# Vectored IO in Hadoop 3.3.5 and GNU/Linux: A Comparative Analysis of the readVectored API and preadv2 System Call

*Marco Rovell, UCLA*

## Abstract

This report provides a comprehensive comparison of the Vectored IO API introduced in Hadoop 3.3.5 and IO system calls of GNU/Linux. In particular, this report looks at the advantages and disadvantages of the readVectored API in Hadoop compared to the preadv2 system call of GNU/Linux. It highlights the features, failure modes/states, and uses of the readVector API that have practical and notable advantages over preadv2.

## 1) Introduction

The readVectored API introduced in Hadoop 3.3.5 and the GNU/Linux preadv2 system call are both similar vectored IO read calls that have provided numerous support for industry-use. It can be daunting to decide which interfaces your team should choose, but this report provides us with the subtle differences between two, the situations in which they should be used, and what support they have for failure.

## 2) Advantages and Disadvantages

The readVectored API was made for use under a distributed file system. This specification provides a multitude of advantages and disadvantages compared to the preadv2 system call.

readVectored and the HDFS in whole improve system throughput for data access as it typically deals with GB-TB of data [1]. readVectored is capable of handling non-contiguous data which is helpful when data is scattered across an entire network. This also means that readVectored requires less calls to read the same amount of data that preadv2 would require. There is also reduced overhead as an API rather than system call. All of these advantages are boosted by the fact that it is intended to be used in a distributed file system.

This can also hinder readVectored's capabilities as it increases complexity of use meaning it puts pressure on the caller to know the extent of its capabilities. This is partially seen where readVectored does not support thread safe use, meaning the caller needs to worry about this overhead. preadv2 IS a thread safe and widely supported under GNU/Linux, whereas readVectored must be used in a file system that provides the necessary interfaces.

## 3) Unique Features

Both readVectored and preadv2 have features that set them apart. readVectored's main feature is the ability to read multiple non-contiguous arrays of data into its buffers in a single call, and allowing this to be done non-synchronously while other tasks can be performed.

preadv2 lacks functionality for non-contiguous access across multiple files or machines. When files are stored across a network like the Hadoop's DFS provides, this vectored IO is essential. In essence, a feature of the readVectored API is the integration of itself into multiple machine networks.

However, what preadv2 has over readVectored is the support for additional flags, which can provide many different benefits. Some ensure that data is on the disk before returning, for consistency reasons. Some ensure higher priority reads, while others specify other user-specific actions that may be needed. This provides flexibility to the user, whereas readVectored only has one specific function albeit a very valuable one.

## 4) Failure Modes and States

Both readVectored and preadv2 account for various types of errors. preadv2 has many error codes for file descriptor errors, from pointing to an incorrect file type, fd is not open for reading, and other common fd errors. On top of this, it also accounts for lseek errors, it throws errors for attempting to access outside of address space, and overflow for its iov types.

readVectored does not work on the file descriptor level, but what it does account for, although minimally, is other read API calls while readVectored is in progress, in which it may block. In general, readVectored does not have a real failure mode or state, it is up to the file system in which the API is used in to manage those data inconsistencies. Thus, the readVectored API can often have undefined output rather than error. readVectored does have some precondition error checking. It will throw an error if the offset and/or length is less than zero, or if the caller does not have authorized permissions to access the file it attempts to read. It also has a postcondition which checks to make sure the buffer byte size is the correct range. It also has a min seek size and max read size which automatically manages the ranges at the time of execution.

For the most part, both readVectored and preadv2 handle error checking very similarly, and it is mostly up to the caller and/or the system to be able to keep track of race conditions that may arise.

## Applications of preadv2 and readVectored

The readVectored API will be most suitable for a situation where you need multiple reads to large non-contiguous

data sets, especially across multiple files and machines. A very applicable situation to use readVectored would be in a company developing software for financial analysis. readVectored will be very helpful for processing large data sets within a distributed file system. Say you need to grab data scattered across many files and machines, this would be the most efficient way to process such data into a single stream for use, as well as improving upon the throughput of the network. preadv2 is simply not capable of this and would not be well equipped for dealing with this job. The number of system calls and their overhead will slow down the process and complicate things further.

On the other hand, preadv2's best feature is its versatility due to the implementation of flags. Many of the flags provide enhanced usage past it's normal control, like RWF_NOWAIT, which lets the process continue if the data isn't available as preadv2 is called [2]. A situation that uses preadv2 to its full potential is during queries and analysis on single structured files. If there is a real-time aspect to this, for example, monitoring sensor data of various devices in an aircraft, a preadv2 system call will be able to efficiently read such data, and specific flags can allow for it to be optimized for such a situation. The RWF_SYNC flag ensures that the write operation is synchronized and that the data will persist onto the storage device. This functionality is not present in the readVectored API, and it does not have flag functionality. The Apache Hadoop DFS documentation implies that added optimization should be done with sub-classes [3].

## References


[1] Apache Software Foundation, "HDFS Architecture Guide," Apache Hadoop Project, 2021. [Online]. Available:https://hadoop.apache.org/docs/r3.3.5/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[2] "preadv2(2) - Linux manual page," Debian Manpages, 2023. [Online]. Available: https://manpages.debian.org/testing/manpages-dev/preadv2.2.en.html.

[3] Apache Software Foundation, "FSDataInputStream - Apache Hadoop HDFS API," Apache Hadoop Project, 2021. [Online]. Available: https://hadoop.apache.org/docs/r3.3.5/hadoop-project-dist/hadoop-common/filesystem/fsdatainputstream.html.