Analytics Accelerator Library for Amazon S3 and Iceberg

Authors: michstub@amazon.co.uk/fbbasik@amazon.co.uk/ahmarsu@amazon.co.uk
Iceberg Mail Thread: https://lists.apache.org/thread/bo1bqzxnd22sosb4r2cw513nbrvyvwk1
AAL Github: https://github.com/awslabs/analytics-accelerator-s3

Motivation

The purpose of this document is to get community feedback on the Analytics Accelerator Library for S3 being the default stream for S3 in Iceberg going forward. To facilitate this we are providing information on the project's goals and details.

Analytics Accelerator Library (AAL) is an open source library for your client applications that accelerates data access to Amazon S3, lowering processing times and compute costs for data intensive workloads. It does this by providing an implementation of the best practices for accessing data in S3. Currently the integration code is merged in both Hadoop S3A and Iceberg S3FileIO but is behind a feature flag that is defaulted to Off. We would like to propose that AAL is the default Stream for Iceberg.

Our testing shows that it accelerates workloads, and customers such as Voodoo.io report 10% improvement with S3FileIO, so we want users to benefit from this by default. By implementing best practices for S3 in one centralized location, Iceberg will continue to benefit as we make further improvements. Making this the default data path for Iceberg will enable future optimizations to build upon the higher performance baseline we have established.

More technical details will be part of a follow-up document based on the outcome of this proposal.

Goals

- Increase read performance for sequential and parquet reader like access patterns.
- Lower processing times and compute costs for data analytics workloads.
- Implement S3 best practices for performance.

Non-Goals

- Write Performance - The library currently only concerns itself with reading data.
- Non-AWS implementations of the S3 API - This library has been tested and is compatible with MinIO, but our focus for performance best practices has been for AWS S3.

What problems does AAL currently solve

Open Range requests - Currently when S3FileIO opens a stream it makes the following Get Request:

```
GetObjectRequest.Builder requestBuilder =
       GetObjectRequest.builder()
           .bucket(location.bucket())
           .key(location.key())
           .range(String.format("bytes=%s-", pos));
```

This leads to over-reading because there is no end to the range and S3 will fetch unnecessary data. AAL instead only does closed range requests. Because of this on a 3TB dataset, for a workload derived from TPC-DS we observed AAL retrieve 53.8% less bytes from S3.

Read Vectored - Currently S3FileIO makes sequential blocking reads to S3. So if the engine knows that it needs 5 ranges it requests them sequentially, i.e. starting read for range 2 waits for range 1 to finish and range 3 waits for range 2. Without Vectored-IO, parquet-mr makes sequential `read(buf[], offset, len)` calls as follows:

```
1. range [80050-160096], length=80,046
2. range [1092787-1093618], length=831 //start when 1st request is done
3. range [1093823-1094031], length=208 //start when 2nd request is done
4. range [1096018-1096087], length=69 //start when 3rd request is done
5. range [1096222-1096360], length=138 //start when 4th request is done
```

With Read vectored Parquet-mr instead passes a list of ranges and buffers to the Stream:

```
readVectored(List<? extends FileRange> ranges, IntFunction<ByteBuffer> allocate)
```

This is useful as it allows us to initiate the requests to S3 and start populating the buffer simultaneously.

```
1. range [80050-160096], length=80,046
2. range [1092787-1093618], length=831 //start at same time as 1
3. range [1093823-1094031], length=208 //start at same time as 1
4. range [1096018-1096087], length=69 //start at same time as 1
5. range [1096222-1096360], length=138 //start at same time as 1
```

Github Issue: https://github.com/apache/iceberg/issues/13254

Footer Pre-Caching - AAL optimizes Parquet file reading by implementing footer pre-caching. When AAL detects a file with a ".par" or ".parquet" extension, it identifies it as a Parquet file and proactively prefetches and caches the file's footer. This caching mechanism prevents multiple requests to S3 for the footer data. For S3FileIO the current implementation makes 2 requests for the footer: one for the last 8 bytes which gives the start position of the footer, and the second for the footer. AAL makes a single request for the last ~32KB/1MB of the file, depending on the file size, which cuts out those two small GETs.

Sequential prefetching - AAL detects sequential read access patterns for the same object and then scales up its request sizes based on that. So instead of making multiple smaller requests it will instead increase the size of the range at an exponential rate to try and speed up access.

If these optimizations seem like something you want to test with as of Iceberg 1.9.1 you can manually enable AAL with the following config:

```
--conf "spark.sql.catalog.<CATALOG_NAME>.s3.analytics-accelerator.enabled=true"
```

Benchmarking

These are the numbers from our latest benchmarks with AAL 1.2.1, however we are planning on one more release of the library (1.2.2) in the next month and the numbers are subject to improvement, we will update this section once that is completed.

The Amazon S3 team executes benchmarks based on an industry standard benchmark derived from TPC-DS at 3 TB scale using two configurations: a baseline using Iceberg 1.9.1 with AAL disabled, and a treatment configuration using Iceberg 1.9.1 with AAL enabled. The total cost calculation includes both S3 requests and EC2 compute costs. To ensure consistency, all benchmark tests are executed within the same region simultaneously. Our data sets are based off of a 3TB dataset, we have found that the shape of the dataset makes a big impact in performance so we have reshaped the datasets with different file sizes and row groups.

Data Shape	**Baseline Performance (s)**	**Treatment Performance (s)**	**Percentage Improvement**	**Total Cost Improvement**
128MB max file size + partitioned	2407.46833	2192.85167	9%	5%
128MB max file size + unpartitioned	1073.70333	1002.92	7%	5%
1GB max file size + unpartitioned	1529.91833	1379.52	10%	7.31%

AAL avoids doing open ended range requests which helps avoid sending unnecessary bytes from S3 to Compute. Based on data gathered from a single execution of our benchmarks we found that while AAL makes 12.7% more Get Requests (2765905 to 3117803) it retrieves 53.8% less data (15.8TB to 7.3TB of data).

See Appendix 4: Calculations if you want to verify these numbers with your workloads.

Considerations

Why a library and not just build this into S3FileIO directly?

We believe that we can build best practices into a central library while still being flexible and abstract enough to target the wider community. Currently we are also doing work to integrate AAL default ON into S3A and we might want to onboard even more connectors in the future.

What input do we need from the Community?

- Iceberg Community feedback on our open PR's: Appendix 3: Open PRs with Iceberg.
- Iceberg Community testing/feedback with AAL and their workloads by enabling AAL.
- Iceberg Community feedback on next steps/action items for the AAL integration going forward.

What is next for AAL?

- We are currently working on some performance improvements for AAL.
- We are working with the S3A community to be the default S3 Input Stream there as well.
- Benchmark and update public data with latest numbers once these improvements are in place.
- Release version 1.2.2 of AAL.

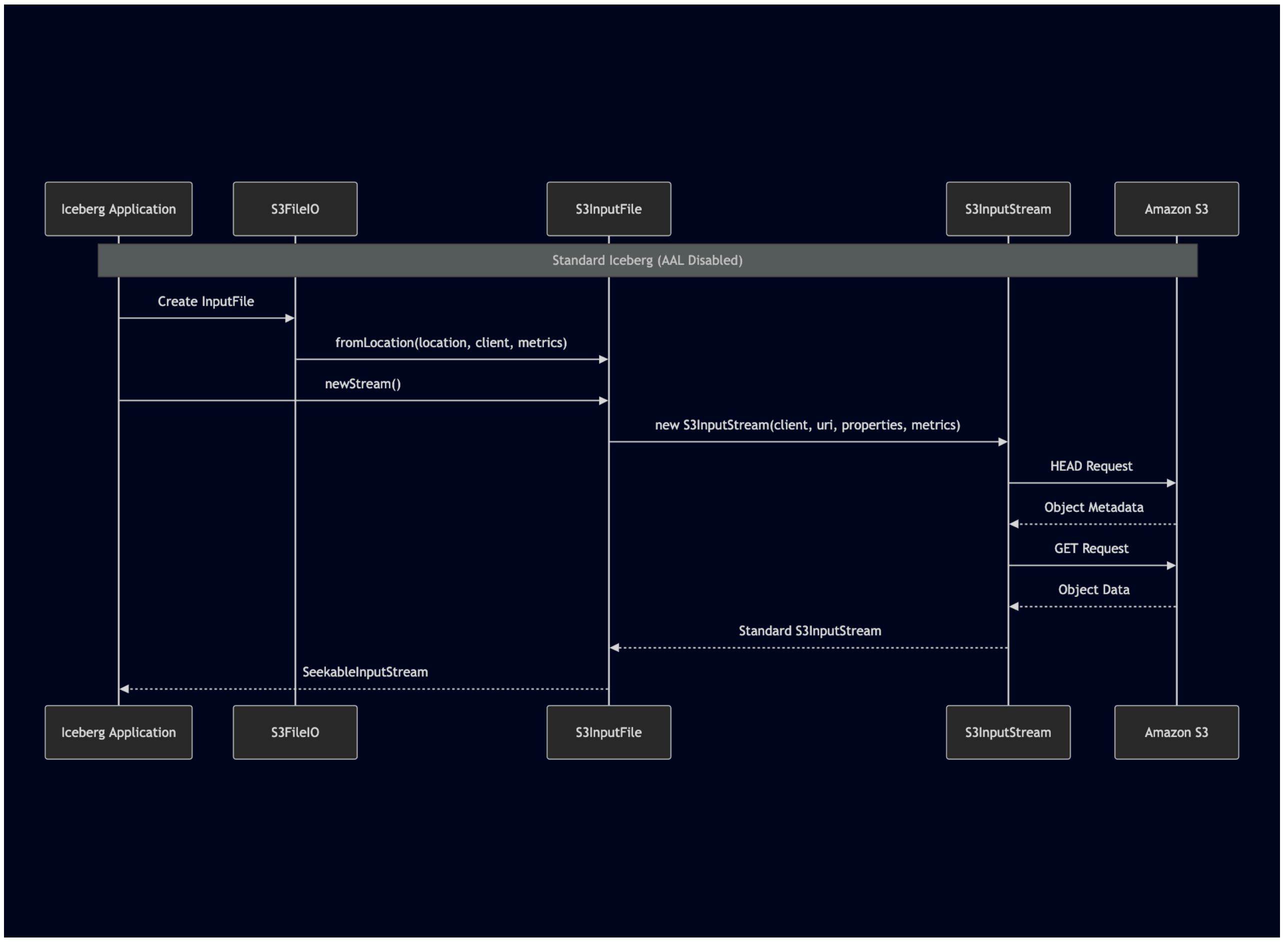
Technical Details

Current AAL integration with S3FileIO

AAL extends the SeekableInputStream interface within Iceberg and provides a factory that can be used to get an AAL stream instead of the default S3 Input Stream. Swapping between the two streams and initializing an async client are the only changes needed on Iceberg's side. There is a factory that we introduced to help create the AAL stream and Async Clients. We still continue to create the default Sync S3 Client because AAL only currently supports read operations, so for operations that require mutations the sync client will still be used. The 2 diagrams below cover the differences between AAL Off and AAL On.

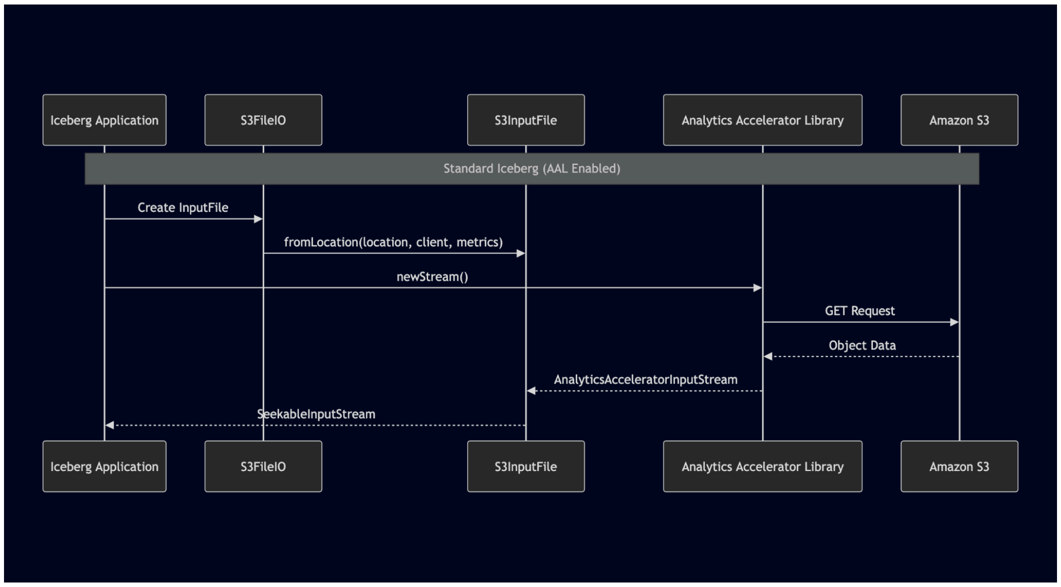
```
@Override
public SeekableInputStream newStream() {
  if (s3FileIOProperties().isS3AnalyticsAcceleratorEnabled()) {
    return AnalyticsAcceleratorUtil.newStream(this);
  }
  return new S3InputStream(client(), uri(), s3FileIOProperties(), metrics());
}
```

Current state:



With our proposed changes AAL will replace the default S3 input stream from above and will apply its optimizations and S3 best practices to an AAL stream which will be consumed instead.

Desired State With AAL:


Iceberg Application
S3FileIO
S3InputFile
Analytics Accelerator Library
Amazon S3
Standard Iceberg (AAL Enabled)
Create InputFile
fromLocation(location, client, metrics)
newStream()
GET Request
Object Data
AnalyticsAcceleratorInputStream
SeekableInputStream

Testing

- AAL has its own set of unit and integration tests.
- Iceberg unit and integration tests are passing with AAL enabled.
- Micro-benchmarks for reading specific object types with AAL.
- Benchmarks on both unpartitioned and partitioned datasets that are showing improvements.

Appendices

Appendix 1. High Level AAL features Matrix:

AAL Feature	Description	Applies To	Pull Request
Sequential prefetching	The library detects sequential read patterns to prefetch data and reduce latency.	All Data Types	https://github.com/awslabs/analytics-accelerator-s3/pull/238
Small object pre-read	AAL prefetches the object if the object size is less than 8MB.	All Data Types	https://github.com/awslabs/analytics-accelerator-s3/pull/258
CRT Integration	AAL lets customers use the CRT client and S3Async client. It does not support the S3Sync client.	All Data Types	https://github.com/awslabs/analytics-accelerator-s3/pull/15

Page Size/Request Re-shaping	Minimum read size is 64KB, and maximum S3 request size is 8MB. The library re-arranges S3 Requests to stay within these boundaries.	All Data Types	https://github.com/awslabs/analytics-accelerator-s3/pull/16
Memory Management	AAL keeps the prefetched data in-memory until the total size of the objects reach a limit defined by the user.	All Data Types	https://github.com/awslabs/analytics-accelerator-s3/pull/221/files
Metadata Caching	The library caches the object metadata (currently contentLength and Etag only) while also letting the caller pass it if they already have one.	All Data Types	https://github.com/awslabs/analytics-accelerator-s3/pull/15
Read Vectored	AAL supports an implementation of the read vectored API. This lets a list of ranges to read as well as buffers to fill. AAL will then asynchronously read and full these buffers. This is beneficial because it no longer waits for one buffer to be read fully before starting to read the next.	Parquet Objects Only	https://github.com/awslabs/analytics-accelerator-s3/pull/270
Parquet footer caching	The library reads the tail of the object with configurable size (1MB by default) as soon as a stream to a Parquet object is opened and caches it in memory. This is done to prevent multiple small GET requests that occur at the tail of the file for the Parquet metadata, `pageIndex`, and bloom filter structures.	Parquet Objects Only	https://github.com/awslabs/analytics-accelerator-s3/pull/35
Predictive column prefetching	The library tracks recent columns being read using parquet metadata. When subsequent Parquet files which have these columns are opened, the library will prefetch these columns. For example, if columns `x` and `y` are read from `A.parquet` , and then `B.parquet` is opened, and it also contains columns named `x` and `y`, the library will prefetch them asynchronously.	Parquet Objects Only	https://github.com/awslabs/analytics-accelerator-s3/pull/65
Sequential Access Prefetching	When the library detects the reader will perform a full sequential read (either through a policy or from file extension), it reads from the current position to the end of the Spark Split Size proactively.	CSV/JSON/TXT	https://github.com/awslabs/analytics-accelerator-s3/pull/238

Appendix 2: Risks

Risk	Description	Mitigation
Increased Resource Usage	**Increased memory usage** AAL keeps all data in memory as it has a caching layer this can reduce available heap memory.	[**partially-mitigated**] Our cache uses a memory threshold and data unlikely to be accessed again is evicted. However we still use more memory than the default stream due to the additional features of AAL. PR
Cost Considerations	**Overfetching data/More Get Requests** - AAL makes more requests to S3 which will increase the S3 bill. - Increased cross region request costs	[**mitigated**] While AAL increases the total number of S3 API calls (which results in higher S3 API costs), it significantly speeds up the access to S3 for compute instances. The reduction in data transfer, combined with faster processing times, leads to lower overall compute costs. Since compute costs typically represent a larger portion of total costs in data analytics workloads compared to S3 costs, the net result is a cost savings. For cross region access you pay per byte sent and with AAL the total bytes read should be decreasing.
Consistency	**Object Changes during a read** - AAL caches the metadata as well as the bytes for an object. There is the risk that the data changes during a read and a second risk that the data changes between reads when the object is already fully cached and we still respond with the old one. **AAL failure affects reading data** - We are introducing a new point of failure to Iceberg. If AAL fails its errors might affect the whole workload, not just the specific AAL operation.	**[mitigated]** For when the data changes during a read, AAL will detect it by comparing the Etags and will clear the object from its cache and will raise a retryable exception and the application will attempt the request again. PR For the case when the data is changed between reads AAL will still respond with the old data. The fix for this is adding TTL or a manual eviction mechanism, there is an open issue on our library to fix this. AAL runs its optimizations like Sequential prefetching and Predictive column prefetching as Async tasks. Even when they fail they just warn and don't raise an error. For normal synchronous reads AAL supports a retry policy that can be configured much in the same way as normal Iceberg. It also wraps all the exceptions from S3 and will return them to Iceberg as is the norm today.

Appendix 3: Open PRs with Iceberg

Number	PR	Depends on	Comments
1	AWS: Add support to run all integration tests when S3 Analytics Accelerator is enabled	-	

2	AWS: Support metrics tracking when using Analytics Accelerator stream	1	
3	AWS: Support RangeReadable in Analytics Accelerator Stream	1	
4	AWS: Support similar S3 Sync Client configurations for S3 Async Clients	1->3	
5	S3FILEIO integration with SSEC support in AAL	1->3	
6	AWS: Add Read Vector IO support to AAL	1->3	
7	AWS: AAL Default ON	1->6	

Appendix 4: Calculations

Note: You need to have enabled S3 access logs to get this data and these queries are run from Athena.

TOTAL Request COUNT

```
TOTAL COUNT requests
SELECT COUNT(*) as total_get_requests
FROM "s3_access_logs_db"."your_db"
WHERE operation LIKE '%GET%'
AND ("timestamp" = '2025/07/13' OR "timestamp" = '2025/07/14')
AND "requestdatetime" BETWEEN '13/Jul/2025:14:01:17 +0000' AND
'14/Jul/2025:14:01:17 +0000'
```

TOTAL Bytes READ

```
SELECT SUM(bytessent) as total_bytes_sent
FROM "s3_access_logs_db"."your_db"
WHERE operation LIKE '%GET%'
AND ("timestamp" = '2025/07/13' OR "timestamp" = '2025/07/14')
AND "requestdatetime" BETWEEN '13/Jul/2025:14:01:17 +0000' AND
'14/Jul/2025:14:01:17 +0000'
```