Item 138
git_comments:

1. SPARK-33532: After SPARK-13883 and SPARK-13989, the parquet read process will no longer enter this branch because `ParquetInputSplit` only be constructed in `ParquetFileFormat.buildReaderWithPartitionValues` and `ParquetPartitionReaderFactory.buildReaderBase` method, and the `rowGroupOffsets` in `ParquetInputSplit` set to null explicitly. We didn't delete this branch because PARQUET-131 wanted to move this to the parquet-mr project.

git_commits:

1. **summary:** [SPARK-33532][SQL] Add comments to a unreachable branch in SpecificParquetRecordReaderBase.initialize method
   message: [SPARK-33532][SQL] Add comments to a unreachable branch in SpecificParquetRecordReaderBase.initialize method ### What changes were proposed in this pull request? This pr mainly adds a comment for the 'rowgroupoffsets! = null' branch in `SpecificParquetRecordReaderBase.init(InputSplit, TaskAttemptContext)` to indicate that spark read parquet process will not enter this branch after SPARK-13883 and SPARK-13989. It is not deleted because PARQUET-131 wants to move `SpecificParquetRecordReaderBase` into the parquet-mr project. ### Why are the changes needed? Add a useful comment. ### Does this PR introduce _any_ user-facing change? No ### How was this patch tested? Pass the Jenkins or GitHub Action Closes #30484 from LuciferYang/SPARK-33532. Authored-by: yangjie01 <yangjie01@baidu.com> Signed-off-by: HyukjinKwon <gurwls223@apache.org>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** [SPARK-33532][SQL] Add comments to a unreachable branch in SpecificParquetRecordReaderBase.initialize method
   body: ### What changes were proposed in this pull request? This pr mainly adds a comment for the 'rowgroupoffsets! = null' branch in `SpecificParquetRecordReaderBase.init(InputSplit, TaskAttemptContext)` to indicate that spark read parquet process will not enter this branch after SPARK-13883 and SPARK-13989. It is not deleted because PARQUET-131 wants to move `SpecificParquetRecordReaderBase` into the parquet-mr project. ### Why are the changes needed? Add a useful comment. ### Does this PR introduce _any_ user-facing change? No ### How was this patch tested? Pass the Jenkins or GitHub Action

github_pulls_comments:

1. **[Test build #131659 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/131659/testReport)** for PR 30484 at commit [`83b85d4`](https://github.com/apache/spark/commit/83b85d4c1101a39e0b85b41d54862badf4247ad5). * This patch **fails Spark unit tests**. * This patch merges cleanly. * This patch adds no public classes.
2. **[Test build #131724 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/131724/testReport)** for PR 30484 at commit [`6cbd321`](https://github.com/apache/spark/commit/6cbd32172abf384c5d8a05d09b256fcc78e6e33f). * This patch **fails PySpark unit tests**. * This patch merges cleanly. * This patch adds no public classes.
3. **[Test build #131733 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/131733/testReport)** for PR 30484 at commit [`a0859e7`](https://github.com/apache/spark/commit/a0859e75e68d0fd248e39a648cdd715c8356a132). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
4. @HyukjinKwon Is it appropriate for us to clean up this code branch now, or do it together when upgrade Apache Parquet version
5. **[Test build #133175 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/133175/testReport)** for PR 30484 at commit [`d210aa4`](https://github.com/apache/spark/commit/d210aa4cb25359b6cadf6766fee2c47499a6c0aa). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
6. @LuciferYang I am very sorry but do you mind pointing out which commit added that codes and removed the usages? It would be much easier to review with that.
7. > @LuciferYang I am very sorry but do you mind pointing out which commit added that codes and removed the usages? It would be much easier to review with that. @HyukjinKwon OK, let me investigate ~ I think it's a very interesting thing ~ haha ~
8. The branch to be deleted in current pr is from SPARK-11787,. In SPARK-11787`SqlNewHadoopPartition` create `UnsafeRowParquetRecordReader` when `spark.parquet.enableUnsafeRowRecordReader` is true and data format is `org.apache.parquet.hadoop.ParquetInputFormat` The key point is the `ParquetInputSplit` pass to `UnsafeRowParquetRecordReader.init` method, if `rowGroupOffsets` in `ParquetInputSplit` is not null will enter the code branch which I want to delete in this pr. `ParquetInputSplit` produce by `SqlNewHadoopRDD#getPartitions` use `ParquetInputFormat.getSplits` method. https://github.com/apache/parquet-mr/blob/32c46643845ea8a705c35d4ec8fc654cc8ff816d/parquet-hadoop/src/main/java/org/apache/parquet/hadoop/ParquetInputFormat.java#L287-L306 ![image](https://user-images.githubusercontent.com/1475305/102896567-92e77d80-44a1-11eb-802c-c778e573a952.png) The `else branch` will create a `ParquetInputSplit` and `rowGroupOffsets` not null, further calls are as follows: ``` ParquetInputFormat#getSplits(JobContext) -> ParquetInputFormat#getSplits(Configuration,List<Footer>) -> ClientSideMetadataSplitStrategy#getSplits -> ClientSideMetadataSplitStrategy#generateSplits -> ClientSideMetadataSplitStrategy.SplitInfo#getParquetInputSplit ``` https://github.com/apache/parquet-mr/blob/32c46643845ea8a705c35d4ec8fc654cc8ff816d/parquet-hadoop/src/main/java/org/apache/parquet/hadoop/ParquetInputFormat.java#L635-L649 ![image](https://user-images.githubusercontent.com/1475305/102896642-af83b580-44a1-11eb-9daa-002efa18818e.png)
9. The first key change from SPARK-13883, ParquetRelation.buildReader method always use `null` to construct `ParquetInputSplit` and pass it to UnsafeRowParquetRecordReader.init method. https://github.com/apache/spark/pull/11709/files#diff-ae45608e2a6173960880aceaeedac157bc9fb30d0a7a6259c152cf33ce6d9fa7R350-R365 ![image](https://user-images.githubusercontent.com/1475305/102894293-d50ec000-449d-11eb-861c-ca90d4db88cf.png)
10. Then SPARK-13989 remove `VectorizedParquetRecordReader` create logical from `SqlNewHadoopRDD`, SPARK-14535 remove `buildInternalScan` from `FileFormat` and SPARK-14596 remove `SqlNewHadoopRDD` from spark code. After these changes, `VectorizedParquetRecordReader` only create by `parquet.DefaultSource#buildReader` method and the `rowGroupOffsets` in `ParquetInputSplit` always `null`. So I think `rowGroupOffsets != null` branch in `SpecificParquetRecordReaderBase.init` method is unreachable now.
11. At least for testing purposes here, you could add a check that it isn't null. Then at least tests would tell us whether that is true in tests. We may or may not leave in the check as an assert or something.
12. Kubernetes integration test starting URL: https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder-K8s/37881/
13. Kubernetes integration test status failure URL: https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder-K8s/37881/
14. **[Test build #133284 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/133284/testReport)** for PR 30484 at commit [`2a57f30`](https://github.com/apache/spark/commit/2a57f3094ae33306b1e7cfcd1919ec4dab98da03). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
15. Kubernetes integration test starting URL: https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder-K8s/38010/
16. Kubernetes integration test starting URL: https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder-K8s/38009/
17. Kubernetes integration test status success URL: https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder-K8s/38010/
18. Kubernetes integration test status success URL: https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder-K8s/38009/
19. Thanks @LuciferYang. LGTM
20. Thx ~ @HyukjinKwon @srowen
21. I manually tested via `./dev/lint-scala`. Merged to master.
22. > I manually tested via ./dev/lint-scala. > > Merged to master. @HyukjinKwon thx ~
23. **[Test build #133419 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/133419/testReport)** for PR 30484 at commit [`bd793cf`](https://github.com/apache/spark/commit/bd793cffab3e7df5ce87240fc25dab3ff47b1f79). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
24. **[Test build #133418 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/133418/testReport)** for PR 30484 at commit [`afa1bf0`](https://github.com/apache/spark/commit/afa1bf0aa13cc63e94831a6f0e7ed773328ba461). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.

github_pulls_reviews:

1. Yeah, let's add an assert. That sounds like a good idea.
2. OK~ Will do it later. a little busy at the moment
3. 2a57f30 add a assert use `Preconditions.checkState`
4. Why don't you just simply use `assert`?
5. `assert` in Java code depends on `-ea` flag to enable, we can't guarantee users execute spark apps with this flag, or do we just want to ensure this `assert` works in UTs?
6. Either is reasonable here. If it should never be true in any execution of Spark, we assert it. If user input or state can cause it to not be true, we need something besides an assert. Most of the point is verifying this at test time, yes.
7. For the current spark code, I think `split.getRowGroupOffsets()` should always be null because after SPARK-13883 and SPARK-13989 `ParquetInputSplit` is created manually in `ParquetRelation(ParquetFileFormat) ` and set `rowGroupOffsets` to null explicitly
8. https://github.com/apache/spark/blob/61881bb6988aa0320b4bacfabbc0ee6f05f287cb/sql/core/src/main/scala/org/apache/spark/sql/execution/datasources/parquet/Par L267
9. https://github.com/apache/spark/blob/61881bb6988aa0320b4bacfabbc0ee6f05f287cb/sql/core/src/main/scala/org/apache/spark/sql/execution/datasources/v2/parquet/ L134
10. After rethinking @LuciferYang, actually this is aimed to be ported to Apache Parquet, see the comments above: ``` * TODO: move this to the parquet-mr project. There are performance benefits of doing it * this way, albeit at a higher cost to implement. This base class is reusable. ``` It's being tracked at [PARQUET-131] (https://issues.apache.org/jira/browse/PARQUET-131). Shall we just simply add a comment that this code is not used in Apache Spark for now?
11. Add a comment and revert the code change?
12. @HyukjinKwon But PARQUET-131 seems to have made no progress in the last three years ... , anyone really do it ? I'm just worried about whether this TODO will really be completed, haha :)
13. Yeah, just simply adding a comment because this is planned to move to Parquet. It's stuck but I think we should port it in long run anyway.
14. OK ~
15. done ~

jira_issues:

1. **summary:** Vectorized Reader In Parquet
   description: Vectorized Query Execution could have big performance improvement for SQL engines like Hive, Drill, and Presto. Instead of processing one row at a time, Vectorized Query Execution could streamline operations by processing a batch of rows at a time. Within one batch, each column is represented as a vector of a primitive data type. SQL engines could apply predicates very efficiently on these vectors, avoiding a single row going through all the operators before the next row can be processed. As an efficient columnar data representation, it would be nice if Parquet could support Vectorized APIs, so that all SQL engines could read vectors from Parquet files, and do vectorized execution for Parquet File Format. Detail proposal: https://gist.github.com/zhenxiao/2728ce4fe0a7be2d3b30

jira_issues_comments:

1. Hi, Thank you very much for creating this! I sincerely appreciate you taking the type to create this proposal! From the Hive side, I have the following feedback: My understanding is that {{ColumnVector}} is an interface so we can provide our own impl. This will be required for Hive since we have our own {{ColumnVector}} impl and it's extremely widely used. I don't think this version of the {{ColumnVector}} interface will provide pluggability for the following reasons: # Impls e.g {{LongVector}} have public members. This same thing was done in Hive (not use getters and setters) but IMO for dubious reasons. No proof was provided that shows JIT does not optimize the getters setters out. # Drill, Hive, etc will be required to extend {{LongVector}} in order to make this work, but that would require massive change on the Hive side. We should provide getters and setters on the interface for the data types so that Hive can simply implement the {{ColumnVector}} interface with our existing implementation. We might also need to provide {{isLongVector}} methods so we know the type of the {{ColumnVector}}. # I don't understanding why {{ColumnVector}} has an {{getEncoding}}. Isn't an encoding a storage feature not a column vector feature?
2. [~brocknoland] Thanks a lot for the comment. I will update the ColumnVector interface, with getters and setters, and getTypes. And all primitive type Vectors are left for implementations. Yes, we are keeping Encoding information in ColumnVector for Presto do lazy materialization. Presto does not materialize the vector until it finishes the filer. It is OK not using this Encoding information.
3. Few thoughts: - I agree with Brock's general comments about having to avoid a parquet canonical representation of the in memory data structure. - For the getter/setters, we need to support bulk transfer and primitive transfer. - We should avoid copy unless necessary. For example, in Drill we often choose to avoid copying the variable length data, instead choosing to use it as is. - The interface should also take in column level filter expression evaluator. Again, this should be a no copy interface. While you may think that with vectorized reads, this isn't necessary, we've found that it actually depends entirely on the selectivity of the filter and whether you are using dictionary encoding. I also would suggest that this be a replacement for the lower layers of the Parquet reader rather than a secondary path. Otherwise, we're always going to have a partial implementation. We're very engaged in trying to think through the ideas here and are definitely going to be pushing this along. One last thought, I'm not entirely convinced that this should be a column at a time interface. I've been thinking that a batch of records at a time is more appropriate. Otherwise, there are too many internal concerns that have to be reimplemented and fancy inter column behaviors have to be implemented multiple times (as well as complex data support). On the flip side I'm not sure any other engines currently have vectorized readers for complex data but I think we're more than happy to push it that direction alone and people can fall back to a higher-level non-vectorized read interface for complex data.
4. [~brocknoland] The gist is updated with ColumnVector interface. We are still discussing with the Drill team about whether to use Primitive Arrays, or ByteBuffer, or byte[] for setters and getters. [~jnadeau] I just updated the gist with a ByteBuffer, hoping both Drill, Hive and Presto could use this kind of generalized ByteBuffer. I will spend time reading Drill's code to see other magics. While, some relevant articles seems showing ByteBuffer is not as efficient/fast as primitive arrays: http://www.evanjones.ca/software/java-bytebuffers.html https://groups.google.com/forum/#!topic/mechanical-sympathy/9I18sXm4bvY http://imranrashid.com/posts/profiling-bytebuffers/ Still thinking primitive arrays could be the most efficient way. Anyway, let's continue discussing about it.
5. Hi [~zhenxiao], [~brocknoland], [~jnadeau] I am working on HIVE-8128 and find here. Thank you for creating and discussing this. From Hive perspective, hope below feedback could help. 1. The current code implementation of vectorization in Hive for reference. * In HIVE-4160, it mainly use data structure {{VectorizedRowBatch}} and {{ColumnVector}} to feed the vectorized sql engine. * For {{VectorizedRowBatch}}, it has an array of {{ColumnVector}} to hold data of each column. And has an int size to indicate the number of rows in this batch. * For {{ColumnVector}}, it has some booleans like noNulls and isRepeating, which help the engine skip some data. Also its subclass representing concrete type (e.g. Long) holds an array of primitive data. * To generate the {{VectorizedRowBatch}}, the reader (of ORC file) was added a new method nextBatch(), which delegate each column to its type-suitable vectorized reader to load data. Similiar with the VectorReader in Zhenxiao's design. 2. A few thoughs. * I agree with Jacques's comment about build a batch of records at a time. Maybe a class {{ParquetRowBatch}} could be added to hold the columns. * {{ColumnVector}} could has the boolean indicators about null or repeating values, as they are computed and set during extracting and building data from storage layer. These values in vector provide useful info to sql engines. * How about give a length to the VectorReader? Maybe there is a possibility that sql engine wants to specify the rows fetched in a batch. * A rough idea: add a readPatch() method in {{InternalParquetRecordReader<T>}}. And when vector mode is on, reader will invoke this method to get {{ParquetRowBatch}}. The sql engine like Hive, Drill will convert this batch to the type they need. Primitive arrays in the vectors of batch might make conversion efficiently. The conversion procedure is reading values in {{ParquetRowBatch}} and set them to XxxRowBatch object, which is sent to sql engine. I will keep going on this work to make things more detailed and joining the discussion.
6. [~dongc], I believe in Hive {{ColumnVector.isRepeating}} is used when it's a partition column. Is that your understanding?
7. Hi [~brocknoland] bq. in Hive ColumnVector.isRepeating is used when it's a partition column I think using for a partition column is one case, and it can also used for normal columns, if the values are same in one column. In Hive, when the {{VectorExpression}} in {{Operator}} consume the {{ColumnVector}}, it will check {{isRepeating}} first. If true, it will skip the array loop and just fetch the 1st element in the vector for computation.
8. Hi, After digging into the code more, I get some thoughts based on the design proposal. In order to describe these thoughts clearly, I uploaded a doc {{Parquet-Vectorized-APIs.pdf}}. The general ideas are: * Parquet internal readers read one row at a time now. I think we don't have to add a series of Readers for vectorization. Maybe we could use these existed readers and just add methods like {{readBatch(T next, int size)}}. * ColumnReader.Binding is responsible for binding low level ValuesReader to the customized record Converter materializing records. We can add new concrete Binding classes in Parquet and new customized Converter classes in SQL engine like Hive, Drill. Then the loaded raw primitive data could be materialized to records in the representation SQL engine expecting. This solution could decouple Parquet iterative raw data reading and SQL engine vectorized records materialization. Parquet will not have to organize the primitive data by itself. It just load the data iteratively for vectorization usage. SQL engines could organize the data as they like.
9. Briefly describe how Presto works with Parquet

10. Hello [~dongc], Today the informal elements of the "parquet vectorization" team in the US met. This included myself, Zhenxiao, Daniel, and Eva for PrestoDB, Parth Jason for Drill, and [~spena] and myself for Hive. I of course thought to invite you but the rest of the team wanted an on-site and I know it's very late in China... h2. Questions *Why does presto read api specify ColumnVector. Do they read one column at a time?* Presto has code which reads all columns in a loop, thus they don't need the batch API. *Original API specified encoding, does the reader use the encoding to materialize?* ColumnVector will not expose Encoding and won't materialize values until getter is called or initialize is called. *Does presto DB use ByteBuffers or primitive arrays (long[], etc)?* They use primitive arrays, like Hive. Drill uses native Buffers. *If API is not going to materialize and gives back raw Buffer, is there any strategy for converting that to long array without copying?* We'll pass in allocator which allocates appropriate Buffer type. Presto and Hive will allocate instances of for exmaple {{LongBuffer}} which gives us access to the primitive array. h2. Next Steps # Update interface to remove Encoding as getters will materialize # Add allocator interface # Netflix will hack together POC (Drill and Hive might do POC on top of this POC) # GSOC byte buffer patch is a requirement, thus we should merge soon. # Finish implementation of Parquet Vector* classes (part of POC) # Finish Drill, Presto and Hive implementation [~dweeks-netflix] - in the meeting it was said that merging the GSOC buffer patch ([PR 49?|https://github.com/apache/incubator-parquet-mr/pull/49]) depended on doing some parquet releases such as mr 1.6/1.7 and format 2.0. I chatted with [~rdblue] and he wasn't sure what that would be?
11. [~dweeks-netflix] actually looks like [PR 6|https://github.com/apache/incubator-parquet-mr/pull/6]. Any thoughts on why you feel releases are required?
12. Thanks [~brocknoland], PrestoDB team, and Drill team for the progress and plan! Adding the allocator interface is a good idea. Looking forward to the POC. And hope I could help on Hive part then.
13. Hi guys, I'm a Tajo (tajo.apache.org) guy. We would also like to participate in this work. Is the current progress POC? We'll share the progress too here.
14. [~jaltekruse] [~parthc] Do you have the allocator interface in one of your PRs? May I get a reference to the PR? Thanks.
15. The pull request is here : https://github.com/apache/incubator-parquet-mr/pull/50
16. Hi all, Some time ago I have sent a message to the parquet dev mailing list about our efforts regarding vector support. I also want to share it here in case some of you have missed it. Even though it's still early work-in-progress any feedback is welcome: https://github.com/zhenxiao/incubator-parquet-mr/pull/1 Thanks
17. Hi [~nezihyigitbasi], The work looks good! I built a HIVE POC (HIVE-8128) based on it and worked. I lest some feedback in the PR: https://github.com/zhenxiao/incubator-parquet-mr/pull/1 P.S. The code in this PR seems already to be merged. Sorry if I left the message in wrong place.
18. Hi [~dongc], thanks for the feedback and glad that it worked for your Hive POC.
19. rebased vectorized parquet code against current master: https://github.com/zhenxiao/incubator-parquet-mr/tree/vector
20. Created a PR for the initial implementation of the vectorized reader: https://github.com/apache/parquet-mr/pull/257
21. Hi How is the vectorized reader going on in this JIRA? Thanks.
22. [~dongc] It's still WIP and there is some work to get it merged.
23. Hi, I see that recently there has been no activity on this JIRA. I wonder whether it is due to the lack of time/interest or was there a technical barrier that stopped progress? Also, how do you perceive the state of uncommitted pull requests? Are they worth revisiting or should the feature be rebuilt from the ground up? Thanks, Zoltan
24. Any news on this?