Item 208
git_comments:

git_commits:

1. **summary:** [FLINK-11775][table-runtime-blink] Use MemorySegmentWritable to BinaryRowSerializer
   message: [FLINK-11775][table-runtime-blink] Use MemorySegmentWritable to BinaryRowSerializer This closes #8775

github_issues:

github_issues_comments:

github_pulls:

1. **title:** [FLINK-11775][runtime][table-runtime-blink] Introduce MemorySegmentWritable to let Segments direct copy to internal bytes
   body: ## What is the purpose of the change Introduce MemorySegmentWritable to let BinaryRow direct copy to internal bytes ## Verifying this change ut ## Does this pull request potentially affect one of the following parts: - Dependencies (does it add or upgrade a dependency): (no) - The public API, i.e., is any changed class annotated with `@Public(Evolving)`: (no) - The serializers: (no) - The runtime per-record code paths (performance sensitive): (no) - Anything that affects deployment or recovery: JobManager (and its components), Checkpointing, Yarn/Mesos, ZooKeeper: (no) - The S3 file system connector: (no) ## Documentation - Does this pull request introduce a new feature? (yes) - If yes, how is the feature documented? (JavaDocs)

github_pulls_comments:

1. Thanks a lot for your contribution to the Apache Flink project. I'm the @flinkbot. I help the community to review your pull request. We will use this comment to track the progress of the review. ## Review Progress * ❓ 1. The [description] looks good. * ❓ 2. There is [consensus] that the contribution should go into to Flink. * ❓ 3. Needs [attention] from. * ❓ 4. The change fits into the overall [architecture]. * ❓ 5. Overall code [quality] is good. Please see the [Pull Request Review Guide](https://flink.apache.org/reviewing-prs.html) for a full explanation of the review process.<details> The Bot is tracking the review progress through labels. Labels are applied according to the order of the review items. For consensus, approval by a Flink committer of PMC member is required <summary>Bot commands</summary> The @flinkbot bot supports the following commands: - `@flinkbot approve description` to approve one or more aspects (aspects: `description`, `consensus`, `architecture` and `quality`) - `@flinkbot approve all` to approve all aspects - `@flinkbot approve-until architecture` to approve everything until `architecture` - `@flinkbot attention @username1 [@username2 ..]` to require somebody's attention - `@flinkbot disapprove architecture` to remove an approval you gave earlier </details>

github_pulls_reviews:

jira_issues:

1. **summary:** Introduce MemorySegmentWritable to let Segments direct copy to internal bytes
   description: Blink new binary format is based on MemorySegment. Introduce MemorySegmentWritable to let DataOutputView direct copy to internal bytes {code:java} /** * Provides the interface for write(Segment). */ public interface MemorySegmentWritable { /** * Writes {@code len} bytes from memory segment {@code segment} starting at offset {@code off}, in order, * to the output. * * @param segment memory segment to copy the bytes from. * @param off the start offset in the memory segment. * @param len The number of bytes to copy. * @throws IOException if an I/O error occurs. */ void write(MemorySegment segment, int off, int len) throws IOException; }{code}   If we want to write a Memory Segment to DataOutputView, we need to copy bytes to byte[] and then write it in, which is less effective. If we let AbstractPagedOutputView have a write(MemorySegment) interface, we can copy it directly. We need to ensure this in network serialization, batch operator calculation serialization, Streaming State serialization to avoid new byte[] and copy.

jira_issues_comments:

1. Hi [~lzljs3620320], could you add some more explainations to show why the API should be added and what benefits we can gain from that API?
2. cc [~sewen], [~pnowojski]
3. Couldn't we provide an implementation of {{DataInputView}} that wraps a {{MemorySegment}}?
4. >> Couldn't we provide an implementation of {{DataInputView}} that wraps a {{MemorySegment}}? In some places, write(DataInput) is inefficient, such as AbstractPagedOutputView, which eventually calls HybridMemorySegment.put(DataInput), which constantly calls putLongBigEndian when it is in direct memory.
5. Sorry for asking maybe stupid question, I'm not very familiar with this code. Shouldn't we in that case try to optimize the {{HybridMemorySegment#put(java.io.DataInput, int, int)}} for off heap cases. For example for cases when {{DataInput}} is backed by array or something that can be easily wrapped as ByteBuffer or something else that's efficient? Like: {code:java} @Override public final void put(DataInput in, int offset, int length) throws IOException { if (address <= addressLimit) { if (heapMemory != null) { in.readFully(heapMemory, offset, length); } else { ByteBuffer src = in.wrapAsByteBuffer(); offHeapBuffer.put(src); } } else { throw new IllegalStateException("segment has been freed"); } } {code} and provide some efficient implementation of {{wrapAsByteBuffer()}} for {{DataInputView}} that are wrapping {{MemorySegment}}?
6. Hi [~pnowojski] I think it's very difficult for DataInputView to provide wrapAsByteBuffer. (For example, there are multiple MemorySegments under AbstractPagedInputView.) I prefer to have DataOutputView provide write(MemorySegment segment) interface.
7. [~lzljs3620320] I think in general the performance optimization makes sense. There is one question for me that I don't understand from the proposal. Are you suggesting: a) {{AbstractPagedOutputView implements DataOutputView, MemorySegmentWritable}} or b) {{AbstractPagedOutputView implements DataOutputView}} and {{DataOutputView extends MemorySegmentWritable}} I am assuming a) and that sounds resaonable to me, but b) does not sound good. Can you give an example of where and how you would use this new functionality. Would you have to cast to {{MemorySegmentWritable}} or are we already typed to {{AbstractPagedOutputView}}. In general, please be aware that MemorySegment is tagged as {{@Internal}} and we should not leak it through a new public interface, so at least I would suggest to also tag {{MemorySegmentWritable}} as internal.
8. I think that having {{AbstractPagedOutputView implements DataOutputView, MemorySegmentWritable}} can work. Can you share how you plan to implement this? Will the implementation of the write method cast the memory segment to a specific type (like {{HybridMemorySegment}}) and then make an unsafe copy? There should still be a generic fallback path that works for all memory segments, like going through a temporary array, or wrapping the MemorySegment as a ByteBuffer and reading data from there into memory structure of the AbstractPagedOutputView.
9. [~sewen] I found that the implementation in {{AbstractPagedOutputView}} was already somehow recently merged as part of a >4k lines PR via FLINK-11856, for now without introducing an interface. It is using {{Segment#copyTo}} which looks good to work for all kinds of segment implementations We could consider two alternatives, something in the direction of Piotrs proposal or casting a special {{DataInputView}} for an optimized path in the already existing method of {{DataOutputView}}. One concern is, that according to the docs {{DataOutputView}} is already supposed to be the interface to interact with memory segments. So here we bypass that abstraction. So might reconsider if the abstraction still fits or (probably applicable here) consider this new interface a secondary, lower level access interface available to code that already operates on the {{MemorySegment}} level. But in that case, I wonder what is currently the benefit of introducing an interface over only implementing this on {{AbstractPagedOutputView}}, as currently done.
10. I think my goal is to optimize the serialization of BinaryRow, which currently occurs on two views: 1. AbstractPagedOutputView: In Sort, HashTable, etc. 2. DataOutputSerializer: (Because bytes is saved to byte[] in DataOutputSerializer, it can be directly copied from MemorySegment.) Scenario 1: It happened in RecordWriter and is about to be sent to the network. Scenario 2: In the serialization of RocksDBValueState.   My original intention was to optimize the serialization of BinaryRow on both views. The current idea is: Let AbstractPagedOutputView and DataOutputSerializer implement MemorySegmentWritable. In AbstractPagedOutputView, implement write(MemorySegment segment, int off, int len) to use MemorySegment.copyTo(MemorySegment)
In DataOutputSerializer, implement write(MemorySegment segment, int off, int len) to use MemorySegment.get(byte[]) Then in BinaryRowSerializer.serialize(), if the outputView isInstanceOf

MemorySegmentWritable, call write(MemorySegment), or whether it is serialized using the DataOutputView interface.   Thanks [~srichter] and [~sewen] and [~pnowojski] for your advice: 1.let DataOutputView implement MemorySegmentWritable is a bad idea. Not every DataOutputView has the ability to deal directly with MemorySegment. 2.keep MemorySegmentWritable as internal is good. Only our Table can touch it.

11. I also think letting AbstractPagedOutputView implements MemorySegmentWritable (not exposing to DataOutputView) is more appropriate. From my understanding, the concept of *page* in AbstractPagedOutputView is similar with MemorySegment we have, or even it is representing MemorySegment, but just with a different name. Conceptually, it will be more align we let AbstractPagedOutputView know MemorySegment and have an optimal way to write it.
12. Does this jira still valid? [~lzljs3620320]
13. It is still valid, we need implement what we discussed.
14. merged in 1.9.0: 99a94edf543fb28ccc8e34b886a67763f328b79c