Item 91
git_comments:

git_commits:

1. **summary:** [SPARK-10772] [STREAMING] [SCALA] NullPointerException when transform function in DStream returns NULL
   message: [SPARK-10772] [STREAMING] [SCALA] NullPointerException when transform function in DStream returns NULL Currently, the ```TransformedDStream``` will using ```Some(transformFunc(parentRDDs, validTime))``` as compute return value, when the ```transformFunc``` somehow returns null as return value, the followed operator will have NullPointerExeception. This fix uses the ```Option()``` instead of ```Some()``` to deal with the possible null value. When ```transformFunc``` returns ```null```, the option will transform null to ```None```, the downstream can handle ```None``` correctly. NOTE (2015-09-25): The latest fix will check the return value of transform function, if it is ```NULL```, a spark exception will be thrown out Author: Jacker Hu <gt.hu.chang@gmail.com> Author: jhu-chang <gt.hu.chang@gmail.com> Closes #8881 from jhu-chang/Fix_Transform.

github_issues:

github_issues_comments:

github_pulls:

1. **title:** [SPARK-10772][Streaming][Scala]: NullPointerException when transform function in DStream returns NULL
   body: Currently, the `TransformedDStream` will using `Some(transformFunc(parentRDDs, validTime))` as compute return value, when the `transformFunc` somehow returns null as return value, the followed operator will have NullPointerExeception. This fix uses the `Option()` instead of `Some()` to deal with the possible null value. When `transformFunc` returns `null`, the option will transform null to `None`, the downstream can handle `None` correctly. NOTE (2015-09-25): The latest fix will check the return value of transform function, if it is `NULL`, a spark exception will be thrown out

github_pulls_comments:

1. LGTM.
2. I'm not sure this is valid. Looking at how the result of `transform()` is used, `None` means "try computing this again". Why would you want or need to return a `null` RDD? instead of an empty one?
3. Looking at it again, @srowen is right. Returning `None` makes `getOrCompute` think that no RDDs have been generated for a given time (artifact of the fact that `None` is returned when a map has no value for a key). So this really is case of returning an empty RDD from your function.
4. You should not be returning in null in the first place from a transform function. May be we need to document this better. At least, in the code, we should check for whether the returned value is null, and throw a SparkException saying "Transform function return null instead of an RDD; this should be avoided . You can return an empty RDD using RDD.empty, if you dont want to return anything. " I think that should be the right solution here. If you think it makes sense, please update the PR accordingly.
5. I think we'd better document the return value of `compute` function, I saw lots of users in the community they're trying to return `None` instead of `RDD.empty` if there's no data, the definition of this function accepts to return `Option`, but actually `None` will introduce some undefined behaviors. I know it is hard to change the interface, at least we could document the exact meaning of this function to avoid misuse.
6. Absolutely agreed. This is something I wish I had done it right 3 years ago :(
7. There exists a case that user needs to skip the operation after the `transform`, especially, the output operation like saveAsXXXFiles (e.g. snapshot the state of updateStateByKey in every 10 batch interval), of course, user can use the `foreachRDD` to achieve this, but needs a little more coding. If `transform` can return `None`, it can make such usage easier. About the undefined behaviors, I think `RDD.empty` will also introduce some undefined behavior here because the new RDD has no relational with the upstream RDD (use `rdd.filter(_ => false)` can avoid this), so the upstream RDD may be not evaluated, if this RDD in state dstream (`updateStateByKey`, `reduceByKeyAndWIndow`) and after some checkpoint, the state may be not right. And creating empty RDD needs to use spark context, the user needs to pay attention for this if the application recover from some checkpoint: needs a redirect way to get the spark context instead of using the spark context directly. I agree that we can avoid NullPointerException by checking NULL and throwing a spark exception, if we forbid the `None` usage, I can modify the PR to check NULL.
8. @jhu-chang you can write application logic that selectively performs additional operations or not on an RDD depending on whether it's empty. It's just an if statement. Why would the lineage of the empty RDD matter? indeed, it's better if it's known to be empty without recomputing anything. Why would something not be computed -- the point is that you're deciding how to continue based on the result of some RDD that _was_ computed.
9. @srowen Yes, user can write his own logical to deal with the case mentioned last post. But user may still try to use the exist API like `saveAsXXXFiles` first rather than start a new one, even though it is very simple for us. About the lineage, I totally agree with you: user must know that the side affect of empty rdd, but as a junior in spark, he/she may not aware of this (Well document may help him/her), In this aspect, return `None` has no difference with empty RDD if he/she knows what he/she does. So if `None` will introduce undefined behavior, the empty RDD also will, just depends how the user uses @jerryshao
10. @jhu-chang I still don't see why you say a user can write logic to return a null RDD, but can't write an if statement to handle it? a user who can use `DStream` is surely capable of this. I think it's more surprising if you're allowed to make a transformation that returns nothing, yet write a program that tries to use it. What is the problem with an empty RDD? Returning an empty RDD is not the same as what you're suggesting. It's an RDD that has nothing in it. It's meaningful to operate on it further. If the application doesn't wish to, it can write logic to do something different for an RDD.
11. Hi @jhu-chang , can you elaborate at what scenario emptyRDD will introduce undefined behavior, I'm afraid I could not clearly catch what you mentioned. If emptyRDD will introduce unexpected behavior, I think it is a bug should be fixed. But I can assure you `None` will bring in problems, so this patch should be changed.
12. @jerryshao This is an extreme sample, it is not a real case, just to demonstrate the issue of RDD.empty, you can see the dependency increasing all the time and final stack over flow will happen (You can say it is usage error). Do you know which kind of operations/dstreams will introduce undefined behavior with `None` and what's the appearance? ``` ssc.socketTextStream("localhost", 9999).map(x => (x,1)).updateStateByKey((inputs : Seq[Int], oldr : Option[Int]) => { inputs.headOption }).transform(rdd => { println(rdd.toDebugString.split("\n").length) sparkContext.makeRDD(Seq[Int]()) }).print ``` Anyway, I will change the PR to throw exception. P.S. The [queueStream] (https://github.com/apache/spark/blob/master/streaming/src/main/scala/org/apache/spark/streaming/dstream/QueueInputDStream.scala) will return `None` on some cases, do we need to JIRA to track it?

13. I cannot remember very clearly, `count()` may introduce incorrect result when you use `None` instead of `RDD.empty` as I remembered. In the early version some operators actually return `None` instead of `RDD.empty`, that's why this interface is designed like this. But `None` will introduce some problem, I remembered TD fixed several this kind of bugs by changing to empty RDD instead of None. TD might better know this issue.
14. `count` with `None` will report NullPointerException since `None` will transform to `null` in [`transform` function](https://github.com/apache/spark/blob/master/streaming/src/main/scala/org/apache/spark/streaming/dstream/TransformedDStream.scala). Actually, all the functions which uses transform function will face the same issue. @srowen The message has been changed, could you check it again?
15. Lgtm
16. [Test build #1824 has finished](https://amplab.cs.berkeley.edu/jenkins/job/NewSparkPullRequestBuilder/1824/console) for PR 8881 at commit [`d068000`](https://github.com/apache/spark/commit/d068000a234bf47f7d5a38c8a7474ad66c87e086). - This patch **fails Scala style tests**. - This patch merges cleanly. - This patch adds no public classes.
17. [Test build #1833 has finished](https://amplab.cs.berkeley.edu/jenkins/job/NewSparkPullRequestBuilder/1833/console) for PR 8881 at commit [`cba60ed`](https://github.com/apache/spark/commit/cba60ed77e1c4812617667f5d1d3e73e588e9f96). - This patch **fails Scala style tests**. - This patch merges cleanly. - This patch adds no public classes.
18. @jhu-chang can you fix up the style problem that fails the build? looks like whitespace at the end of the lines
19. @jhu-chang Could you fix the style issue and one minor issue that I pointed out. Style issues: ``` [error] /home/jenkins/workspace/NewSparkPullRequestBuilder/streaming/src/test/scala/org/apache/spark/streaming/BasicOperationsSuite.scala:213:0: Whitespace at end of line [error] /home/jenkins/workspace/NewSparkPullRequestBuilder/streaming/src/test/scala/org/apache/spark/streaming/BasicOperationsSuite.scala:221:10: Whitespace at end of line ```
20. Hi, @tdas, I have checked in the fix for those issues, could you check again?
21. LGTM except some small comments :).
22. [Test build #1865 has finished](https://amplab.cs.berkeley.edu/jenkins/job/NewSparkPullRequestBuilder/1865/console) for PR 8881 at commit [`0d660ce`](https://github.com/apache/spark/commit/0d660ce1c8ad953b20ffc78f5056e701f3e45e21). - This patch **fails MiMa tests**. - This patch merges cleanly. - This patch adds no public classes.
23. [Test build #1866 has finished](https://amplab.cs.berkeley.edu/jenkins/job/NewSparkPullRequestBuilder/1866/console) for PR 8881 at commit [`2cc4fab`](https://github.com/apache/spark/commit/2cc4faba0da2f8a137ab3c00ce9da32cbf37126e). - This patch **fails MiMa tests**. - This patch merges cleanly. - This patch adds no public classes.
24. Hi, @srowen , @tdas @jerryshao Do you know what's the reason of this failure?
25. It might be the problem of Jenkins, you'd better run the unit test again.
26. On my local environment, the unit test is ok.
27. I mean you need to trigger the Jenkins test again by typing "Jenkins, retest this please." in Github comment box.
28. Or let the committer trigger the test for you if you don't have such permission.
29. Jenkins, retest this please.
30. [Test build #1871 has finished](https://amplab.cs.berkeley.edu/jenkins/job/NewSparkPullRequestBuilder/1871/console) for PR 8881 at commit [`2cc4fab`](https://github.com/apache/spark/commit/2cc4faba0da2f8a137ab3c00ce9da32cbf37126e). - This patch **passes all tests**. - This patch merges cleanly. - This patch adds no public classes.
31. Merged to master

github_pulls_reviews:

1. nit: The formatting here is pretty weird. Maybe take the transformation method out and separate it out as a `def`, so this method call can be in a single line: ``` def transformMethod(r: DStream[Int]): Option[RDD] = { r.transform { rdd => if (rdd != null && !rdd.isEmpty()) rdd.map(_.toString) else null } } testOperation(input, transformMethod, input.filter(!_.isEmpty).map(_.map(_.toString)), 4, false) ```
2. I don't have a strong opinion on this, but this could be done with `require`. Generates a different exception -- `IllegalArgumentException`, but a lower-level generic exception seems pretty reasonable. Can I suggest tightening the text to something like "Transform function may not return null. Return RDD.empty to return no elements as the result of the transformation."
3. I would say "must not return null". Saying "may not" is a little ambiguous.
4. May not relevant to this PR, it would be better to rearrange the import ordering.
5. "Return RDD.empty instead to represent no element as the result of transformation" ?
6. Do you mean to import the scala lib first and others in alpha order?
7. Yes, you could refer to other source codes to change the import ordering like that.
8. The `SparkException` was in the right place; it's really alpha by package, and then alpha within the package. At least, that's the convention. But the `scala` import can stand alone in a section above these.
9. I am sooo sorry for the correction, but its not RDD.empty, but `SparkContext.emptyRDD()`. Can you fix it? Everything else is LGTM.
10. I reverted the import and move the scala import before the spark import. @srowen @jerryshao could you please check if it is ok?
11. @tdas I corrected the message, could you please check again?

jira_issues:

jira_issues_comments: