FLASHARRAY™ FILE SERVICES

Unifying block and file storage in FlashArray systems

As storage array capacities grow larger,[1] the need for separate block and file servers is determined more by array capabilities than by capacity limitations. Already established as the premier all-flash block storage for data centers, FlashArray systems present *virtual volumes* (LUNs) to host computers whose software imposes file system structures and semantics on them. *FlashArray File Services* adds the ability to host shared file systems such as home directories, project shares, VDI profiles, and so forth.

From a software architecture perspective, FlashArray File Services is a peer of volume services; both utilize the same underlying virtual and physical storage management. This brief describes how FlashArray File Services extends the FlashArray architecture to deliver the same reliability, performance, security, ease of use, and Evergreen™ longevity for shared file systems.

FEATURES

FlashArray File Services supports the features that enterprises require from file servers:

- Hundreds of millions of files in thousands of separate exports (*shares*[2])[3].
- Thin provisioning—byte ranges in which no client has written data occupy no space.
- Concurrent SMB and NFSv3 client access. SMB protocol support is entirely Pure Storage-developed; it does not utilize any pre-existing or third-party technology.
- Nested exports of subtrees, each with its own client access, snapshot, and quota policies.
- Active Directory (AD) and Lightweight Directory Access Protocol (LDAP) authentication (with Kerberos and NTLM for SMB access) and AUTH_SYS for NFS.
- Ad hoc and automatic (policy-based) space-saving snapshots of *managed directory* contents and policies. (Managed directories are described on page 7.)
- Quotas that limit the storage space consumed by managed directories.
- Replication of exports' data and policies to remote FlashArrays.

When FlashArray File Services is enabled, the array dedicates a portion of its resources to it. The file and volume data they store are intermixed on flash, however. No per-file system provisioning or capacity reservation is necessary, or indeed possible.

---

[1] With typical 5:1 compression, a single-chassis FlashArray//C60 system can store almost 7PB of user data.

[2] This brief uses the terms *export* and *share* interchangeably.

[3] As of early 2023, File Services supports up to one billion files and 5,000 separate exports in a single FlashArray system.

ARCHITECTURE

The conceptual storage system software model in Figure 1 is useful for explaining how FlashArray File Services fits into the Purity//FA software architecture. Whether a storage system presents blocks, files, or objects to clients, it typically includes the four layers shown in the figure:

Protocol(s)
Communicates with and executes commands from client computers (usually called *hosts* in block storage contexts).

Data Model
Organizes stored data as files, volumes, or objects which it presents to clients, implements data model semantics, and manages the storage presented by the virtual storage layer.

Virtual Storage
Provides storage to the **Data Model,** typically virtualized to enhance I/O performance, data reliability, flexibility, or a combination. Common virtualization techniques include caching, mirroring, striping, and erasure coding.

Physical Storage
Ultimately, systems store data in devices containing persistent memory, such as flash, magnetic disk, and tape. This layer controls devices and transfers data to and from them.

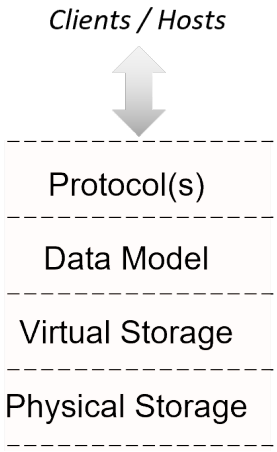

Figure 1: A Generic Storage System Model

MODELING FLASHARRAY FILE SERVICES

Figure 2 illustrates the Purity//FA software architecture in the context of the model in Figure 1. In particular, the figure shows how Purity//FA provides coequal volume and file services:

Protocol(s)
At the protocol layer, **Volume Services** communicates with hosts using any of the FCP, iSCSI or NVMe protocols over Fibre Channel or Ethernet networks. File Services communicates with clients using any of the Server Message Block (SMB), Network File System (NFS), or HTTP protocols via TCP/IP connections on Ethernet networks.

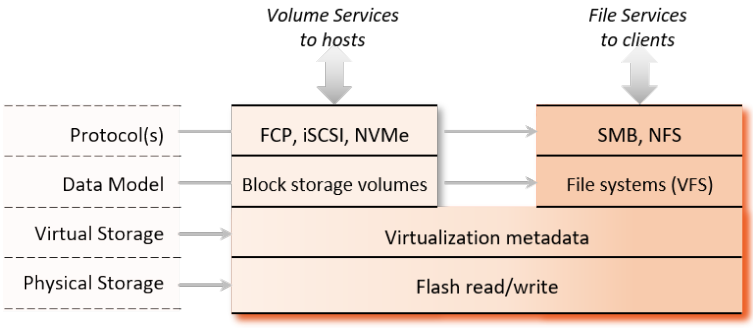

Figure 2: Purity//FA's Layered Architecture

Data Model
Volume Services presents disk-like virtual volumes to hosts, and manages host access,

volume resizing, snapshots, and replication of *protection groups* of volumes.
File Services exports directory hierarchies to clients via NFS and/or SMB, and manages client access, quotas, snapshots, and file system replication.

Virtual Storage
Both Volume and File Services utilize the same virtual storage layer. Purity//FA presents blocks of virtual storage to the **Data Model** layer via metadata structures that indicate the blocks' locations on physical storage. The **Virtual Storage** layer minimizes consumption by *reducing* (deduplicating and compressing) data prior to storing it. Deduplication is array-wide; a single copy of identical content may be shared by multiple volumes and/or files.

Physical Storage
Purity//FA allocates flash in *segments* distributed across multiple *DirectFlash® Modules* (DFMs) to optimize I/O performance and resilience. Multiple checksums and erasure codes detect errors and protect against data loss due to read failures.

INSIDE FLASHARRAY FILE SERVICES

As Figure 2 illustrates, FlashArray Volume and File Services are architectural peers. Both utilize the same virtual and physical storage substrate. Two key advantages of common underpinnings are:

Identical Features
Volume and File Services organize data and store it persistently on flash in the same way, so the well-known features of FlashArray volumes—data reduction, automatic back-end load balancing, always-on encryption, recovery from read failures, and so forth—proven by nearly a decade of field experience in tens of thousands of arrays, also apply to files.

Simultaneous Volume and File Access
Newer FlashArray models support Volume and File Services simultaneously, in many cases eliminating the need for separate storage systems for file and block data.

THE DATA MODEL LAYER

The FlashArray File Services **Data Model** implementation uses a protocol-independent *Virtual File System* (**VFS**) to provide services to SMB and NFS clients via the **Protocol** layer. Part of **VFS** is a **Data Store** (similar to a conventional object store) that interacts with the **Virtual Storage** layer. The **Data Store** manages virtual storage for **VFS,** providing the scalability and performance needed to support hundreds of millions of files.

The Data Store Component of VFS

Most conventional file server software manages storage directly. The Purity//FA **Virtual** and **Physical Storage** layers are more versatile in that they reduce data, protect against read failures, and balance back-end load automatically for both files and block volumes.

The **Data Store** component of **VFS** isolates namespace management and semantics from direct interaction with storage. It is a highly scalable *key-value store* that provides a single flat namespace of *items* (key-value pairs). It has no awareness of directory hierarchies or file system semantics. **VFS** uses it to implement namespaces and to store directory structures, file attributes, and data.

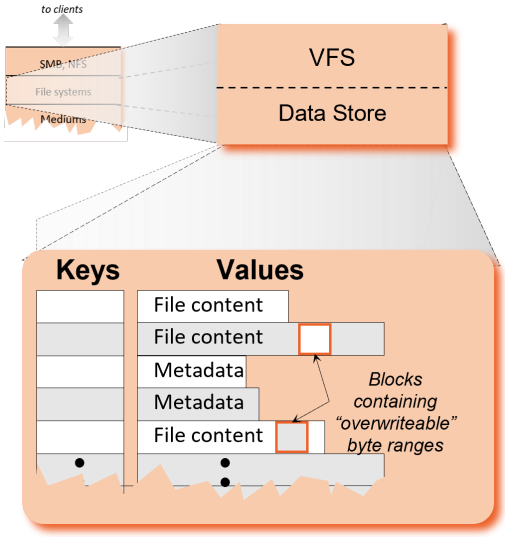

Figure 3: The Data Store

The **Data Store** resembles a conventional object store in that each key has a corresponding value that holds data and/or metadata. It differs, however, in that it supports overwriting ranges of bytes within a value (Figure 3), a capability required to make it possible for clients to overwrite byte ranges within files. When a client overwrites data in a file, **VFS** overwrites only the part of the file's object that contains the data written by the client.[4] The storage layers append overwritten blocks to the array's log and update metadata to reflect their new locations on flash.

[4] Purity//FA stores data in an append-only log that intermixes file and volume data, so "overwriting" is conceptual. Overwrites are appended to the log and metadata is updated accordingly. Space occupied by overwritten content is reclaimed by a background "garbage collection" process. The Appendix (page 15) describes Purity//FA virtual and physical storage management.

The VFS Hierarchy

VFS uses the **Data Store** to implement a master hierarchy that is not exposed to clients. The hierarchy contains metadata and file system root directories, both of which are only used internally. Figure 4 illustrates its major elements:

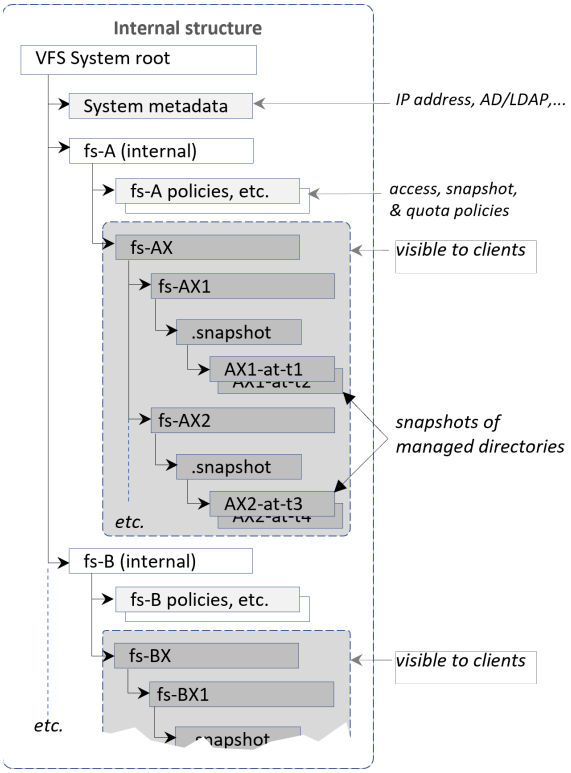

Figure 4: The VFS Master Hierarchy

VFS System root

Root of the **VFS** internal hierarchy. *Not exposed to clients.*

Internal file system roots (fs-A, fs-B,...)

A **VFS**-internal root for each file system. Contains metadata (e.g., client access, quota, and snapshot policies). *Not exposed to clients.*

Exports (fs-AX, fs-AX1, fs-AX2, fs-BX, fs-BX1 in Figure 4**)**

Managed directories (described on page 7) exported to clients. **VFS** creates top-level managed directories during file system creation. Each one can contain up to seven levels of managed subdirectories (e.g., **fs-AX1**, **fs-AX2**, **fs-BX1** in the figure) that can be exported separately with their own policies (which are subordinate to those of the parent). For example, a managed subdirectory's space quota or user permissions may not exceed those of its parent.

.snapshot Subdirectories

FlashArray File Services stores snapshots as subdirectories of a top-level **.snapshot** subdirectory of the managed directory (e.g., **AX1-at-t1**, **AX1-at-t2**, etc.). Snapshots are *immutable*—array administrators can destroy them, but while they exist, their contents cannot be altered, so they are exact point-in-time copies of export data and policies. Clients can:

- browse snapshots (read the contents of their directories and files), including listing previous versions for Windows clients.
- restore deleted or corrupted files by copying older versions of them from snapshots to their original locations.
- copy files from snapshots to other locations.

FlashArray File Services Structural Integrity

Many common operations require modifications to multiple file system data structures. For example, to move a file to a different directory requires modifying both original and target directory structures as well as the file's timestamp. To guarantee file system structural integrity, these operations must be *atomic*—should a failure occur while they are in progress, the file system must either show them as completed or never to have occurred after it restarts.

All operations that affect the **VFS** file system structure (e.g., creating, appending, truncating, moving, or removing files, changing attributes and policies, etc.), both internal and client-initiated, are *transactional* (i.e., atomic). If an array fails before Purity//FA has signaled completion of an outstanding operation to the client, then after restart, the **VFS** persistent hierarchy reflects either the pre- or post-operation state but no intermediate state that could result in structural inconsistencies.

*FlashArray File Services protects file system **structural** integrity, but as with any file system, client applications that use files must provide for recovery from client, server, and network failures that occur while application operations on files are in progress.*

Storing Data and Metadata

VFS uses the **Data Store** to construct the hierarchies illustrated in Figures 5 and 4 and to persist data and metadata on flash. Each **Data Store** item consists of a key and a value (a data or metadata item) up to 64 terabytes in size. The **Data Store** uses **Virtual Storage** services to write items on flash and retrieve them on client request or for FlashArray File Services internal use.

The **Data Store** manages files and file system metadata similarly to the way in which Volume Services manages volumes, using metadata structures to map virtual addresses (i.e., {inode, offset} tuples) to the locations of current data on flash. Purity//FA therefore delivers the same performance, resiliency, and data reduction properties for both files and volumes.[5]

The **Physical Storage** layer allocates flash in which it stores volume and file data from a single system-wide pool. Like FlashArray volumes, files are inherently *thin-provisioned*—aside from the small amount of metadata that describes them, only data that clients write occupies physical storage. The **Physical Storage** layer stores incoming file and volume data and metadata in a log in approximate order of arrival. The Appendix (page 15) contains a brief description of Purity//FA flash organization and management.

[5] That the software architecture originally designed to support hundreds of volumes scales to hundreds of millions of files is a testament to its versatility.

MANAGED DIRECTORIES

VFS *managed directories* are a unique FlashArray File Services construct that is largely responsible for its versatility. Unlike ordinary directories that can be manipulated by clients with appropriate permissions, only array administrators can control managed directories. FlashArray File Services supports up to eight levels of managed directories in a file system hierarchy. They differ from ordinary directories in that:

- Only array administrators can create, delete, and export them and manage their policies.
- Only array administrators can attach policies to them to limit virtual space consumption, control client access, and schedule snapshots.
- When clients move files or subtrees between managed directories, the moved files' inode numbers change.
- Hard links may not cross managed directory boundaries.

Figure 5 is an example of a three-level managed directory structure. File system **fs-A** is exported to clients as **fs-AX**. Managed subdirectories **fs-BX** and **fs-CX** are exported, potentially to different clients and with different policies. Clients of **fs-AX** have access to its entire hierarchy, including the **fs-BX** and **fs-CX** sub-hierarchies, whereas clients with **fs-BX** and **fs-CX** access only have access to those exports' hierarchies and are subject to their policies.

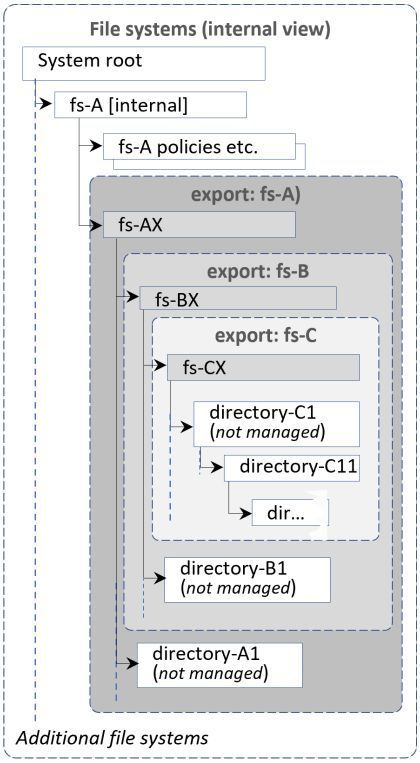

Figure 5: Managed Directories

A FlashArray File Services file system can support thousands of managed directories, each exported independently. Individually exported managed directories, each with its own policies for client access, snapshots, and quotas, largely eliminate the need for different applications to use separate file systems to isolate their data.

FlashArray File Services only replicates entire file systems (i.e., top-level managed directories) to remote arrays. A file system's entire tree is replicated, including managed subdirectory contents and policies and ordinary subdirectory contents. Each file system can be replicated to a single target. Target array administrators can export replicated file systems and managed subdirectories for read-only access by clients.

SECURING FLASHARRAY FILE SERVICES

FlashArray File Services controls client computer and user access to data and protects data against both misappropriation and loss due to user and administrator errors.

CONTROLLING CLIENT AND USER ACCESS

Array administrators can restrict access to FlashArray File Services to subsets of an array's Ethernet ports by creating *virtual network interfaces* (**VIF**s) through which arrays present exports.

Arrays utilize LDAP (for NFS), Active Directory (for NFS and SMB), or NTLM (for SMB) to authenticate users before granting access to exports for which array administrators have authorized them. Alternatively, administrators can disable user mapping to bypass NFS client authentication (a feature known as AUTH_SYS).

PROTECTING DATA AGAINST MISAPPROPRIATION

Purity//FA's **Physical Storage** layer encrypts all data and metadata it writes to flash and NVRAM using AES-256. Encryption is "always-on"—it is not an option. The software manages encryption keys internally; they are never exposed on any external interface.

Thus, even if an attacker removed flash devices from an array and could somehow retrieve their contents, stored data would not be exposed.

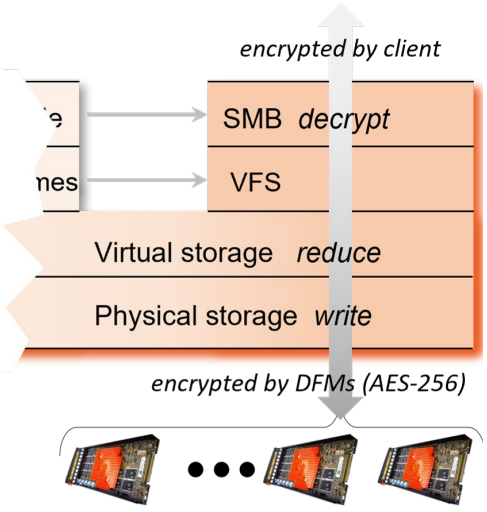

Figure 6: SMB "on the wire" Encryption (writing)

For situations in which network security cannot be guaranteed, FlashArray File Services can be configured to use SMB's *on-the-wire* data encryption. With on-the-wire encryption, SMB clients encrypt data before sending it to the array. The **Protocol** layer decrypts incoming data prior to processing to allow deduplication and compression of the clear text. DFMs encrypt all (reduced) file and volume data and metadata before staging it in NVRAM or writing it on flash. Encrypting data on the SMB client-to-array path secures it against interception from origin to retrieval without sacrificing the benefit of data reduction.

PROTECTING DATA AGAINST USER AND ADMINISTRATOR ERRORS

Purity//FA provides two facilities that protect against file and volume data loss due to user, administrator, and application errors:

Snapshots

Purity//FA takes *point-in-time* (logically instantaneous) snapshots of managed directory contents, either automatically according to policies or on administrator command. Array administrators can destroy snapshots but cannot alter their contents. They are exposed to clients as descendants of the top-level **`.snapshot`** subdirectories of managed directories.

To restore files to a point in time prior to a data loss or corruption event, a user would delete the corrupt files and replace them by copying earlier versions from a snapshot taken prior to the event.

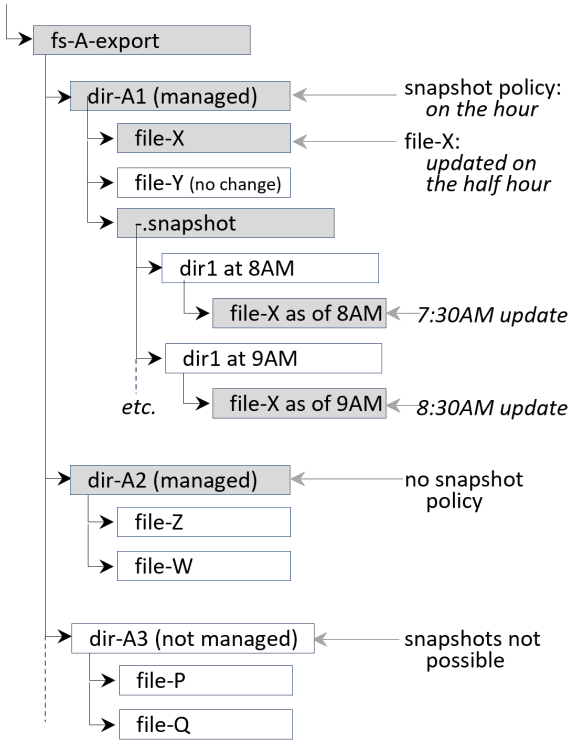

Figure 7: Snapshots of Managed Directories

Snapshots consume physical storage only when clients alter data in the managed directories on which they are based. Array administrators typically automate snapshot management by attaching policies that specify frequency (as often as every five minutes) and retention to managed directories. Purity//FA automatically eradicates snapshots after their retention periods have elapsed.

Eradication delays

When an array administrator explicitly *destroys*[6] a snapshot, it becomes inaccessible to clients immediately, but it remains recoverable for 24 hours. During the 24-hour *eradication delay period*, the administrator can restore it for client use. Administrators can explicitly *eradicate* destroyed snapshots (e.g., if physical space is urgently required), causing background reclamation of the storage they occupy to commence immediately. Once eradication of a snapshot begins, whether due to lapse of its retention period or by administrator command, it cannot be restored.

[6] Pure Storage CLI commands use the verb *destroy* to direct removal of objects that contain user data. For other objects, the CLI uses the more conventional *delete*.

SafeMode

Snapshots can help recover from ransomware and other forms of malware attacks by preserving previous images of data that attackers have encrypted or corrupted. Attackers realize this, and often attempt to infiltrate array administrator accounts so they can eradicate snapshots that could be used for recovery before they mount attacks on live data.

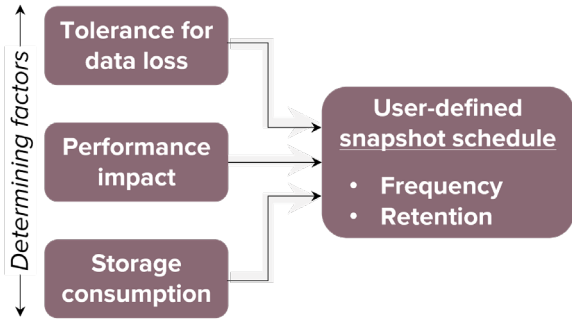

Figure 8: Determining Snapshot Schedules

To prevent attackers from eradicating snapshots prematurely or altering schedules to cause premature eradication, FlashArray owners can enable Purity//FA's *SafeMode* feature. With SafeMode enabled, destroying snapshots or altering schedules requires live (e.g., by telephone or video conference) cooperation between Pure Storage Technical Services engineers and designated trusted user representatives. Enabling SafeMode protects both volume (or protection group) and file system (top-level managed directory) snapshots from premature eradication.

SafeMode does not create snapshots; data owners must determine their malware protection requirements, the impact of snapshots on performance, and the cost of incremental storage consumption, and define appropriate schedules for protection groups and file systems.

USING FLASHARRAY FILE SERVICES

Most FlashArray models support FlashArray File Services.[7] Readers should consult with Pure Storage representatives to identify the products that meet their consolidated file and volume storage and performance requirements.

CONTROLLING CLIENT ACCESS TO FILE DATA

Client computers connect to FlashArray File Services exports via **VIF**s that include some or all of an array's Ethernet ports. In order to make transparent failover and non-disruptive upgrade possible, each VIF must include one or more ports on each array controller. Arrays connected to switches on which the *Link Aggregation Control Protocol* (LACP) is enabled support LACP for enhanced performance and availability.

Array administrators create *access policies* that specify rules for client access. Access policies are independent objects; any policy can be associated with any export. Administrators manage them by adding and removing client access permission rules and by enabling and disabling entire policies. Modifications to a policy immediately affect all exports with which it is associated.

ACCESSING FILES VIA SMB AND NFS

FlashArray File Services uses **VFS** to read, write, and manage files whether clients access them via SMB (versions 1-3[8]) or NFSv3. The **Protocol** layer adheres to SMB and NFS locking rules and blocks attempts by clients using NFS to access byte ranges locked by SMB clients. However, because NFSv3 is stateless, issues may arise when using it to access data used by applications that rely on SMB state. Two significant examples are:

NFSv3 access to files that are 'open exclusive' by SMB
Because NFSv3 does not have an explicit open operation, it does not respect SMB's *open exclusive* state. NFS clients can read and write files that are open exclusive by SMB clients.

NFSv3 client access to data with SMB oplocks
NFSv3 client accesses to data within the scope of an SMB client's oplock generate *break notifications* to the SMB client per the protocol specification. Typically, client software degrades or releases oplocks as appropriate to the circumstances. This is usually transparent to applications.

For these and similar reasons, however, Pure Storage does not recommend enabling NFSv3 access to exports used by applications that rely on SMB locks for correct operation.

---

[7] Some older FlashArray models (e.g., X50R2) do not support concurrent use of FlashArray File and Volume Services. X10 models do not support FlashArray File Services.

[8] Pure Storage discourages use of SMB version 1 due to security deficiencies that are remedied in newer versions. FlashArray File Services does not support SMB Version 3 *continuously available shares* or multi-channel links.

QUOTAS

Array administrators can assign quotas to managed directories[9] to limit the amount of data that clients may store in them. Quota limits apply to data as written by clients, prior to reduction by Purity//FA.

FlashArray File Services supports *nested* quotas. An administrator can assign separate quotas to a managed directory and its managed subdirectories. No subdirectory quota can exceed that of the parent. The sum of all subdirectory quotas might exceed a parent's quota, but the parent quota limits space consumption for the entire tree, as Figure 9 illustrates.

For example, an export and each of its managed subdirectories might all have 100GB quotas. The file system's quota can be consumed in any way, including by its managed subdirectories, but total consumption cannot exceed 100GB.

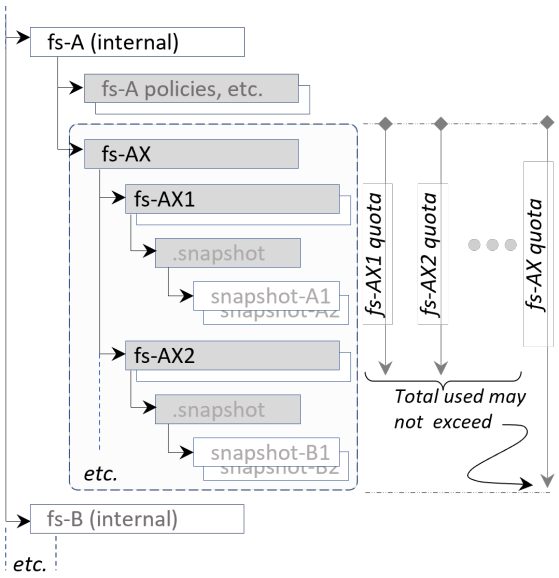

Figure 9: Nested Quotas

FlashArray File Services alerts managed directory owners when storage consumption reaches 80% of quota, and again when it reaches 90%. Alerts can also be sent to users, groups, or both. Thus, for example, a project file system might be configured to alert either the project's manager or all project group members when available space runs low. Managed subdirectories might be owned by individual team leaders or users, and subdirectory alerts directed only to them.

STORAGE ALLOCATION AND ARRAY CAPACITY

FlashArray files and volumes are inherently thin-provisioned. Aside from a small amount of metadata that describes them, they consume physical storage only for data written by clients. Administrators and users do not reserve physical storage for specific volumes or files.

Purity//FA is designed to deliver full performance even when an array reports that its physical flash is 100% occupied. Arrays achieve full performance at 100% reported occupancy by:

- Reserving an amount of physical capacity for the software's internal use. Arrays do not explicitly report reserved capacity.[10]
- *Throttling* (slowing down) writes when occupancy approaches 100% of reported capacity to allow time for the software to free space occupied by overwritten data.

[9] Quotas cannot be assigned to ordinary (unmanaged) directories.

[10] Administrators new to FlashArray may be surprised to discover arrays reporting that their storage is more than 100% occupied.

- Alerting array administrators and Pure1 to allow them to alleviate close-to-full situations by eradicating unneeded data or installing additional physical capacity.

Throttling client and host writes slightly increases response times but avoids abrupt failure.[11] When occupancy falls below 100%, the software resumes writing at full speed.

High Availability and Failover

If a FlashArray's primary controller fails, its secondary controller assumes the primary role. Each **VIF** used by FlashArray File Services must therefore include Ethernet ports on both controllers, so that after failover, the new primary controller responds to clients on the same virtual IP address. **VIF**s having ports on both array controllers also makes non-disruptive upgrade possible.

Replication

Pods are Purity//FA management objects that contain groups of volumes and/or top-level managed directories to be replicated to other arrays. Pods can be *stretched,* causing the software to replicate them from a *source* array to a (usually remote) *target* array. Replication is continuous—source pod updates are sent to the target array every few minutes. The contents of the target pod may therefore lag those of the source by an amount that varies based on source array and network loading but is usually no more than a few minutes.

Array administrators can move top-level managed directories (and volumes) into and out of pods configured for continuous replication. Replicated data on target arrays can be exported (managed directories) or mounted (volumes) for read-only access by clients or hosts.

Purity//FA supports multiple pods on an array; each may be stretched to a different replication target. Array administrators can reverse source and target array roles, for example when restoring a source site's data as part of recovering from a site disaster.

Monitoring FlashArray File Services Performance

Administrators can monitor array performance history via the Purity//FA CLI and GUI. Performance history displays may include all activity or may be filtered to restrict displays to volume or file services, to NFS or SMB activity, or to specific managed directories.

[11] Read performance is not materially affected by array occupancy.

SUMMING UP

FlashArray File Services expands FlashArray capabilities with native file system support based on Purity//FA's **Virtual** and **Physical Storage** layers, proven by a decade of service with an installed base of tens of thousands of arrays. **VFS** and the **Data Store** combine to provide robust, protocol-independent file services to clients using either SMB or NFS to read and write file data. Both protocol implementations are entirely Pure Storage-developed; they do not rely on external packages such as Samba. Unlike file servers originally developed for one protocol and adapted to support a second, the FlashArray File Services SMB and NFS protocols are peers; both use the same **VFS** and **Virtual** and **Physical Storage** layers to a provide uniformly high quality of service to both client communities. For applications that require it, FlashArray File Services supports simultaneous client access via both protocols.

FlashArray File Services allows data centers to consolidate on premise storage for volume and file applications such as home directories, project directories, backup targets, and others in highly available, high-performing, cost-effective, space-efficient arrays that may be configured with up to 1.4 petabytes of physical flash.

APPENDIX

A Brief Overview of the Purity//FA Storage Layers

The FlashArray architecture is the foundation for products that store digital data efficiently, affordably, and with absolute *integrity*—they return exactly what hosts have written when it is retrieved, regardless of what may have happened to the array or occurred in its environment. Of course, performance is due in large part to the speed of the underlying flash, but the arrays' efficiency, affordability, integrity, and general enterprise suitability result from the unique way in which the Purity//FA software organizes flash and manages data for storage and retrieval.

FLASH STORAGE DESIGN CHALLENGES

Flash memory and magnetic disk are fundamentally different data storage media. Flash read and write performance are much higher, but there are other important differences, summarized in the table below, that require different approaches to storage system design for the two.

Property	Disk	Flash	Flash Storage System Design Principle Suggested by the Property
Access time	Depends on current head position and target data location.	Nearly instantaneous regardless of target data location	Media layout and data organization can be independent of access time.
Read vs. write performance	Essentially equal	Write latency >> Read	Use techniques such as NVRAM staging to ensure consistently high write performance.
Random access	Sectors can be written independently.	Large blocks must be pre-erased; Pages within a pre-erased block must be written sequentially.	Stage data persistently (e.g., in high-performing NVRAM) so writes to flash can be large and well-aligned.
Media wear	Negligible over device lifetime.	Limited overwrites before media becomes unreliable.	Consolidate writes to physical flash and align them to maximize media lifetime.

PURITY//FA FLASH ORGANIZATION

Purity//FA organizes the flash in an array's *Direct Flash Modules* (DFMs) in large fixed-length blocks called *allocation units* (AUs). It allocates storage for writing and erasure-coding data in *segments*—dynamically chosen groups of AUs on separate DFMs as suggested by Figure 10.

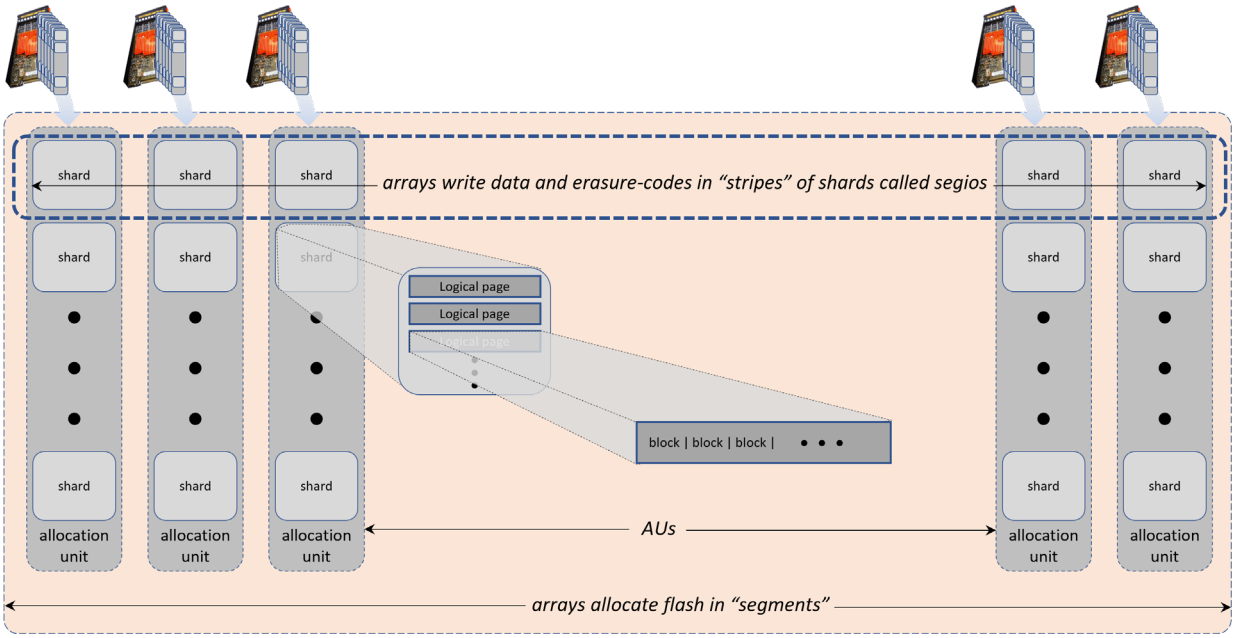

Figure 10: Purity//FA Media Organization

The software selects the AUs for a segment randomly among its DFMs to balance back-end I/O and minimize flash wear. Segments include space for data and/or metadata and for erasure codes. Each AU in a segment is treated as a column of contiguous blocks called *shards* that correspond to the DRAM buffers from which arrays write data and metadata to flash. Stripes of corresponding shards are called *segios*.

Purity//FA *reduces* the data written by hosts and clients by (a) removing repeating byte patterns, (b) eliminating sequences of sectors that contain duplicate data, and (c) compressing what's left.[12] The software packs reduced data into buffers to be written to flash. When the buffers for a *stripe* fill, it calculates erasure codes over them and *flushes* (writes) them to the shards of a flash segio. Segio writes are "full-stripes" in the sense that the software calculates erasure codes entirely from buffer contents—calculations do not require access to already-stored data, so there is no "write amplification." With the Purity//FA erasure codes it is possible to reconstruct data from any two simultaneous read failures in a stripe as well as from many other read failure scenarios.

[12] ***Transient*** data (data that clients overwrite soon after creating it) is only reduced inline as it enters an array. Longer-lasting data undergoes more exhaustive after-the-fact ***deep reduction*** by background tasks to minimize the space it occupies.

Log Structured Data Storage

Purity//FA does not overwrite data in place. Segios are effectively time-ordered logs of the data written by clients and hosts.[13] As the software writes data to flash, it updates *medium graphs* that relate virtual data addresses (for files, {GUID, offset}; for volumes, {volume, LBA}) to the flash locations of the content most recently written to them. Medium graphs are what make array-wide deduplication of file and volume data possible.

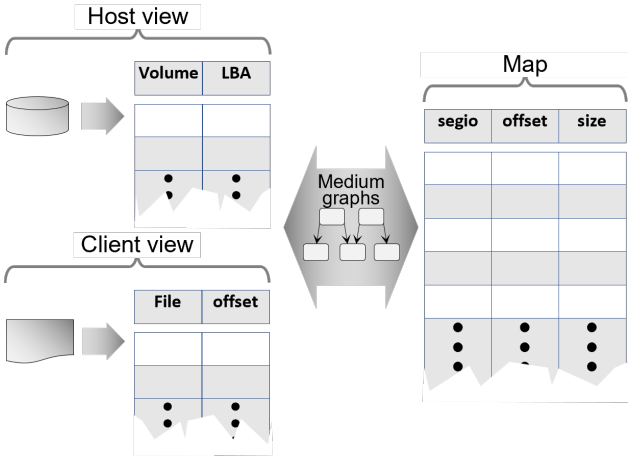

Figure 11: File and Volume Mapping

Medium graphs resemble trees—they terminate in *leaf nodes* that point to entries in a system-wide map that relates host and client data addresses to the flash locations of the corresponding data.

An array's map indicates the flash locations of all data stored in it. As it ingests, reduces, and stores incoming data, it updates its map to reflect the flash location of the data most recently written to each file or volume virtual address.

Locating Stored Data: The Map "Pyramid"

Purity//FA does not overwrite its stored data map directly. Each time the software writes a segio, it creates a new map *layer* containing the locations of newly written items. The overall map is thus conceptually a pyramid of data locations as Figure 12 suggests.

When executing client or host read commands, the software traverses the medium graph to locate the entries corresponding to the requested data. It retrieves the (reduced) data from flash, expands it to its original form, and returns it to the requester. During the process, it validates the data in multiple ways.

When a client or host overwrites data, its new flash locations appear in the topmost layer of the map pyramid. The locations of the overwritten data appear in a lower layer. When executing read commands, the software searches map layers in new-to-old order, stopping when it locates entries that represent the requested data.

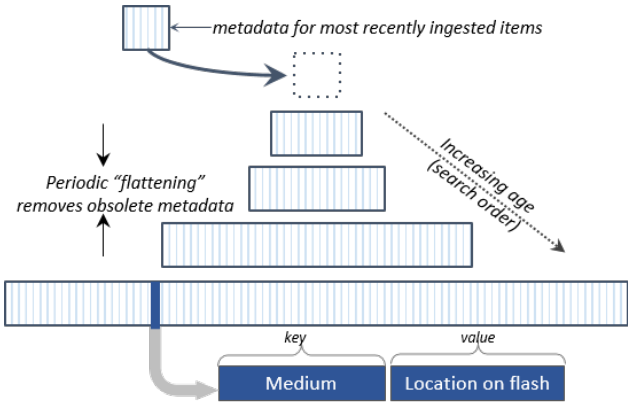

Figure 12: The Map "Pyramid"

[13] Time-ordering is approximate because the software typically constructs segios for multiple segments concurrently.

"Garbage Collection" and Flattening

As any log-structured storage system must, Purity//FA *garbage collects* in a background task to reclaim storage occupied by data that clients or hosts have overwritten. A companion task periodically *flattens* the map pyramid, reducing the number of layers by removing entries that point to overwritten data and consolidating those that represent adjacent data in virtual file or volume address spaces.

Most FlashArray models, including the FlashArray//C, do not implement read cache per se because searching the map, which *is* cached, and retrieving data from flash is fast enough to deliver consistently good response to clients and hosts.

Medium Graphs and Deduplication

Medium graphs are key to deduplication. If multiple files or volumes contain regions of identical data, their medium graphs terminate in leaf nodes that point to a map entry for a single representation on flash as Figure 13 illustrates. As the figure suggests, mediums for different files and volumes may point to the same map entry, which in turn points to a single instance of data.

As an array ingests file or volume data, the software searches for likely duplicates of already-stored data. When it encounters possible duplicates, it first reads the stored data to verify that stored and ingested data are indeed identical. If they are, it represents the ingested data with a medium leaf node that points to the map entry for the already-stored representation.

Medium graphs make snapshot creation almost instantaneous. Taking a snapshot of a volume or managed directory essentially consists of creating a medium graph for the new object. Initially, a snapshot's content is identical to that of its source, so the new medium points to the source object's graph. As clients alter data:

Volumes

Volume medium graphs diverge from those of their snapshots (which never change) as hosts write data to them. The medium graphs of clones (writable copies of volumes) diverge from those of their source volumes when host write to either volume or clone.

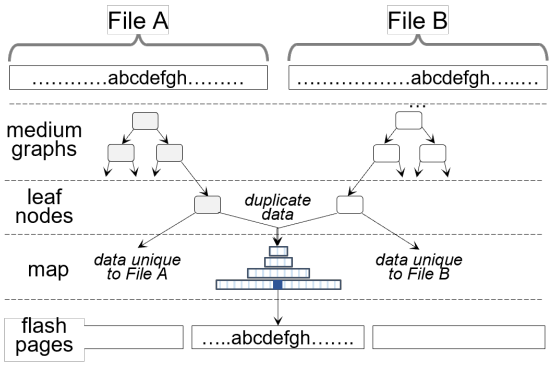

Figure 13: Using Mediums to Deduplicate

Files

Because the number of files in a system can be very large, the software defers creating new mediums for files in managed directories when snapshots are taken until clients actually modify data. When clients modify files in a managed directory that has active snapshots, the software attaches the medium graphs of the file snapshots to the `.snapshot` subdirectory entry that represents the snapshot of the managed directory affected by the modification (Figure 7 on page 9).

***FlashArray File Services**:*

Simplifying the modern data experience