Best Practices Design Patterns: Optimizing Amazon S3 Performance

AWS Whitepaper

Best Practices Design Patterns: Optimizing Amazon S3 Performance: AWS Whitepaper

Table of Contents

Best Practices Design Patterns: Optimizing Amazon S3 Performance

Initial publication date: **June 2019** (*Document Revisions* (p. 10))

Abstract

When building applications that upload and retrieve storage from Amazon S3, follow the AWS best practices guidelines to optimize performance. AWS also offers more detailed Performance Design Patterns.

Introduction

Your applications can easily achieve thousands of transactions per second in request performance when uploading and retrieving storage from Amazon S3. Amazon S3 automatically scales to high request rates. For example, your application can achieve at least 3,500 `PUT/COPY/POST/DELETE` and 5,500 `GET/HEAD` requests per second per prefix in a bucket. There are no limits to the number of prefixes in a bucket. You can increase your read or write performance by parallelizing reads. For example, if you create 10 prefixes in an Amazon S3 bucket to parallelize reads, you could scale your read performance to 55,000 read requests per second.

Some data lake applications on Amazon S3 scan many millions or billions of objects for queries that run over petabytes of data. These data lake applications achieve single- instance transfer rates that maximize the network interface use for their Amazon EC2 instance, which can be up to 100 Gb/s on a single instance. These applications then aggregate throughput across multiple instances to get multiple terabits per second.

Other applications are sensitive to latency, such as social media messaging applications. These applications can achieve consistent small object latencies (and first- byte-out latencies for larger objects) of roughly 100–200 milliseconds.

Other AWS services can also help accelerate performance for different application architectures. For example, if you want higher transfer rates over a single HTTP connection or single-digit millisecond latencies, use Amazon CloudFront or Amazon ElastiCache for caching with Amazon S3.

Additionally, if you want fast data transport over long distances between a client and an S3 bucket, use Amazon S3 Transfer Acceleration. Transfer Acceleration uses the globally distributed edge locations in CloudFront to accelerate data transport over geographical distances.

If your Amazon S3 workload uses server-side encryption with AWS Key Management Service (SSE-KMS), see AWS KMS Limits in the *AWS Key Management Service Developer Guide* for information about the request rates supported for your use case.

The following topics describe best practice guidelines and design patterns for optimizing performance for applications that use Amazon S3.

This guidance supersedes any previous guidance on optimizing performance for Amazon S3. For example, previously Amazon S3 performance guidelines recommended randomizing prefix naming with hashed characters to optimize performance for frequent data retrievals. You no longer have to randomize prefix naming for performance, and can use sequential date-based naming for your prefixes. Refer to *Performance Guidelines* and *Performance Design Patterns* for the most current information about performance optimization for Amazon S3.

Performance Guidelines for Amazon S3

To obtain the best performance for your application on Amazon S3, AWS recommends the following guidelines.

Topics

Measure Performance

When optimizing performance, look at network throughput, CPU, and Dynamic Random Access Memory (DRAM) requirements. Depending on the mix of demands for these different resources, it might be worth evaluating different Amazon EC2 instance types. For more information about instance types, see Instance Types in the *Amazon EC2 User Guide for Linux Instances*.

It's also helpful to look at DNS lookup time, latency, and data transfer speed using HTTP analysis tools when measuring performance.

Scale Storage Connections Horizontally

Spreading requests across many connections is a common design pattern to horizontally scale performance. When you build high performance applications, think of Amazon S3 as a very large distributed system, not as a single network endpoint like a traditional storage server. You can achieve the best performance by issuing multiple concurrent requests to Amazon S3. Spread these requests over separate connections to maximize the accessible bandwidth from Amazon S3. Amazon S3 doesn't have any limits for the number of connections made to your bucket.

Use Byte-Range Fetches

Using the Range HTTP header in a GET Object request, you can fetch a byte-range from an object, transferring only the specified portion. You can use concurrent connections to Amazon S3 to fetch different byte ranges from within the same object. This helps you achieve higher aggregate throughput versus a single whole-object request. Fetching smaller ranges of a large object also allows your application to improve retry times when requests are interrupted. For more information, see Getting Objects.

Typical sizes for byte-range requests are 8 MB or 16 MB. If objects are PUT using a multipart upload, it's a good practice to GET them in the same part sizes (or at least aligned to part boundaries) for best performance. GET requests can directly address individual parts; for example, `GET ?partNumber=N`.

Retry Requests for Latency-Sensitive Applications

Aggressive timeouts and retries help drive consistent latency. Given the large scale of Amazon S3, if the first request is slow, a retried request is likely to take a different path and quickly succeed. The AWS SDKs have configurable timeout and retry values that you can tune to the tolerances of your specific application.

Combine Amazon S3 (Storage) and Amazon EC2 (Compute) in the Same AWS Region

Although S3 bucket names are globally unique, each bucket is stored in a Region that you select when you create the bucket. To optimize performance, we recommend that you access the bucket from Amazon EC2 instances in the same AWS Region when possible. This helps reduce network latency and data transfer costs.

For more information about data transfer costs, see Amazon S3 Pricing.

Use Amazon S3 Transfer Acceleration to Minimize Latency Caused by Distance

Amazon S3 Transfer Acceleration manages fast, easy, and secure transfers of files over long geographic distances between the client and an S3 bucket. Transfer Acceleration takes advantage of the globally distributed edge locations in Amazon CloudFront. As the data arrives at an edge location, it is routed to Amazon S3 over an optimized network path. Transfer Acceleration is ideal for transferring gigabytes to terabytes of data regularly across continents. It's also useful for clients that upload to a centralized bucket from all over the world.

You can use the Amazon S3 Transfer Acceleration Speed Comparison tool to compare accelerated and non-accelerated upload speeds across Amazon S3 Regions. The Speed Comparison tool uses multipart uploads to transfer a file from your browser to various Amazon S3 Regions with and without using Amazon S3 Transfer Acceleration.

Use the Latest Version of the AWS SDKs

The AWS SDKs provide built-in support for many of the recommended guidelines for optimizing Amazon S3 performance. The SDKs provide a simpler API for taking advantage of Amazon S3 from within an application and are regularly updated to follow the latest best practices. For example, the SDKs include logic to automatically retry requests on HTTP 503 errors and are investing in code to respond and adapt to slow connections.

The SDKs also provide the Transfer Manager, which automates horizontally scaling connections to achieve thousands of requests per second, using byte-range requests where appropriate. It's important to use the latest version of the AWS SDKs to obtain the latest performance optimization features.

You can also optimize performance when you are using HTTP REST API requests. When using the REST API, you should follow the same best practices that are part of the SDKs. Allow for timeouts and retries on slow requests, and multiple connections to allow fetching of object data in parallel. For information about using the REST API, see the Amazon Simple Storage Service API Reference.

Performance Design Patterns for Amazon S3

When designing applications to upload and retrieve storage from Amazon S3, use our best practices design patterns for achieving the best performance for your application. We also offer Performance Guidelines for you to consider when planning your application architecture.

To optimize performance, you can use the following design patterns.

Topics

Using Caching for Frequently Accessed Content

Many applications that store data in Amazon S3 serve a "working set" of data that is repeatedly requested by users. If a workload is sending repeated GET requests for a common set of objects, you can use a cache such as Amazon CloudFront, Amazon ElastiCache, or AWS Elemental MediaStore to optimize performance. Successful cache adoption can result in low latency and high data transfer rates. Applications that use caching also send fewer direct requests to Amazon S3, which can help reduce request costs.

Amazon CloudFront is a fast content delivery network (CDN) that transparently caches data from Amazon S3 in a large set of geographically distributed points of presence (PoPs). When objects might be accessed from multiple Regions, or over the internet, CloudFront allows data to be cached close to the users that are accessing the objects. This can result in high performance delivery of popular Amazon S3 content. For information about CloudFront, see the Amazon CloudFront Developer Guide.

Amazon ElastiCache is a managed, in-memory cache. With ElastiCache, you can provision Amazon EC2 instances that cache objects in memory. This caching results in orders of magnitude reduction in GET latency and substantial increases in download throughput. To use ElastiCache, you modify application logic to both populate the cache with hot objects and check the cache for hot objects before requesting them from Amazon S3. For examples of using ElastiCache to improve Amazon S3 GET performance, see the blog post Turbocharge Amazon S3 with Amazon ElastiCache for Redis.

AWS Elemental MediaStore is a caching and content distribution system specifically built for video workflows and media delivery from Amazon S3. MediaStore provides end-to-end storage APIs specifically for video, and is recommended for performance- sensitive video workloads. For information about MediaStore, see the AWS Elemental MediaStore User Guide.

Timeouts and Retries for Latency-Sensitive Applications

There are certain situations where an application receives a response from Amazon S3 indicating that a retry is necessary. Amazon S3 maps bucket and object names to the object data associated with them. If

an application generates high request rates (typically sustained rates of over 5,000 requests per second to a small number of objects), it might receive HTTP 503 *slowdown* responses. If these errors occur, each AWS SDK implements automatic retry logic using exponential backoff. If you are not using an AWS SDK, you should implement retry logic when receiving the HTTP 503 error. For information about back-off techniques, see Error Retries and Exponential Backoff in AWS in the *Amazon Web Services General Reference*.

Amazon S3 automatically scales in response to sustained new request rates, dynamically optimizing performance. While Amazon S3 is internally optimizing for a new request rate, you will receive HTTP 503 request responses temporarily until the optimization completes. After Amazon S3 internally optimizes performance for the new request rate, all requests are generally served without retries.

For latency-sensitive applications, Amazon S3 advises tracking and aggressively retrying slower operations. When you retry a request, we recommend using a new connection to Amazon S3 and performing a fresh DNS lookup.

When you make large variably sized requests (for example, more than 128 MB), we advise tracking the throughput being achieved and retrying the slowest 5 percent of the requests. When you make smaller requests (for example, less than 512 KB), where median latencies are often in the tens of milliseconds range, a good guideline is to retry a GET or PUT operation after 2 seconds. If additional retries are needed, the best practice is to back off. For example, we recommend issuing one retry after 2 seconds and a second retry after an additional 4 seconds.

If your application makes fixed-size requests to Amazon S3, you should expect more consistent response times for each of these requests. In this case, a simple strategy is to identify the slowest 1 percent of requests and to retry them. Even a single retry is frequently effective at reducing latency.

If you are using AWS Key Management Service (AWS KMS) for server-side encryption, see Quotas in the *AWS Key Management Service Developer Guide* for information about the request rates that are supported for your use case.

Horizontal Scaling and Request Parallelization for High Throughput

Amazon S3 is a very large distributed system. To help you take advantage of its scale, we encourage you to horizontally scale parallel requests to the Amazon S3 service endpoints. In addition to distributing the requests within Amazon S3, this type of scaling approach helps distribute the load over multiple paths through the network.

For high-throughput transfers, Amazon S3 advises using applications that use multiple connections to GET or PUT data in parallel. For example, this is supported by Amazon S3 Transfer Manager in the AWS Java SDK, and most of the other AWS SDKs provide similar constructs. For some applications, you can achieve parallel connections by launching multiple requests concurrently in different application threads, or in different application instances. The best approach to take depends on your application and the structure of the objects that you are accessing.

You can use the AWS SDKs to issue GET and PUT requests directly rather than employing the management of transfers in the AWS SDK. This approach lets you tune your workload more directly, while still benefiting from the SDK's support for retries and its handling of any HTTP 503 responses that might occur. As a general rule, when you download large objects within a Region from Amazon S3 to Amazon EC2, we suggest making concurrent requests for byte ranges of an object at the granularity of 8–16 MB. Make one concurrent request for each 85–90 MB/s of desired network throughput. To saturate a 10 Gb/s network interface card (NIC), you might use about 15 concurrent requests over separate connections. You can scale up the concurrent requests over more connections to saturate faster NICs, such as 25 Gb/s or 100 Gb/s NICs.

Measuring performance is important when you tune the number of requests to issue concurrently. We recommend starting with a single request at a time. Measure the network bandwidth being achieved and the use of other resources that your application uses in processing the data. You can then identify the bottleneck resource (that is, the resource with the highest usage), and hence the number of requests that are likely to be useful. For example, if processing one request at a time leads to a CPU usage of 25 percent, it suggests that up to four concurrent requests can be accommodated.

Measurement is essential, and it is worth confirming resource use as the request rate is increased.

If your application issues requests directly to Amazon S3 using the REST API, we recommend using a pool of HTTP connections and re-using each connection for a series of requests. Avoiding per-request connection setup removes the need to perform TCP slow-start and Secure Sockets Layer (SSL) handshakes on each request. For information about using the REST API, see the Amazon S3 REST API Introduction.

Finally, it's worth paying attention to DNS and double-checking that requests are being spread over a wide pool of Amazon S3 IP addresses. DNS queries for Amazon S3 cycle through a large list of IP endpoints. But caching resolvers or application code that reuses a single IP address do not benefit from address diversity and the load balancing that follows from it. Network utility tools such as the `netstat` command line tool can show the IP addresses being used for communication with Amazon S3, and we provide guidelines for DNS configurations to use. For more information about these guidelines, see Request routing.

Using Amazon S3 Transfer Acceleration to Accelerate Geographically Disparate Data Transfers

Amazon S3 Transfer Acceleration is effective at minimizing or eliminating the latency caused by geographic distance between globally dispersed clients and a regional application using Amazon S3. Transfer Acceleration uses the globally distributed edge locations in CloudFront for data transport. The AWS edge network has points of presence in more than 50 locations. Today, it is used to distribute content through CloudFront and to provide rapid responses to DNS queries made to Amazon Route 53.

The edge network also helps to accelerate data transfers into and out of Amazon S3. It is ideal for applications that transfer data across or between continents, have a fast internet connection, use large objects, or have a lot of content to upload. As the data arrives at an edge location, data is routed to Amazon S3 over an optimized network path. In general, the farther away you are from an Amazon S3 Region, the higher the speed improvement you can expect from using Transfer Acceleration.

You can set up Transfer Acceleration on new or existing buckets. You can use a separate Amazon S3 Transfer Acceleration endpoint to use the AWS edge locations. The best way to test whether Transfer Acceleration helps client request performance is to use the Amazon S3 Transfer Acceleration Speed Comparison tool. Network configurations and conditions vary from time to time and from location to location. So, you are charged only for transfers where Amazon S3 Transfer Acceleration can potentially improve your upload performance. For information about using Transfer Acceleration with different AWS SDKs, see Amazon S3 Transfer Acceleration Examples.

Contributors

Contributors to this document include:

- Mai-Lan Tomsen Bukovec, VP, Amazon S3
- Andy Warfield, Senior Principal Engineer, Amazon S3
- Tim Harris, Principal Engineer, Amazon S3

Document Revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

update-history-change	update-history-description	update-history-date
Updated (p. 10)	Reviewed for technical accuracy	March 10, 2021
Initial publication (p. 10)	Initial publication	June 1, 2019

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.