6. Segmented Log

Let's learn about segmented log and its usage.

We'll cover the following

Background#

A single log can become difficult to manage. As the file grows, it can also become a performance bottleneck, especially when it is read at the startup. Older logs need to be cleaned up periodically or, in some cases, merged. Doing these operations on a single large file is difficult to implement.

Definition#

Break down the log into smaller segments for easier management.

Solution#

A single log file is split into multiple parts, such that the log data is divided into equal-sized log segments. The system can roll the log based on a rolling policy, either a configurable period of time (e.g., every 4 hours) or

rolling policy - either a configurable period of time (e.g., every 4 hours) or a configurable maximum size (e.g., every 1GB).

Examples#

- **Cassandra** uses the segmented log strategy to split its commit log into multiple smaller files instead of a single large file for easier operations. As we know, when a node receives a write operation, it immediately writes the data to a commit log. As the Commit Log grows in size and reaches its threshold in size, a new commit log is created. Hence, over time, several commit logs will exist, each of which is called a segment. Commit log segments reduce the number of seeks needed to write to disk. Commit log segments are truncated when Cassandra has flushed corresponding data to SSTables. A commit log segment can be **archived**, **deleted**, or **recycled** once all its data has been flushed to SSTables.
- **Kafka** uses log segmentation to implement storage for its partitions. As Kafka regularly needs to find messages on disk for purging, a single long file could be a performance bottleneck and error-prone. For easier management and better performance, the partition is split into segments.

← Back

5. Write-ahead Log

Next →

7. High-Water Mark

Mark as Completed

Report an Issue