Item 368
git_comments:

1. fall on an 8-byte offset
2. The starting byte offset into 'in'.
3. Align the output to 8 byte boundary.
4. Metadata length contains integer prefix plus byte padding
5. Add extra padding bytes so that length prefix + metadata is a multiple of 8 after alignment
6. * * Deserializes a RecordBatch knowing the size of the entire message up front. This * minimizes the number of reads to the underlying stream.
7. Deserializes a record batch given the Flatbuffer metadata and in-memory body
8. Metadata size in the Block account for the size prefix
9. Read the message size. There is an i32 little endian prefix.
10. Now read the body
11. Now read the record batch body
12. Read just the header. This demonstrates being able to read without need to deserialize the buffer.

git_commits:

1. **summary:** ARROW-499: Update file serialization to use the streaming serialization format.
   message: ARROW-499: Update file serialization to use the streaming serialization format. Author: Wes McKinney <wes.mckinney@twosigma.com> Author: Nong Li <nongli@gmail.com> Closes #292 from nongli/file and squashes the following commits: 18890a9 [Wes McKinney] Message fixes. Fix Java test suite. Integration tests pass f187539 [Nong Li] Merge pull request #1 from wesm/file-change-cpp-impl e3af434 [Wes McKinney] Remove unused variable 664d5be [Wes McKinney] Fixes, stream tests pass again ba8db91 [Wes McKinney] Redo MessageSerializer with unions. Still has bugs 21854cc [Wes McKinney] Restore Block.bodyLength to long 7c6f7ef [Nong Li] Update to restore Block behavior 27b3909 [Nong Li] [ARROW-499]: [Java] Update file serialization to use the streaming serialization format.

github_issues:

github_issues_comments:

github_pulls:

1. **title:** ARROW-499: Update file serialization to use the streaming serialization format.
   body:

github_pulls_comments:

1. see https://github.com/wesm/arrow/commit/c4059e61cfabb73163e46a1fee928a6efae6c5f0 -- that was at least one additional problem with the deserialization path for files
2. i'm not sure what else is wrong
3. Also, I don't quite understand the semantics of when you need to call `ArrowBuf.release`
4. With this commit https://github.com/wesm/arrow/commit/367367b0b1e697ddf6a8fae0960ffc26e976bcde, the tests pass except for 1, but I think the failure is correct
5. OK, if you pull this commit https://github.com/wesm/arrow/commit/97d61202cfbbacede7a1c36e5f2d0d2faff8402d, you should get a green build including integration tests
6. can you update the title to start with `ARROW-499:` to appease the merge tool?
7. @wesm Done
8. LGTM

github_pulls_reviews:

1. The new serialized batches aren't aligned. Whats the requirement/goal for alignment? This would be easy to update in MessageSerializer but I'm not sure what the current desired behavior is.
2. 8 byte alignment is sufficient
3. These offsets should be long / int64
4. What do you want aligned. The start of buffers? various metadata pieces? The serialized format is now: message header with size prefix rowbatch metadata header with size prefix buffers We can pad between any of them/none of them. Currently its none of them.
5. One downside of this change getting rid of the Blocks, is that reading a record batch from the file will now require 3 reads from the source (length, metadata, then body) -- before, you could read the whole payload in 1 read if you wanted. not sure how much that matters
6. BigInteger?
7. Only the starts of buffers need to be aligned (the buffers are already supposed to be padded). So only need to add padding to the serialized metadata
8. The serialization would be quite a bit simpler if we enforced that Message was also fixed length. It would be similarly easy if we enforced that the current fields in Message always came first. i.e. read(sizeof(Message)) was always going to be valid and if the header grew, you could tell by looking header.version.
9. The problem with combining the metadata and body length is that it makes it difficult / impossible to do partial reads / field projections in a record batch. If you can inspect the metadata without reading the body, you can determine the byte ranges you need to read only a certain subset of fields.
10. Repeating my comment from slack here for the others: the message.header could be arbitrarily large, so are you saying that the new fixed-size message would contain the version, message type (as an enum), metadata length, and body length with padding that makes 16 bytes i guess
11. I think I'm confused. I'm referring to this: https://github.com/apache/arrow/blob/master/format/Message.fbs#L276 Is this expected to be arbitrarily large?
12. The `header` union could be arbitrarily large since it grows with the number of fields in the record batch or schema
13. Hmm, this is not what I'm interpreting the union to mean. I'm interpreting it to basically mean enum which could grow I guess if there many more message types. It doesn't contain the thing inside the union. https://google.github.io/flatbuffers/md__schemas.html I've very little experience with flatbufs though. Am I totally off?
14. Yeah, the object you put in the union must be appended as part of constructing the root flatbuffer. See for example: https://github.com/apache/arrow/blob/master/cpp/src/arrow/ipc/metadata-internal.cc#L312 When you read the Flatbuffer in C++, if gives you the functions `header_type()` (which gives you the enum) and `header()`, which is a `const void*` that you can cast to the correct Flatbuffer type (e.g. `reinterpret_cast<const flatbuf::Schema*>`) depending on the value of the enum. The Flatbuffers documentation around unions is not very good; it remember it took me a while to figure out
15. OK, I see what you did in the Java code, and why we are confused. You are writing the embedded message header separately, not actually storing the data in the union. So if I looked at the Message in your stream, the `header` field would have the header union enum set properly, but the union data payload would be nullptr https://github.com/apache/arrow/blob/6811d3fcfc9da65e24b6d0f2ad5d5d348d879f11/java/vector/src/main/java/org/apache/arrow/vector/stream/MessageSerializer.ja
16. Here is what flatc generates for Message: https://gist.github.com/wesm/3a9c2106b3a3634d8af1e6e66260836f When you serialize the header you aren't calling `Message.addHeader`

jira_issues:

1. **summary:** Update file serialization to use streaming serialization format
   description: As discussed in the PR for ARROW-474, it makes sense if the two share the underlying serialization

jira_issues_comments:

1. Issue resolved by pull request 292 [https://github.com/apache/arrow/pull/292]