Qumulo

Qumulo File Fabric
Technical Overview

Contents

Modern, Highly Scalable Storage

Qumulo File Fabric (QF2) is a modern, highly scalable file storage system that spans the data center and the public cloud. It scales to billions of files and costs less and has lower TCO than legacy storage appliances. It is also the highest performance file storage system on premises and in the cloud. Its built-in, real-time analytics let administrators easily manage data no matter how large the footprint or where it's located globally.

QF2 moves file data where it's needed when it's needed, for example, between on-premises clusters and clusters that run in the cloud. QF2 lets you store and manage massive data sets, in multiple environments, anywhere in the world. QF2's software runs on industry standard hardware on premises or on cloud instances and cloud storage resources. QF2 was designed from the ground up to meet today's requirements of scale and data mobility. It is the world's first universal-scale file storage system.

Qumulo is a new kind of storage company, based entirely on advanced software. Qumulo is part of a larger industry trend of software increasing in value compared to hardware. Commodity hardware running advanced, distributed software is the unchallenged basis of modern low-cost, scalable computing. This is just as true for file storage at large scale.

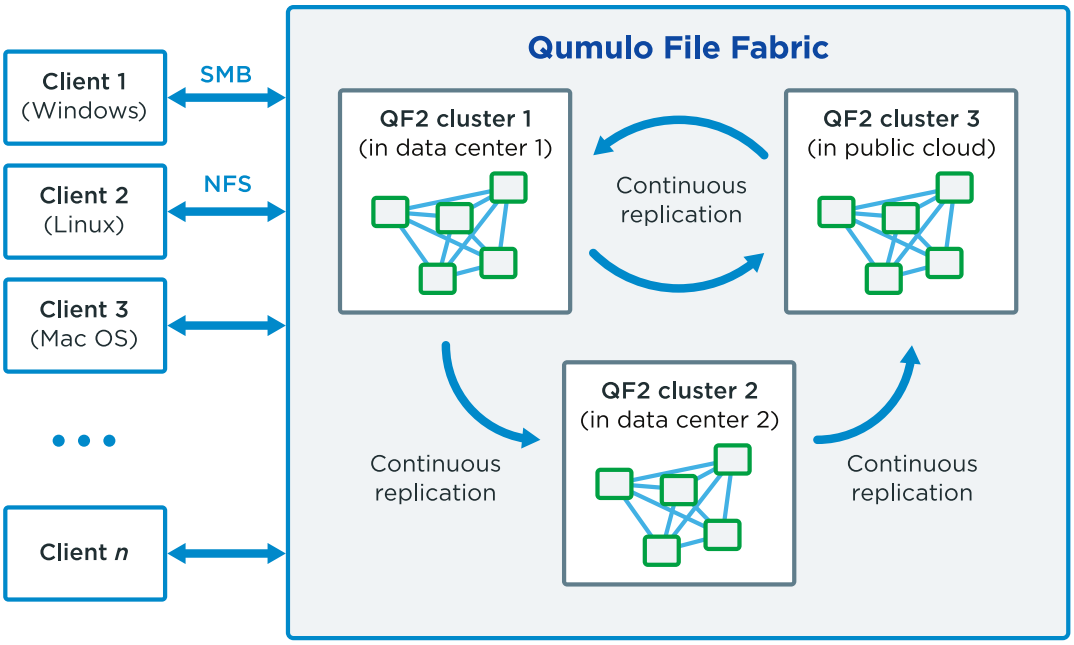
When people see QF2 in action for the first time, they often ask, "How can it do that?" Answering that question is the reason for this paper. We'll go under the hood and explain some of the advanced techniques that give QF2 its unique abilities.

Data mobility, linear scalability and global reach

For scalability, QF2 has a distributed architecture where many individual computing nodes work together to form a cluster with scalable performance and a single, unified file system. QF2 clusters, in turn, work together to form a globally distributed but highly connected storage fabric tied together with continuous replication relationships.

Customers interact with QF2 clusters using industry-standard file protocols, the QF2 REST API and a webbasedgraphical user interface (GUI) for storage administrators.

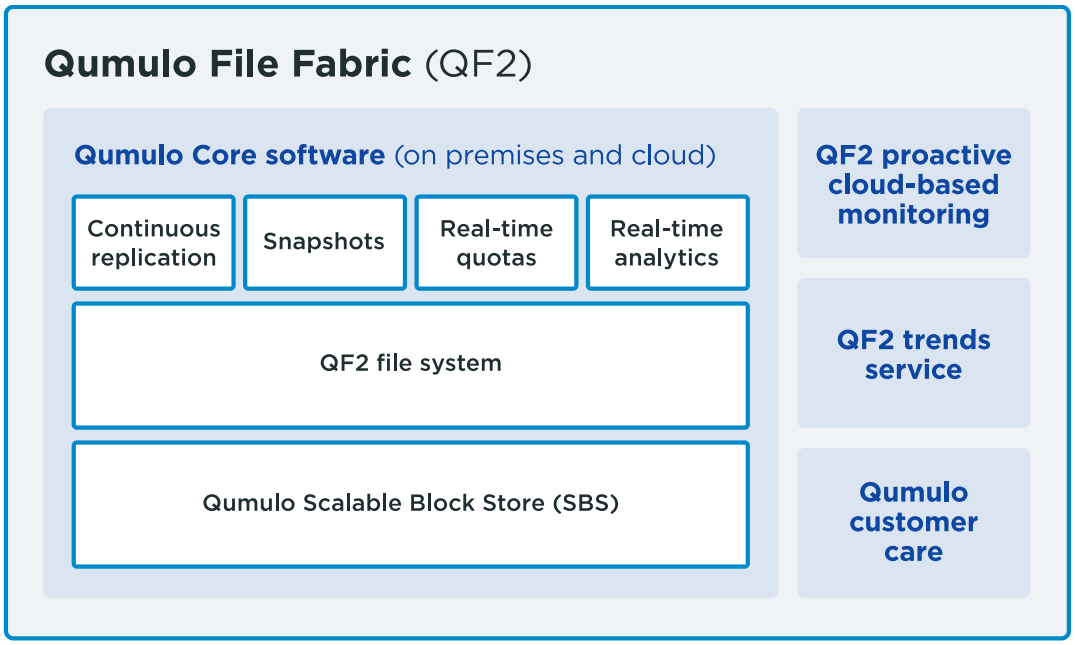
This diagram shows the connections between clients, QF2 clusters comprised of nodes running Qumulo Core, and multiple QF2 clusters, comprising the fabric, running in multiple environments and geographic locations.



QF2 is a modular system. As demands increase on a QF2 cluster, you simply add nodes or instances. Capacity and performance scale linearly, and this is true whether the QF2 cluster is operating in an onpremises data center or in the public cloud.

What's included in QF2

QF2 is comprised of both software and services. It's available with simple and transparent subscriptionbased pricing. Here's a diagram of what's included in a QF2 subscription. Qumulo Core is the software that runs on each node of a QF2 cluster. It includes powerful real-time analytics and capacity quotas, as well as continuous replication and snapshots. These features are built on the highly scalable QF2 file system that, unlike other systems, includes integrated real-time aggregation of file metadata.



The QF2 file system is built on top of a powerful, state-of-the art data management system called the Qumulo Scalable Block Store (SBS). SBS uses the principles of massively scalable distributed databases and is optimized for the specialized needs of file-based data. The Scalable Block Store is the block layer of the QF2 file system, making that file system simpler to implement and extremely robust. SBS also gives the file system massive scalability, optimized performance and data protection.

QF2 provides cloud-based monitoring and trends analysis as part of the QF2 subscription, along with comprehensive customer support. Cloud monitoring includes proactive detection of events such as disk failures to prevent problems before they happen. Access to historical trends helps lower costs and optimize workflows for best use of your storage investment.

Your Choice of Operating Environments

Qumulo Core software runs in the public cloud and on industry standard hardware in your data center. Qumulo Core currently runs on Qumulo's own QC-series of white box servers, and on Hewlett-Packard Enterprise (HPE) Apollo servers. Because QF2 is hardware independent, it does not rely on any one hardware vendor, and in fact, you can expect Qumulo will be adding support for servers from a number of OEMs.

QF2's software-only approach means that there are no dependencies on expensive, proprietary components such as NVRAM, InfiniBand switches and proprietary flash storage. Instead, QF2 relies on hard disk drives (HDD) and solid state drives (SSD) with standard firmware that is available from the drive manufacturers. The combination of standard SSDs and HDDs for automatic tiering of hot and cold data is one of the reasons that QF2 gives flash performance at the price of archive storage.

You can also create QF2 clusters in the public cloud (currently, AWS). The nodes of cloud-based QF2 clusters run the same Qumulo Core software as on-premises QF2 clusters. Unlike other cloud-based file storage systems, a QF2 cluster running in the public cloud has no hard limits for performance and capacity. Both can be increased by adding nodes (compute instances and block storage). QF2 is the only solution for cloud that lets you flexibly scale both capacity and performance.

When running in the cloud, QF2 uses cloud block storage in a way that's similar to the SSD/HDD combination on premises. QF2 uses low-latency block storage as a cache in front of economical higherlatency block storage. In the rest of the paper, we'll talk about SSDs and HDDs, but the concepts discussed apply equally to QF2's user of lower- and higher-latency block storage resources in the public cloud.

On each node of a QF2 cluster, Qumulo Core runs in Linux user space rather than kernel space. Kernel mode is primarily for device drivers that work with specific hardware. By operating in user space, QF2 can run in a wide variety of configurations and environments, and also deliver features at a much faster pace.

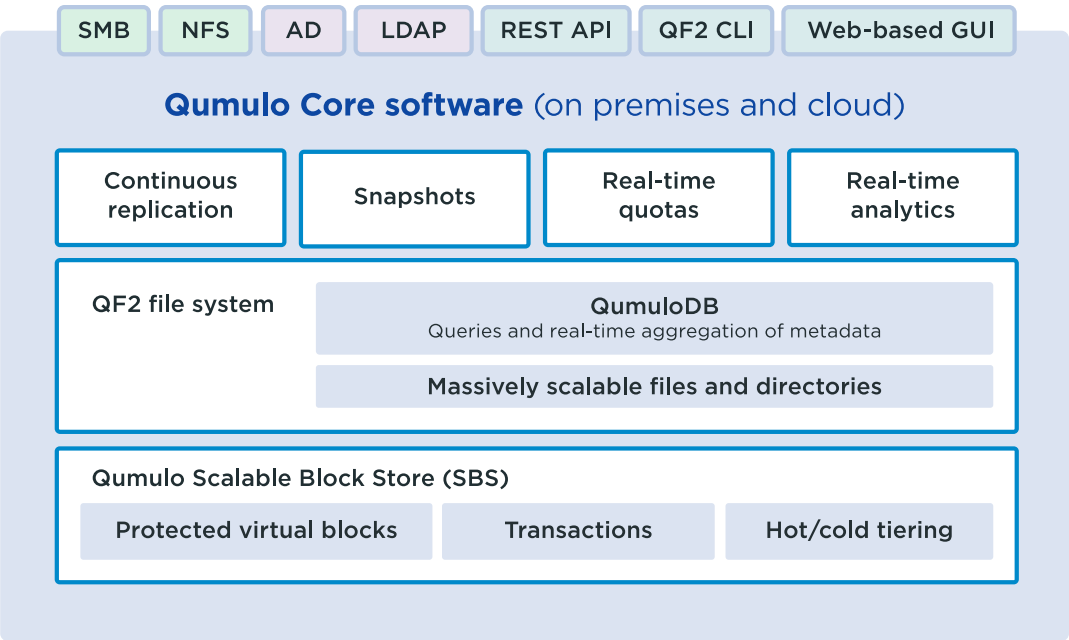
Running in user space also means QF2 can have its own implementations of crucial protocols such as SMB, NFS, LDAP and Active Directory. For example, QF2's implementation of NFS runs as a system service and has its own notions of users and groups separate from the underlying operating system on which it runs.

Running in user space also improves QF2 reliability. As an independent user-space process, QF2 is isolated from other system components that could introduce memory corruption, and the QF2 development processes can make use of advanced memory verification tools that allow memory-related coding errors to be detected prior to software release. By using a dual partition strategy for software upgrades, Qumulo can automatically update both the operating system and Qumulo Core software for fast and reliable upgrades. You can easily restart QF2 without having to reboot the OS, node or cluster.

Massively scalable files and directories

When you have a large numbers of files, the directory structure and file attributes themselves become big data. As a result, sequential processes such as tree walks, which are fundamental to legacy storage, are no longer computationally feasible. Instead, querying a large file system and managing it requires a new approach that uses parallel and distributed algorithms.

QF2 does just that. It is unique in how it approaches the problem of scalability. Its design implements principles similar to those used by modern, large-scale, distributed databases. The result is a file system with unmatched scale characteristics. In contrast, legacy storage appliances were simply not designed to handle the massive scale of today's data footprint.



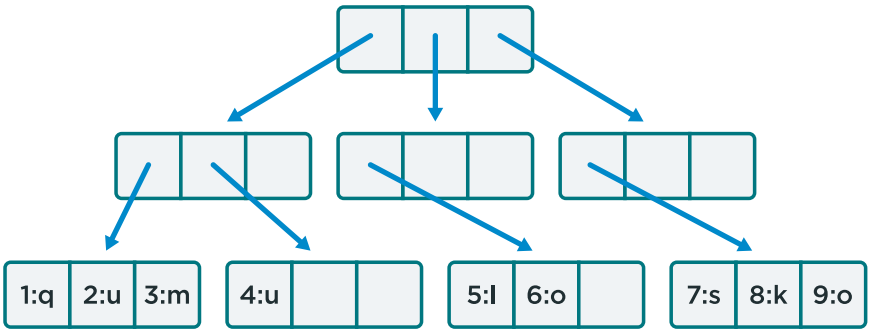
The QF2 file system sits on top of a virtualized block layer called SBS. In QF2, time consuming work such as protection, rebuilds, and deciding which disks hold which data occurs in the SBS layer, beneath the file system. (We'll talk more about protected virtual blocks and transactions later in the paper.)

The virtualized protected block functionality of SBS is a huge advantage for the QF2 file system. In legacy storage systems that do not have SBS, protection occurs on a file by file basis or using fixed RAID groups, which introduces many difficult problems such as long rebuild times, inefficient storage of small files and costly management of disk layouts. Without a virtualized block layer, legacy storage systems also must implement data protection within the metadata layer itself, and this additional complexity limits the ability of these systems to optimize distributed transactions for their directory data structures and metadata.

For scalability of files and directories, the QF2 file system makes extensive use of index data structures known as B-trees[1]. B-trees are particularly well suited for systems that read and write large numbers of data blocks because they are "shallow" data structures that minimize the amount of I/O required for each operation as the amount of data increases. With B-trees as a foundation, the computational cost of reading or inserting data blocks grows very slowly as the amount of data increases. B-trees are ideal for file systems and very large database indexes, for example.

1. https://en.wikipedia.org/wiki/B-tree

In QF2, B-trees are block-based. Each block is 4096 bytes. Here is an example that shows the structure of aB-tree.



Each 4K block may have pointers to other 4K blocks. This particular B-tree has a branching factor of 3, where a branching factor is the number of children (subnodes) at each node in the tree.

Even if there are millions of blocks, the height of a B-tree in SBS, from the root down to where the data is located, might only be two or three levels. For example, in the diagram, to look up the label q with index value 1, the system traverses only two levels of the tree.

The QF2 file system uses B-trees for many different purposes.

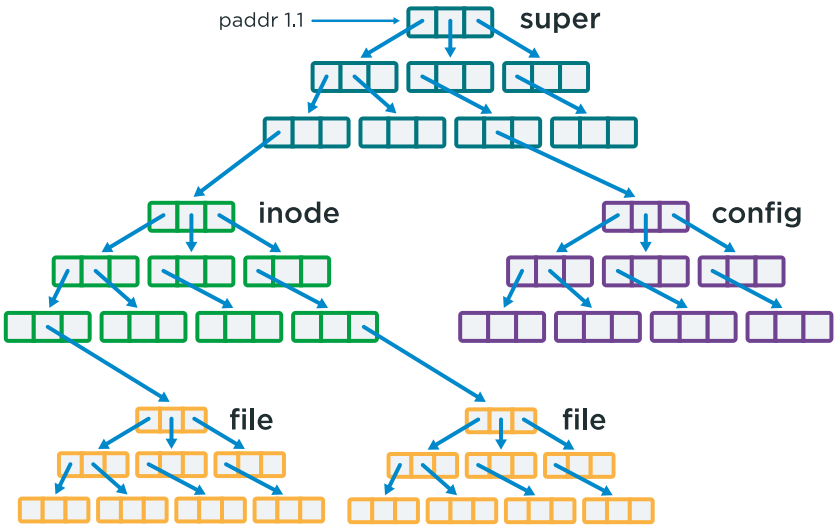
There is an inode[2] B-tree, which acts as an index of all the files. The inode list is a standard file-system implementation technique that makes checking the consistency of the file system independent of the directory hierarchy. Inodes also help to make update operations such as directory moves efficient.

Files and directories are represented as B-trees with their own key/value pairs, such as the file name, its size and its access control list (ACL) or POSIX permissions.

Configuration data is also a B-tree and contains information such as the IP address and the cluster.

2. https://en.wikipedia.org/wiki/Inode

In fact, the QF2 file system can be thought of as a tree of trees. Here is an example.



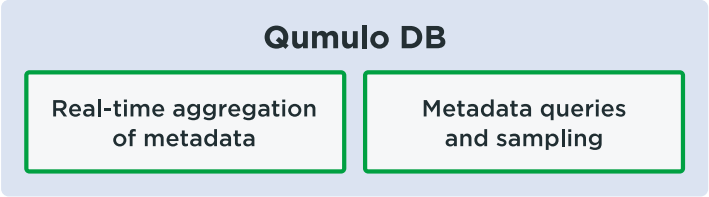
The top-level tree is called the super B-tree. When the system starts, processing begins there. The root is always stored in the first address ("paddr 1.1") of the array of virtual protected blocks. The root tree references other B-trees. If the QF2 file system needs to access configuration data, for instance, it goes to the config B-tree. To find a particular file, the system queries the inode B-tree using the inode number as the index. The system traverses the inode tree until it finds the pointer to the file B-tree. From there, the file system can look up anything about the file. Directories are handled just like files.

Relying on B-trees that point to virtualized protected block storage in SBS is one of the reasons that in QF2 a file system with a trillion files is feasible.

Real-time visibility and control with QumuloDB

QF2 is designed to do more than store file data. It also lets you manage your data and users in real time. Administrators of legacy storage appliances are hampered by "data blindness." They cannot get an accurate picture of what is happening in their file system. QF2 is designed to give exactly that kind of visibility, no matter how many files and directories there are. You can, for example, get immediate insight into throughput trending and hotspots. You can also set real-time capacity quotas, which avoid the timeconsuming quota provisioning overhead of legacy storage. Information is accessible through a graphical user interface and there is also a REST API that allows you to access the information programmatically.

The integrated analytics features of the QF2 file system are provided by a component called QumuloDB.



When people are introduced to QF2's real-time analytics and watch them perform at scale, their first question is usually, "How can it be so fast?" The breakthrough performance of QF2's real-time analytics is possible for three reasons.

First, in the QF2 file system, QumuloDB analytics are built-in and fully integrated with the file system itself. In legacy systems, metadata queries are answered outside of the core file system by an unrelated software component.

Second, because the QF2 file system relies on B-trees, QumuloDB analytics can use an innovative system of real-time aggregates, which we describe in the next section.

Finally, QumuloDB analytics are possible because the QF2 file system has a streamlined design due to its use of the B-tree indexes and the virtualized protected blocks and transactions of the Qumulo Scalable Block Store (SBS).

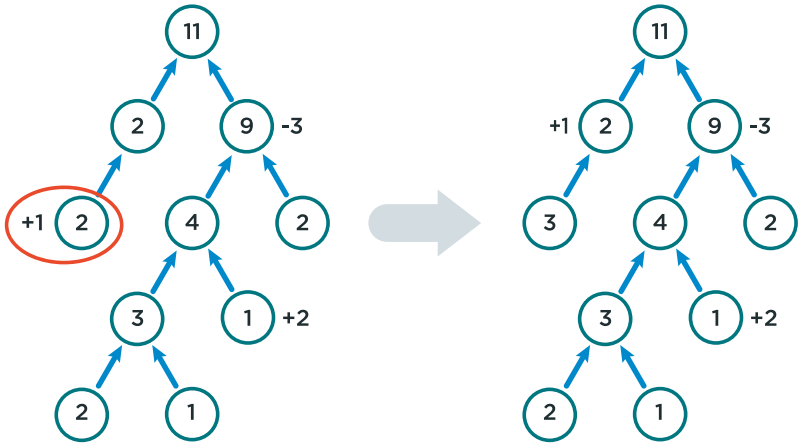
Real-time aggregation of metadata

In the QF2 file system, metadata such as bytes used and file counts are aggregated as files and directories are created or modified. This means that the information is available for timely processing without expensive file system tree walks.

QumuloDB maintains up-to-date metadata summaries. It uses the file system's B-trees to collect information about the file system as changes occur. Various metadata fields are summarized inside the file system to create a virtual index. The performance analytics that you see in the GUI and can pull out with the REST API are based on sampling mechanisms that are built into the QF2 file system. Statistically valid sampling techniques are possible because of the availability of up-to-date metadata summaries that allow sampling algorithms to give more weight to larger directories and files.

Aggregating metadata in QF2 uses a bottom-up and top-down approach.

As each file (or directory) is updated with new aggregated metadata, its parent directory is marked "dirty" and another update event is queued for the parent directory. In this way, file system information is gathered and aggregated while being passed up the tree. The metadata propagates up from the individual inode, at the lowest level, to the root of the file system as data is accessed in real time. Each file and directory operation is accounted for, and this information eventually propagates up to the very core of the file system. Here is an example.

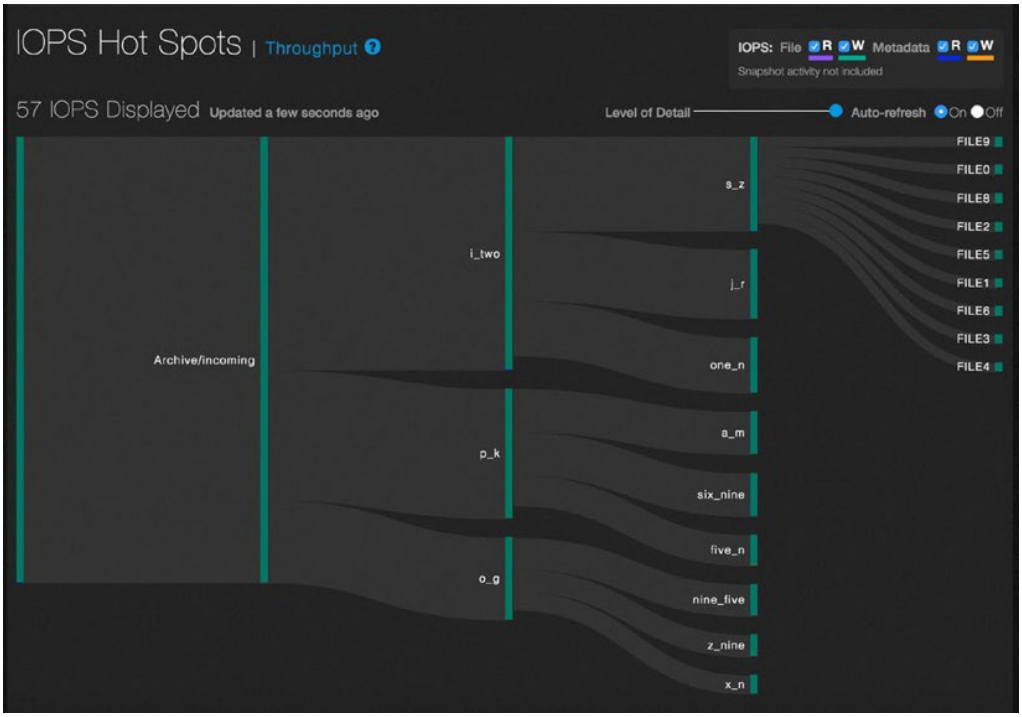


The tree on the left is aggregating file and directory information and incorporating it into the metadata. An update is then queued for the parent directory. The information moves up, from the leaves to the root.

In parallel to the bottom-up propagation of metadata events, a periodic traversal starts at the top of the file system and reads the aggregate information present in the metadata. When the traversal finds recently updated aggregate information, it prunes its search and moves on to the next branch. It assumes that aggregated information is up-to-date in the file system tree from this point down towards the leaves (including all contained files and directories) and does not have to go any deeper for additional analytics. Most of the metadata summary has already been calculated, and, ideally, the traversal only needs to summarize a small subset of the metadata for the entire file system. In effect, the two parts of the aggregation process meet in the middle with neither having to explore the complete file system tree from top to bottom.

Sampling and metadata queries

One example of QF2's real-time analytics is its performance hot spots reports. Here is an example from theGUI.



To represent every throughput operation and IOPS within the GUI would be infeasible in large file systems. Instead, QumuloDB queries use probabilistic sampling to provide a statistically valid approximation of this information. Totals for IOPS read and write operations, as well as I/O throughput read and write operations, are generated from samples gathered from an in-memory buffer of 4,096 entries that is updated every few seconds.

The report shown here displays the operations that are having the largest impact on the cluster. These are represented as hotspots in the GUI.

QF2's ability to use statistically valid probabilistic sampling is only possible because of the summarized metadata for each directory (bytes used, file counts) that is continually kept up to date by QumuloDB. It is a unique benefit of QF2's advanced software techniques that are found in no other file storage system.[3]

Real-time quotas

Just as real-time aggregation of metadata enables QF2's real-time analytics, it also enables real-time capacity quotas. Quotas allow administrators to specify how much capacity a given directory is allowed to use for files. For example, if an administrator sees that a rogue user is causing a subdirectory to grow too quickly, the administrator can instantly limit the capacity of that user's directory.

3. Blog post, "Getting People to Delete Data", https://qumulo.com/blog/getting-people-to-delete-data/

In QF2, quotas are deployed immediately and do not have to be provisioned. They are enforced in real time and changes to their capacities are immediately implemented. A side benefit is that quotas assigned to directories move with them and directories themselves can be moved into and out of quota domains.

Unlike some other systems, Qumulo quotas do not require breaking the file system into volumes. Also, with QF2, moving a quota, or moving directories across quota domains, involves no lengthy tree walks, job scheduling or cumbersome, multi-step processes. Quotas in QF2 can be applied to any directory, even nested ones. If an allocation has more than one quota because of nested directories, all quotas must be satisfied in order to allocate the requested space.

Because quota limits in QF2 are recorded as metadata at the directory level, quotas can be specified at any level of the directory tree. When a write operation occurs, all relevant quotas must be satisfied. These are hard limits. The precision and agility of QF2's real-time quotas are possible because the built-in aggregator continually keeps the summary of the total amount of storage used per directory up to date.

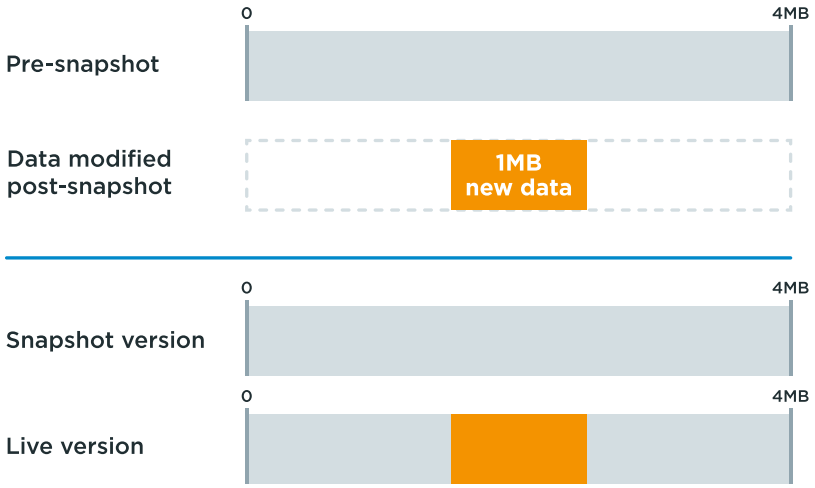
Snapshots

Snapshots let system administrators preserve the state of a file system or directory at a given point in time. If a file or directory is modified or deleted unintentionally, users or administrators can revert it to its saved state.

Snapshots in QF2 have an extremely efficient and scalable implementation. A single QF2 cluster can have a virtually unlimited number of concurrent snapshots without performance or capacity degradation.

Snapshots are implemented as out-of-place block writes. When a snapshot is taken and blocks are modified, they are written to a new spine of B-tree blocks. Existing blocks that were not modified are linked to the new spine and are now shared. The new modified spine has been written "out of place" but still references existing data, sharing the unchanged blocks. No space is consumed by the snapshot until data is modified or deleted.

As an example, assume you add a 4MB file to the QF2 file system and then take a snapshot. After the snapshot is created, the amount of capacity used is still just 4MB. Then, you modify a 1MB region of your file. The new data (1MB) is written out of place, and associated with the "live" version of the file. 3MB of the original 4MB of data is shared between the live version and the version captured in the snapshot. The total storage usage for this file is now 5MB.



Snapshots are a safety feature that help make the file system resilient in case users mistakenly delete or modify file data. For example, if some of the data is accidentally deleted, users can recover files from a previously taken snapshot. Single files or whole directories can be restored by copying them back into the live file system.

When space runs low, administrators often decide to delete snapshots to free up storage. Because snapshots share data, deleting a snapshot does not usually reclaim an amount of space equal to the sum of all of the files it contains. In legacy systems, it's difficult to know how much space would, in fact, be reclaimed.

Snapshots in QF2 take advantage of the intelligence built into the file system. Administrators can calculate how much storage would actually be reclaimed if they delete a set of snapshots.

Continuous replication

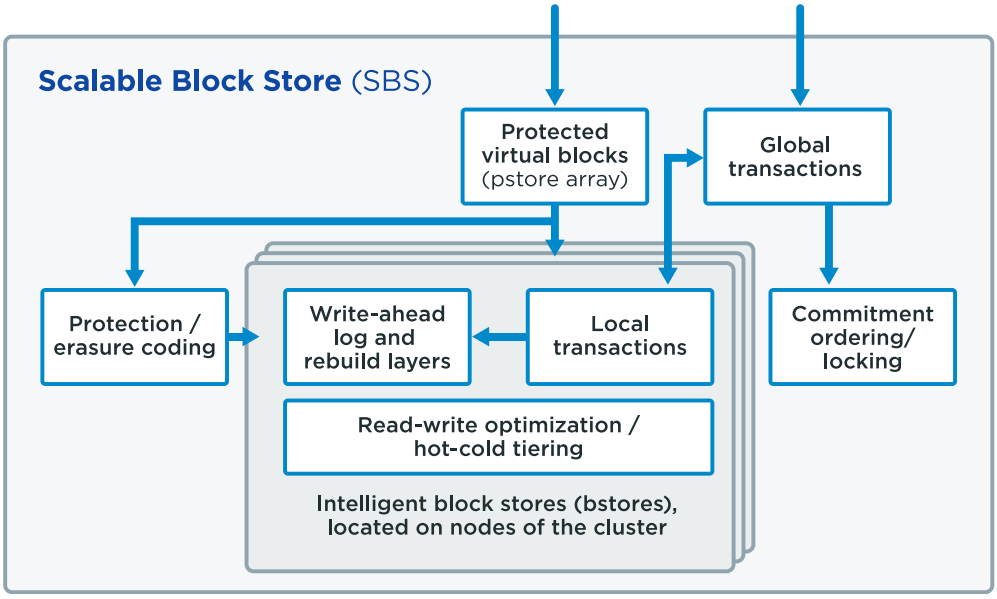
QF2 provides continuous replication across storage clusters, whether on premises or in the public cloud. Once a replication relationship between a source cluster and a target cluster has been established and synchronized, QF2 automatically keeps data consistent.

Continuous replication in QF2 leverages QF2's advanced snapshot capabilities to ensure consistent data replicas. With QF2 snapshots, a replica on the target cluster reproduces the state of the source directory at exact moments in time. QF2 replication relationships can be established on a per-directory basis for maximum flexibility.

Qumulo applies smart algorithms to manage replication, so you don't have to. QF2 replicates as often as practical without negatively impacting overall cluster performance. Continuous replication in QF2 is oneway asynchronous as of today; that is, there's a source and a read-only target. Changes to the source directory are propagated asynchronously to the target.

The Qumulo Scalable Block Store (SBS)

Now that we've looked at the internals of the QF2 file system, let's turn to QF2's advanced strategies for distributed block management found in the Qumulo Scalable Block Store. Here is an overview of what's inside SBS.



SBS provides a transactional virtual layer of protected storage blocks. Instead of a system where every file must figure out its protection for itself, data protection exists beneath the file system, at the block level.

QF2's block-based protection, as implemented by SBS, provides outstanding performance in environments that have petabytes of data and workloads with mixed file sizes.

SBS has many benefits, including:

- Fast rebuild times in case of a failed disk drive
- The ability to continue normal file operations during rebuild operations
- No performance degradation due to contention between normal file writes and rebuild writes
- Equal storage efficiency for small files as for large files
- Accurate reporting of usable space
- Efficient transactions that allow QF2 clusters to scale to many hundreds of nodes
- Built-in tiering of hot/cold data that gives flash performance at archive prices.

To understand how SBS achieves these benefits we need to look at how it works.

Protected virtual blocks

The entire storage capacity of a QF2 cluster is organized conceptually into a single, protected virtualaddress space, shown here.

Each protected address within that space stores a 4K block of bytes. By "protected" we mean that all blocks are recoverable even if multiple disks were to fail. Protection will be explained in more detail later in this paper.

The entire file system is stored within the protected virtual address space provided by SBS, including user data, file metadata, the directory structure, analytics, and configuration information. In other words, the protected store acts as an interface between the file system and block-based data recorded on attached block devices. These devices might be virtual disks formed by combining SSDs and HDDs or block-storage resources in the cloud.

Note that the blocks in the protected address space are distributed across all of the nodes or instances of the QF2 cluster. However, the QF2 file system sees only a linear array of fully-protected blocks.

Data protection based on erasure coding

Protecting against data loss in the event of disk failure always includes some form of redundancy or duplication of information across storage devices. The simplest form of data protection is mirroring. Mirroring means that there are two or more full copies of the data being protected. Each copy resides on a different disk so that it is recoverable if one of the disks fails.

Mirroring is simple to implement but has disadvantages compared to more modern protection techniques. Mirroring is wasteful in terms of the space required to protect the data, and it only handles a single disk failure, which generally isn't a high enough level of safety as node density and cluster sizes increase.

Other strategies for data protection include RAID striping[4]. RAID comes with extremely complex administration and slow rebuild times that force the admin to choose between unacceptably long rebuild and unacceptable storage efficiency.[5]

SBS implements its block-based data protection with an efficient technique known as erasure coding (EC). SBS uses Reed-Solomon codes[6]. EC is faster, more configurable and more space efficient than alternatives such as mirroring and RAID striping.

4. https://en.wikipedia.org/wiki/RAID
5. http://www.networkcomputing.com/storage/raid-vs-erasure-coding/1792588127
6. https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction

EC encodes block data using redundant segments that are stored across a set of different physical media. Because of EC's efficiency, more of the disk is available for data as compared with RAID and mirroring schemes, and this lowers the cost per usable TB.

Erasure coding can be configured with tradeoffs for performance, recovery time in the case of failed physical media, and the number of allowable simultaneous failures. In this paper, we'll use the notation (m, n) to indicate a specific EC configuration, where m indicates the total number of blocks of physical media that will be used to safely encode n user blocks. The encoding has the property that up to m – n blocks can be destroyed without loss of user data. In other words, the survival of any collection of n disks is enough to recover all the user data, even if some of the failed disks contained user data. The efficiency of the encoding can be calculated as the number n / m, or the ratio of user blocks divided by all blocks.

EC is easiest to understand with examples. Here is a simple example called (3,2) encoding.

A (3,2) encoding requires three blocks (m = 3), stored on three distinct physical devices to safely encode two blocks (n = 2). Two of the blocks contain the user data we want to protect and the third is called a parity block. The contents of the parity block are calculated by the erasure coding algorithm. Even this simple scheme is more efficient than mirroring—you are only writing one parity block for every two data blocks. In a (3, 2) encoding, if the disk containing any one of the three blocks fails, the user data in blocks 1 and 2 is safe.

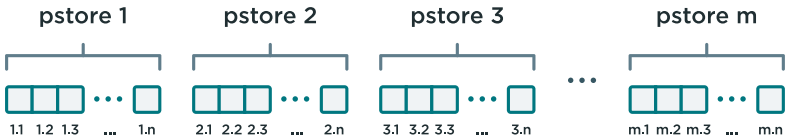
Here's how it works. If data block 1 is available, then you simply read it. The same is true for data block 2. However, if data block 1 is down, the EC system reads data block 2 and the parity block and reconstructs the value of data block 1 using the Reed-Solomon formula (which in this particular example is just bitwise XOR). Similarly, if data block 2 resides on the failed disk, the systems read data block 1 and the parity block. SBS makes sure that the blocks are on different spindles so the system can read from blocks simultaneously.

A (3,2) encoding has efficiency of 2 / 3 (n/m), or 67%. While it is better than the 50% efficiency of mirroring in terms of data storage, (3,2) encoding can still only protect against a single disk failure.

At a minimum, QF2 uses (6, 4) encoding, which stores a third more user data in the same amount of space as mirroring but can tolerate two disk failures instead of just one as mirroring does. Even if two blocks containing user data are unavailable, the system still only needs to read the two remaining data blocks and the two parity blocks to recover the missing data.

Distribution of protected virtual blocks across nodes

There are many practical considerations to take into account when implementing EC in systems with massive scalability. To make the process of writing the required parity blocks easier (and to restore data when a disk fails), SBS divides its virtual address space of 4K blocks into logical segments called protected stores, or pstores.



SBS manages each pstore individually, which makes the mapping scheme that associates the protected address space to the disks more flexible. All pstores are the same size. Data protection is entirely implemented at the pstore level of the system.

A pstore can be thought of as a table that maps ranges of protected virtual block addresses to contiguous regions of storage that reside on virtual disks of the nodes of the QF2 cluster. The contiguous regions are called bstores.

The map of pstores to bstores is stored by each node of the cluster. For reliability, nodes of the cluster use a distributed algorithm called Paxos[7] to maintain consensus about globally shared knowledge such as the pstore-to-bstore map. The cluster forms a quorum of nodes to ensure the safety of the cluster's critical data structures.

Each bstore uses a segment of a specific virtual disk (that is, the bstore is associated with a particular SSD and HDD pair). Each bstore is allocated contiguous space on its associated HDD, while space on the bstore's associated SDD is dynamically allocated. Metadata about a bstore also exists on its associated SSD. Bstore metadata includes information such as the addresses in use and a map that indicates which block addresses in the bstore reference SSD storage and which are on HDD.

During a read or write, the pstore decides which bstores need to be accessed.

When a client of the file system initiates a write operation, it goes into SBS as a stream of raw data. The system figures out which bstores to write the data to, calculates the parity data, and writes both the raw data and parity data to the SSDs at the same time, even if the SSDs are on many different nodes. Once the data is written, the client gets an acknowledgement that the write has taken place.

Data blocks that contain user data and parity blocks are both written to bstores. A particular bstore, for its lifetime, either contains parity blocks or data blocks but not both. Because EC is implemented at the pstore layer of SBS, bstores that contain parity blocks and bstores that contain data blocks behave identically.

The amount of storage allocated to a bstore depends on the choice of EC. To keep the pstore size consistent, the system's bstore size changes according to the coding scheme. For example, if the pstore is 70GB, then, with (6,4) encoding, each bstore is about 17.5GB, which divides the pstore into 4 bstores. For (10, 8) encoding, the bstores will be about half that size.

7. https://en.wikipedia.org/wiki/Paxos_(computer_science)

In the cloud, QF2 uses the same data protection scheme as it does on-premises, with one exception. Onpremises, the data protection scheme requires that there be at least four nodes in a cluster. In the cloud, it's possible to have a single-node cluster because QF2 can use the built-in mirroring that is in every elastic storage block. Single-node QF2 clusters in the cloud use (5, 4) erasure coding.

Fast rebuild times

QF2 rebuild times are measured in hours. In contrast, legacy storage systems, which were designed for workloads with far less data, have rebuild times that are measured in days. Large numbers of files, mixed workloads, and increasing disk density have all contributed to the crisis in rebuild times for legacy storage appliances. QF2's dramatic advantage in rebuild times is a direct result of SBS's advanced block-based protection.

Block-based protection is ideal for today's modern workloads where there are petabytes of data and millions of files, many of which are small. The SBS protection system doesn't need to do time-consuming tree walks or file-by-file rebuild operations. Instead, the rebuild operations work on the pstores. The result is that rebuild times aren't affected by file size. Small files are handled as efficiently as large files, and protecting millions of files is completely feasible.

In addition, QF2 is designed so that rebuild times aren't adversely affected by cluster size. In fact, the opposite is true. In QF2, larger clusters have faster rebuild times than smaller clusters. The reason for this is that SBS distributes the rebuild calculation effort across nodes of the cluster. The more nodes, the less work each node needs to do during a rebuild.

Legacy storage appliances with slow rebuild times are vulnerable to additional failures that can occur during the prolonged process of rebuilding. In other words, slow rebuild times have a negative impact on reliability. Typically, administrators compensate for this by overprovisioning (that is, decreasing efficiency by adding data redundancy). With QF2's fast rebuild times, administrators can maintain their Mean Time To Data Loss (MTTDL) targets without a great deal of redundancy, which saves both money and time.

Rebuilding the pstores

When a disk fails, the protected store still exists. It can always be read to and written from. However, some pstores will have missing or damaged bstores. These are called degraded pstores. Because of EC, you can continue to read the degraded pstores, but the data is no longer fully protected. In other words, at the first failure, you still have data integrity but are one disk closer to data loss.

To reprotect the data, the system works pstore by pstore (rather than file by file with RAID groups, as in legacy systems) to reconstruct the bstores that were located on the failed disk drive. SBS allocates a small amount of extra disk space so there is room to do this. This is called sparing.

Since the global pstore-to-bstore map contains the ID of the bstore's associated virtual disk, this information makes it easy to know which pstores need processing when a particular disk fails. Since the map that associates pstores with bstores is small enough to reside in every node's memory, nodes can quickly translate virtual block addresses from pstore to bstore.

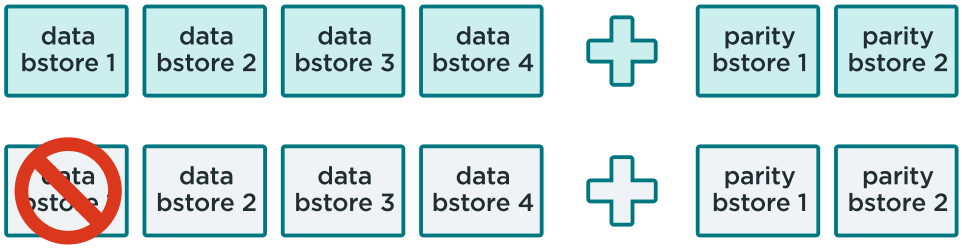
During the rebuild process, SBS reads and writes bstores sequentially. Since bstores are laid out contiguously on the disk, degraded pstores can be rebuilt very quickly. Sequential operations are much faster than many small I/O operations, which can be slow and cause disk contention. SBS's rebuild process is efficient—disks are involved in exactly one read or write stream at a time during the rebuild process. No random-I/O is required. Also, bstores are small enough so that the reprotect work is efficiently distributed across the entire cluster.

Normal file operations unaffected by rebuilds

In legacy file systems, lock contention affects rebuild times and slows down standard file system operations during the rebuild. This is because these file operations compete with the rebuild/reprotect threads. QF2 uses write layering with independent locking schemes so that rebuild operations don't contend with normal use of the file system.

When there is a failure, it makes no sense to write to the incomplete set of bstores in the degraded pstores. The new writes would not be fully protected and it would complicate the bstore reconstruction work. However, the cluster must not experience downtime during the rebuild operation, and as a result userinitiated write operations cannot wait for a pstore to be reprotected.

To perform those writes, the system adds a new layer of virtual bstores to the degraded pstore. This is called "pushing a layer." Writes go to the new layer of bstores and reads combine the values from each layer. Here is an example.



New writes go into the top layer of bstores. A read combines the values from the top layer and the bottom layer by using EC. Once the bstore is reconstructed the push layer goes away. The layers are constructed using components of SBS's transaction system in a way that makes them nonblocking.

Small files are as efficient as large files

Because the QF2 file system uses block-based protection, small files are as efficient as large files. They can share stripes with other files, and they can share the protection. Each file consists of the data blocks, at least one inode block, and any other blocks that are required. Very small files are inlined into the inode block. The system uses 4K blocks and all the blocks are protected at the system protection ratio.

QF2's efficiency with small files is a big advantage compared to legacy storage appliances, which use inefficient mirroring for small files and system metadata.t

All provisioned capacity is available for user files

QF2 user files can occupy 100% of provisioned capacity, while legacy scale-out recommends only using 80% to 85%. QF2's block-based protection requires no user-provisioned capacity for reprotection, other than a small amount of space for sparing, which is excluded from user-provisioned capacity. In contrast, legacy storage appliances implement protection either with fixed RAID groups or with erasure coding on a file-by-file basis, which means that reprotection also happens at the file level and requires user-provisioned capacity to recover.

In addition, the QF2 file system accurately reports capacity available for user files. Again, this predictability is a consequence of block-based protection. In legacy systems, storage use depends on file size, so these systems can only report on raw space—administrators are left to guess how much space they actually have.

When comparing QF2 to legacy systems, you'll want take into consideration how much user-provisioned capacity is actually available for you to use in each type of system.

Transactions

In SBS, reads and writes to the protected virtual address space are transactional. This means that, for example, when a QF2 file system operation requires a write operation that involves more than one block, the operation will either write all the relevant blocks or none of them. Atomic read and write operations are essential for data consistency and the correct implementation of file protocols such as SMB and NFS.

For optimum performance, SBS uses techniques that maximize parallelism and distributed computing while maintaining transactional consistency of I/O operations. For example, SBS is designed to avoid serial bottlenecks, where operations would proceed in a sequence rather than in parallel.

SBS's transaction system uses principles from the ARIES[8] algorithm for non-blocking transactions, including write-ahead logging, repeating history during undo actions and logging undo actions. However, SBS's implementation of transactions has several important differences from ARIES.

SBS takes advantage of the fact that transactions initiated by the QF2 file system are predictably short, in contrast to general-purpose databases where transactions may be long-lived. A usage pattern with shortlived transactions allows SBS to frequently trim the transaction log for efficiency. Short-lived transactions allow faster commitment ordering.

Also, SBS's transactions are highly distributed and do not require globally defined, total ordering of ARIESstyle sequence numbers for each transaction log entry. Instead, transaction logs are locally sequential in each of the bstores and coordinated at the global level using a partial ordering scheme that takes commitment ordering constraints into account.

Qumulo DB uses a two-phase locking (2PL) protocol to implement serializability for consistent commitment ordering. Serializable operations are performed by distributed processing units (bstores) and have the property that the intended sequence of the operations can be reconstructed after the fact. The advantage of SBS's approach is that the absolute minimum amount of locking is used for transactional I/O operations, and this allows QF2 clusters to scale to many hundreds of nodes.

8. http://www.vldb.org/conf/1999/P1.pdf

Hot/cold tiering for read/write optimization

SBS includes built-in tiering of hot and cold data to optimize read/write performance.

When running on premises, QF2 takes advantage of the speed of solid-state drives (SSDs) and the costeffectiveness of hard disk drives (HDDs). SSDs are paired with commodity HDDs on each node. This pair is called a virtual disk. There is a virtual disk for every HDD in the system.

Data is always written first to the SSDs. Because reads typically access recently written data, the SSDs also act as a cache. When the SSDs are approximately 80% full, less frequently accessed data is pushed down to the HDDs. The HDDs provide capacity and sequential read/writes of large amounts of data.

When running in the cloud, QF2 optimizes the use of block storage resources by matching highperformance block storage with cost-effective lower-performance block storage.

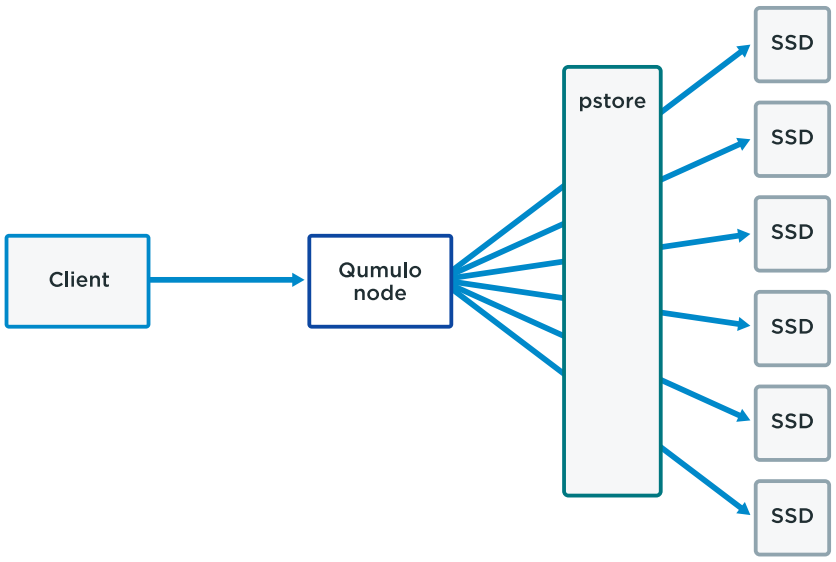
Let's look at the following aspects of SBS's hot/cold tiering:

- How and where data is written
- Where metadata is written
- How data is expired
- How data is cached and read

The initial write

To write to a cluster, a client sends some data to a node. That node picks a pstore or multiple pstores where that data will go, and, in terms of hardware, it always writes to the SSDs or to low-latency block storage if using cloud resources.[9] These SSDs will be on multiple nodes.

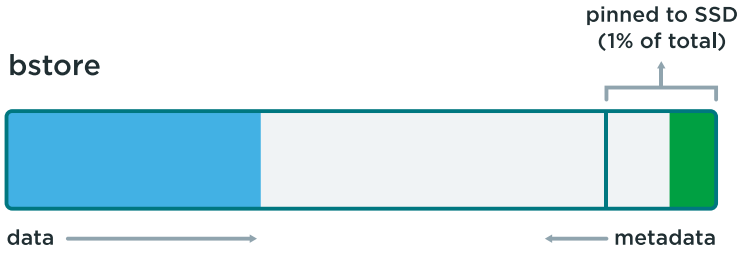
All writes occur on SSDs; SBS never writes directly to the HDD. Even if an SSD is full, the system makes space for the new data by purging previously cached data.



9. Recall that we use SSD to mean both on-premises SSDs and low-latency block storage in the public cloud; the behavior is similar.

Handling metadata

Generally, metadata stays on the SSD. Data is typically written to a bstore at the lowest available address so data grows from the beginning of the bstore to the end. Metadata starts at the end of the bstore and grows toward the beginning. This means all the metadata is to the right of the data. Here is an illustration of where metadata sits on a bstore.



QF2 allocates up to 1% of the bstore on the SSD to metadata and never expires it. Nothing in that 1% goes to the HDD. If metadata ever grows past that 1% it could expire but, for a typical workload, there is approximately 0.1% metadata. The space isn't wasted if there isn't enough metadata to fill it. Data can use that space as well.

Expiring data

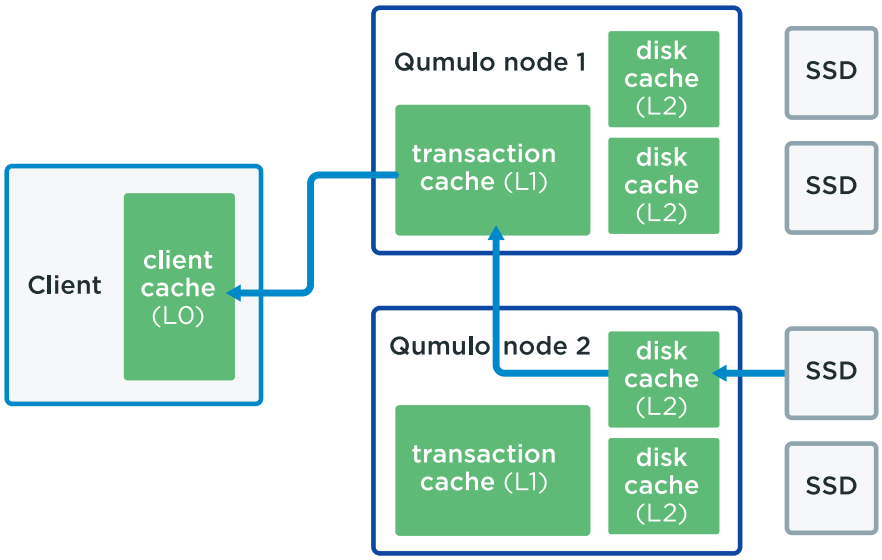
At some point, the system needs more space on the SSD so some of the data is expired, or moved from the SSD to the HDD. The data is copied from the SSD to the HDD and then, once it's on the HDD, it's deleted from the SSD.

Expiration starts when an SSD is at least 80% full and stops when it gets back to less than 80% full. The 80% threshold is a heuristic that optimizes performance—writes are faster when the SSDs are between 0% and 80% and expirations aren't happening at the same time.

When data from an SSD is moved to HDD, SBS optimizes the writes sequentially to HDD in a way that optimizes disk performance. Bursts of large, contiguous bytes are the most efficient method possible to write to HDD.t

Caching data

The following illustration shows all the QF2 caches. Everything in green is a place that can hold data, and it can be on SSD or HDD.



QF2 I/O operations use three different types of caches. The client always has some cache on its side and there are two types of caches on the nodes. One is the transaction cache, which can be thought of as the file system data that the client is directly requesting. The other type is the disk cache, which are blocks from that disk that are kept in memory.

As an example, assume that a client that is connected to node 1 initiates a read of file X. Node 1 discovers that those blocks are allocated on node 2 so it notifies node 2 that it wants data, which in this example is stored in one of the SSDs of node 2. Node 2 reads the data and puts it into the disk cache that is associated with this SSD. Node 2 replies to node 1 and sends the data. At this point the data goes into the node 1 transaction cache, which notifies the client that it has the data for file X.

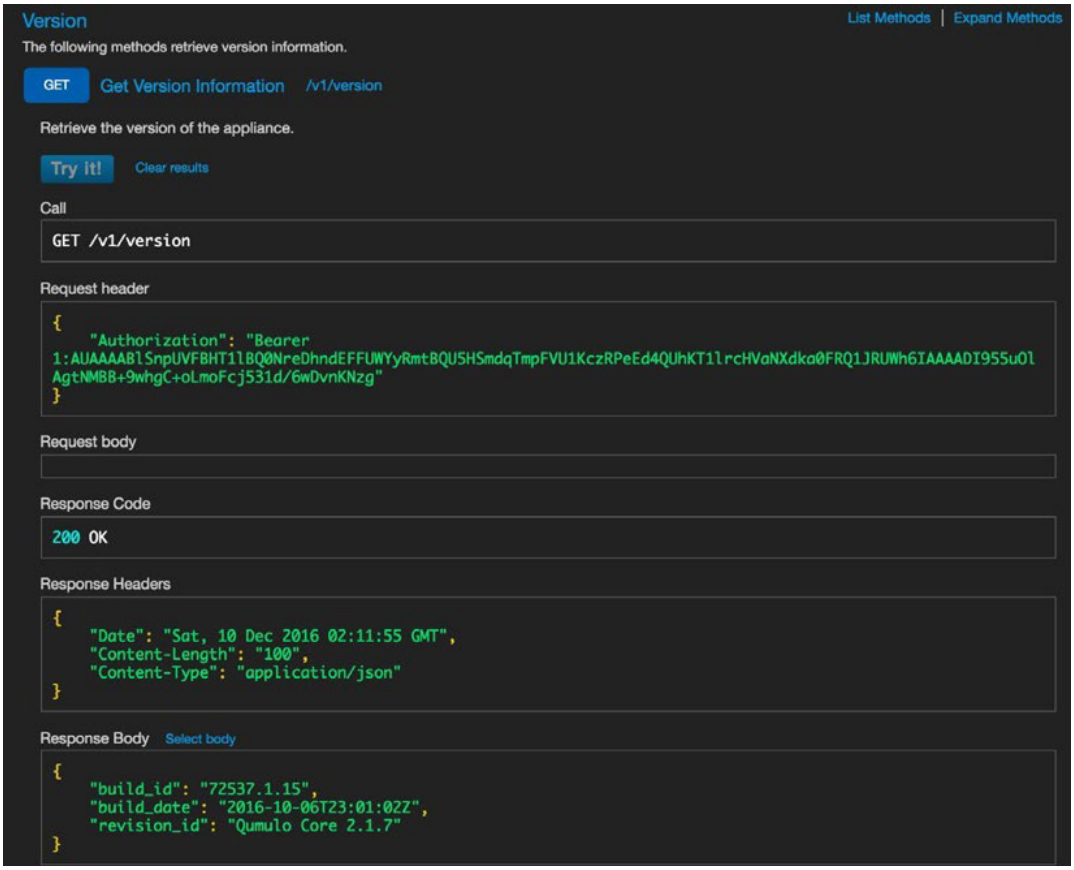
The node that the disk is attached to is where the disk cache is populated. The node that the client is attached to is where the transaction cache gets populated. The disk cache always holds blocks and the transaction cache holds data from the actual files. The transaction cache and the disk cache share memory, although there's not a specific amount allocated to either one.

Industry-standard Protocols

QF2 uses standard NFSv3 and SMBv2.1 protocols.

REST API

QF2 includes a comprehensive REST API. In fact, all the information represented in the QF2 GUI is generated from calls to the QF2 REST API. A tab within the GUI provides a self-documenting resource of the available REST API calls. You can experiment with the API by executing calls against the live cluster and inspecting the requests and results in real time. Here is an example screenshot.



All the information presented in the Analytics tab in the GUI can be programmatically retrieved with REST calls against the API and stored externally in a database or sent to another application such as Splunk or Tableau. Most file system operations can also be invoked with the REST API.

Conclusion

We hope that this paper has demystified the inner workings of QF2 and given you some insight into why QF2's performance and scalability breakthroughs are possible. If you'd like additional information, please visit our web site, www.qumulo.com, and contact us.

Key Points

Here are the key points made in this paper.

- Data is growing at an explosive rate and modern workloads are petabytes in size, can have billions of files, and those files are of mixed sizes.
- Most storage systems use decades-old technologies that were never meant to handle modern workloads.
- QF2 is a modern storage system designed specifically to address modern workloads.
- QF2 runs on standard hardware on premises and in the cloud.
- QF2 has a hybrid architecture that uses SSDs and HDDs.
- QF2 uses a block protection scheme that works below the actual file system.
- QF2 has fast rebuild times, measured in hours. They are by far the fastest in the industry.
- User files can take up to 100% of provisioned capacity.
- Small files are as efficient as large files.
- The entire file system exists as a single volume.
- QF2 applies the same techniques used by large, distributed databases to its file system.
- Real-time analytics give visibility into what's happening in the file system right now.
- There is accurate reporting of how much usable space is available.
- Administrators can apply quotas in real time.
- Administrators know how much actual space they save by deleting snapshots.
- QF2 includes asynchronous data replication at scale.
- QF2 uses standard protocols such as NFS and SMB and includes a REST API.