ILM and object lifecycle

StorageGRID

NetApp
December 03, 2025

Table of Contents

ILM and object lifecycle

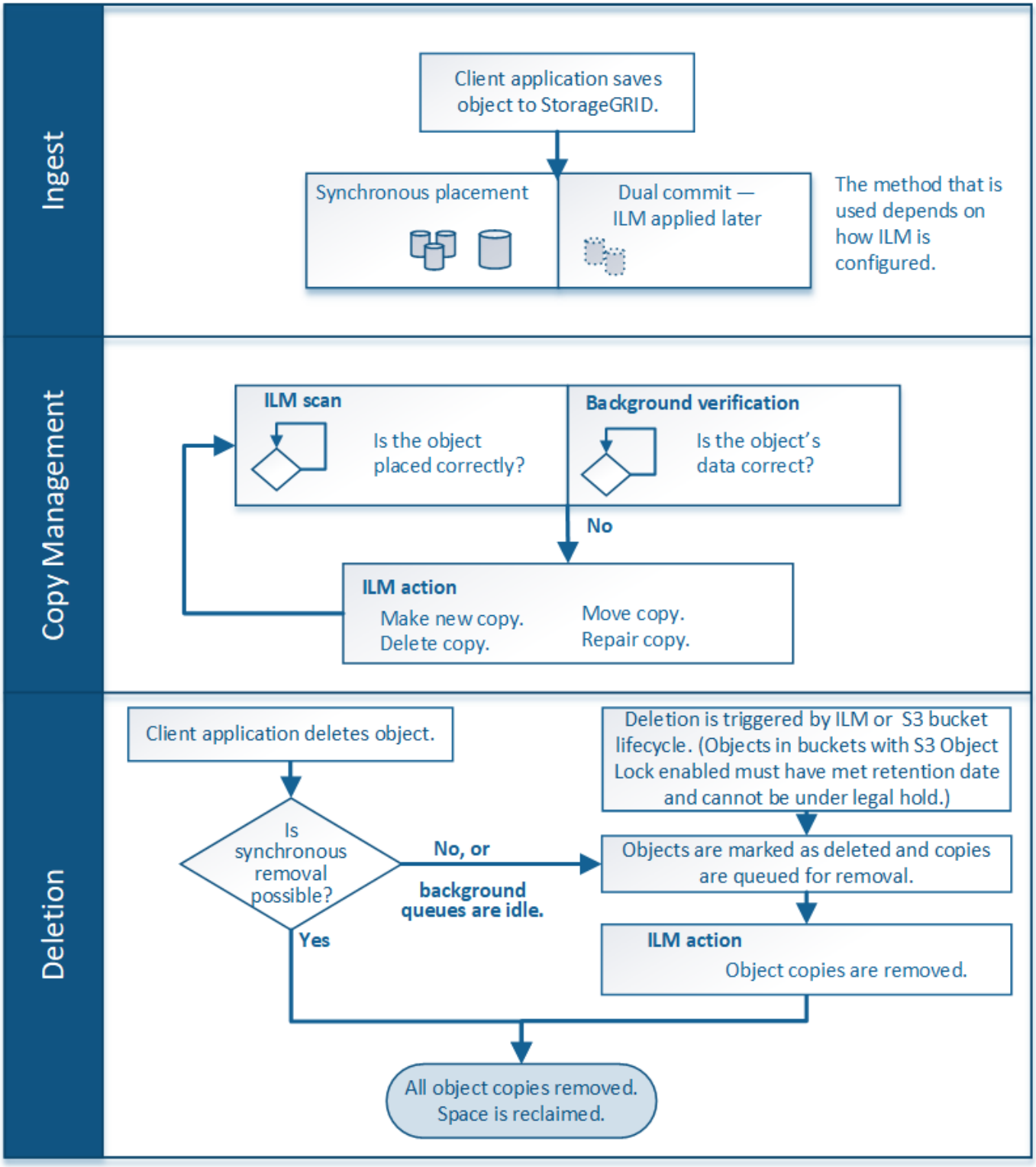
How ILM operates throughout an object's life

Understanding how StorageGRID uses ILM to manage objects during every stage of their life can help you design a more effective policy.

- **Ingest**: Ingest begins when an S3 or Swift client application establishes a connection to save an object to the StorageGRID system, and is complete when StorageGRID returns an "ingest successful" message to the client. Object data is protected during ingest either by applying ILM instructions immediately (synchronous placement) or by creating interim copies and applying ILM later (dual commit), depending on how the ILM requirements were specified.
- **Copy management**: After creating the number and type of object copies that are specified in the ILM's placement instructions, StorageGRID manages object locations and protects objects against loss.
  - **ILM scanning and evaluation**: StorageGRID continuously scans the list of objects stored in the grid and checks if the current copies meet ILM requirements. When different types, numbers, or locations of object copies are required, StorageGRID creates, deletes, or moves copies as needed.
  - **Background verification**: StorageGRID continuously performs background verification to check the integrity of object data. If a problem is found, StorageGRID automatically creates a new object copy or a replacement erasure-coded object fragment in a location that meets current ILM requirements. See Verify object integrity.
- **Object deletion**: Management of an object ends when all copies are removed from the StorageGRID system. Objects can be removed as a result of a delete request by a client, or as a result of deletion by ILM or deletion caused by the expiration of an S3 bucket lifecycle.

> Objects in a bucket that has S3 Object Lock enabled can't be deleted if they are under a legal hold or if a retain-until-date has been specified but not yet met.

The diagram summarizes how ILM operates throughout an object's lifecycle.



How objects are ingested

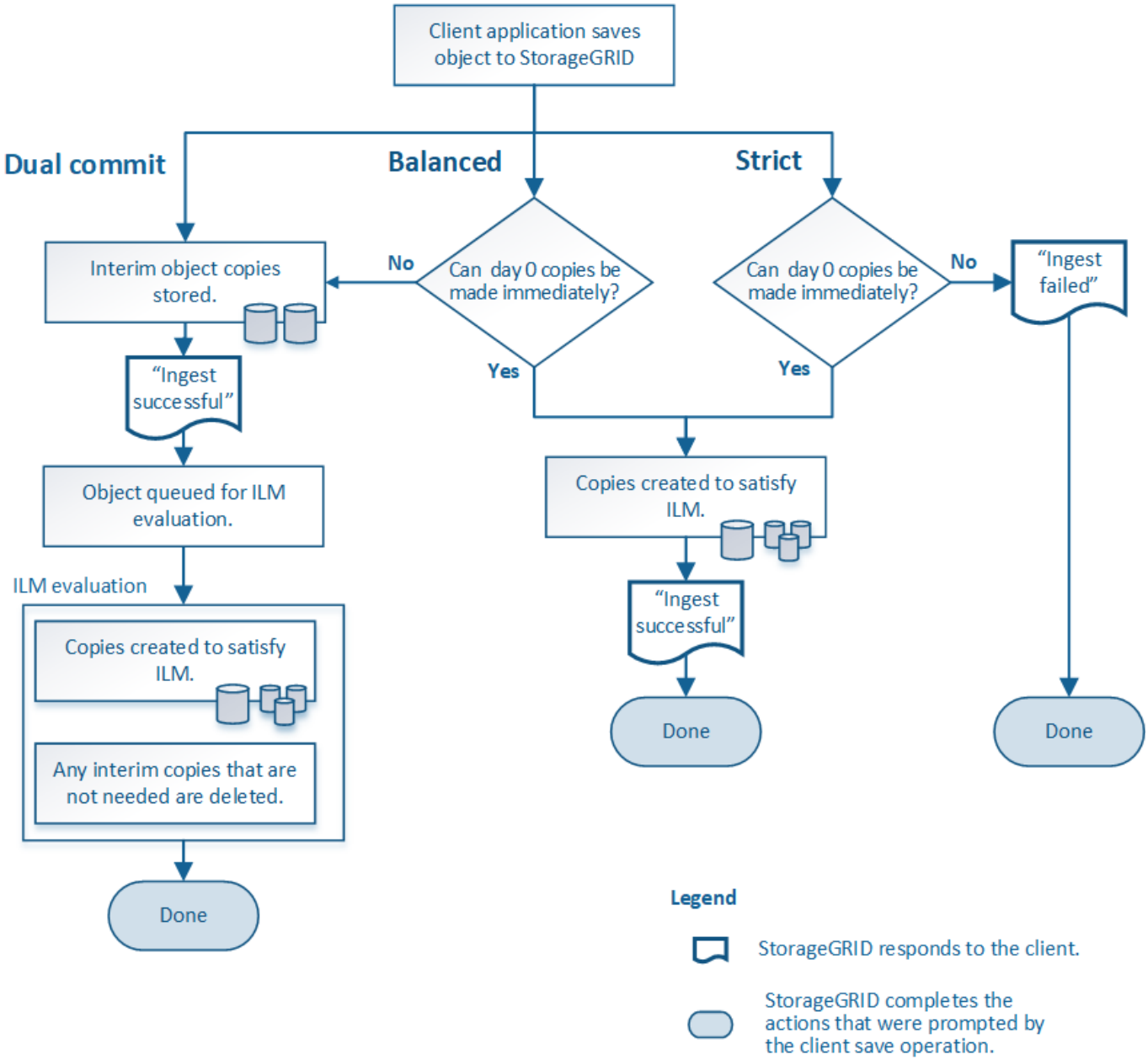
Ingest options

When you create an ILM rule, you specify one of three options for protecting objects at ingest: Dual commit, Strict, or Balanced.

Depending on your choice, StorageGRID makes interim copies and queues the objects for ILM evaluation later, or it uses synchronous placement and immediately makes copies to meet ILM requirements.

Flowchart of ingest options

The flowchart shows what happens when objects are matched by an ILM rule that uses each of the three ingest options.



Dual commit

When you select the Dual commit option, StorageGRID immediately makes interim object copies on two different Storage Nodes and returns an "ingest successful" message to the client. The object is queued for ILM evaluation, and copies that meet the rule's placement instructions are made later. If the ILM policy can't be processed immediately after the dual commit, site-loss protection could take time to achieve.

Use the Dual commit option in either of these cases:

- You are using multi-site ILM rules and client ingest latency is your primary consideration. When using Dual commit, you must ensure your grid can perform the additional work of creating and removing the dual-commit copies if they don't satisfy ILM. Specifically:
  - The load on the grid must be low enough to prevent an ILM backlog.
  - The grid must have excess hardware resources (IOPS, CPU, memory, network bandwidth, and so on).
- You are using multi-site ILM rules and the WAN connection between the sites usually has high latency or limited bandwidth. In this scenario, using the Dual commit option can help prevent client timeouts. Before choosing the Dual commit option, you should test the client application with realistic workloads.

Balanced (default)

When you select the Balanced option, StorageGRID also uses synchronous placement on ingest and immediately makes all copies specified in the rule's placement instructions. In contrast with the Strict option, if StorageGRID can't immediately make all copies, it uses Dual commit instead. If the ILM policy uses placements on multiple sites and immediate site-loss protection can't be achieved, the **ILM placement unachievable** alert is triggered.

Use the Balanced option to achieve the best combination of data protection, grid performance, and ingest success. Balanced is the default option in the Create ILM rule wizard.

Strict

When you select the Strict option, StorageGRID uses synchronous placement on ingest and immediately makes all object copies specified in the rule's placement instructions. Ingest fails if StorageGRID can't create all copies, for example, because a required storage location is temporarily unavailable. The client must retry the operation.

Use the Strict option if you have an operational or regulatory requirement to immediately store objects only in the locations outlined in the ILM rule. For example, to satisfy a regulatory requirement, you might need to use the Strict option and a Location Constraint advanced filter to guarantee that objects are never stored at certain data centers.

See Example 5: ILM rules and policy for Strict ingest behavior.

Advantages, disadvantages, and limitations of the ingest options

Understanding the advantages and disadvantages of each of the three options for protecting data at ingest (Balanced, Strict, or Dual commit) can help you decide which one to select for an ILM rule.

For an overview of ingest options, see Ingest options.

Advantages of the Balanced and Strict options

When compared to Dual commit, which creates interim copies during ingest, the two synchronous placement options can provide the following advantages:

- **Better data security**: Object data is immediately protected as specified in the ILM rule's placement instructions, which can be configured to protect against a wide variety of failure conditions, including the failure of more than one storage location. Dual commit can only protect against the loss of a single local copy.
- **More efficient grid operation**: Each object is processed only once, as it is ingested. Because the

StorageGRID system does not need to track or delete interim copies, there is less processing load and less database space is consumed.

- **(Balanced) Recommended**: The Balanced option provides optimal ILM efficiency. Using the Balanced option is recommended unless Strict ingest behavior is required or the grid meets all of the criteria for using Dual commit.
- **(Strict) Certainty about object locations**: The Strict option guarantees that objects are immediately stored according to the placement instructions in the ILM rule.

Disadvantages of the Balanced and Strict options

When compared to Dual commit, the Balanced and Strict options have some disadvantages:

- **Longer client ingests**: Client ingest latencies might be longer. When you use the Balanced or Strict options, an "ingest successful" message is not returned to the client until all erasure-coded fragments or replicated copies are created and stored. However, object data will most likely reach its final placement much faster.
- **(Strict) Higher rates of ingest failure**: With the Strict option, ingest fails whenever StorageGRID can't immediately make all copies specified in the ILM rule. You might see high rates of ingest failure if a required storage location is temporarily offline or if network issues cause delays in copying objects between sites.
- **(Strict) S3 multipart upload placements might not be as expected in some circumstances**: With Strict, you expect objects either to be placed as described by the ILM rule or for ingest to fail. However, with an S3 multipart upload, ILM is evaluated for each part of the object as it is ingested, and for the object as a whole when the multipart upload completes. In the following circumstances this might result in placements that are different than you expect:
  - **If ILM changes while an S3 multipart upload is in progress**: Because each part is placed according to the rule that is active when the part is ingested, some parts of the object might not meet current ILM requirements when the multipart upload completes. In these cases, ingest of the object does not fail. Instead, any part that is not placed correctly is queued for ILM re-evaluation and is moved to the correct location later.
  - **When ILM rules filter on size**: When evaluating ILM for a part, StorageGRID filters on the size of the part, not the size of the object. This means that parts of an object can be stored in locations that don't meet ILM requirements for the object as a whole. For example, if a rule specifies that all objects 10 GB or larger are stored at DC1 while all smaller objects are stored at DC2, at ingest each 1 GB part of a 10-part multipart upload is stored at DC2. When ILM is evaluated for the object, all parts of the object are moved to DC1.
- **(Strict) Ingest does not fail when object tags or metadata are updated and newly required placements cannot be made**: With Strict, you expect objects either to be placed as described by the ILM rule or for ingest to fail. However, when you update metadata or tags for an object that is already stored in the grid, the object is not re-ingested. This means that any changes to object placement that are triggered by the update aren't made immediately. Placement changes are made when ILM is re-evaluated by normal background ILM processes. If required placement changes can't be made (for example, because a newly required location is unavailable), the updated object retains its current placement until the placement changes are possible.

Limitations on object placements with the Balanced and Strict options

The Balanced or Strict options can't be used for ILM rules that have any of these placement instructions:

- Placement in a Cloud Storage Pool at day 0.
- Placement in an Archive Node at day 0.

- Placements in a Cloud Storage Pool or an Archive Node when the rule has a User defined creation time as its Reference time.

These restrictions exist because StorageGRID can't synchronously make copies to a Cloud Storage Pool or an Archive Node, and a User defined creation time could resolve to the present.

How ILM rules and consistency interact to affect data protection

Both your ILM rule and your choice of consistency affect how objects are protected. These settings can interact.

For example, the ingest behavior selected for an ILM rule affects the initial placement of object copies, while the consistency used when an object is stored affects the initial placement of object metadata. Because StorageGRID requires access to both an object's data and metadata to fulfill client requests, selecting matching levels of protection for the consistency and ingest behavior can provide better initial data protection and more predictable system responses.

Here is a brief summary of the consistency values that are available in StorageGRID:

- **All**: All nodes receive object metadata immediately or the request will fail.
- **Strong-global**: Object metadata is immediately distributed to all sites. Guarantees read-after-write consistency for all client requests across all sites.
- **Strong-site**: Object metadata is immediately distributed to other nodes at the site. Guarantees read-after-write consistency for all client requests within a site.
- **Read-after-new-write**: Provides read-after-write consistency for new objects and eventual consistency for object updates. Offers high availability and data protection guarantees. Recommended for most cases.
- **Available**: Provides eventual consistency for both new objects and object updates. For S3 buckets, use only as required (for example, for a bucket that contains log values that are rarely read, or for HEAD or GET operations on keys that don't exist). Not supported for S3 FabricPool buckets.

Before selecting a consistency value, read the full description of consistency. You should understand the benefits and limitations before changing the default value.

Example of how consistency and ILM rules can interact

Suppose you have a two-site grid with the following ILM rule and the following consistency:

- **ILM rule**: Create two object copies, one at the local site and one at a remote site. Use Strict ingest behavior.
- **consistency**: Strong-global (object metadata is immediately distributed to all sites).

When a client stores an object to the grid, StorageGRID makes both object copies and distributes metadata to both sites before returning success to the client.

The object is fully protected against loss at the time of the ingest successful message. For example, if the local site is lost shortly after ingest, copies of both the object data and the object metadata still exist at the remote site. The object is fully retrievable.

If you instead used the same ILM rule and the strong-site consistency, the client might receive a success message after object data is replicated to the remote site but before object metadata is distributed there. In this case, the level of protection of object metadata does not match the level of protection for object data. If the local site is lost shortly after ingest, object metadata is lost. The object can't be retrieved.

The inter-relationship between consistency and ILM rules can be complex. Contact NetApp if you need assistance.

Related information

- Example 5: ILM rules and policy for Strict ingest behavior

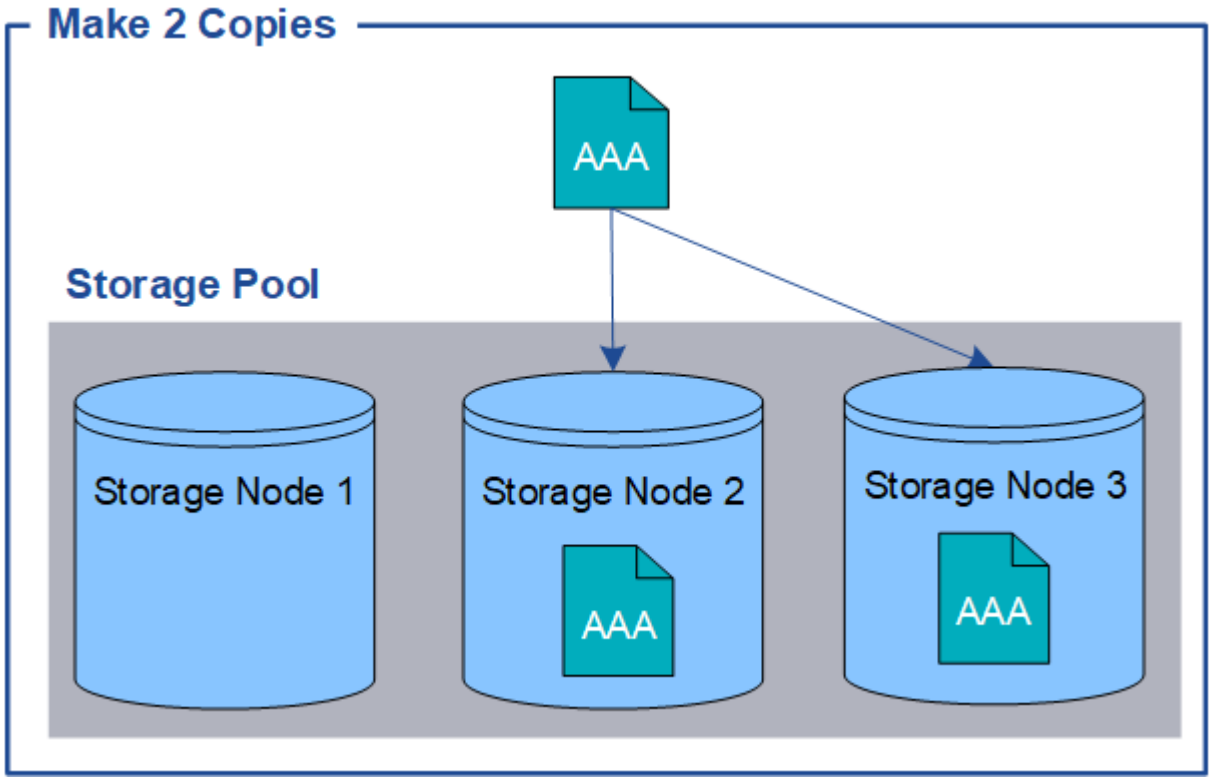
How objects are stored (replication or erasure coding)

What is replication?

Replication is one of two methods used by StorageGRID to store object data. When objects match an ILM rule that uses replication, the system creates exact copies of object data and stores the copies on Storage Nodes or Archive Nodes.

When you configure an ILM rule to create replicated copies, you specify how many copies should be created, where those copies should be placed, and how long the copies should be stored at each location.

In the following example, the ILM rule specifies that two replicated copies of each object be placed in a storage pool that contains three Storage Nodes.



When StorageGRID matches objects to this rule, it creates two copies of the object, placing each copy on a different Storage Node in the storage pool. The two copies might be placed on any two of the three available Storage Nodes. In this case, the rule placed object copies on Storage Nodes 2 and 3. Because there are two copies, the object can be retrieved if any of the nodes in the storage pool fails.

StorageGRID can store only one replicated copy of an object on any given Storage Node. If your grid includes three Storage Nodes and you create a 4-copy ILM rule, only three copies will be made—one copy for each Storage Node. The **ILM placement unachievable** alert is triggered to indicate that the ILM rule could not be completely applied.

Related information

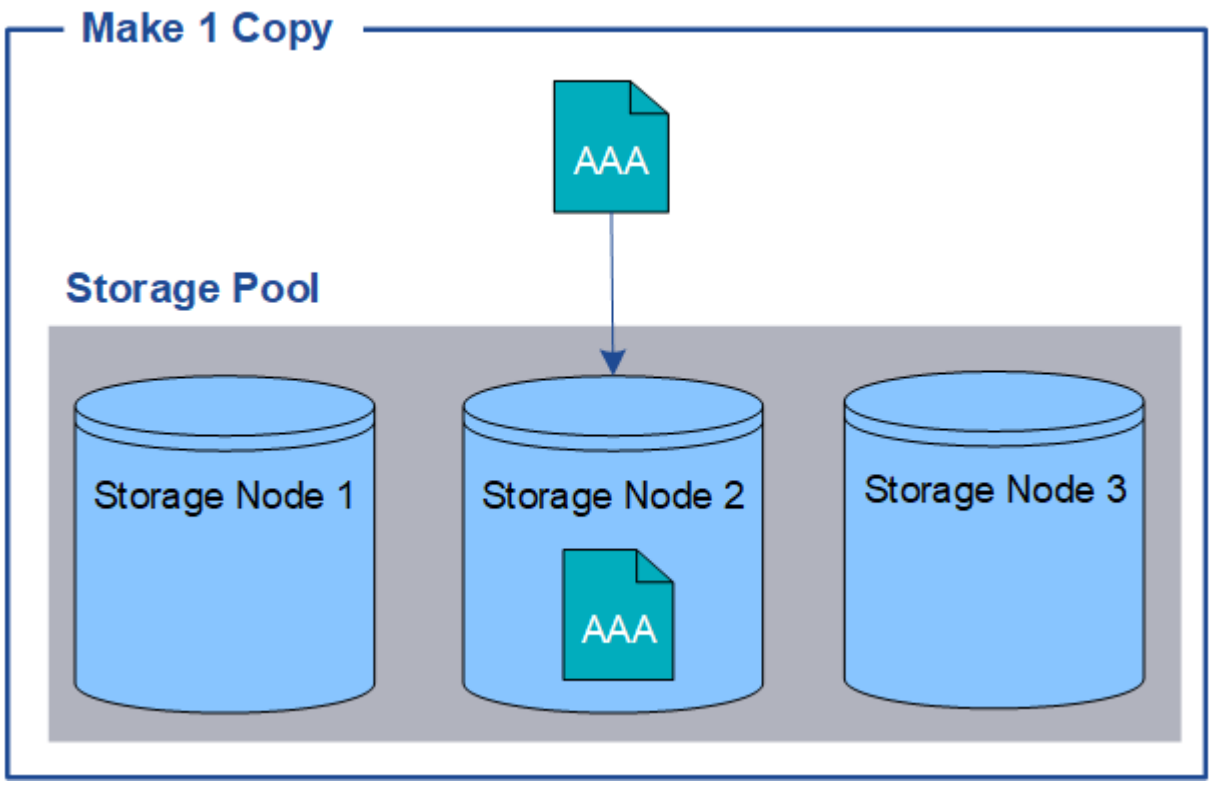
- What is erasure coding
- What is a storage pool
- Enable site-loss protection using replication and erasure coding

Why you should not use single-copy replication

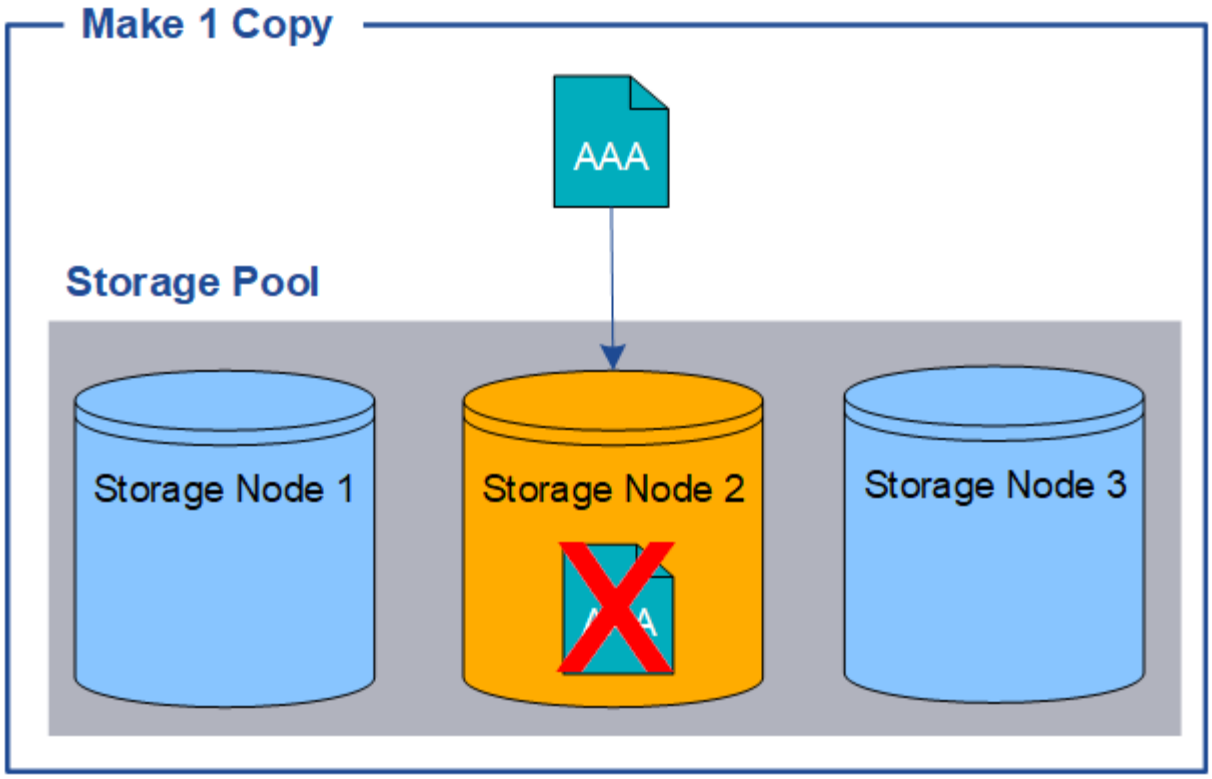
When creating an ILM rule to create replicated copies, you should always specify at least two copies for any time period in the placement instructions.

Don't use an ILM rule that creates only one replicated copy for any time period. If only one replicated copy of an object exists, that object is lost if a Storage Node fails or has a significant error. You also temporarily lose access to the object during maintenance procedures such as upgrades.

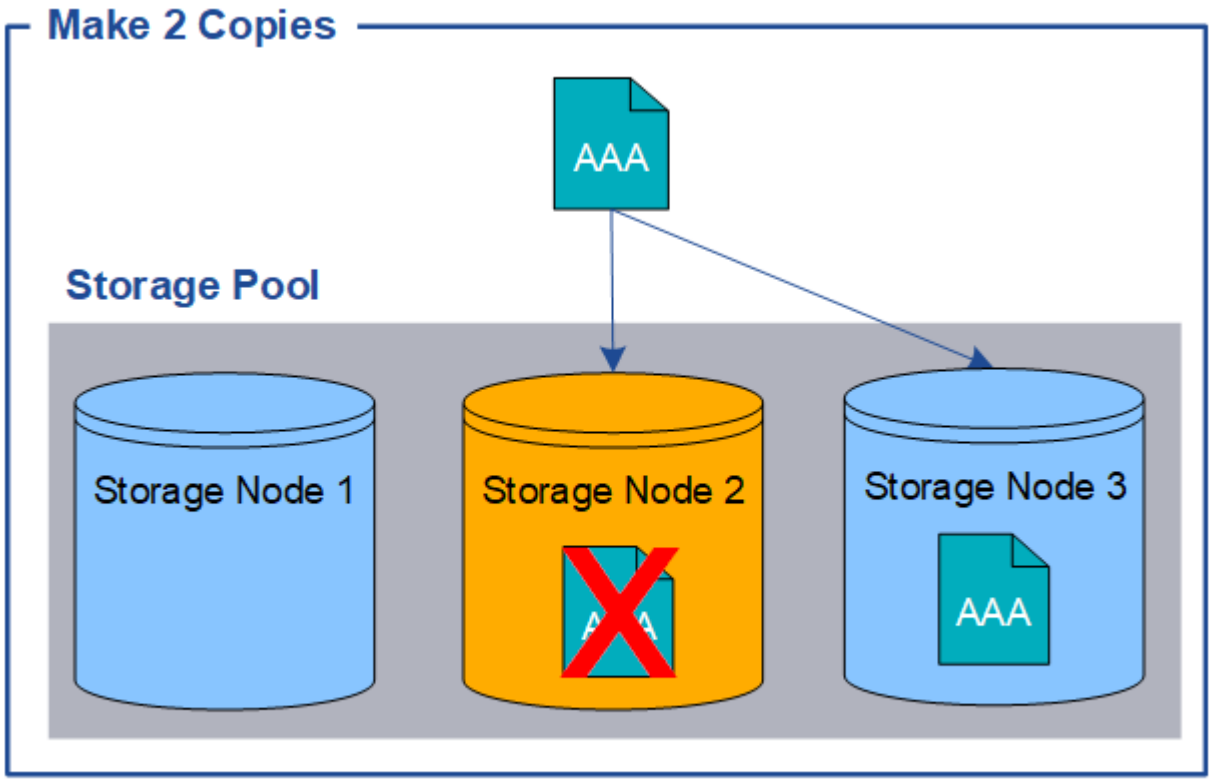
In the following example, the Make 1 Copy ILM rule specifies that one replicated copy of an object be placed in a storage pool that contains three Storage Nodes. When an object is ingested that matches this rule, StorageGRID places a single copy on only one Storage Node.



When an ILM rule creates only one replicated copy of an object, the object becomes inaccessible when the Storage Node is unavailable. In this example, you will temporarily lose access to object AAA whenever Storage Node 2 is offline, such as during an upgrade or other maintenance procedure. You will lose object AAA entirely if Storage Node 2 fails.



To avoid losing object data, you should always make at least two copies of all objects you want to protect with replication. If two or more copies exist, you can still access the object if one Storage Node fails or goes offline.



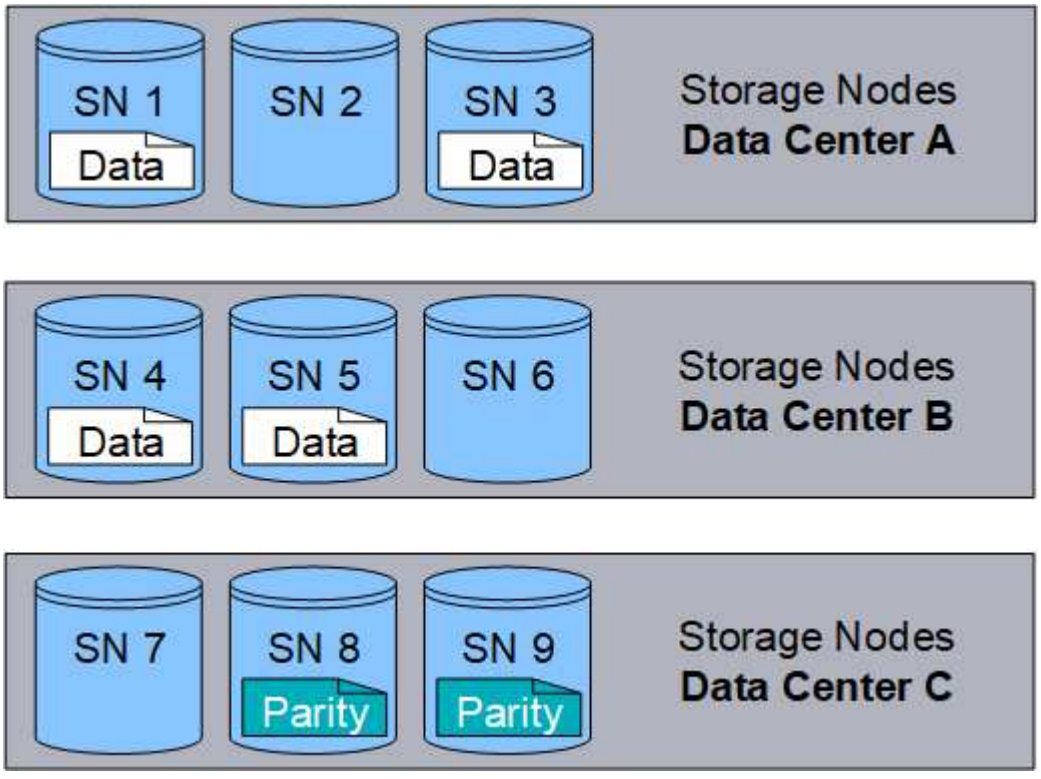
What is erasure coding?

Erasure coding is one of two methods StorageGRID uses to store object data. When objects match an ILM rule that uses erasure coding, those objects are sliced into data fragments, additional parity fragments are computed, and each fragment is stored on a different Storage Node.

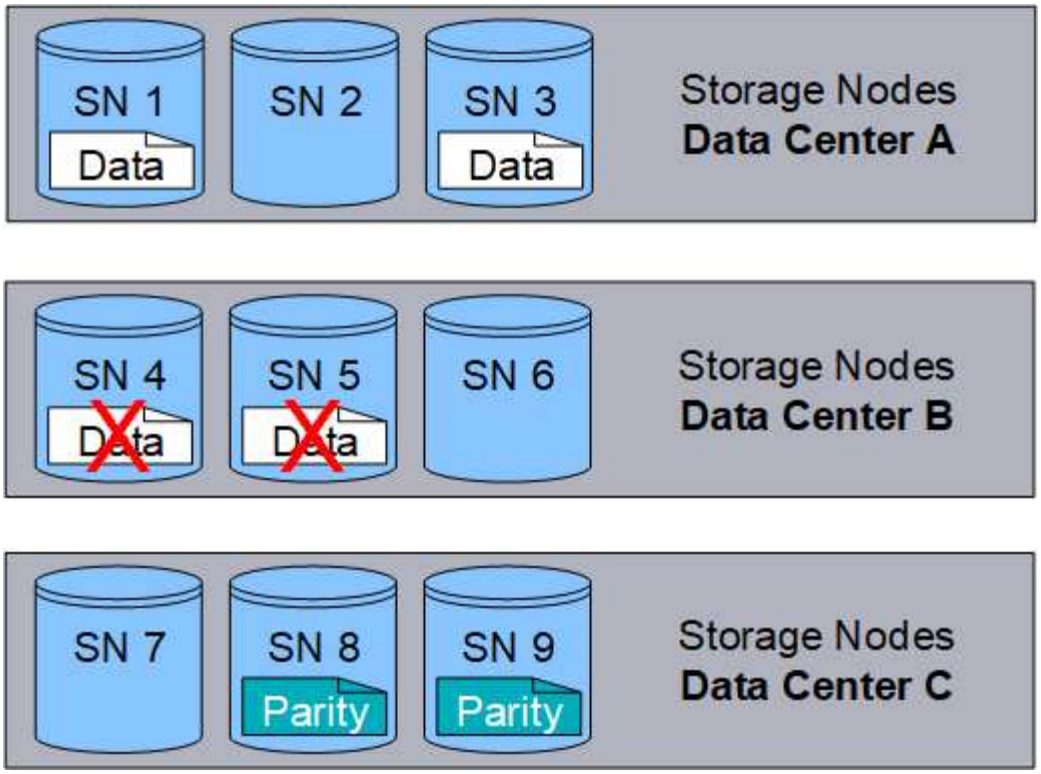
When an object is accessed, it is reassembled using the stored fragments. If a data or a parity fragment becomes corrupt or lost, the erasure-coding algorithm can recreate that fragment using a subset of the remaining data and parity fragments.

As you create ILM rules, StorageGRID creates erasure-coding profiles that support those rules. You can view a list of erasure-coding profiles, rename an erasure-coding profile, or deactivate an erasure-coding profile if it is not currently used in any ILM rules.

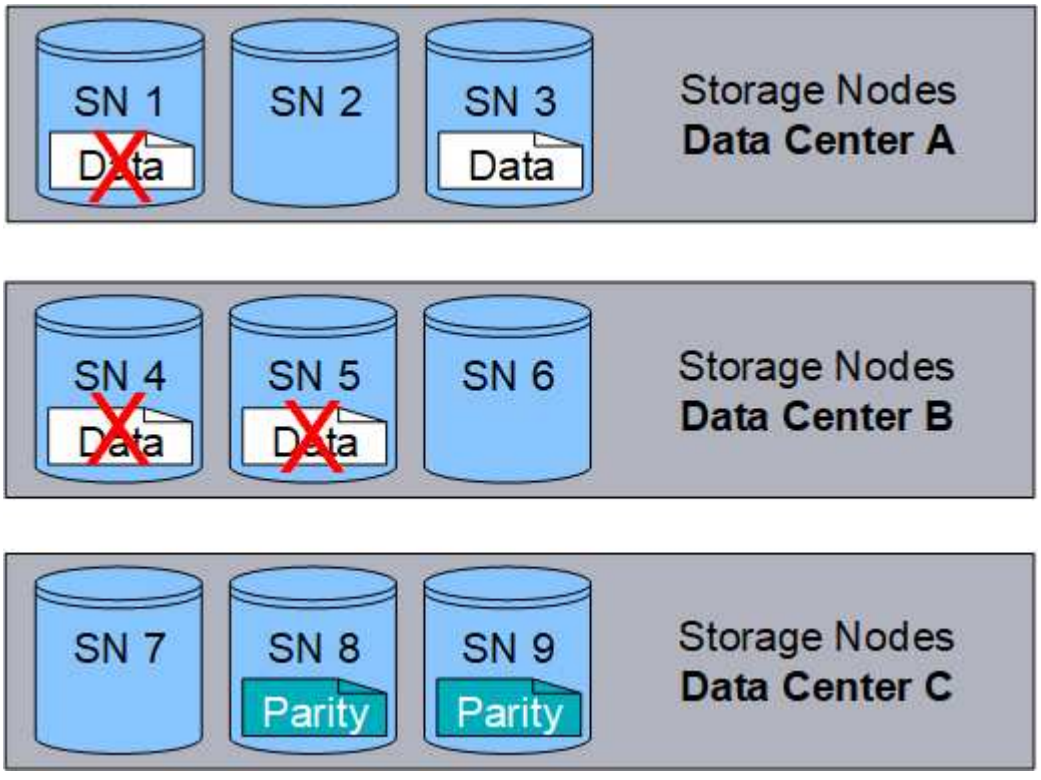
The following example illustrates the use of an erasure-coding algorithm on an object's data. In this example, the ILM rule uses a 4+2 erasure-coding scheme. Each object is sliced into four equal data fragments, and two parity fragments are computed from the object data. Each of the six fragments is stored on a different node across three data center sites to provide data protection for node failures or site loss.



The 4+2 erasure-coding scheme can be configured in various ways. For example, you can configure a single-site storage pool that contains six Storage Nodes. For site-loss protection, you can use a storage pool containing three sites with three Storage Nodes at each site. An object can be retrieved as long as any four of the six fragments (data or parity) remain available. Up to two fragments can be lost without loss of the object data. If an entire site is lost, the object can still be retrieved or repaired, as long as all of the other fragments remain accessible.



If more than two Storage Nodes are lost, the object is not retrievable.



Related information

- What is replication
- What is a storage pool
- What are erasure-coding schemes

- Rename an erasure-coding profile
- Deactivate an erasure-coding profile

What are erasure-coding schemes?

Erasure-coding schemes control how many data fragments and how many parity fragments are created for each object.

When you configure the erasure-coding profile for an ILM rule, you select an available erasure-coding scheme based on how many Storage Nodes and sites make up the storage pool you plan to use.

The StorageGRID system uses the Reed-Solomon erasure-coding algorithm. The algorithm slices an object into `k` data fragments and computes `m` parity fragments. The `k + m = n` fragments are spread across `n` Storage Nodes to provide data protection. An object can sustain up to `m` lost or corrupt fragments. To retrieve or repair an object, `k` fragments are needed.

When selecting the storage pool to use for a rule that will create an erasure-coded copy, use the following guidelines for storage pools:

- The storage pool must include three or more sites, or exactly one site.

  > ⓘ You can't use erasure coding if the storage pool includes two sites.

  - Erasure-coding schemes for storage pools containing three or more sites
  - Erasure-coding schemes for one-site storage pools
- Don't use a storage pool that includes the default site, All Sites.
- The storage pool should include at least *k+m* `+1` Storage Nodes that can store object data.

  > ⓘ Storage Nodes can be configured during installation to contain only object metadata and not object data. For more information, see Types of Storage Nodes.

  The minimum number of Storage Nodes required is `k+m`. However, having at least one additional Storage Node can help prevent ingest failures or ILM backlogs if a required Storage Node is temporarily unavailable.

The storage overhead of an erasure-coding scheme is calculated by dividing the number of parity fragments (`m`) by the number of data fragments (`k`). You can use the storage overhead to calculate how much disk space each erasure-coded object requires:

*disk space = object size + (object size * storage overhead)*

For example, if you store a 10 MB object using the 4+2 scheme (which has 50% storage overhead), the object consumes 15 MB of grid storage. If you store the same 10 MB object using the 6+2 scheme (which has 33% storage overhead), the object consumes approximately 13.3 MB.

Select the erasure-coding scheme with the lowest total value of `k+m` that meets your needs. Erasure-coding schemes with a lower number of fragments are overall more computationally efficient, as fewer fragments are created and distributed (or retrieved) per object, can show better performance due to the larger fragment size, and can require fewer nodes be added in an expansion when more storage is required. (For information about planning a storage expansion, see the instructions for expanding StorageGRID.)

Erasure-coding schemes for storage pools containing three or more sites

The following table describes the erasure-coding schemes currently supported by StorageGRID for storage pools that include three or more sites. All of these schemes provide site-loss protection. One site can be lost, and the object will still be accessible.

For erasure-coding schemes that provide site-loss protection, the recommended number of Storage Nodes in the storage pool exceeds `k+m +1` because each site requires a minimum of three Storage Nodes.

Erasure-coding scheme (*k+m*)	Minimum number of deployed sites	Recommended number of Storage Nodes at each site	Total recommended number of Storage Nodes	Site loss protection?	Storage overhead
4+2	3	3	9	Yes	50%
6+2	4	3	12	Yes	33%
8+2	5	3	15	Yes	25%
6+3	3	4	12	Yes	50%
9+3	4	4	16	Yes	33%
2+1	3	3	9	Yes	50%
4+1	5	3	15	Yes	25%
6+1	7	3	21	Yes	17%
7+5	3	5	15	Yes	71%

> StorageGRID requires a minimum of three Storage Nodes per site. To use the 7+5 scheme, each site requires a minimum of four Storage Nodes. Using five Storage Nodes per site is recommended.

When selecting an erasure-coding scheme that provides site protection, balance the relative importance of the following factors:

- **Number of fragments**: Performance and expansion flexibility are generally better when the total number of fragments is lower.
- **Fault tolerance**: Fault tolerance is increased by having more parity segments (that is, when `m` has a higher value.)
- **Network traffic**: When recovering from failures, using a scheme with more fragments (that is, a higher total for `k+m`) creates more network traffic.
- **Storage overhead**: Schemes with higher overhead require more storage space per object.

For example, when deciding between a 4+2 scheme and 6+3 scheme (which both have 50% storage overhead), select the 6+3 scheme if additional fault tolerance is required. Select the 4+2 scheme if network

resources are constrained. If all other factors are equal, select 4+2 because it has a lower total number of fragments.

If you are unsure of which scheme to use, select 4+2 or 6+3, or contact technical support.

Erasure-coding schemes for one-site storage pools

A one-site storage pool supports all of the erasure-coding schemes defined for three or more sites, provided that the site has enough Storage Nodes.

The minimum number of Storage Nodes required is `k+m`, but a storage pool with `k+m +1` Storage Nodes is recommended. For example, the 2+1 erasure-coding scheme requires a storage pool with a minimum of three Storage Nodes, but four Storage Nodes is recommended.

Erasure-coding scheme (*k+m*)	Minimum number of Storage Nodes	Recommended number of Storage Nodes	Storage overhead
4+2	6	7	50%
6+2	8	9	33%
8+2	10	11	25%
6+3	9	10	50%
9+3	12	13	33%
2+1	3	4	50%
4+1	5	6	25%
6+1	7	8	17%
7+5	12	13	71%

Advantages, disadvantages, and requirements for erasure coding

Before deciding whether to use replication or erasure coding to protect object data from loss, you should understand the advantages, disadvantages, and the requirements for erasure coding.

Advantages of erasure coding

When compared to replication, erasure coding offers improved reliability, availability, and storage efficiency.

- **Reliability**: Reliability is gauged in terms of fault tolerance—that is, the number of simultaneous failures that can be sustained without loss of data. With replication, multiple identical copies are stored on different nodes and across sites. With erasure coding, an object is encoded into data and parity fragments and distributed across many nodes and sites. This dispersal provides both site and node failure protection.

When compared to replication, erasure coding provides improved reliability at comparable storage costs.

- **Availability**: Availability can be defined as the ability to retrieve objects if Storage Nodes fail or become inaccessible. When compared to replication, erasure coding provides increased availability at comparable storage costs.
- **Storage efficiency**: For similar levels of availability and reliability, objects protected through erasure coding consume less disk space than the same objects would if protected through replication. For example, a 10 MB object that is replicated to two sites consumes 20 MB of disk space (two copies), while an object that is erasure-coded across three sites with a 6+3 erasure-coding scheme only consumes 15 MB of disk space.

> Disk space for erasure-coded objects is calculated as the object size plus the storage overhead. The storage overhead percentage is the number of parity fragments divided by the number of data fragments.

Disadvantages of erasure coding

When compared to replication, erasure coding has the following disadvantages:

- An increased number of Storage Nodes and sites is recommended, depending on the erasure-coding scheme. In contrast, if you replicate object data, you need only one Storage Node for each copy. See Erasure-coding schemes for storage pools containing three or more sites and Erasure-coding schemes for one-site storage pools.
- Increased cost and complexity of storage expansions. To expand a deployment that uses replication, you add storage capacity in every location where object copies are made. To expand a deployment that uses erasure coding, you must consider both the erasure-coding scheme in use and how full existing Storage Nodes are. For example, if you wait until existing nodes are 100% full, you must add at least `k+m` Storage Nodes, but if you expand when existing nodes are 70% full, you can add two nodes per site and still maximize usable storage capacity. For more information, see Add storage capacity for erasure-coded objects.
- There are increased retrieval latencies when you use erasure coding across geographically distributed sites. The object fragments for an object that is erasure-coded and distributed across remote sites take longer to retrieve over WAN connections than an object that is replicated and available locally (the same site to which the client connects).
- When you use erasure coding across geographically distributed sites, there is higher WAN network traffic usage for retrievals and repairs, especially for frequently retrieved objects or for object repairs over WAN network connections.
- When you use erasure coding across sites, the maximum object throughput declines sharply as network latency between sites increases. This decrease is due to the corresponding decrease in TCP network throughput, which affects how quickly the StorageGRID system can store and retrieve object fragments.
- Higher usage of compute resources.

When to use erasure coding

Erasure coding is best suited for the following requirements:

- Objects greater than 1 MB in size.

> Erasure coding is best suited for objects greater than 1 MB. Don’t use erasure coding for objects smaller than 200 KB to avoid the overhead of managing very small erasure-coded fragments.

- Long-term or cold storage for infrequently retrieved content.
- High data availability and reliability.
- Protection against complete site and node failures.
- Storage efficiency.
- Single-site deployments that require efficient data protection with only a single erasure-coded copy rather than multiple replicated copies.
- Multiple-site deployments where the inter-site latency is less than 100 ms.

How object retention is determined

StorageGRID provides options for both grid administrators and individual tenant users to specify how long to store objects. In general, any retention instructions provided by a tenant user take precedence over the retention instructions provided by the grid administrator.

How tenant users control object retention

Tenant users have three primary ways to control how long their objects are stored in StorageGRID:

- If the global S3 Object Lock setting is enabled for the grid, S3 tenant users can create buckets with S3 Object Lock enabled and then use the S3 REST API to specify retain-until-date and legal hold settings for each object version added to that bucket.
  - An object version that is under a legal hold can’t be deleted by any method.
  - Before an object version’s retain-until-date is reached, that version can’t be deleted by any method.
  - Objects in buckets with S3 Object Lock enabled are retained by ILM "forever." However, after its retain-until-date is reached, an object version can be deleted by a client request or the expiration of the bucket lifecycle. See Manage objects with S3 Object Lock.
- S3 tenant users can add a lifecycle configuration to their buckets that specifies an Expiration action. If a bucket lifecycle exists, StorageGRID stores an object until the date or number of days specified in the Expiration action are met, unless the client deletes the object first. See Create S3 lifecycle configuration.
- An S3 or Swift client can issue a delete object request. StorageGRID always prioritizes client delete requests over S3 bucket lifecycle or ILM when determining whether to delete or retain an object.

How grid administrators control object retention

Grid administrators use ILM placement instructions to control how long objects are stored. When objects are matched by an ILM rule, StorageGRID stores those objects until the last time period in the ILM rule has elapsed. Objects are retained indefinitely if "forever" is specified for the placement instructions.

Regardless of who controls how long objects are retained, ILM settings control what types of object copies (replicated or erasure-coded) are stored and where the copies are located (Storage Nodes, Cloud Storage Pools, or Archive Nodes).

How S3 bucket lifecycle and ILM interact

When an S3 bucket lifecycle is configured, the lifecycle expiration actions override the ILM policy for objects that match the lifecycle filter. As a result, an object might be retained on the grid even after any ILM instructions for placing the object have lapsed.

Examples for object retention

To better understand the interactions between S3 Object Lock, bucket lifecycle settings, client delete requests, and ILM, consider the following examples.

Example 1: S3 bucket lifecycle keeps objects longer than ILM

ILM

Store two copies for 1 year (365 days)

Bucket lifecycle

Expire objects in 2 years (730 days)

Result

StorageGRID stores the object for 730 days. StorageGRID uses the bucket lifecycle settings to determine whether to delete or retain an object.

> If the bucket lifecycle specifies that objects should be kept longer than specified by ILM, StorageGRID continues to use the ILM placement instructions when determining the number and type of copies to store. In this example, two copies of the object will continue to be stored in StorageGRID from days 366 to 730.

Example 2: S3 bucket lifecycle expires objects before ILM

ILM

Store two copies for 2 years (730 days)

Bucket lifecycle

Expire objects in 1 year (365 days)

Result

StorageGRID deletes both copies of the object after day 365.

Example 3: Client delete overrides bucket lifecycle and ILM

ILM

Store two copies on Storage Nodes "forever"

Bucket lifecycle

Expire objects in 2 years (730 days)

Client delete request

Issued on day 400

Result

StorageGRID deletes both copies of the object on day 400 in response to the client delete request.

Example 4: S3 Object Lock overrides client delete request

S3 Object Lock

Retain-until-date for an object version is 2026-03-31. A legal hold is not in effect.

Compliant ILM rule

Store two copies on Storage Nodes "forever"

Client delete request

Issued on 2024-03-31

Result

StorageGRID will not delete the object version because the retain-until-date is still 2 years away.

How objects are deleted

StorageGRID can delete objects either in direct response to a client request or automatically as a result of the expiration of an S3 bucket lifecycle or the requirements of the ILM policy. Understanding the different ways that objects can be deleted and how StorageGRID handles delete requests can help you manage objects more effectively.

StorageGRID can use one of two methods to delete objects:

- Synchronous deletion: When StorageGRID receives a client delete request, all object copies are removed immediately. The client is informed that deletion was successful after the copies have been removed.
- Objects are queued for deletion: When StorageGRID receives a delete request, the object is queued for deletion and the client is informed immediately that deletion was successful. Object copies are removed later by background ILM processing.

When deleting objects, StorageGRID uses the method that optimizes delete performance, minimizes potential delete backlogs, and frees space most quickly.

The table summarizes when StorageGRID uses each method.

Method of performing deletion	When used
Objects are queued for deletion	When **any** of the following conditions are true: • Automatic object deletion has been triggered by one of the following events: ◦ The expiration date or number of days in the lifecycle configuration for an S3 bucket is reached. ◦ The last time period specified in an ILM rule elapses. **Note:** Objects in a bucket that has S3 Object Lock enabled can’t be deleted if they are under a legal hold or if a retain-until-date has been specified but not yet met. • An S3 or Swift client requests deletion and one or more of these conditions is true: ◦ Copies can’t be deleted within 30 seconds because, for example, an object location is temporarily unavailable. ◦ Background deletion queues are idle.

Method of performing deletion	When used
Objects are removed immediately (synchronous deletion)	When an S3 or Swift client makes a delete request and **all** of the following conditions are met: • All copies can be removed within 30 seconds. • Background deletion queues contain objects to process.

When S3 or Swift clients make delete requests, StorageGRID begins by adding objects to the delete queue. It then switches to performing synchronous deletion. Making sure that the background deletion queue has objects to process allows StorageGRID to process deletes more efficiently, especially for low concurrency clients, while helping to prevent client delete backlogs.

Time required to delete objects

The way that StorageGRID deletes objects can affect how the system appears to perform:

- When StorageGRID performs synchronous deletion, it can take StorageGRID up to 30 seconds to return a result to the client. This means that deletion can appear to be happening more slowly, even though copies are actually being removed more quickly than they are when StorageGRID queues objects for deletion.
- If you are closely monitoring delete performance during a bulk delete, you might notice that the deletion rate appears to be slow after a certain number of objects have been deleted. This change occurs when StorageGRID shifts from queuing objects for deletion to performing synchronous deletion. The apparent reduction in the deletion rate does not mean that object copies are being removed more slowly. On the contrary, it indicates that on average, space is now being freed more quickly.

If you are deleting large numbers of objects and your priority is to free space quickly, consider using a client request to delete objects rather than deleting them using ILM or other methods. In general, space is freed more quickly when deletion is performed by clients because StorageGRID can use synchronous deletion.

The amount of time required to free space after an object is deleted depends on several factors:

- Whether object copies are synchronously removed or are queued for removal later (for client delete requests).
- Other factors such as the number of objects in the grid or the availability of grid resources when object copies are queued for removal (for both client deletes and other methods).

How S3 versioned objects are deleted

When versioning is enabled for an S3 bucket, StorageGRID follows Amazon S3 behavior when responding to delete requests, whether those requests come from an S3 client, the expiration of an S3 bucket lifecycle, or the requirements of the ILM policy.

When objects are versioned, object delete requests don’t delete the current version of the object and don’t free space. Instead, an object delete request creates a zero-byte delete marker as the current version of the object, which makes the previous version of the object "noncurrent." An object delete marker becomes an expired object delete marker when it is the current version and there are no noncurrent versions.

Even though the object has not been removed, StorageGRID behaves as though the current version of the object is no longer available. Requests to that object return 404 NotFound. However, because noncurrent object data has not been removed, requests that specify a noncurrent version of the object can succeed.

To free space when deleting versioned objects, or to remove delete markers, use one of the following:

- **S3 client request**: Specify the object version ID in the S3 DELETE Object request (`DELETE /object?versionId=ID`). Keep in mind that this request only removes object copies for the specified version (the other versions are still taking up space).
- **Bucket lifecycle**: Use the `NoncurrentVersionExpiration` action in the bucket lifecycle configuration. When the number of NoncurrentDays specified is met, StorageGRID permanently removes all copies of noncurrent object versions. These object versions can't be recovered.

  The `NewerNoncurrentVersions` action in the bucket lifecycle configuration specifies the number of noncurrent versions retained in a versioned S3 bucket. If there are more noncurrent versions than `NewerNoncurrentVersions` specifies, StorageGRID removes the older versions when the NoncurrentDays value has elapsed. The `NewerNoncurrentVersions` threshold overrides lifecycle rules provided by ILM, meaning that a noncurrent object with a version within the `NewerNoncurrentVersions` threshold is retained if ILM requests its deletion.

  To remove expired object delete markers use the `Expiration` action with one of the following tags: `ExpiredObjectDeleteMarker`, `Days`, or `Date`.

- **ILM**: Clone an active policy and add two ILM rules to the new policy:
  - First rule: Use "Noncurrent time" as the Reference time to match the noncurrent versions of the object. In Step 1 (Enter details) of the Create an ILM rule wizard, select **Yes** for the question, "Apply this rule to older object versions only (in S3 buckets with versioning enabled)?"
  - Second rule: Use **Ingest time** to match the current version. The "Noncurrent time" rule must appear in the policy above the **Ingest time** rule.

  > ⓘ ILM cannot be used to remove current object delete markers. Use an S3 client request or S3 Bucket Lifecycle to remove current object delete markers.

- **Delete objects in bucket**: Use the tenant manager to delete all object versions, including delete markers, from a bucket.

  When a versioned object is deleted, StorageGRID creates a zero-byte delete marker as the current version of the object. All objects and delete markers must be removed before a versioned bucket can be deleted.

  - Delete markers created in StorageGRID 11.7 or earlier can only be removed through S3 client requests, they are not removed by ILM, bucket lifecycle rules, or Delete objects in bucket operations.
  - Delete markers from a bucket that was created in StorageGRID 11.8 or later can be removed by ILM, bucket lifecycle rules, Delete objects in bucket operations, or an explicit S3 client deletion. Expired delete markers in StorageGRID 11.8 or later must be removed by bucket lifecycle rules or by an explicit S3 client request with a version ID specified.

Related information

- Use S3 REST API
- Example 4: ILM rules and policy for S3 versioned objects

Copyright information

Trademark information