Laziness Kills: The Downside of Spark's Lazy Evaluation

Madeleine Schneider
dept. Electrical Engineering and Computer Science
United States Military Academy
West Point, United States
madeleine.schneider@westpoint.edu

Kyle King
dept. Electrical Engineering and Computer Science
United States Military Academy
West Point, United States
kyle.king@westpoint.edu

***Abstract*—Apache Spark sells itself as being the newest, fastest, and easiest tool for big data analysis. Many users, however, find themselves buried in troubleshooting woes as their code moves much beyond a basic analysis. This paper walks through how a seemingly rudimentary sample problem exploded into a plethora of issues surrounding Spark's lazy evaluation and considers why a tool specifically catered to big data could have such a glaring flaw.**

***Index Terms*—Apache Spark, Elastic Map Reduce, Parallel Computing, Lazy Evaluation**

I. Introduction

Apache spark is a unified analytics engine for large-scale data processing. It promises speedup when compared to tools such as Hadoop. It can be used with Python, Java, Scala, SQL, and R. It runs on a standalone cluster mode, Hadoop, Apache Mesos, Kubernetes, or on a cloud service such as an Amazon EC2 cluster [4]. Despite being used by a fairly large community, many users face complications that eventually drive them to other tools.

This paper looks at downfalls with Spark's lazy evaluation method, and how to avoid potential pitfalls. It is broken into three sections. First, it will lay out the example problem solved using Spark. Next, it will review how some of the common issues in Spark's lazy evaluation appeared in this problem. Finally, it will explain what steps were taken to run the spark job.

II. Sample Problem

To test the capabilities of Apache spark, we considered a seemingly basic problem. Using Amazon's open repository of web crawl data, *Common Crawl*, we attempted to look through all of the text on the Internet [1]. The crawl stores text records in WET files in a world readable Amazon S3 bucket [2]. The goal was to read through the files and identify any ".ic.gov" email address on the web. We then stored the domain names that an email address was seen on in a Data Frame [3]. The Data Frame grouped on email address, such that someone could easily find all of the domains where an address appeared. We hoped that looking for classified email addresses would keep the Data Frame small, given their infrequency on the Internet.

Department of Electrical Engineering and Computer Science

III. Lazy Evaluation

Lazy evaluation is a method that Spark uses to reduce the time necessary to execute data structure evaluations [5]. Essentially, Spark will build a plan for best executing changes to a data structure instead of immediately computing every time an operation is called. Theoretically, this should allow it for ideal parallelism and optimize its operations. As its primary data structure, Spark uses RDDs, resilient distributed datasets, or more recently, Data Frames [6]. Operations on RDD's are broken in to two types, transformations and actions.

Transformations, are lazy. This means that when a transformation, such as .groupBy, is called, the data structure itself will not be immediately changed. Instead, Spark maintains a record of what transformations have been called. The plan it creates for execution on an RDD is called a lineage. A lineage includes where the RDD read in the data, if the RDD changes names, conversion to a Data Frame, or any other transformation [7]. It is not until an action, such as .show, is called that Spark actually makes the changes to the data structure.

A. One action after many big data transformations

Lazy evaluation became a problem when reading in all of the WET files from Amazon S3 storage. As described above, transformations do not result in immediate action. If an RDD is being transformed in a loop, where each iteration uses new data, that data will not be read in until the RDD is acted on.

The code, for our problem, originally looped through the files, transforming an RDD each time. It was not until the end of the loop that an action was called on the RDD, saving it in our own S3 bucket. Running this code resulted in a hanging shell. During this time, we also noticed that only the driver node had any load. Spark was building it's execution plan as it saw transformations, but had not yet seen an action that required parallelism, thus it looked as if nothing was occurring.

Given that this loop could take over 15 minutes, we spent a lot of time assuming nothing was happening and trying to find an alternative solution for reading in data. By adding a print statement every 1000 times through the loop, we were at least able to see that it was making progress building its execution plan. When we eventually switched from a loop to a .map on the WET file path array, we had confidence that it was working.

B. Acting inside a loop

An immediate reaction to not seeing progress may be to put an action inside of the loop. This would then pull in and act on data one chunk at a time. This is problematic for a few reasons. For one, it reduces the usefulness of Spark's parallelism. Additional time will be spent sending information back and forth between the executor and driver nodes instead of computing information about the data. This occurs because each time data is read it must be split up and sent to the executors, and each time an action is called, the result must be sent back to the driver.

The truly destructive nature of Spark's lazy evaluation in a loop has to do with an RDD's lineage. The current state of an RDD is not necessarily saved in memory. Thus, when an action is called, Spark may have to recompute the same transformations from the start of the lineage. One way to view this is by looking at the number of stages in each action or loop. This can be seen in the Spark UI. When we ran our code, acting on every 25 loops, the Spark UI would show an additional 25 stages each time. For each loop, the execution time increased, and eventually we reached a Java heap out of memory error.

Spark has an action that breaks a lineage, .checkpoint. This action was applied every five files to try and truncate the lineage and save memory. Despite this, Spark continued to compute on every file in the lineage.

Ultimately, unless you are working on very small datasets, loops with actions inside should not be used in Spark.

C. Union inside a loop

One seemingly harmless transformation to put inside a loop is the union transformation. With this transformation, you can build on an RDD each time through the loop. The loop should run perfectly fine, and with small datasets, there may be no errors. However, RDD lineage, with many files, may again cause problems. When acting on the complete RDD, the lineage may be so long that it causes a Stack-overflow error. To break the lineage, either create a checkpoint every few unions, or use SparkContext.union [8].

IV. Running the full job

The final job was run on a cluster of 20 m3xlarge EC2 instances. The configurations were: spark.driver.maxResultSize = 3g, maximumResourceAllocation = true, spark.maxRemoteBlockSizeFetchToMem = 250m, spark.driver.memory 6g. The job was run on only 1000 files taking about 15 minutes. This was due to the cost constraints which demanded a small size cluster. Code can be found at: https://github.com/usma-eecs/commoncrawl-analytics/blob/master/src/main/scala/edu/usma/cc/SimpleApp.scala

V. Conclusion

Apache Spark has many features which makes it used by a wide array of data analysts. The learning curve however, and complications with its lazy evaluation may make it's benefits not worth the struggle. Developers should take great caution if planning to write Spark code using loops.

References

[1] So youre ready to get started, Common Crawl. [Online]. Available: http://commoncrawl.org/. [Accessed: 14-Dec-2018].

[2] Amazon Web Services (AWS) - Cloud Computing Services, Amazon. [Online]. Available: https://aws.amazon.com/. [Accessed: 14-Dec-2018].

[3] Spark SQL, DataFrames and Datasets Guide, Apache Spark - Unified Analytics Engine for Big Data. [Online]. Available: https://spark.apache.org/docs/latest/sql-programming-guide.html. [Accessed: 14-Dec-2018].

[4] Apache Spark - Unified Analytics Engine for Big Data, Apache Spark - Unified Analytics Engine for Big Data. [Online]. Available: https://spark.apache.org/. [Accessed: 14-Dec-2018].

[5] Lazy Evaluation in Apache Spark A Quick guide, DataFlair, 21-Nov-2018. [Online]. Available: https://data-flair.training/blogs/apache-spark-lazy-evaluation/. [Accessed: 14-Dec-2018].

[6] RDD Programming Guide, Apache Spark - Unified Analytics Engine for Big Data. [Online]. Available: https://spark.apache.org/docs/latest/rdd-programming-guide.html. [Accessed: 14-Dec-2018].

[7] Spark RDDs Simplified, Bigdata ML Notebook, 04-Feb-2016. [Online]. Available: http://vishnuviswanath.com/spark_rdd.html. [Accessed: 14-Dec-2018].

[8] Spark when union a lot of RDD throws stack overflow error, Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/30522564/spark-when-union-a-lot-of-rdd-throws-stack-overflow-error. [Accessed: 14-Dec-2018].