Item 351
git_comments:

1. * * The result of writing a batch of rows to Bigtable. Rows are written to bigtable in batches (based * on the runner-chosen bundle size). Once each batch finishes, a single {@link BigtableWriteResult} * is emitted.
2. * The number of rows written in this batch.
3. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
4. * A coder for {@link BigtableWriteResult}.
5. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
6. * * A {@link PTransform} that writes to Google Cloud Bigtable and emits a {@link * BigtableWriteResult} for each batch written. See the class-level Javadoc on {@link BigtableIO} * for more information. * * @see BigtableIO
7. * * Returns a {@link BigtableIO.WriteWithResults} that will emit a {@link BigtableWriteResult} * for each batch of rows written.
8. See Wait.on
9. The windowing of `moreData` must be compatible with `data`, see {@link org.apache.beam.sdk.transforms.Wait#on} for details.
10. * * Tests that the outputs of the Bigtable writer are correctly windowed, and can be used in a * Wait.on transform as the trigger.
11. * Tests that at least one result is emitted per element written in the global window.
12. * Tests that at least one result is emitted per element written in each window.
13. * * A DoFn used to generate N outputs, where N is the input. Used to generate bundles of >= 1 * element.

git_commits:

1. **summary:** Merge pull request #7805: [BEAM-3061] Done notifications for BigtableIO.Write
   message: Merge pull request #7805: [BEAM-3061] Done notifications for BigtableIO.Write

github_issues:

github_issues_comments:

github_pulls:

1. **title:** [BEAM-3061] Done notifications for BigtableIO.Write
   body: This adds support for "done" notification in `BigtableIO.write()`. For each bundle+window that is written, a single "Void" is output by the PTransform. Doing so allows flows to effectively wait for all writes to finish. Note that windowing is preserved, and pipeline writers can choose to deal with early firings, etc as they desire. A common use-case of this is to wait for all writes to finish, and then do something else, this can be accomplished using the built-in `Wait.on` transform with the output of the write, like so: ``` val writeResults = mutations.apply("WriteToBigtable", BigtableIO.write()...) val collectionToProcessAfterWrites = Create.of("derp") collectionToProcessAfterWrites .apply(Wait.on(writeResults)) .apply(ParDo.of(someFn)) ``` An important point to note here is this only ensures writes have been applied **to a single cluster**, it does not ensure that all writes **have replicated to all clusters**. I had also made a previous PR a few months ago (https://github.com/apache/beam/pull/3997), this is an enhancement of that. ---------- Thank you for your contribution! Follow this checklist to help us incorporate your contribution quickly and easily: - [ ] [**Choose reviewer(s)**] (https://beam.apache.org/contribute/#make-your-change) and mention them in a comment (`R: @username`). - [x] Format the pull request title like `[BEAM-XXX] Fixes bug in ApproximateQuantiles`, where you replace `BEAM-XXX` with the appropriate JIRA issue, if applicable. This will automatically link the pull request to the issue. - [x] If this contribution is large, please file an Apache [Individual Contributor License Agreement](https://www.apache.org/licenses/icla.pdf). Post-Commit Tests Status (on master branch) ---------- Lang | SDK | Apex | Dataflow | Flink | Gearpump | Samza | Spark --- | --- | --- | --- | --- | --- | --- | --- Go | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Go/lastCompletedBuild/badge/icon)](https://builds.apache.org/job/beam_PostCommit_Go/lastCompletedBuild/) | --- | --- | --- | --- | --- | --- Java | [![Build Status](https://builds.apache.org/job/beam_PostCommit_Java/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java/lastCompletedBuild/) | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Apex/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Apex/lastCompletedBuild/) | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Dataflow/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Dataflow/lastCompletedBuild/) | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Flink/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Flink/lastCompletedBuild/)
[![Build Status] (https://builds.apache.org/job/beam_PostCommit_Java_PVR_Flink_Batch/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java_PVR_Flink_Batch/lastCompletedBuild/)
[![Build Status] (https://builds.apache.org/job/beam_PostCommit_Java_PVR_Flink_Streaming/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java_PVR_Flink_Streaming/lastCompletedBuild/) | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Gearpump/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Gearpump/lastCompletedBuild/) | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Samza/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Samza/lastCompletedBuild/) | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Spark/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Java_ValidatesRunner_Spark/lastCompletedBuild/) Python | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Python_Verify/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Python_Verify/lastCompletedBuild/) | --- | [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Py_VR_Dataflow/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Py_VR_Dataflow/lastCompletedBuild/) </br> [![Build Status] (https://builds.apache.org/job/beam_PostCommit_Py_ValCont/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PostCommit_Py_ValCont/lastCompletedBuild/) | [![Build Status] (https://builds.apache.org/job/beam_PreCommit_Python_PVR_Flink_Cron/lastCompletedBuild/badge/icon)] (https://builds.apache.org/job/beam_PreCommit_Python_PVR_Flink_Cron/lastCompletedBuild/) | --- | --- | ---

github_pulls_comments:

1. @kennknowles
2. I think @chamikaramj is probably best for review, not me.
3. This pull request has been marked as stale due to 60 days of inactivity. It will be closed in 1 week if no further activity occurs. If you think that’s incorrect or this pull request requires a review, please simply write any comment. If closed, you can revive the PR at any time and @mention a reviewer or discuss it on the dev@beam.apache.org list. Thank you for your contributions.
4. @steveniemitz can you please rebase and update it to pass the tests. @chamikaramj can you PTAL this looks like a nice change and is already stalled because of review.

5. Sorry about the delay. So IIUC, for each bundle (not window) a null value will be emitted as the output ? Since this signals that part of a data has been written, not all data, how do you think users will use this notification ? Also, please update documentation and add unit tests. Also make sure to test this with a real BT cluster and/or add an integration test (optional).
6. I mean adding an integration test part is optional :)
7. hah I had forgotten I even had this PR open still...answers are inline. > Sorry about the delay. > > So IIUC, for each bundle (not window) a null value will be emitted as the output ? Both really, for every bundle that arrives at the BigtableIO, when all writes in the bundle complete, it will emit a single Void (in this case it is null, but the value isn't important) for every window in the bundle. > Since this signals that part of a data has been written, not all data, how do you think users will use this notification ? I included an example of how it would be used above in my initial comment, was that unclear? The typical use-case would be to do something after a window closes and all writes have completed. > Also, please update documentation and add unit tests. Also make sure to test this with a real BT cluster and/or add an integration test (optional). I'll add unit tests and rebase this soon. We've been using this in production for months now, so it's gotten battle tested pretty well.
8. Right, but given that you emit the value at "finishBundle" there'll be one null emitted per bundle (not per window) ?
9. I mean for every window seen by every bundle. A given window may have elements in multiple bundles.
10. We should clearly document this behavior with examples in BigTableIO.java
11. > I mean for every window seen by every bundle. A given window may have elements in multiple bundles. Correct. One element per bundle+window, so there will likely be many elements per window total. Using the `Wait.on` transform handles this for you and will only fire once the window closes.
12. I ignore most of the inner working of the `Wait` transform but after a quicklook at other Writes that return collections e.g. `JdbcIO`, `TextIO`, `SpannerIO` I do not see any managing windowing explicitly. I suppose this is done to do the notification per bundle+window but I am wondering if this can have a consequence when using this change to do `Wait.on`. Do you have any comment on this @chamikaramj ?
13. Wondering this because it is not clear to me why we want to care about the bundle and not just about the Window like `Wait` does. And maybe we can just simplify and behave like the others Write transforms do.
14. Many of the IOs that did this I had looked at rewindow into the global window, so there isn't a need to manage the windows inside the IO. This doesn't (because, at least for my use case, I need to preserve the windows), so it needs to track them internally.
15. @steveniemitz From the description ` A common use-case of this is to wait for all writes to finish, and then do something else` isn't this what basic `Wait` does? What is the use case of doing this per bundle? (just trying to understand).
16. > @steveniemitz From the description ` A common use-case of this is to wait for all writes to finish, and then do something else` isn't this what basic `Wait` does? What is the use case of doing this per bundle? (just trying to understand). I don't see any great use-case for per-bundle, but `Wait.on` needs some kind of trigger, which is what this provides (you can't Wait.on a PDone). If there were a simple way to have this only emit one element per-window, that'd be cool, but I can't think of anything efficient off the top of my head that wouldn't involve a reshuffle. For example, in our topologies, we hook up a `Sample.any` after the BigtableIO.Write (and remove all triggers from the input), which gives us a single firing when the window closes.
17. Have you tried to just return the type without any Windowing changes? I am surprised to see that [JdbcIO can achieve this] (https://github.com/apache/beam/blob/d754094802c4c8767e4e2289563398d96e126d16/sdks/java/io/jdbc/src/main/java/org/apache/beam/sdk/io/jdbc/JdbcIO.java#L9 without doing any extra stuff. [Tests as reference] (https://github.com/apache/beam/blob/d754094802c4c8767e4e2289563398d96e126d16/sdks/java/io/jdbc/src/test/java/org/apache/beam/sdk/io/jdbc/JdbcIOTest.java#
18. Hmm, are you suggesting just changing it to return a `PCollection[Void]` but never actually outputting anything? I think I did try that before and it didn't work correctly in streaming, but it would have been awhile ago so I don't remember the specifics. I can give it a try again at some point.
19. I was surprised too by JdbcIO not returning anything and even wondering if it works correctly but the test looks ok. After your comment i wonder if this could be a Streaming only issue but not sure.
20. Actually other transforms that return on Write return Void for the success case too, e.g. [SpannerIO.Write] (https://github.com/apache/beam/blob/c6d38fa31afa7de4f28abe1c901083bf21c43422/sdks/java/io/google-cloud-platform/src/main/java/org/apache/beam/sdk/io/gcp/spanner/SpannerIO.java#L1204) without further rewindowing.
21. interesting, I'll give it another try just changing the type to return void. If it works, that'll really simplify things.
22. I still have trouble understanding how people will actually use this (even with the example above). Basically users will see a stream of Voids/nulls but how can they act on that without mapping that to the actual data that was written ? For example, for file-based IO I believe we return a PCollection names of files that were actually written.
23. I have the impression it works because it is used just to signal, and in the case of Wait the only thing that matters is a signal to know we can continue (well the accumulated signals + window).
24. Maybe a 'richer' approach is to save more info as `Spanner.Write` but maybe Void could be a valid first approach for the intended use case.
25. so thinking back a little more I realized something else. While we have been focused on the `Wait.on` use case, we're also using the output of this to feed directly (via a `Sample.any`) into another DoFn. If the bigtable DoFn didn't emit any elements, the DoFn after it would never be invoked. The topology looks something like: ``` input .apply(BigtableIO.write(...)) //write some stuff .apply(Sample.any()) // wait for the window to close .apply(ParDo.of(new SomeDoFn()) ``` In order for SomeDoFn to actually be invoked, BigtableIO.write needs to emit (at least one) element.
26. Yeah, which also means that that the solution which does not output anything will not work for streaming mode. I think it'll be good to output some information about batches of mutations that were written for a given bundle+window combination instead of not outputting anything or outputting just null. How about just updating this recordsWritten counter to be per window and outputting that ? https://github.com/apache/beam/blob/master/sdks/java/io/google-cloud-platform/src/main/java/org/apache/beam/sdk/io/gcp/bigtable/BigtableIO.java#L702 Another option might be to output failed records. But this should be optional since if everything get written to BigTable without errors, nothing will get output from the write step. BTW it might be worth summarizing this discussion in the dev list.
27. > How about just updating this recordsWritten counter to be per window and outputting that ? > https://github.com/apache/beam/blob/master/sdks/java/io/google-cloud-platform/src/main/java/org/apache/beam/sdk/io/gcp/bigtable/BigtableIO.java#L702 Is this really any more useful than a `null`? I can't think of how anyone would ever use the number for anything other than metrics (and there are other ways to get it in that case). > Another option might be to output failed records. But this should be optional since if everything get written to BigTable without errors, nothing will get output from the write step. The way the IO is designed mutations never fail, either the whole batch succeeds or it retries until it does (or the pipeline fails). ;)
28. > > How about just updating this recordsWritten counter to be per window and outputting that ? > > https://github.com/apache/beam/blob/master/sdks/java/io/google-cloud-platform/src/main/java/org/apache/beam/sdk/io/gcp/bigtable/BigtableIO.java#L702 > > Is this really any more useful than a `null`? I can't think of how anyone would ever use the number for anything other than metrics (and there are other ways to get it in that case). > I think number of records written probably combined with the spanner instance ID might be a useful signal but I agree that this is debatable. For example, this will allow us to compute the number of records written to each spanner instance if we update SpannerIO to support dynamic destinations. At least consider returning some sort of a metadata object instead of null which will allow us to extend this functionality without breaking existing pipelines. > > Another option might be to output failed records. But this should be optional since if everything get written to BigTable without errors, nothing will get output from the write step. > > The way the IO is designed mutations never fail, either the whole batch succeeds or it retries until it does (or the pipeline fails). ;) Yeah, you are correct. So this is not an option.
29. > At least consider returning some sort of a metadata object instead of null which will allow us to extend this functionality without breaking existing pipelines. This seems reasonable. Maybe we just make a `BigtableWriteResult` object for now with just the instance info, and can update it w/ more metadata as needed in the future?
30. Yeah, that makes sense. Thanks.
31. @chamikaramj one question, looking at the [SpannerWriteResult implementation] (https://github.com/apache/beam/blob/c6d38fa31afa7de4f28abe1c901083bf21c43422/sdks/java/io/google-cloud-platform/src/main/java/org/apache/beam/sdk/io/gcp/spanner/SpannerWriteResult.java#L42) they are also returning PCollection<Void> but wrapped, if we apply the same approach in other IOs e.g. JdbcIO will it unblock the streaming case mentioned above?
32. And question 2: `SpannerWriteResult` does not seem to be `Serializable` or to have an associated `Coder` so I am wondering what will happen once you connect another transform, isn't this an issue too?
33. `SpannerWriteResult` is itself a POutput, I think the name is a little confusing. I was suggesting a `PCollection<BigtableWriteResult>` for the bigtable writer.
34. Agree with what @steveniemitz said. BTW will it make sense to group these objects so that we only output one per window not one per batch ? Also it might also make sense to introduce something like 'SpannerWriteResult' so that we only output these objects if users explicitly ask for additional output. (specially if we do grouping due to the cost of additional shuffle). I think we should summarize the discussion here in dev list to in case others have additional opinions. This might end up being a pattern for other sinks.

35. Yes @chamikaramj this is worth a discussion in dev@ because we are doing it differently in each IO (Spanner, Jdbc, TextIO and now here) so probably we should agree on a common ground that works correctly for all cases. Can you please submit the subject to the ML, probably worth to cc @jkff because his advice could be definitely worth.
36. > BTW will it make sense to group these objects so that we only output one per window not one per batch ? Also it might also make sense to introduce something like 'SpannerWriteResult' so that we only output these objects if users explicitly ask for additional output. (specially if we do grouping due to the cost of additional shuffle). I don't think it's worth adding the GBK directly into the transform. Users can either: - Use Wait.on directly with this, which already does a shuffle itself - Use the output directly, to handle when each bundle is committed - Can just add a Sample.any themselves to get only a single firing when the window closes. Adding the GBK in the transform just removes flexibility for no real gain in simplicity imo. Also, there have been a couple discussions on the mailing list about this: https://lists.apache.org/thread.html/ddcdf93604396b1cbcacdff49aba60817dc90ee7c8434725ea0d26c0@%3Cuser.beam.apache.org%3E https://lists.apache.org/thread.html/8d5970c101c14c5b85fe0d3b53aaab169ab067ceb84941f4f69e4f44@%3Cdev.beam.apache.org%3E I'm not sure if there is a "one size fits all" solution, because many IOs operate differently (some can fail single elements, some rewindow into the global window, etc), but I agree it'd be good to try to set a standard.
37. Agree it is hard to have a one solution for all cases, but we can give recommendations for the given cases and update the [PTransform style guide](https://beam.apache.org/contribute/ptransform-style-guide/) with this info.
38. what's the status here? I just fixed that spotless error, is this good to land?
39. ping?
40. Looks like there are some comments from @jkff that have not been addressed ? I think we can get this in with the experimental tag when the comments are addressed with the discussion pending to decide on the recommended approach for Beam. @iemejia do you agree ?
41. > Looks like there are some comments from @jkff that have not been addressed ? I think those have been resolved.

github_pulls_reviews:

1. Just a heads up, this is a backward incompatible change, it will break people's code of the form: ``` public class MyTransform extends PTransform<..., PDone> { expand { return blah.apply(BigtableIO.write()); } } ``` Up to you whether this is a big concern or no. If you want to make it compatible, follow the example of TextIO.write().withOutputFilenames() (or something like that).
2. Seems like this would be much cleaner in terms of Java readability as a simple `HashMap<BoundedWindow, Instant>`. However, looking at code below, you just need a `HashSet<BoundedWindow>`, and then use window.maxTimestamp() instead of the timestamp of a non-deterministically chosen element within the window.
3. the class is marked as experimental, which means it can have backwards-incompatible changes right? ;) I'd be ok with either way, but if we're going by the letter of the `@Experimental` rules, it seems like it'd be cleaner to just change the interface rather than adding a new option.
4. hmm, yeah that seems reasonable!
5. Thanks @jkff for pointing out that this is backwards incompatible. @steveniemitz even though this is marked experimental there are many of users who use this connector. So it's preferable if we can not break user pipelines (which can be easily achieved with a slight change here).
6. hah, I was mostly just teasing since like 50% of beam is marked as experimental at this point. What's the path forward here for merging this in, if I make this change are we good to go?
7. Please document what this means.
8. I think it's better to aggregate records per window and output the number of records written for a given window. Otherwise if a following step aggregate values across windows they will see a total more than the total number of records written. WDYT ? Either way we should clearly document what we are outputting here.
9. Please add a comment.
10. Let's add a "BigTableIO.write().withWriteResults()" method that return the updated transform so that this change does not break backwards compatibility. Also, please add documentation there describing this feature.
11. I think we kind of talked about this before. If someone wanted that metric, they could do so using a Sum on the output of this. I don't see a good reason to do that in here, doing so would require a GBK on the output which I don't think this transform should force onto people.
12. I agree with @chamikaramj here: as-is, the returned value is kinda useless because it can't be correctly aggregated - if we see 3 records in W1 and 5 records in W2 in a bundle, the current code will output (8, W1) and (8, W2) with no way for a follow-up Sum to untangle them back. You don't need a GBK, you just need a `HashMap<BoundedWindow, Integer>`.
13. Please add a test that demonstrates usage of this API with Wait.on(), as this is its primary use case.
14. I think you don't need TestStream for this test, you just need Create.timestamped(). Would be good to also make this test a little more involved to catch the issue with correct counting per window.
15. I'd rather just go back to outputting null at this point...if you used a map of window -> counts you'd need to then figure out when the correct time to expire the items from the map was as well. edit: oh maybe I misunderstood the suggestion, you're saying keep a count that resets each bundle, but tracks the counts per window?
16. Not sure what's the issue with expiration here? Just clear the map at the beginning of the bundle; you get 1 result per window per bundle. However if you feel doubtful, then I'm also fine with removing BigtableWriteResult and getting back to outputting null, but please make it a `PCollection<?>` rather than `PCollection<Void>` so that later we can reintroduce BigtableWriteResult without breaking compatibility.
17. alright, I'll pick this up later when I have more free time. It'll be a bunch of refactoring to plumb this around like TextIO does, essentially copying all the options onto another transform and proxying them through.
18. I think you don't necessarily need to copy all the options. You'd only need that if you want to support code like "BigTableIO.write().withWriteResults().withTable()..and so on", but if you're ok with allowing .withWriteResults() only in the last position on the builder, then you can just capture the whole original Write object, and the object returned by withWriteResults() doesn't need any more setters of its own. It's still a bit of refactoring, but not as much boilerplate.
19. should I just remove the `@Experimental` annotation and comment while I'm in here? If we're saying that we can't make backwards-incompatible changes to the transform, I'm not sure what use having that in here still is. Also, Bigtable came out of beta in 2016 so that note is a little out of date too :P
20. Yeah, I think it makes sense to remove the experimental annotation from BigTable connector since it has been around for some time. Beta reference should definitely be removed :) cc: @aaltay @sduskis
21. Is really a PR that is literally changing the current API the best place to decide that we should remove the `Experimental` characteristic of the IO? I mean the APIs have been very stable but this decision is probably worth of a different issue, and given the uncertain future of Writes with return type I think probably it is wise to let it at least for the Write signature for some time.
22. Also with the ongoing work on [BEAM-7615](https://issues.apache.org/jira/browse/BEAM-7615) I am not even sure the API will stay stable for the Read part too, so definitely -1 on any API stability annotation change at least for this PR.
23. thats fine with me...I just want to land this at this point :P
24. I think we can remove the experimental tag from stable transforms (Read, Write). This PR does not break backwards compatibility and I don't think we should break backwards compatibility for these transforms in future PRs either. This does not prevent adding new features or adding a new "ReadAll" form transform when we have SDF. I agree that this is better done in a separate JIRA/PR. Created https://issues.apache.org/jira/browse/BEAM-7631.
25. I am curious about this comment. I understand the logic of it, but I am curious if we can tackle the case of triggering the 'next' part after full writing a PCollection, aka the Batch case. Any hints @jkff ? (I thought this already worked also for that case or maybe I just misread it)
26. You are right about backwards compatibility @chamikaramj, I had in mind that we were still changing the return time so it is ok. `BigtableIO.ReadAll` may require refactors, but it can indeed be still backwards compatible as we did for `HBaseIO.ReadAll`. So probably good to go then in the other issue. I think we can do that for HBaseIO too since it has barely changed in 2 years (apart of the ReadAll addition.
27. We should probably mark `BigtableIO.ReadAll` experimental for a bit of time. Just created https://issues.apache.org/jira/browse/BEAM-7633 for the HBaseIO case.
28. If we go with this change, how about returning PCollection<?>. This means we will allow ourselves, in a forwards compatible way, to return something other than Void (e.g. some kind of a status about what was written) in the future.
29. Can you add the `@Experimental` annotation here too since this is public API.
30. Mmm actually after looking at the [Wait javadoc](https://github.com/iemejia/beam/blob/1ad61fd384bcd1edd11086a3cf9d7dddb154d934/sdks/java/core/src/main/java/org/apache/beam/sdk/transforms/Wait.java#L68 I have the impression the global window is covered too but it requires the input to be bounded. So maybe this comment is not needed. `If signal is globally windowed, main input must also be. This typically would be useful only in a batch pipeline, because the global window of an infinite PCollection never closes, so the wait signal will never be ready. `

31. @robertwb may have a point but it `?` have other trade-offs too. I am going to open a discussion thread on the ML around the return types of Write transforms to maybe have other opinions and to not fill this PR with the pros/cons of each approach.
32. Email [sent](https://lists.apache.org/thread.html/d1a4556a1e13a661cce19021926a5d0997fbbfde016d36989cf75a07@%3Cdev.beam.apache.org%3E). @robertwb @chamikaramj does python already have this Write with return pattern in some IOs? Is this an issue there? Or the lack of types makes things ‘easier’?
33. oh yeah, this works for batch...we even use it in batch mode already. I'm not sure why I added that comment. I'll remove it.
34. Thanks for sending the email. Replied there.
35. "Does not modify this object" is redundant, it is true for all methods of all builders in Beam.
36. Could you delegate the regular BigtableIO.write().expand() to this one, to avoid them diverging in the future? (i.e. implement write().expand() as "input.apply(this.withWriteResults()); return PDone.in(input.getPipeline())")
37. Same might hold for other methods - validate, populateDisplayData, toString.
38. Not sure what the conclusion of https://lists.apache.org/thread.html/3b5c96ce91994c19ba1e848b96b8beb0ca2c346a00658b9350013437@%3Cdev.beam.apache.org%3E was, but maybe there's now a way to avoid manually writing this coder? Fine if not, it's trivial enough.
39. 🤷 I was just being consistent with all the other methods in this class. I removed it from this one though
40. I changed everything except for toString to delegate to withWriteResults()
41. yeah, probably could have used DelegateCoder as well since its just one field. I don't know what ever came out of that thread either, but like you said, this is a very simple coder.

jira_issues:

1. **summary:** BigtableIO writes should support emitting "done" notifications
   description: There was some discussion of this on the dev@ mailing list [1]. This approach was taken based on discussion there. [1] https://lists.apache.org/thread.html/949b33782f722a9000c9bf9e37042739c6fd0927589b99752b78d7bd@%3Cdev.beam.apache.org%3E

jira_issues_comments:

1. GitHub user steveniemitz opened a pull request: https://github.com/apache/beam/pull/3997 [BEAM-3061] Done notification for BigtableIO.write() Follow this checklist to help us incorporate your contribution quickly and easily: - [x] Make sure there is a [JIRA issue](https://issues.apache.org/jira/projects/BEAM/issues/) filed for the change (usually before you start working on it). Trivial changes like typos do not require a JIRA issue. Your pull request should address just this issue, without pulling in other changes. - [x] Each commit in the pull request should have a meaningful subject line and body. - [x] Format the pull request title like `[BEAM-XXX] Fixes bug in ApproximateQuantiles`, where you replace `BEAM-XXX` with the appropriate JIRA issue. - [x] Write a pull request description that is detailed enough to understand what the pull request does, how, and why. - [x] Run `mvn clean verify` to make sure basic checks pass. A more thorough check will be performed on your pull request automatically. - [x] If this contribution is large, please file an Apache [Individual Contributor License Agreement] (https://www.apache.org/licenses/icla.pdf). --- This adds support for "done" notification in `BigtableIO.write()`. For each bundle that is completely written, a single "Void" is output by the PTransform. Doing so allows flows to effectively wait for all writes to finish. This works well for batch flows (and I've been using for a few weeks now in some large ones), but there is some limitation on the streaming side. For streaming writes, all "done" notifications are output into the global window, rather than the window(s?) that was(were) written in that bundle. I'm not very familiar with the interactions of bundles, windows, and panes in the streaming API, so I haven't tried to solve this at all, although for this IO specifically it seems doable. You can merge this pull request into a Git repository by running: $ git pull https://github.com/tc-dc/beam bigtable-done-notification Alternatively you can review and apply these changes as the patch at: https://github.com/apache/beam/pull/3997.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #3997 ---- commit b6848d54c4eac2942abed6ce363aaa5ae3f2ba05 Author: steve <sniemitz@twitter.com> Date: 2017-09-25T14:21:40Z Done notification for BigtableIO ----