Kafka Characteristics

Let's explore different characteristics of Kafka.

We'll cover the following

Storing messages to disks#

Kafka writes its messages to the local disk and does not keep anything in RAM. Disks storage is important for durability so that the messages will not disappear if the system dies and restarts. Disks are generally considered to be slow. However, there is a huge difference in disk performance between random block access and sequential access. Random block access is slower because of numerous disk seeks, whereas the sequential nature of writing or reading, enables disk operations to be thousands of times faster than random access. Because all writes and reads happen sequentially, Kafka has a very high throughput.

Writing or reading sequentially from disks are heavily optimized by the OS, via **read-ahead** (prefetch large block multiples) and **write-behind** (group small logical writes into big physical writes) techniques.

Also, modern operating systems cache the disk in free RAM. This is called Pagecache (https://en.wikipedia.org/wiki/Page_cache). Since Kafka stores

messages in a standardized binary format unmodified throughout the whole flow (producer → broker → consumer), it can make use of the zero-copy (https://en.wikipedia.org/wiki/Zero-copy) optimization. That is when the operating system copies data from the Pagecache directly to a socket, effectively bypassing the Kafka broker application entirely.

Kafka has a protocol that groups messages together. This allows network requests to group messages together and reduces network overhead. The server, in turn, persists chunks of messages in one go, and consumers fetch large linear chunks at once.

All of these optimizations allow Kafka to deliver messages at near network-speed.

Record retention in Kafka#

By default, Kafka retains records until it runs out of disk space. We can set time-based limits (configurable retention period), size-based limits (configurable based on size), or compaction (keeps the latest version of record using the key). For example, we can set a retention policy of three days, or two weeks, or a month, etc. The records in the topic are available for consumption until discarded by time, size, or compaction.

Client quota#

It is possible for Kafka producers and consumers to produce/consume very high volumes of data or generate requests at a very high rate and thus monopolize broker resources, cause network saturation, and, in general, deny service to other clients and the brokers themselves. Having quotas protects against these issues. Quotas become even more important in large multi-tenant clusters where a small set of badly behaved clients can degrade the user experience for the well-behaved ones.

In Kafka, quotas are byte-rate thresholds defined per `client-ID`. A `client-ID` logically identifies an application making a request. A single

`client-ID` can span multiple producer and consumer instances. The

quota is applied for all instances as a single entity. For example, if a `client-ID` has a producer quota of 10 MB/s, that quota is shared across all instances with that same ID.

The broker does not return an error when a client exceeds its quota but instead attempts to slow the client down. When the broker calculates that a client has exceeded its quota, it slows the client down by holding the client's response for enough time to keep the client under the quota. This approach keeps the quota violation transparent to clients. This also prevents clients from having to implement special back-off and retry behavior.

Kafka performance#

Here are a few reasons behind Kafka's performance and popularity:

Scalability: Two important features of Kafka contribute to its scalability.

- A Kafka cluster can easily expand or shrink (brokers can be added or removed) while in operation and without an outage.
- A Kafka topic can be expanded to contain more partitions. Because a partition cannot expand across multiple brokers, its capacity is bounded by broker disk space. Being able to increase the number of partitions and the number of brokers means there is no limit to how much data a single topic can store.

Fault-tolerance and reliability: Kafka is designed in such a way that a broker failure is detectable by ZooKeeper and other brokers in the cluster. Because each topic can be replicated on multiple brokers, the cluster can recover from broker failures and continue to work without any disruption of service.

Throughput: By using consumer groups, consumers can be parallelized, so that multiple consumers can read from multiple partitions on a topic,

allowing a very high message processing throughput.

Low Latency: 99.99% of the time, data is read from disk cache and RAM; very rarely, it hits the disk.

← Back

Kafka Delivery Semantics

Next →

Summary: Kafka

Mark as Completed

Report an Issue