a direct impact on a company's productivity and bottom line.

How It Works

Atlas Core creates potentially several layers of cache depending on the type of memory and storage media the system has. All OpenDrives storage systems have RAM memory, which provides the top layer of read cache known as ARC (adaptive replacement cache). The more RAM memory a system has, the more Atlas Core can store in ARC. Keep in mind that RAM memory provides the best performance (in the nanosecond range), so reads from ARC are incredibly fast and contribute best to smooth execution of data reads. Of course, Atlas Core uses RAM memory for other processes as well, so RAM is a finite and valuable resource. Therefore, Atlas Core enacts intelligent assessment of the data which is placed into ARC and how long it remains there.

NVMe storage provides the next layer of read cache, known as L2ARC. While not quite as fast as RAM, NVMe storage operates in the microsecond range so is still significantly faster than disk. L2ARC within NVMe serves as a type of overflow for the top-level RAM-based ARC cache. However, because of the intelligent logic Atlas Core runs, it can also prefetch a fairly large amount of information within the storage pools depending on the size of L2ARC. Ultimately, Atlas Core determines the data being explicitly requested versus the data which (though not explicitly requested at any given point) might provide a higher payout somewhere within the workflow or workload being run.

To make this determination, Atlas Core must distinguish between different types of data, because all data simply isn't equal in terms of immediate usability (by the user or application) and a subsequent uplift in performance. Data that is most frequently used (MFU) has a much higher probability of being useful and therefore requested than data that has been most recently used (MRU). Therefore, Atlas Core gives a higher priority to MFU data than to MRU data. Also, data that's all been written approximately at the same time has a higher probability of being requested if a portion has already been read.

The intelligent treatment of these different types of data and how to fetch and place the highest-priority information into ARC, then into L2ARC, is the real value statement behind our caching capability. However, this is only half the story. Filling these two layers of cache is the first part, but also determining which data needs to be evicted to make way for higher-priority information is equally critical. Again, Atlas Core performs eviction determination even before the actual data needs to be deprecated (from ARC to L2ARC) or removed from cache entirely. Even when deprecated from ARC, data may still exist in L2ARC until the intelligent logic determines that higher-priority MFU or MRU data needs to replace it. All of this is completely transparent to the user. The user simply sees much better performance through system responsiveness and the elimination of lag when requesting data.

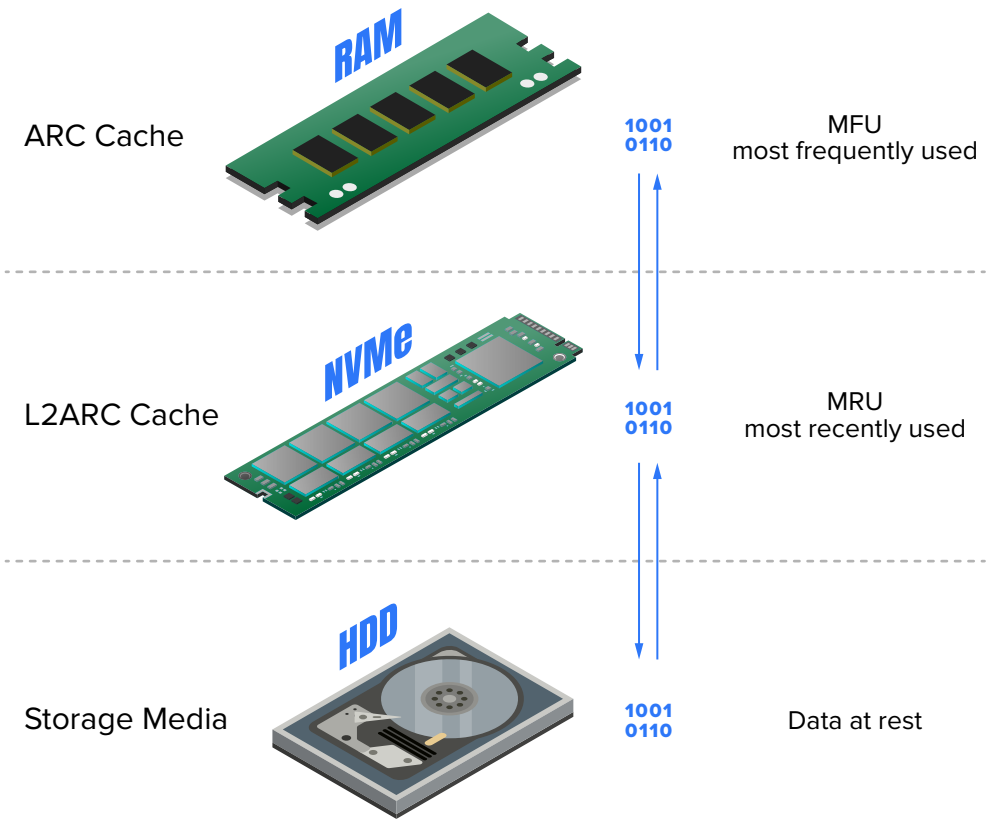
Note: Prefetch is a type of data caching and retrieval based on the same notion of grabbing information in cache even before it has been requested by the user or application. Caching is the mechanism of storing data in the different layers of cache and managing the persistence and eviction, while prefetching is based on specific data requests—especially in linear streams—and caching then reading additional data prior to the explicit request.

Characteristics

The caching capability has the following characteristics:

- Multiple layers of cache (ARC, L2ARC) depending on the series and system configuration
- Intelligent operation that distinguishes between different types of data (MFU, MRU) and manages the location, deprecation, and ultimate eviction of data
- Persistent caching means that Atlas Core has a much better determination of MFU and MRU data—upon reboot or failure, it accesses transaction logs to try to restore as much in cache as possible
- L2ARC provides the prefetching capability, and if a system has considerable free space in L2ARC it will use intelligent logic to prefetch based on current data use or MFU/MRU determinations
- Media with high frame rates really benefits from our caching and prefetching capabilities
- Cache often maintains multiple copies of data, both in ARC and in L2ARC until delisting/deprecation from ARC occurs, all maintained by Atlas Core via a master cache file
- If the system has L2ARC, Atlas Core has more ways to tune caching operations
- To see a tangible example of caching, you can play some media for a given duration and then stop—by looking at the dashboard in Atlas Core you can see the bandwidth continue to increase, which indicates a freshly booted system attempting to continue to cache MFU data

The following diagram illustrates the caching layers and deprecation/eviction processes. The diagram displays both ARC and L2ARC cache, the storage media, and movement of data between these layers.



Further Reading

As previously stated, caching drives the prefetch operation. For more information about prefetching, please refer to the Prefetch Capability Sheet which explains in more detail the mechanics behind ZFS' prefetch capabilities which we've implemented within Atlas Core.

sales@opendrives.com | +1 888-778-5491