How Apache Spark can fail or be confusing and what you can do about it

Yin Huai
Strata, 03/16/2017
About me

• Software engineer at Databricks

• Apache Spark committer and PMC member

• Recent focus: making Spark easier to use, debug, and manage

• Before joining Databricks: Ohio State University
Data management services @ Databricks powered by Apache Spark

- Notebooks & Dashboards
- Secure SQL Server for BI Tools
- REST API
- Jobs & Workflows

Open Source Spark + Databricks Managed Services

Connectors and Optimized AWS S3 Access Layer

Databricks Enterprise Security Framework
Making big data simple

• Our mission: Make Spark and clusters powered by Spark easier to use, easier to debug, and easier to manage.

• Like other distributed systems and data management systems, Spark’s internal is complicated.

• But, Spark can still be very approachable.
In this talk, I am going to share our experience at Databricks on making Spark easier to use, easier to debug, and easier to manage.
Outline

• Confusing errors and behaviors, and how to debug them

• New challenges introduced by cloud environments

• Tips on writing debuggable code
My goals for this talk

• Help you get to know:
  • approaches and tools to debug confusing error and behaviors
  • tips on how to avoid of causing confusing errors or behaviors

• Get you interested in the work on making Spark easier to use, debug, and manage

• Get to know users/developers who share the same interest and hear your awesome ideas on how to make big data simple
Confusing errors and behaviors
&
How to debug them
Confusing errors and behaviors

• Seems my job is not making progress. What’s going on?

• My cluster somehow gradually has less and less free disk space. But, I do not think I have stored anything explicitly.

• My Spark executor was lost suddenly and the executor log does not show anything wrong.

• I get a random error message. Not sure about what it means.
Common causes of confusions

• Hanging Spark jobs
• Memory or file leaks
• Service failures
• An exception does not show the original error (swallowed exceptions or suppressed exceptions)
Hanging Spark jobs

• Symptom: my Spark job have been running for a long time. I am not sure if it is doing anything useful.

• What to do?

  Take several executor thread dumps
  Using jstack or Spark UI

  Hanging or slow?

  Slowness
  Profile and find the bottleneck

  Hanging
  Find the cause of hanging
jstack

- A tool used to print Java stack trace for a given Java process
- Example: Get thread dump for a Spark task

  > jps ## Finds the process of Spark’s executor
  7718 Jps
  7623 CoarseGrainedExecutorBackend
  6761 Worker

  > jstack 7623
"Executor task launch worker for task 9" #71 daemon prio=5 os_prio=0 tid=0x00007fc8f8034000 nid=0x1e58 waiting on condition [0x00007fc8c13f9000]
java.lang.Thread.State: TIMED_WAITING (sleeping)
...
at scala.collection.Iterator$$anon$11.next(Iterator.scala:328)
at org.apache.spark.util.Utils$.getIteratorSize(Utils.scala:1762)
at org.apache.spark.rdd.RDD$$anonfun$count$1.apply(RDD.scala:1158)
at org.apache.spark.rdd.RDD$$anonfun$count$1.apply(RDD.scala:1158)
at org.apache.spark.SparkContext$$anonfun$runJob$5.apply(SparkContext.scala:1952)
at org.apache.spark.SparkContext$$anonfun$runJob$5.apply(SparkContext.scala:1952)
at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:322)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
Thread dump provided by Spark UI

<table>
<thead>
<tr>
<th>Executor ID</th>
<th>Address</th>
<th>Status</th>
<th>RDD Blocks</th>
<th>Storage Memory</th>
<th>Disk Used</th>
<th>Cores</th>
<th>Active Tasks</th>
<th>Failed Tasks</th>
<th>Complete Tasks</th>
<th>Total Tasks</th>
<th>Task Time (GC Time)</th>
<th>Task Time</th>
<th>Input</th>
<th>Shuffle Read</th>
<th>Shuffle Write</th>
<th>Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver</td>
<td>10.0.169.30:42567</td>
<td>Active</td>
<td>0</td>
<td>0.0 B / 13.5 GB</td>
<td>0.0 B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 ms (0 ms)</td>
<td>0 ms</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10.0.221.106:37425</td>
<td>Active</td>
<td>0</td>
<td>0.0 B / 13.1 GB</td>
<td>0.0 B</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 ms (0 ms)</td>
<td>0 ms</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10.0.170.11:42440</td>
<td>Active</td>
<td>0</td>
<td>0.0 B / 13.1 GB</td>
<td>0.0 B</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 ms (0 ms)</td>
<td>0 ms</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10.0.182.66:41264</td>
<td>Active</td>
<td>0</td>
<td>0.0 B / 13.1 GB</td>
<td>0.0 B</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 ms (0 ms)</td>
<td>0 ms</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td></td>
</tr>
</tbody>
</table>
Example: SPARK-18971

Some Spark tasks are hanging for a long time

Using jatack on the executor (or using thread dump link on the Spark UI) shows that Spark tasks are hanging at fetching shuffle blocks.

parking to wait for <0x0000000498c249c0> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
at java.util.concurrent.locks.LockSupport.park(LockSupport.java:189)
at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:442)
at org.apache.spark.storage.ShuffleBlockFetcherIterator.next(ShuffleBlockFetcherIterator.scala:332)
at org.apache.spark.storage.ShuffleBlockFetcherIterator.next(ShuffleBlockFetcherIterator.scala:58)
at scala.collection.Iterator$$anon$12.nextCur(Iterator.scala:434)
...
at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:97)
at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:54)
at org.apache.spark.scheduler.Task.run(Task.scala:114)
at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:323)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1147)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)
at java.lang.Thread.run(Thread.java:834)
Example: SPARK-18971

Looking at the shuffle client, which is used to fetch shuffle blocks when shuffle service is enabled.

"shuffle-client-7-4" daemon prio=5 tid=97 RUNNABLE
at io.netty.util.Recycler$Stack.scavengeSome(Recycler.java:504)
at io.netty.util.Recycler$Stack.scavenge(Recycler.java:454)
at io.netty.util.Recycler$Stack.pop(Recycler.java:435)
at io.netty.util.Recycler.get(Recycler.java:144)
at io.netty.buffer.PooledUnsafeDirectByteBuf.newInstance(PooledUnsafeDirectByteBuf.java:39)
...
a cycle linked list of io.netty.util.Recycler.WeakOrderQueue

Potential endless loop in Recycler.Stack.scavengeSome

#6153
zxwing opened this issue on Dec 21, 2016 · 15 comments

zxwing commented on Dec 21, 2016

**Expected behavior**
Recycler.Stack.scavengeSome should never run forever.

**Actual behavior**
Recycler.Stack.scavengeSome may not stop and it prevents the event loop from running other tasks.

**Steps to reproduce**
I cannot reproduce it. But I found out a cycle linked list of io.netty.util.Recycler.WeakOrderQueue when I was investigating an issue that a netty client was hung forevery.
Memory or file leaks

• Symptom: have less and less memory/disk space overtime without a reasonable explanation

• What to do?
  • Use jmap (jmap -histo) to check the number of objects and see if there is any suspicious number (e.g. the number of instances is way higher).
  • Then, analyze the heap dump to find how those objects could not be GCed.
jmap

- A tool used to print memory-related statistics for a running VM

  > jps ## Finds the process of Spark’s executor
  7718 Jps
  7623 CoarseGrainedExecutorBackend
  6761 Worker

  > jmap -histo 7623 ## prints histogram of heap
  > jmap -histo:live 7623 ## prints histogram of heap after a full GC
  > jmap -dump:format=b,file=filename 7623 ## dumps the heap
jmap

> jmap –histo:live 7623 ## prints histogram of java

<table>
<thead>
<tr>
<th>num</th>
<th>#instances</th>
<th>#bytes</th>
<th>class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>389127</td>
<td>184602296</td>
<td>[B</td>
</tr>
<tr>
<td>2:</td>
<td>291211</td>
<td>35891976</td>
<td>[C</td>
</tr>
<tr>
<td>3:</td>
<td>9158</td>
<td>20316048</td>
<td>[I</td>
</tr>
<tr>
<td>4:</td>
<td>74442</td>
<td>1786608</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>5:</td>
<td>13114</td>
<td>969880</td>
<td>[Ljava.lang.Object;</td>
</tr>
<tr>
<td>6:</td>
<td>5916</td>
<td>656504</td>
<td>java.lang.Class</td>
</tr>
</tbody>
</table>

...  

Total 1013412 253156840
Example: SPARK-18553

- Symptoms: Unneeded shuffle files on the disk are not GCed.

- Using jmap on the driver shows unexpected high number of ShuffleDependency instances.

- Heap dump shows that those ShuffleDependency instances are kept by TaskScheduler.
Service failures

• Symptom: a Spark driver or an executor suddenly crashed

• What to do?
  • Check driver log or executor log to find errors that caused the failure
  • If the service OOMed, try to reproduce the issue and then use jmap or heap dump to see how the heap was full.
  • If the log does not show any error, check dmesg and see if kernel’s OOM killer killed the service

Memory cgroup out of memory:
Kill process 7536 (java) score 1907 or sacrifice child
An exception does not show the original error

• Symptom: There is an error, which seems random.

• What to do?
  • Try to write a snippet to reproduce it and find the real cause
  • Spark related?
    - File a jira (more useful information will help the jira get resolved faster)
    - Even better: Spot the specific problem and submit a PR to improve the code
Example: SPARK-14149

A Databricks internal test failed because Spark could not deserialize a task

Caused by: java.lang.IllegalStateException: unread block data
at java.io.ObjectInputStream$BlockDataInputStream.setBlockDataMode(ObjectInputStream.java:2431)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1383)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2000)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1924)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1801)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:76)
at org.apache.spark.serializer.JavaSerializerInstance.deserialize(JavaSerializer.scala:115)
at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:194)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
private Object readObject0(boolean unshared) throws IOException {
    ...  // Working on something that may throw an exception
    try {
        ...
    } finally {
        depth--;
        bin.setBlockDataMode(oldMode);
    }
}
Inside `ObjectInputStream.readObject0`

```java
private Object readObject0(boolean unshared) throws IOException {
    ...
    try {
        ... // Working on something that may throw an exception
    } finally {
        depth--;
        bin.setBlockDataMode(oldMode);
    }
}
```

This line may also fail if the try block throws an exception.

`java.lang.NoClassDefFoundError` was the real error, which was caused by a misconfiguration of the classpath. We found the real error by writing a code snippet to simulate the behavior of Spark’s task deserialization logic.
How to make this issue easier to debug?

[SPARK-14149] Log exceptions in tryOrIOException

## What changes were proposed in this pull request?
We ran into a problem today debugging some class loading problem during deserialization, and JVM was masking the underlying exception which made it very difficult to debug. We can however log the exceptions using try/catch ourselves in serialization/deserialization. The good thing is that all these methods are already using Utils.tryOrIOException, so we can just put the try catch and logging in a single place.

## How was this patch tested?
A logging change with a manual test.

Author: Reynold Xin <rxin@databricks.com>

Closes #11951 from rxin/SPARK-14149.

master (#1) v2.1.0 ... 2.0.0-preview

rxin committed on Mar 25, 2016
How to make this issue easier to debug?

```scala
// Execute a block of code that returns a value, re-throwing any non-fatal uncaught exceptions as IOException. This is used when implementing Externalizable and Serializable's read and write methods, since Java's serializer will not report non-IOException properly;

def tryOrIOException[T](block: => T): T = {
  try {
    block
  } catch {
    case e: IOException => throw e
    case NonFatal(t) => throw new IOException(t)
    case e: IOException =>
      logError("Exception encountered", e)
      throw e
    case NonFatal(e) =>
      logError("Exception encountered", e)
      throw new IOException(e)
  }
}
```
New challenges in cloud environments
New challenges introduced by new environments

• Running Spark on top of a layer of systems managed by a cloud provider
• These systems have their specific behaviors
• These systems may have undocumented behaviors
• Two examples:
  • Spot instances are terminated by AWS
  • Temporary network connectivity issue between EC2 instances
1: Spot instances are terminated by AWS

• Spot instances: spare EC2 instances
• You can bid on spot instances with a low price
• If the Spot price exceeds your bid price for a given instance, AWS will automatically terminate the instance

• Impacts
  • Positive: You can save cost while using spot instances
  • Negative: Your cluster using spot instances may suddenly lose nodes, which cause failures or lower performance
Reducing the negative impacts

- Databricks cluster manager will add back nodes when some spot instances are terminated because of spot price and will fall back to use on-demand nodes if needed.

- Users can set retry policy for scheduled jobs in Databricks
2: Temporary network connectivity issue

- Two EC2 instances may not be able to talk to each other for several minutes
- Example: An Spark worker node could not reach master node for several minutes
- Result: multiple executors are launched in the single worker node and they may compete resources that should be just allocated to a single executor.
How could it happen?

Spark Master

t1: Worker must be lost

Spark worker

heartbeats

t2: There is a new worker

t3: Launch an executor

Time

t4: Worker launches a new executor

Time
Solution

[SPARK-13522][CORE] Executor should kill itself when it's unable to heartbeat to driver more than N times

## What changes were proposed in this pull request?

Sometimes, network disconnection event won't be triggered for other potential race conditions that we may not have thought of, then the executor will keep sending heartbeats to driver and won't exit.

This PR adds a new configuration `spark.executor.heartbeat.maxFailures` to kill Executor when it's unable to heartbeat to the driver more than `spark.executor.heartbeat.maxFailures` times.

## How was this patch tested?

unit tests

Author: Shixiong Zhu <shixiong@databricks.com>

Closes #11401 from zsxwing/SPARK-13522.

master (2) v2.1.0 2.0.0-preview

zsxwing committed with Andrew Or on Feb 29, 2016
Tips on writing debuggable code
Provide good error messages

```java
require(length == 1) throw new java.lang.IllegalArgumentException:
requirement failed

require(length == 1, "The given length is not allowed.")

require(
    length == 1,
    s"The given length ($length) is not allowed. " +
    "Only 1 is allowed.")
```
def allocateMemory(sizeInBytes: Long): Page = {
    val allocatedPage =
        memoryManager.allocatePage(sizeInBytes)
    if (allocatedPage == null || allocatedPage.size < sizeInBytes)
        ... // If necessary, free allocated memory

    throw new OutOfMemoryError(
        s"Unable to acquire $sizeInBytes bytes of memory, got $got")
}
Log the context

```scala
def allocateMemory(sizeInBytes: Long): Page = {
  val allocatedPage =
    memoryManager.allocatePage(sizeInBytes)
  if (allocatedPage == null || allocatedPage.size < sizeInBytes)
    // If necessary, free allocated memory
    throw new OutOfMemoryError(
      s"Unable to acquire $sizeInBytes bytes of memory, got $got")
}
```

What is the cause of this OOM?

What is the cause of this OOM?
def allocateMemory(sizeInBytes: Long): Page = {
    val allocatedPage =
        memoryManager.allocatePage(sizeInBytes)
    if (allocatedPage == null || allocatedPage.size < sizeInBytes)
        ... // If necessary, free allocated memory
        memoryManager.logMemoryUsage()
    throw new OutOfMemoryError(
        s"Unable to acquire $sizeInBytes bytes of memory, got $got")
}
Ask the following questions

• How will my code fail?

• What do I need to do to make my life easier when there is something wrong with my code and I need to debug it?
Interested in this talk and our work?
Let’s chat

• Find me after this talk; or
• Find me at Databricks booth: 2:40 to 3:40pm

Join Databricks and work together to make big data simple!
Thank You!