Hadoop 3.3.5 Vectored IO API vs. GNU/Linux preadv2 Analysis

Roland Yang, *UCLA*

1 Introduction

While seeking is a fundamental operation to manage data, it comes with an inherent computational cost that can significantly impact system performance. As the quantity of data in the world continues to rapidly increase with time and innovation, it has become more critical to optimize operating systems. Hadoop's Vectored IO API and GNU/Linux's preadv2 offer a solution through the utility to efficiently execute reads across multiple vectored data buffers. Although both aim to address similar needs, they differ in functionality and consequently operate more robustly in different environments. This paper provides an in-depth comparative analysis between the two APIs.

2 Advantages & Disadvantages

The readVectored API finds itself most advantageous in situations with large data reads. Its method of dividing work (potentially parallelizing it if its system permits) and then coalescing the resulting ranges back together in buffers sees around a 10%-50% boost in runtime performance [1, Fig. 1]. On top of that the readVectored API was designed to be extensively modular and compatible with several different file systems while preserving the same user interface. At the release, there was support for local file systems, the checksum file system, and the S3A file system.[2] Although preadv2 can also work with several different file systems, it may introduce a layer of complexity modifying and wrapping methods to ensure proper behavior.

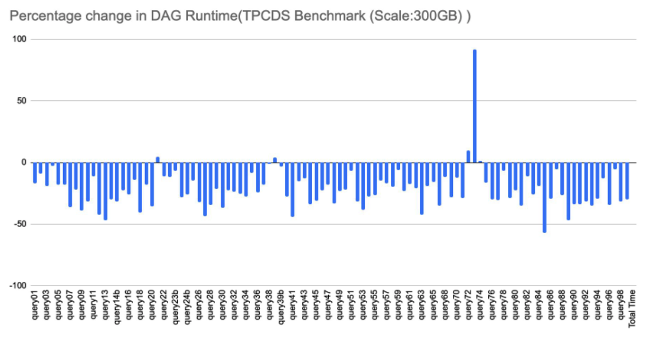

Fig. 1. Percentage change in DAG Runtime with 300 GB of TPCDS Data cast as Strings across several queries. [1]

However, while the underlying implementation of readVectored enhances runtime performance for larger datasets, it incurs a cost of decreased performance with smaller data ranges. The overhead of tasking asynchronous reads and remerging them outweighs traditional system reads, entailing as much as a negative 90% performance decrease as seen with several small queries [1, Fig. 1]. This asynchronous approach also introduces the disadvantages of non-sequential data retrieval, which preadv2 does not deal with as a sequential reader.[1] Operations dependent on prior data to arrive must either wait for all data to arrive or abort, potentially causing performance degradation. Unlike preadv2 which can update its file descriptor, readVectored lacks a method of file tracking, rendering calls to getPos() post execution invalid. Further inflexibility is also demonstrated as readVectored can be blocked with normal read API calls, while multiple preadv2 calls can be concurrently executed.[2]

3 Features Comparison

Though both APIs serve similar primary functions, their implementations differ in several features.

Namely, preadv2 offers more extensive customizability in its settings through special low-level flags. These flags modify system behavior on a per-call basis. For example, the RWF_NOWAIT flag modifies a call to not wait for data that is not immediately available and instead instantly return whatever it retrieves. The RWF_HIPRI flag allows block-based file systems to use polling of a device giving it higher priority and thus lower latency at the cost of additional resources.[3] Proper usage of these flags can lead to dramatic performance improvements.

Another noteworthy difference is that preadv2 only contiguously reads data, while readVectored in addition to reading contiguously can also read non-contiguously from several different user-defined ranges as well. Preadv2 sequentially populates several data buffers in comparison with readVectored which parallelizes reading the ranges and then coalesces them into an internal buffer as they process.[1]

Moreover, data transfers with preadv2 are guaranteed to be atomic, meaning it will completely read a contiguous block from file all at once regardless of other operations.[3]

ReadVectored on the other hand acknowledges that other operations can potentially interfere with it and opts to declare undefined output under those cases.[2]

4 Failure Modes and States Comparison

Whilst all APIs regularly encounter failures, they all tend to handle them differently through unique modes and states.

With readVectored, for any operations that fail, the contents of the destination buffer are stated to be undefined. Additionally, as comes with parallelization, any changes to a resource occurring while readVectored is processing it will have undefined output. Its output will be subject to potentially having old data, new data, or in many cases a mix. For all invalid values such as ranges or arguments, the appropriate Java exception is raised.[2]

Preadv2 in contrast follows the same error reporting and codes as its predecessors pread and read, also in addition to system call lseek. When an error occurs -1 is returned and errno is set appropriately. In the context of preadv2's new features, for instance, if an unknown flag is specified then errno is set to EOPNOTSUPP. If no bytes were read with the RWF_NOWAIT flag then errno is set to EAGAIN. If io_ven values overflow or iovcnt is outside of the permitted range then errno is set to EINVAL.[3]

5 Example Applications

Hadoop's readVectored primarily shows performance improvements with big data processing, while preadv2 proves to be more impactful in real-time applications.

For example, say an aerospace corporation is looking to perform data analysis on prior aircraft flight data and maintenance to identify potential trends and issues. Aircraft data is often characterized by mass amounts of data including navigation, communications, and hardware metrics. ReadVectored would be more suited to handle this application because it would be able to process datasets of this capacity significantly quicker. Additionally, this application is analyzing prior data, so it is not changing; in other words, the data can be processed out of order which fits into readVector's requisites. Preadv2 could still work, but readVectored's speed performance would substantially outweigh preadv2's usage.

In another instance, say an aerospace corporation is looking to handle live communications and metric monitoring with an aircraft from ground control. Preadv2 would be more suited in this case because of its features to reliably and quickly process live data streams. Special flags such as RWF_HIPRI would decrease latency, important for real-time communication. Furthermore, guaranteeing that data is processed in array order with preadv2 is important in this context, as out-of-order data here could lead to catastrophic misinterpretations and miscommunications that could lead to disaster.

6 Hadoop 3.3.5 to 3.3.6 Changes

Between Hadoop 3.3.5 and 3.3.6, there were no relevant direct changes to the Vectored IO API.[4] However, there were several minor enhancements to the S3A file system, which is one of the major file systems readVectored shows notable improvements on. S3A DiskBlocks now support the upload of files greater than 2 GB, which previously was capped at about 2 GB.[5] API requests to disabled S3 stores are properly rejected by the logging auditor.[6] Furthermore, S3A is now bindable to other filesystem schemes such as "s3://".[7]

7 Conclusion

Ultimately the comparative analysis of Hadoop's Vectored IO API and GNU/Linux's preadv2 reveals several inherent strengths and limitations among the two solutions. While each holds its distinct features, it is more importantly the responsibility of the developer to appropriately select which implementation aligns most with their specific use case for optimal operating system performance.

References

[1] M. Thakur, Hadoop Vectored IO: your Data just got Faster!, https://www.apachecon.com/acna2022/slides/02_Thakur_Hadoop_Vectored_IO.pdf (accessed Dec. 6, 2023).

[2] "Class org.apache.hadoop.fs.fsdatainputstream," Apache Hadoop 3.3.6 Documentation, https://hadoop.apache.org/docs/r3.3.6/hadoop-project-dist/hadoop-common/filesystem/fsdatainputstream.html (accessed Dec. 6, 2023).

[3] "System Calls Manual: preadv2," manpages, https://manpages.debian.org/testing/manpages-dev/preadv2.2.en.html (accessed Dec. 6, 2023).

[4] "Apache Hadoop Changelog Release 3.3.6 - 2023-06-18," Apache Hadoop, https://hadoop.apache.org/docs/r3.3.6/hadoop-project-dist/hadoop-common/release/3.3.6/CHANGELOG.3.3.6.html (accessed Dec. 6, 2023).

[5] H. Gupta, "S3A to support upload of files greater than 2 GB using DiskBlocks [HADOOP-18637]," Apache Issues, https://issues.apache.org/jira/browse/HADOOP-18637 (accessed Dec. 6, 2023).

[6] S. Loughran, "S3A: Reject multipart copy requests when disabled [HADOOP-18695]," Apache Issues, https://issues.apache.org/jira/browse/HADOOP-18695 (accessed Dec. 6, 2023).

[7] H. Gupta, “S3A filesystem to support binding to other URI schemes [HADOOP-18684],” Apache Issues, https://issues.apache.org/jira/browse/HADOOP-18684 (accessed Dec. 6, 2023).