Technical reports

How to enable StorageGRID in your environment

NetApp
April 26, 2024

Table of Contents

Technical reports

NetApp StorageGRID and big data analytics

NetApp StorageGRID use cases

NetApp StorageGRID object storage solution offers scalability, data availability, security, and high performance. Organizations of all sizes and across various industries use StorageGRID S3 for a wide range of use cases. Let's explore some typical scenarios:

Big data analytics: StorageGRID S3 is frequently used as a data lake, where businesses store large amounts of structured and unstructured data for analysis using tools like Apache Spark, Splunk Smartstore and Dremio.

Data Tiering: NetApp customers use ONTAP's FabricPool feature to automatically move data between a high-performance local tier to StorageGRID. Tiering frees up expensive flash storage for hot data while keeping cold data readily available on low-cost object storage. This maximizes performance and savings.

Data backup and disaster recovery: Businesses can use StorageGRID S3 as a reliable and cost-effective solution for backing up critical data and recovering it in case of a disaster.

Data storage for applications: StorageGRID S3 can be used as a storage backend for applications, enabling developers to easily store and retrieve files, images, videos, and other types of data.

Content delivery: StorageGRID S3 can be used to store and deliver static website content, media files, and software downloads to users around the world, leveraging StorageGRID's geo distribution and global namespace for fast and reliable content delivery.

Data Tiering: NetApp customers use ONTAP FabricPool feature to automatically move data between a high-performance local tier to StorageGRID. Tiering frees up expensive flash storage for hot data while keep cold data readily available from low-cost object storage. This maximizes performance and savings.

Data Archive: StorageGRID offers different storage types and supports tiering to public long term low-cost storage options, make it an ideal solution for archiving and long-term retention of data that needs to be retained for compliance or historical purposes.

Object storage use cases

[StorageGRID use case diagram]

Among the above, big data analytics is one of the topmost use cases and its usage is trending upward.

Why StorageGRID for data lakes?

- Increased collaboration - Massive shared multi-site, multi-tenancy w/industry standard API access
- Decreased operational costs - Operational simplicity of a single, self-healing, automated scale-out architecture
- Scalability - Unlike traditional Hadoop and data warehouse solutions, StorageGRID S3 object storage decouples storage from compute and data, allowing business to scale their storage needs as they grew.
- Durability and reliability - StorageGRID provides 99.999999999% durability, meaning that data stored is highly resistant to data loss. It also offers high availability, ensuring that data is always accessible.
- Security - StorageGRID offers various security features, including encryption, access control policy, data

lifecycle management, object lock and versioning to protect data stored in S3 buckets

StorageGRID S3 Data Lakes

[StorageGRID datalake example]

Which data warehouse or data lake works best with S3 object storage

NetApp benchmarked StorageGRID with three data warehouse/lake house ecosystems - Hive, Delta Lake and Dremio. Apache Iceberg: The Definitive Guide includes brief introduction of data warehouse and data lake house and pro/cons of these two architectures.

- Benchmark Tool - TPC-DS - https://www.tpc.org/tpcds/
- Big data ecosystems
  - Cluster of 5 VMs, each with 128G RAM and 24 vCPU, SSD storage for system disk
  - Hadoop 3.3.5 with Hive 3.1.3 (1 name node + 4 data nodes)
  - Delta Lake with Spark 3.2.0 (1 master + 4 workers) and Hadoop 3.3.5
  - Dremio v23 (1 master + 4 executors)
- Object storage
  - NetApp® StorageGRID® 11.6 with 3 x SG6060 + 1x SG1000 load balancer
  - Object protection - 2 copies
- Database size 1000GB
- Cache disabled on all 3 ecosystems to get consistent result for each query test.

TPC-DS comes with 99 complex SQL queries for query benchmarking. We measured the total minutes to complete all 99 queries and we dove deeper by breaking down the type and number of S3 requests to analyze the result. The first table below shows the total duration of all 99 queries and the second table summarizes the number and types of S3 requests each ecosystem sent to StorageGRID.

TPC-DS query result

Ecosystem	Hive	Delta Lake	Dremio
Storage layer	NetApp® StorageGRID®	NetApp® StorageGRID®	NetApp® StorageGRID®
Drive type	HDD	HDD	HDD
Table format	Parquet	Parquet	Parquet [1]
Database size	1000G	1000G	1000G
TPCDS 99 queries total minutes	1084 [2]	55	47

[1] Tested both Parquet and Iceberg table format, result is similar.

[2] Hive unable to complete query number 72.

TPC-DS queries - S3 requests breakdown

S3 Requests	Hive	Delta Lake	Dremio
GET	1,117,184	2,074,610	4,414,227
observation: all range GET	80% range get of 2KB to 2MB from 32MB objects, 50 - 100 requests/sec	73% range get below 100KB from 32MB objects, 1000 - 1400 requests/sec	90% 1M byte range get from 256MB objects, 2000 - 2300 requests/sec
List objects	312,053	24,158	240
HEAD (non-existent object)	156,027	12,103	192
HEAD (existent object)	982,126	922,732	1,845
Total requests	2,567,390	3,033,603	4,416,504

From the first table, we can see Delta Lake and Dremio are much faster than Hive. From the second table, we notice that Hive sent lots of S3 list-objects requests which is typically slow in all object storage platforms, especially if dealing with a bucket containing many objects. This increases overall query duration significantly. Another observation is Dremio was able to send high number of GET requests in parallel, 2,000 to 2,300 requests per second versus 50 - 100 requests per second in Hive. Hive and Hadoop S3A mimic standard filesystem contributes to Hive slowness to S3 object storage.

Using Hadoop (either on HDFS or S3 object storage) with Hive or Spark requires extensive knowledge of Hadoop and Hive/Spark and how the settings from each service interact - together they have 1000+ settings. Very often, the settings are inter-related and cannot be changed alone. It takes tremendous amounts of time and effort to find the optimal combination of settings and values to use.

Dremio is a data lake engine that uses end-to-end Apache Arrow to dramatically increase query performance. Apache Arrow provides a standardized columnar memory format for efficient data sharing and fast analytics. Arrow employs a language-agnostic approach, designed to eliminate the need for data serialization and deserialization, improving the performance and interoperability between complex data processes and systems.

Dremio's performance is mostly driven by computing power on the Dremio cluster. Though Dremio uses Hadoop's S3A connector for S3 object storage connection, Hadoop is not required and most of Hadoop's fs.s3a settings are not used by Dremio. This makes tuning Dremio performance easy without spending time to learn and test various Hadoop s3a settings.

From this benchmark result, we can conclude that big data analytic system that optimized for S3-based workload is a major performance factor. Dremio optimizes query execution, efficiently utilizes metadata, and provides seamless access to S3 data, resulting in better performance compared to Hive when working with S3 storage. Refer to this page to configure Dremio S3 data source with StorageGRID.

Visit the links below to learn more about how StorageGRID and Dremio work together to provide a modern and efficient data lake infrastructure and how NetApp migrated from Hive + HDFS to Dremio + StorageGRID to dramatically enhance big data analytic efficiency.

- Boost performance for your big data with NetApp StorageGRID
- Modern, powerful, and efficient data lake infrastructure with StorageGRID and Dremio
- How NetApp is Redefining the Customer Experience with Product Analytics

Hadoop S3A tuning

Hadoop S3A connector facilitates seamless interaction between Hadoop-based applications and S3 object storage. Tuning the Hadoop S3A Connector is essential to optimize performance when working with S3 object storage. Before we go into tuning details, let's have a basic understanding of Hadoop and its components.

What is Hadoop?

Hadoop is a powerful open-source framework designed to handle large-scale data processing and storage. It enables distributed storage and parallel processing across clusters of computers.

The three core components of Hadoop are:

- **Hadoop HDFS (Hadoop Distributed File System)**: This handles storage, breaking data into blocks and distributing them across nodes.
- **Hadoop MapReduce**: Responsible for processing data by dividing tasks into smaller chunks and executing them in parallel.
- **Hadoop YARN (Yet Another Resource Negotiator):** Manages resources and schedules tasks efficiently

Hadoop HDFS and S3A connector

HDFS is a vital component of the Hadoop ecosystem, playing a critical role in efficient big data processing. HDFS enables reliable storage and management. It ensures parallel processing and optimized data storage, resulting in faster data access and analysis.

In big data processing, HDFS excels at providing fault-tolerant storage for large datasets. It achieves this through data replication. It can store and manage large volumes of structured and unstructured data in a data warehouse environment. Moreover, it seamlessly integrates with leading big data processing frameworks, such as Apache Spark, Hive, Pig, and Flink, enabling scalable and efficient data processing. It is compatible with Unix-based (Linux) operating systems, making it an ideal choice for organizations that prefer using Linux-based environments for their big data processing.

As the volume of data has grown over time, the approach of adding new machines to the Hadoop cluster with their own compute and storage has become inefficient. Scaling linearly creates challenges for using resources efficiently and managing the infrastructure.

To address these challenges, the Hadoop S3A connector offers high-performance I/O against S3 object storage. Implementing a Hadoop workflow with S3A helps you leverage object storage as a data repository and enables you to separate compute and storage, which in turn enables you to scale compute and storage independently. Decoupling compute and storage also enable you to dedicate the right amount of resources for your compute jobs and provide capacity based on the size of your data set. Therefore, you can reduce your overall TCO for Hadoop workflows.

Hadoop S3A connector tuning

S3 behaves differently from HDFS, and some attempts to preserve the appearance of a file system are aggressively suboptimal. Careful tuning/testing/experimenting is necessary to make the most efficient use of S3 resources.

Hadoop options in this document are based on Hadoop 3.3.5, refer to Hadoop 3.3.5 core-site.xml for all available options.

Note – the default value of some Hadoop fs.s3a settings are different in each Hadoop version. Be sure to

check out the default value specific to your current Hadoop version. If these settings are not specified in Hadoop core-site.xml, default value will be used. You can override the value at run time using Spark or Hive configuration options.

You must go to this Apache Hadoop page to understand each fs.s3a options. If possible, test them in non-production Hadoop cluster to find the optimal values.

You should read Maximizing Performance when working with the S3A Connector for other tuning recommendations.

Let's explore some key considerations:

1. Data compression

Do not enable StorageGRID compression. Most of big data systems use byte range get instead of retrieving the entire object. Using byte range get with compressed objects degrade the GET performance significantly.

2. S3A committers

In general, magic s3a committer is recommended. Refer to this common S3A committer options page to get a better understanding of magic committer and its related s3a settings.

Magic Committer:

The Magic Committer specifically relies on S3Guard to offer consistent directory listings on the S3 object store.

With consistent S3 (which is now the case), the Magic Committer can be safely used with any S3 bucket.

Choice and Experimentation:

Depending on your use case, you can choose between the Staging Committer (which relies on a cluster HDFS filesystem) and the Magic Committer.

Experiment with both to determine which best suits your workload and requirements.

In summary, the S3A Committers provide a solution to the fundamental challenge of consistent, high-performance, and reliable output commitment to S3. Their internal design ensures efficient data transfer while maintaining data integrity.

[S3A Options Table]

3. Thread, connection pool sizes and block size

- Each **S3A** client interacting with a single bucket has its own dedicated pool of open HTTP 1.1 connections and threads for upload and copy operations.
- You can tune these pool sizes to strike a balance between performance and memory/thread usage.
- When uploading data to S3, it is divided into blocks. The default block size is 32 MB. You can customize this value by setting the fs.s3a.block.size property.
- Larger block sizes can improve performance for large data uploads by reducing the overhead of managing multipart parts during upload. Recommended value is 256 MB or above for large data set.

[S3A Options Table]

4. Multipart upload

s3a committers **always** use MPU (multipart upload) to upload data to s3 bucket. This is needed to allow for: task failure, speculative execution of tasks, and job aborts before commit. Here are some key specifications related to multipart uploads:

- Maximum object size: 5 TiB (terabytes).
- Maximum number of parts per upload: 10,000.
- Part numbers: Ranging from 1 to 10,000 (inclusive).
- Part size: Between 5 MiB and 5 GiB. Notably, there is no minimum size limit for the last part of your multipart upload.

Using a smaller part size for S3 multipart uploads has both advantages and disadvantages.

Advantages:

- Quick Recovery from Network Issues: When you upload smaller parts, the impact of restarting a failed upload due to a network error is minimized. If a part fails, you only need to re-upload that specific part rather than the entire object.
- Better Parallelization: More parts can be uploaded in parallel, taking advantage of multi-threading or concurrent connections. This parallelization enhances performance, especially when dealing with large files.

Disadvantage:

- Network overhead: Smaller part size means more parts to upload, each part requires its own HTTP request. More HTTP requests increase overhead of initiating and completing individual requests. Managing a large number of small parts can impact performance.
- Complexity: Managing the order, tracking parts, and ensuring successful uploads can be cumbersome. If the upload needs aborted, all the parts that already uploaded need to be tracked and purged.

For Hadoop, 256MB or above part size is recommended for fs.s3a.multipart.size. Always set the fs.s3a.mutlipart.threshold value to 2 x fs.s3a.multipart.size value. For example if fs.s3a.multipart.size = 256M, fs.s3a.mutlipart.threshold should be 512M.

Use larger part size for large data set. It is important to choose a part size that balances these factors based on your specific use case and network conditions.

A multipart upload is a three-step process:

1. The upload is initiated, StorageGRID returns an upload-id.
2. The object parts are uploaded using the upload-id.
3. Once all the object parts are uploaded, sends complete multipart upload request with upload-id. StorageGRID constructs the object from the uploaded parts, and client can access the object.

If the complete multipart upload request isn't sent successfully, the parts stay in StorageGRID and will not create any object. This happens when jobs are interrupted, failed, or aborted. The parts remain in the Grid until multipart upload completes or is aborted or StorageGRID purges these parts if 15 days elapsed since upload was initiated. If there are many (few hundreds thousand to millions) in-progress multipart uploads in a bucket, when Hadoop sends 'list-multipart-uploads' (this request does not filter by upload id), the request may take a long time to complete or eventually time out. You may consider set fs.s3a.mutlipart.purge to true with an appropriate fs.s3a.multipart.purge.age value (e.g. 5 to 7 days, do not use default value of 86400 i.e. 1 day). Or engage NetApp support to investigate the situation.

[S3A Options Table]

5. Buffer write data in memory

To enhance performance, you can buffer write data in memory before uploading it to S3. This can reduce the number of small writes and improve efficiency.

[S3A Options Table]

Remember that S3 and HDFS work in distinct ways. Careful tuning/test/experiment is necessary to make the most efficient use of S3 resources.