Inside the Amazon S3 Object Index, Part 1: Why Key Order Creates Hot Partitions

Julien Laurenceau | Published 2026-11-13 | Updated 2026-11-13

https://pepitedata.com/s3-object-index-part-1-key-order-hot-partitions/

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



This is Part 1 of a two-part series on the Amazon S3 object index: how S3 decides which partition a key lives in, why that decision made hot partitions a real problem, and what AWS does not publish about it. Part 2 follows what AWS told customers to do about hot prefixes, from the 2012 salting advice to the 2026 user guide, and checks six widely repeated claims against the sources.

Amazon S3 stores an object by writing its bytes in one place and its name in another. The name lives in the object index, a distributed key-value map that turns a key into metadata and a location. In the re:Invent 2025 session on S3 performance (STG335), AWS engineers describe S3 as three components working together: a front end that authenticates, logs and orchestrates requests, the index, and a storage layer that owns the disk fleet. Every GET, PUT, HEAD, LIST and DELETE passes through the index, including the operations that never touch object data at all.

The one-line version: **S3 partitions the index by key prefix, in UTF-8 binary order, and the key name decides which partition stores it. That choice buys prefix listing for free and makes a sequential prefix dangerous. Salting a prefix never changed the partition function, it moved a key to a different region of the same sorted index.**

What is the S3 object index?

The S3 object index is a distributed key-value store that maps an object's key name to its metadata: the key itself, where the bytes are, and when the object was created. AWS engineers describe S3 as a front end, an index and a storage layer, and call the index "a very very large distributed key value store" (re:Invent 2025, STG335).

The index sits on the critical path of every data-plane request, because a HEAD or a LIST needs metadata and never needs bytes.

In the re:Invent 2025 session Deep dive on Amazon S3, AWS engineers put the request volume in perspective: more requests reach the index than reach the storage layer, because metadata operations outnumber byte reads. That is a talk, not a document, and I mark it as such wherever it appears here. The architectural split itself is documented: S3 is built from over 300 software microservices, and object data is stored by ShardStore, the storage-node service AWS presented at SOSP 2021 as a key-value storage node.

The index is therefore the hard part of object storage, and not because of size. It is hard because S3 promises two things that pull against each other: a flat namespace over a virtually unbounded number of keys, and listings returned in lexicographic order. One of those promises is cheap to keep in a sharded system. The other one is the reason this series exists.

S3 partitions the index by key prefix, in binary order

S3 partitions the index by the key prefix, and the key is represented internally as the string `bucketname/keyname`. AWS stated this in 2012, in an engineering post by Doug Grismore, then Director of Storage Operations: each key in the internal keymap is "stored and retrieved based on the name provided when the object is first put into S3."

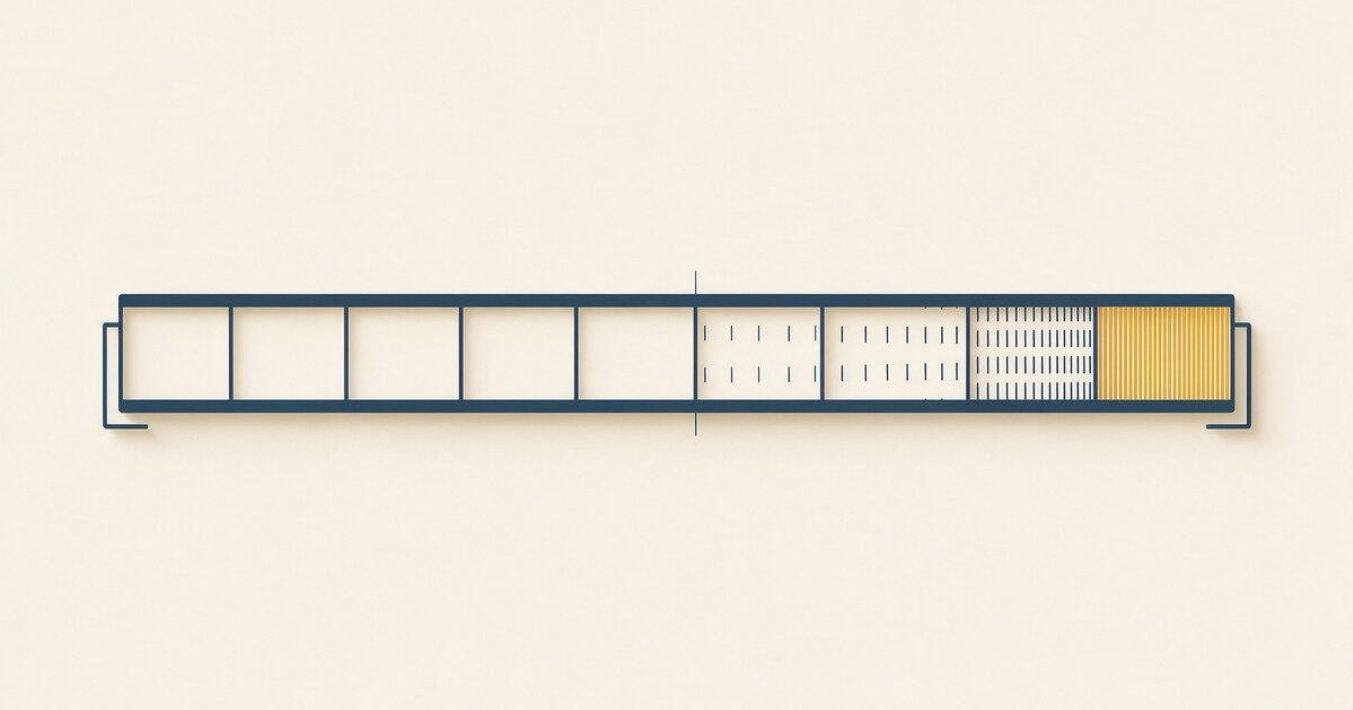
The same post states that "keys in S3 are partitioned by prefix." The 2017 Developer Guide said the same thing from the customer side: "Object keys are stored in UTF-8 binary ordering across multiple partitions in the index. The key name dictates which partition the key is stored in."

This is a placement rule. The mapping is not `hash(key) mod N`, and no AWS source describes a hash of the key deciding the partition. The key's own bytes decide it, in sorted order, which is why a `ListObjects` call with a prefix is a range scan over an ordered structure rather than a scatter-gather over hash buckets.

The trade-off is worth stating plainly, because it explains every hot-partition symptom you will ever see on S3:

Placement scheme	Prefix listing	Load distribution	Where it breaks
Ordered range partitioning (S3's documented scheme)	Native. A prefix scan is one range read.	Emergent, from key diversity and automatic splitting	Low-entropy prefixes: one partition absorbs the writes
`hash(key) mod N`	Impossible without a full scan	Uniform by construction	Adding or removing a node remaps almost every key
Consistent hashing	Impossible without a full scan	Uniform, with bounded remapping	No lexicographic order, so no prefix listing

S3 chose the first row. Prefix listing and uniform spread are alternatives, not complements.



Load concentrates at one edge of an ordered range. Keys that share a prefix are adjacent, so one partition absorbs the writes.

The 2005 S3 patent describes a prefix-indexed keymap

Amazon's original S3 patent, US 7,716,180 (provisional filed 29 December 2005, granted 11 May 2010), calls the object index a keymap and describes keymap instances that "index entries according to prefixes of their corresponding keys." The application was filed on 8 March 2006, six days before S3 launched. It is the earliest public description of the index, seven years before the 2012 post.

The patent's worked example makes the ordering concrete. Instead of a balanced index that splits 600 entries into three groups of 200, it defines groups by key range: keys starting with a to l, keys starting with m to x, and keys starting with y, z or a digit. Group membership is "a function of its corresponding key," not of how many entries a group holds, and the index is a trie whose nodes are key prefixes such as "al", "ali" and "alic".

Hashing does appear in the patent, in a different layer. Keymap data is split into partitions held by storage units called bricks, replicated three times in the illustrated embodiment, and a separate keymap cache may be "indexed by hashes of keys" to spread cache load across hosts. The patent hashes keys to distribute a cache, not to place index entries.

Read the patent as design intent, not as a specification of today's S3. Every mechanism in it is phrased as one possible embodiment, and AWS has never said which of them shipped or survived. What it adds is independent agreement: the 2005 design, the 2012 post and the 2017 Developer Guide all describe placement decided by the key's own characters, in order.

Which Amazon patents describe the S3 index?

Amazon Technologies, Inc. holds eight patents or groups of related patents, filed between 2005 and 2021, whose text describes an object index, its consistency or how object data is stored, in terms that match the public record of S3. Read together, they repeat one pattern: index entries are placed and split by key prefix, in key order, and hashing spreads load across cache nodes. AWS has not said that S3 implements any of them.

Patent	Priority / granted	What the text describes	Why it matters here
US 7,716,180 B2, Distributed storage system with web services client interface	2005 / 2010	A web-services storage system whose keymap indexes entries by key prefix, splits keymap data into partitions on replicated bricks, uses quorum reads and writes, and hashes keys only to spread a keymap cache.	The earliest public description of ordered, prefix-based placement. Continuations of the same application were granted until 2024 (US 11,895,188 B2).
US 7,702,640 B1, Stratified unbalanced trees for indexing of data items	2005 / 2010	Hierarchical index trees in which every ancestor node's tag is a prefix of the key it leads to, with a keymap for a web-based object store as the example application.	The trie behind the keymap's prefix index, sharing the original application's 29 December 2005 priority date.
US 8,688,912 B2, Management of object mapping information	2009 / 2014	Keymap coordinators that cache mapping information and put a placeholder in the cache while a change to an entry is in flight.	Cache coherence on the keymap path, the layer AWS later named as the main source of stale reads.
US 8,468,134 B1 and US 8,396,840 B1, measuring and improving consistency	2010 / 2013	Bloom filters assigned to time periods that detect stale reads of replicated objects, then target the least consistent objects for repair.	Consistency treated as a measured property a decade before the December 2020 change.
US 10,528,537 B2 and US 11,126,605 B2, versioned keymap entries	2010 / 2020 and 2021	Composite keys made of a user key plus a version ID; a keymap cache in a distributed hash table routed by consistent hashing of the user-key part, so all versions of a key land together; listings in lexicographic key order, then by version.	Hashing again routes cache traffic, while listing order stays lexicographic.
US 11,741,078 B1, Witness service for ensuring data consistency, with US 12,032,562 B1, US 11,704,033 B1 and US 11,537,516 B1	2021 / 2022 to 2024	A transaction sequence witness that tells a storage node whether its copy is stale; partitions assigned to the longest matching key prefix and split by measured request rate; cells defined by key ranges; hot-key cache reads spread over several nodes.	The closest match to the witness AWS described in 2021 and to automatic prefix splitting. Filed ten months after strong consistency launched.
US 12,436,709 B1, Persistent object storage with sequential updates	2021 / 2025	A key-value store for object data on drives written extent by extent, including shingled magnetic recording drives.	Its inventors include five authors of the 2021 paper on ShardStore, the S3 storage node.
US 11,356,120 B1, Hierarchical erasure coding for multi-region storage	2020 / 2022	Two levels of erasure coding: shards spread across fault domains, each shard re-encoded inside its domain, and a manifest to rebuild the object.	Below the index: how object bytes, not index entries, survive the loss of a fault domain such as a region.

Priority and grant years from Google Patents, checked on 13 November 2026. Each mechanism is one embodiment; AWS has not confirmed any of them as part of S3.

This is a reading list, not a legal analysis: I did not assess claims, scope or validity. Two patents that come up in searches for S3 fail the test above. US 10,621,055 B2 (priority 2017) covers failure recovery in clusters of network-attachable data transfer devices, tamper-proof storage appliances rather than the S3 service. US 2015/0379009 A1 is a Cohesity application for the metadata store of a distributed file system and does not mention Amazon or S3.

Why ordered partitioning creates hot partitions

Ordered partitioning creates hot partitions because keys that share a long prefix are adjacent in the index and land in the same partition. A timestamp, a sequence number or an alphabetical prefix writes every new key at the same edge of the ordered space, so one partition absorbs the incoming rate while the partitions holding older keys go cold.

The 2017 Developer Guide names the failure mode directly: a sequential prefix "increases the likelihood that Amazon S3 will target a specific partition for a large number of your keys, overwhelming the I/O capacity of the partition."

The 2012 post adds a second-order cost that is easy to miss. When new content keeps arriving at one edge of the keyspace, "all the partitions holding slightly older (and generally less 'hot') content get cold much faster than other naming conventions, effectively wasting the available operations per second that each partition can support." A date-first layout is the canonical example. A key like `logs/2026-11-13/app-01/...` puts every writer for every application into the partition that owns today's date, and abandons the partition that owns yesterday's.

This is a property of order, not a defect that was fixed. Any system that keeps keys sorted and lets a customer choose the sort key inherits it. That includes the object stores you run yourself: the same date-first prefix that concentrates requests on S3 concentrates directory entries, inodes and metadata transactions on a MinIO or Ceph cluster. I cover that mapping to self-hosted stores at the end of this article.

Splitting a partition is not instant and not free

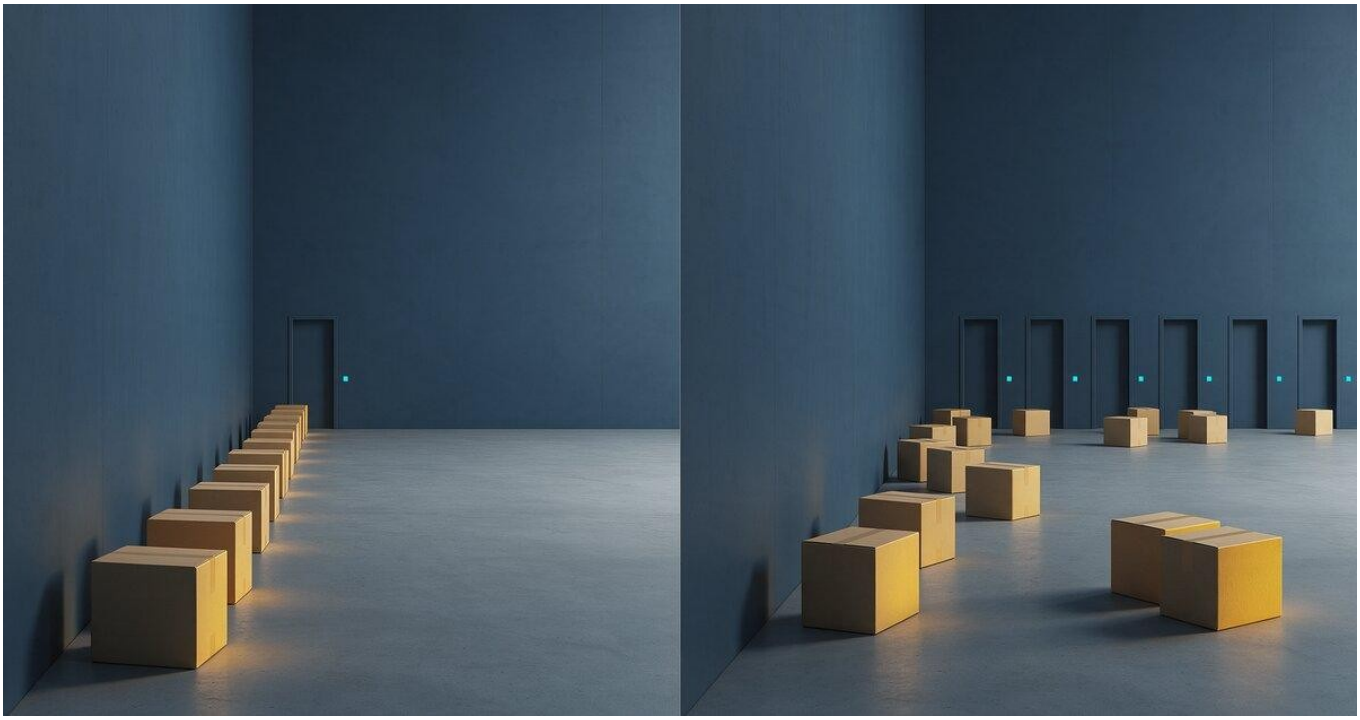
Automatic splitting reduces a hot partition's load, but it is not a synchronous fix and it costs something while it runs. AWS's 2012 account is explicit that there "is overhead in moving keys into newly created partitions," and that the moment "request rates significantly increase on a single partition, partition splits become detrimental to request performance."

The current documentation describes the same window in customer terms: S3 "automatically scales in response to sustained new request rates," and while it is "internally optimizing for a new request rate, you will receive HTTP 503 request responses temporarily until the optimization completes."

Two consequences follow for anyone designing a namespace:

- **Bursts show up as 503s.** A burst that outruns the splitting machinery returns `503 Slow Down`. The documented mitigation is exponential backoff with jitter plus a gradual ramp, not a retry storm.
- **Date-first splits expire.** In the re:Invent 2025 session STG335, AWS engineers said splits walk the key from left to right, so partitions built for one day's date prefix do nothing for the next day's. That is a talk, not a document, but it matches the 2012 cold-partition warning above.

I keep the 2012 date in view deliberately. The idea that S3 introduced automatic partitioning in 2018 is one of the most durable pieces of folklore about the service, and Part 2 shows what the 2012 post says about it.



The same work through one doorway or through many. On S3 the doorway is the partition your key prefix selects.

The index stores names, not object bytes

The index stores names and locations, not object bytes. Object data is erasure-coded into shards spread across the fleet and written by ShardStore, which AWS presented as a validated key-value storage node rather than as an index component. That distinction matters because "shard" in S3's public material means a fragment of object data.

No AWS document reviewed for this series describes index sharding, an index shard count, or a partition function, and I treat any specific number of index shards as invention.

It is also worth separating the index from the metadata caches in front of it. The December 2020 consistency change was widely reported as an index rebuild. What AWS actually described, in Werner Vogels's April 2021 post and in the same re:Invent session cited above, is a coherence problem in the front-end metadata cache. That story belongs to Part 2; the point here is only that "the index" and "the thing that serves your read" are not the same component.

What is not published

Several questions that a deep dive on the S3 index would naturally answer are not answered in public. Listing them is more useful than filling them with plausible engineering, because a wrong internal model leads to wrong sizing and wrong naming decisions. What no source reviewed for this series states:

- **The partition function.** Ordering is documented and the key decides the partition, but the function that maps a key onto a partition is not. Anyone who tells you S3 hashes keys across index shards is guessing. Amazon patents describe a prefix trie (2005) and assignment of each key to the partition with the longest matching prefix (2021). Both agree with the documented ordering; neither is a statement about the current service.
- **Split granularity and rebalancing.** How large a partition is, exactly when a split is triggered, and how index partitions are moved or rebalanced across hosts are all unpublished. A 2021 Amazon patent splits a partition at the prefix that divides its measured request rate as close to 50/50 as possible, which is design intent, not a published S3 parameter.

- **A single hot key's index entry.** Hot prefixes are documented and handled automatically. Whether one key's entry can be split or spread is not described, even though sustained rates to a small number of objects are called out as a 503 risk. A 2021 Amazon patent spreads reads of a hot key across several cache nodes, which addresses the cache, not the index entry.
- **Where the 3,500 and 5,500 figures are enforced.** AWS states them as a minimum ("at least") per partitioned prefix. Whether they are admission control per partition, per prefix, or both, is not described.
- **Whether 2018 changed the mapping or only the scaling rate.** The announcement describes an external effect, not an internal change. The stronger reading, that the mapping was already ordered and already self-splitting, is supported by the 2012 and 2017 sources, traced in Part 2, but is not stated by AWS as a change.

What this means if you run MinIO or Ceph

If you run MinIO or Ceph instead of S3, the mechanism differs but the lesson transfers directly: key naming decides how work spreads across whatever component holds your namespace. On a self-hosted store that component is not an AWS index, it is your metadata path and your filesystem layout, and the same date-first prefix that concentrates writes on S3 concentrates directory entries and inode pressure on your hardware.

This is the part of the S3 story I use most in reviews, because the failure looks identical from the client side and the fix is on your side of the API. If your object store is slow on writes that all arrive under one prefix, or if a filesystem reports inode exhaustion while capacity looks fine, the naming scheme is the first thing I check, before the hardware quote and before the erasure-coding profile. Two posts go deeper on the self-hosted side: inode accounting and prefix design on MinIO and XFS, and the cluster-level guide to object storage for data platforms at scale.

FAQ

Does S3 hash keys to decide the partition?

No. The documented rule is ordered range partitioning over the `bucketname/keyname` string, in UTF-8 binary order, with the key name dictating the partition. Hashing appears in AWS guidance only as a customer-side naming technique, where a hash prefix is prepended to the key so that keys with a shared logical prefix stop being adjacent. Amazon's patents point the same way: from 2005 to 2021 they place index entries by key prefix and use hashing only to spread load across cache nodes.

Why does ListObjects return keys in order?

Because the index is ordered. Keys are stored in UTF-8 binary order across the index partitions, so a listing is a range scan and returns keys in that same order. This is the property a hash-based placement function would destroy, and it is why S3 can offer lexicographic listing and prefix filtering at all.

Is a hot prefix a capacity problem or a request-rate problem?

It is a request-rate problem on one partition, not a capacity problem for the bucket. A bucket can hold effectively unlimited keys while a single prefix sustains a write rate that one partition cannot serve. That is why the symptoms are throttling and latency rather than a full bucket, and why the fix is distributing the keys, not adding storage.

If I run MinIO or Ceph, do I need to salt prefixes?

Start by measuring, not by salting. Salting buys spread at the cost of listing order and application complexity, and it is worth it only when one prefix is measurably saturating a component in your own stack. On self-hosted stores the constraining component is often the metadata path or the filesystem rather than a partition, so identify which one you are hitting before changing your key scheme.

Sources

- A. H. Vermeulen et al., US 7,716,180 B2, Distributed storage system with web services client interface, Amazon Technologies, Inc.; provisional filed 29 December 2005, application filed 8 March 2006, granted 11 May 2010 (the keymap, prefix-indexed unbalanced tries, keymap partitions among bricks, quorum protocols, the hash-indexed keymap cache). A patent describes possible embodiments; AWS has not stated that S3 implements this one. Figure reference numerals are omitted from quotations.
- W. Vogels, Diving Deep on S3 Consistency, All Things Distributed, 20 April 2021 (the metadata cache as "the primary source of S3's eventual consistency," keeping the cache, replication logic that orders operations per object, the witness component as a read barrier).
- A. H. Vermeulen, A. K. Fischman, US 7,702,640 B1, Stratified unbalanced trees for indexing of data items within a computer system, Amazon Technologies, Inc.; priority 29 December 2005, granted 20 April 2010 (prefix-tagged index trees, the keymap example).
- J. C. Sorenson III et al., US 8,688,912 B2, Management of object mapping information corresponding to a distributed storage system, Amazon Technologies, Inc.; priority 31 July 2009, granted 1 April 2014 (keymap coordinators, cache placeholders).
- J. G. McHugh et al., US 8,468,134 B1 and US 8,396,840 B1, Amazon Technologies, Inc.; filed September 2010, granted 2013 (measuring consistency with time-bucketed Bloom filters, targeted consistency improvement).
- J. G. McHugh et al., US 10,528,537 B2 and US 11,126,605 B2, Amazon Technologies, Inc.; priority 23 December 2010, granted 2020 and 2021 (composite keys, consistent hashing of the user-key part for the keymap cache, lexicographic listing order).
- V. Narendra et al., US 11,741,078 B1, Witness service for ensuring data consistency in a distributed storage system, Amazon Technologies, Inc.; filed 30 September 2021, granted 29 August 2023 (the transaction sequence witness, partitions by longest matching key prefix, request-rate splits). Related filings of the same date: US 12,032,562 B1, US 11,704,033 B1 and US 11,537,516 B1 (hot-key reads spread across cache nodes). A patent describes possible embodiments; AWS has not stated that S3 implements this one.
- T. G. Slatton et al., US 12,436,709 B1, Persistent object storage with sequential updates, Amazon Technologies, Inc.; filed 30 June 2021, granted 7 October 2025 (extent-based key-value store, shingled magnetic recording drives).
- S. W. Markle et al., US 11,356,120 B1, Hierarchical erasure coding for multi-region storage, Amazon Technologies, Inc.; filed 15 September 2020, granted 7 June 2022 (two-level erasure coding across fault domains, reconstruction manifest).
- J. Bornholt et al., Using lightweight formal methods to validate a key-value storage node in Amazon S3, SOSP 2021 (author list used to match the inventors of US 12,436,709 B1).
- D. Grismore, Amazon S3 Performance Tips & Tricks, AWS News Blog, March 2012 (the keymap, `bucketname/keyname`, partitioning by prefix, split automation, cold-partition waste, split overhead).
- Amazon S3 Developer Guide, Request Rate and Performance Considerations, snapshot of 5 July 2017 (UTF-8 binary ordering across index partitions, the key name dictating the partition, the

sequential-prefix warning).

- Amazon S3 User Guide, Performance design patterns for Amazon S3 and Performance guidelines for Amazon S3 (automatic scaling, temporary HTTP 503 responses, distributing requests across prefixes).
- AWS News Blog, Twenty years of Amazon S3 and building what's next, 13 March 2026 (S3 launched on 14 March 2006).
- AWS Storage Blog, How automated reasoning helps Amazon S3 innovate at scale, 20 October 2021 (over 300 microservices; ShardStore as a validated storage node).
- J. Bornholt et al., Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3, SOSP 2021.
- AWS re:Invent 2025, Deep dive on Amazon S3 (STG407), AWS S3 engineers. Cited for the index versus storage request volume and for the December 2020 cache-coherence account; the session recording is the only source for both.
- AWS re:Invent 2025, Amazon S3 performance: architecture, design, and optimization (STG335), AWS S3 engineers. Cited for the three-component description, the index as "a very very large distributed key value store", and splits walking the key from left to right; the recording is the only source for these.
- M. Brooker, Exponential Backoff and Jitter, AWS Architecture Blog, 4 March 2015.

Reviewed against AWS documentation, AWS announcements, the linked recordings and the patent text on 13 November 2026. Where a claim rests on a conference session rather than a document, the text and the source list say so. This article explains documented behavior; it is not a benchmark, and I performed no test on S3 for it. I have no affiliation with any vendor cited here, and none of them reviewed or endorsed this article.

Related posts

- Object storage for data platforms at scale (the guide this series belongs to)
- MinIO on XFS: inode exhaustion and prefix design
- Why file count matters as much as file size

If you are sizing an object-storage platform, choosing a namespace scheme or checking whether a naming convention is already costing you throughput, I cover those questions in MinIO, S3 and object-storage engagements. For a wider review, see the data infrastructure audit or book a free 15-minute intro call.

PDF snapshot of the public article. The online version is authoritative; links remain clickable.