

KIP-863: Reduce CompletedFetch#parseRecord() memory copy

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

Status

Current state: *Accepted*

Discussion thread: <https://lists.apache.org/thread/tbhmkf44jhjf8lqmo7w2whynbgttf1o6>

Voting thread: <https://lists.apache.org/thread/z6v4qyhgydl1tj0s3ycn6v4hv408gx2t>

PR: <https://github.com/apache/kafka/pull/12545>

JIRA: [KAFKA-14944 - Getting issue details...](#) STATUS

Motivation

Currently we use `Deserializer#deserialize(String topic, Headers headers, byte[] data)` in `CompletedFetch#parseRecord(TopicPartition, RecordBatch, Record)` to deserialize key&value, we first call `Utils.toArray(ByteBuffer)` to convert `ByteBuffer` into `byte[]` and then call `Deserializer#deserialize(String topic, Headers headers, byte[] data)` which will cause memory allocation and memory copying.

The default implementation of this method would still call `Utils.toNullableArray(ByteBuffer)` and then leverage on the existing method. But for the following cases we can use `ByteBuffer` instead of `byte[]` for deserialization, which will reduce memory allocation and memory copying:

- For built-in `StringDeserializer` and `ByteBufferDeserializer`, we will do the deserialization directly on with the overloaded method with `ByteBuffer`.
- If user-customized `Deserializers` override this overloaded method's default implementation, they also can reduce memory allocation and memory copying.

Public Interfaces

We propose adding default method `Deserializer#deserialize(String, Headers, ByteBuffer)`.

Class	Method
Deserializer	<pre>default T deserialize(String topic, Headers headers, ByteBuffer data) { return deserialize(topic, headers, Utils.toNullableArray(data)); }</pre>
ByteBufferDeserializer	<pre>@Override public ByteBuffer deserialize(String topic, Headers headers, ByteBuffer data) { return data; }</pre>

StringDeserializer	<pre> @Override public String deserialize(String topic, Headers headers, ByteBuffer data) { if (data == null) { return null; } try { if (data.hasArray()) { return new String(data.array(), data.position() + data. arrayOffset(), data.remaining(), encoding); } else { return new String(Utils.toArray(data), encoding); } } catch (UnsupportedEncodingException e) { throw new SerializationException("Error when deserializing ByteBuffer to string due to unsupported encoding " + encoding); } } </pre>
--------------------	--

Proposed Changes

- *Deserializer* add default method *deserialize(String, Headers, ByteBuffer)*;
- Invoke *Deserializer#deserialize(String, Headers, ByteBuffer)* instead of *Deserializer#deserialize(String, Headers, byte[])* in *CompletedFetch#parseRecord(TopicPartition, RecordBatch, Record)*.

Compatibility, Deprecation, and Migration Plan

- This proposal has no compatibility issues, we just add default method *deserialize(String, Headers, ByteBuffer)* which is compatible with the existing *Deserializers*.
- If someone wants the deserializer to be compatible with older versions of the kafka-clients library they should always implement the byte array-based *deserialize* methods.

Rejected Alternatives

Another solution I thought of is *PoolArea*, just like *Netty*, but this solution is more complicated.