

KIP-872: Add Serializer#serializeToByteBuffer() to reduce memory copying

- [Status](#)
- [Motivation](#)
- [Proposed Changes](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)

Status

Current state: *Accepted*

Discussion thread: <https://lists.apache.org/thread/5fmg151x7h3gsx9mn7xq0n28j700rys0>

Voting thread: <https://lists.apache.org/thread/jgtzlb4l60hs6t3o67x53w8gonk2ffvo>

PR: <https://github.com/apache/kafka/pull/12685>

JIRA:



Motivation

Currently, we use `Serializer#serialize(String, Headers, T)` in `KafkaProducer#doSend(ProducerRecord, Callback)` to serialize key and value. First, we call `Serializer#serialize(String, Headers, T)` to convert `T` into `byte[]`, then use `Utils#wrapNullable(byte[])` to convert `byte[]` into `ByteBuffer`, and finally write `ByteBuffer` into `MemoryRecordsBuilder` through `DefaultRecord#writeTo(DataOutputStream, int, long, ByteBuffer, ByteBuffer, Header[])`.

Why don't we add a `serializeToByteBuffer(String, Headers, T)` method to `Serializer`, and then use `Serializer#serializeToByteBuffer(String, Headers, T)` in `KafkaProducer#doSend(ProducerRecord, Callback)`? If `T` is an instance of `ByteBuffer` or `T` is based on `ByteBuffer`, we would reduce a lot of memory allocation and memory copying.

Additionally, I plan to ultimately replace `byte[]` with `ByteBuffer` in `Serializer`: in 3.x versions, both `Serializer#serializeToByteBuffer()` and `Serializer#serialize()` exist. Starting from version 4.0, `Serializer#serializeToByteBuffer()` will be removed and its implementation will replace `Serializer#serialize()`, which will return `ByteBuffer` by default, and `Serializer#serialize()` will not modify the input `ByteBuffer`.

Public Interfaces

We propose adding default method `Serializer#serializeToByteBuffer(String, T)`, `Serializer#serializeToByteBuffer(String, Headers, T)` and `Partitioner#partition(String, Object, ByteBuffer, Object, ByteBuffer, Cluster)`:

Class/Interface	Method
Serializer	<pre>default ByteBuffer serializeToByteBuffer(String topic, T data) { return wrapNullable(serialize(topic, data)); } default ByteBuffer serializeToByteBuffer(String topic, Headers headers, T data) { return wrapNullable(serialize(topic, headers, data)); }</pre>

ByteBufferSerializer	<pre> /** * Note that this method will modify the position and limit of the input ByteBuffer. * * @param topic topic associated with data * @param data typed data * @return serialized ByteBuffer */ @Override public ByteBuffer serializeToByteBuffer(String topic, ByteBuffer data) { if (data == null) { return null; } // Consider that ByteBuffer#wrap(byte[]) return a ByteBuffer that does not need to call flip(). if (data.position() > 0) { data.flip(); } return data; } @Override public ByteBuffer serializeToByteBuffer(String topic, Headers headers, ByteBuffer data) { return serializeToByteBuffer(topic, data); } </pre>
Partitioner	<pre> default int partition(String topic, Object key, ByteBuffer keyBytes, Object value, ByteBuffer valueBytes, Cluster cluster) { return partition(topic, key, toNullableArray(keyBytes), value, toNullableArray(valueBytes), cluster); } </pre>

Proposed Changes

There are the following changes:

- **Serializer**
 - Adding default method `Serializer#serializeToByteBuffer(String, T)` and `Serializer#serializeToByteBuffer(String, Headers, T)`;
 - Implement `serializeToByteBuffer(String, T)` and `serializeToByteBuffer(String, Headers, T)` for `ByteBufferSerializer`;
 - Using `Serializer#serializeToByteBuffer(String, Headers, T)` to serialize key&value in `KafkaProducer#doSend(ProducerRecord, Callback)`.
- **Partitioner**
 - Adding default method `Partitioner#partition(String, Object, ByteBuffer, Object, ByteBuffer, Cluster)`;
 - Implement `partition(String, Object, ByteBuffer, Object, ByteBuffer, Cluster)` for `RoundRobinPartitioner`;
 - Using `Partitioner#partition(String, Object, ByteBuffer, Object, ByteBuffer, Cluster)` in `KafkaProducer#partition(ProducerRecord, ByteBuffer, ByteBuffer, Cluster)`.
- **Utils**
 - Adding method `murmur2(ByteBuffer data)`.

Compatibility, Deprecation, and Migration Plan

- **Compatibility**
 - We just add default methods `Serializer#serializeToByteBuffer(String, T)`, `Serializer#serializeToByteBuffer(String, Headers, T)` and `Partitioner#partition(String, Object, ByteBuffer, Object, ByteBuffer, Cluster)` which is compatible with the existing Serializers.
 - The impact on ByteBuffer offsets is consistent after calling `ByteBufferSerializer#serialize(String, ByteBuffer)` and `ByteBufferSerializer#serializeToByteBuffer(String, Headers, T)`.

Rejected Alternatives

There is no alternative.