Spark 2.x Troubleshooting Guide

IBM Big Data Performance
Jesse Chen, jesse.f.chen@gmail.com

3/2017

Troubleshooting Spark 2.x

- Building Spark
- Running Spark
  - '--verbose'
  - Missing external JARs
  - OOM on Spark driver
  - OOM on executors
  - GC policies
  - Spark Thrift Server for JDBC apps
  - HDFS block distribution
  - HDFS blocksize vs Parquet blocksize
- Profiling Spark
  - Collecting thread & heap dumps in-flight
  - Collecting core dumps after jobs fail

Lots of errors when building a new Spark release on my own…

- Run '`make-distribution.sh`' (generates 'bin/spark-shell', 'bin/spark-submit', etc.)
- Does not always work
  - Wrong JRE version or no JRE found
  - No Maven installed
  - Support for certain components not default, e.g., 'hive' support
- TIP #1: Always explicitly set the following in '.bashrc' for 'root'

```
# for Spark distribution compiling
export JAVA_HOME=/usr/jdk64/java-1.8.0-openjdk-1.8.0.77-0.b03.el7_2.x86_64
export JRE_HOME=$JAVA_HOME/jre
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib:$CLASSPATH

#set maven environment
M2_HOME=/TestAutomation/downloads/tmp/spark-master/build/apache-maven-3.3.9
export MAVEN_OPTS="-Xms256m -Xmx2048m -XX:MaxPermSize=512m"
export PATH=$M2_HOME/bin:$PATH
```

- TIP #2: Specify support you want explicitly
  - To build Spark with YARN and Hive support, do:

```
./dev/make-distribution.sh --name spark-master-2.1 --tgz -Pyarn -Phadoop-2.7 -
Dhadoop.version=2.7.2 -Phive -Phive-thriftserver
```

Building a Spark release is extremely slow …

- Use more cores to speed up the build process (default uses only 1 core)
- Rebuild only modified source code (default is "clean")

```
Edit the file './dev/make-distribution.sh', change line

BUILD_COMMAND=("$MVN" -T 1C clean package -DskipTests $@)

To:

BUILD_COMMAND=("$MVN" -T 48C package -DskipTests $@)

** Assuming your have 48 cores on your build machine
** Assuming you don't need to always build clean, for iterative changes
```

- Can cut build time from 45 min to 15 min on a typical 128GB-RAM 48-core node

Don't know what settings used when running Spark …

- Always use '–-verbose' option on 'spark-submit' command to run your workload
- Prints
  - All default properties
  - Command line options
  - Settings from spark 'conf' file
  - Settings from CLI
- Example output

```
Spark properties used, including those specified through
 --conf and those from the properties file /TestAutomation/spark-2.0/conf/spark-defaults.conf:
  spark.yarn.queue -> default
  spark.local.dir -> /data1/tmp,/data2/tmp,/data3/tmp,/data4/tmp
  spark.history.kerberos.principal -> none
  spark.sql.broadcastTimeout -> 800
  spark.hadoop.yarn.timeline-service.enabled -> false
  spark.yarn.max.executor.failures -> 3
  spark.driver.memory -> 10g
  spark.network.timeout -> 800
  spark.yarn.historyServer.address -> node458.xyz.com:18080
  spark.eventLog.enabled -> true
  spark.history.ui.port -> 18080
  spark.rpc.askTimeout -> 800
  …
```

- Example command:

```
spark-submit  --driver-memory 10g --verbose --master yarn --executor-memory ….
```

Missing external jars

- Compiled OK, but run-time NoClassDefFoundError:

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/kafka/clients/producer/KafkaProducer
        at java.lang.Class.getDeclaredMethods0(Native Method)
        at java.lang.Class.privateGetDeclaredMethods(Class.java:2701)
        at java.lang.Class.privateGetMethodRecursive(Class.java:3048)
        at java.lang.Class.getMethod0(Class.java:3018)
```

- Use '--packages' to include comma-separated list of Maven coordinates of JARs
- Example

```
spark-submit --driver-memory 12g --verbose --master yarn-client --executor-memory 4096m --num-executors 20
--class com.ibm.biginsights.pqa.spark.SparkStreamingTest --packages org.apache.spark:spark-streaming-
kafka_2.10:1.5.1 …
```

- This includes JARs on both driver and executor classpaths
- Order of look-up
  - The local Maven repo – local machine
  - Maven central - Web
  - Additional remote repositories specified in –repositories

OutOfMemory related to Spark driver

- Types of OOM related to Spark driver heap size

```
15/10/06 17:10:00 ERROR akka.ErrorMonitor: Uncaught fatal error from thread [sparkDriver-
akka.actor.default-dispatcher-29] shutting down ActorSystem [sparkDriver]
java.lang.OutOfMemoryError: Java heap space
Exception in thread "task-result-getter-0" java.lang.OutOfMemoryError: Java heap space

Subsequent error: Exception in thread "ResponseProcessor for block
BP-1697216913-9.30.104.154-1438974319723:blk_1073847224_106652" java.lang.OutOfMemoryError: Java heap
space

WARN nio.AbstractNioSelector: Unexpected exception in the selector loop.
java.lang.OutOfMemoryError: Java heap space at
org.jboss.netty.buffer.HeapChannelBuffer.<init>(HeapChannelBuffer.java:42)
```

- Increase '--driver-memory' usually resolves these
- Default 512M is usually too small for serious workloads
- Example: 8GB minimum needed for Spark SQL running TPCDS @ 1TB
- Typical workloads that need large driver heap size
  - Spark SQL
  - Spark Streaming

OOM – GC overhead limit exceeded

```
15/12/09 19:57:02 WARN scheduler.TaskSetManager: Lost task 175.0 in stage 68.0 (TID 7588,
rhel8.cisco.com): java.lang.OutOfMemoryError: GC overhead limit exceeded
        at org.apache.spark.sql.catalyst.expressions.UnsafeRow.copy(UnsafeRow.java:478)
        at org.apache.spark.sql.catalyst.expressions.UnsafeRow.copy(UnsafeRow.java:55)
```

- Too much time is being spent in garbage collection (98% of the total time)
- Less than 2% of the heap is recovered
- From '`top`', often see "1 CPU core fully used at 100%" but no work is done
- Tuning #1: Increase executor heapsize

```
spark-submit  … --executor-memory 4096m --num-executors 20 …
```

- OR Tuning #2: Change GC policy (next slide)

GC policies

- Choose between `-XX:UseG1GC` & `-XX:UseParallelGC`
- Show current GC settings

```
% /usr/jdk64/java-1.8.0-openjdk-1.8.0.45-28.b13.el6_6.x86_64/bin/java -XX:+PrintFlagsFinal

uintx GCHeapFreeLimit                          = 2                                   {product}
uintx GCLockerEdenExpansionPercent             = 5                                   {product}
uintx GCLogFileSize                            = 8192                                {product}
uintx GCTimeLimit                              = 98                                  {product}
uintx GCTimeRatio                              = 99                                  {product}
bool UseG1GC                                   = false                               {product}
bool UseParallelGC                            := true                                {product}
```

- Tuning options
  - Spark default is `-XX:UseParallelGC`
  - Try overwrite with `-XX:G1GC`
- Performance Impact: “Mythical at best”, “It depends”
- **Default is pretty good!**
- Databricks blog on Tuning GC for Spark
  - https://databricks.com/blog/2015/05/28/tuning-java-garbage-collection-for-spark-applications.html

Support JDBC Apps via Spark Thrift Server

- Spark SQL can act as a distributed query engine using its JDBC/ODBC interface
- Supported by running the Thrift JDBC/ODBC server
- Has a single SparkContext with multiple sessions supporting
  - Concurrency
  - re-usable connections (pool)
  - Shared cache (e.g., catalog, tables, etc.)
- Can specify any amount of memory, CPUs through standard Spark-submit parameters:
  - Driver-memory
  - Executor-memory
  - Num-executors, etc.
- Example, to start Thrift Server with 2.3TB of memory, 800 cores and YARN mode:

```
% $SPARK_HOME/sbin/start-thriftserver.sh --driver-memory 12g --verbose --master yarn --executor-memory 16g
--num-executors 100 --executor-cores 8 --conf spark.hadoop.yarn.timeline-service.enabled=false --conf
spark.yarn.executor.memoryOverhead=8192 --conf spark.driver.maxResultSize=5g
```

- Default number of workers (sessions) = 500
- Client tool bundled with Spark 2.0: Beeline

```
% $SPARK_HOME/bin/beeline -u "jdbc:hive2://node460.xyz.com:10013/my1tbdb" -n spark --force=true -f /test/
query_00_01_96.sql
```

Not all CPUs are busy …

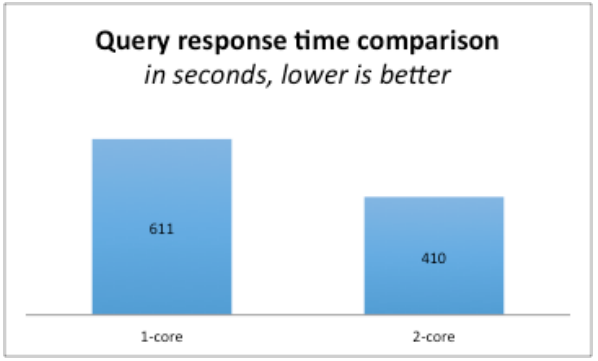
- Designed for big data
- More cores and more memory always better (well, until it breaks!)
- Ways to max out your cluster, for example:
  - 40 vCores per node
  - 128GB memory per node
  - 5-node cluster = 200 vCores, ~500GB RAM
- Method #1 – Start with evenly divided memory and cores

```
--executor-memory 2500m --num-executors 200
Total # of executors = 200 (default: 1-core each)
# of executors/node  = 40  (fully using all cores)
Total memory used    = 500 GB
```

- Method #2 – When heap size non-negotiable

```
--executor-memory 6g --num-executors 80
Total # of executors = 80 (1-core each)
# of executors/node  = 16 (40% CPU utilization)
Total memory used   ~= 500 GB

Can increase cores per executor as:
--executor-memory 6g --num-executors 80 –executor-cores 2
Forcing 80% utilization, boosting 33% performance!
```



Spread out Spark "scratch" space

- Typical error

```
stage 89.3 failed 4 times, most recent failure:
Lost task 38.4 in stage 89.3 (TID 30100, rhel4.cisco.com): java.io.IOException: No space left on device
        at java.io.FileOutputStream.writeBytes(Native Method)
        at java.io.FileOutputStream.write(FileOutputStream.java:326)
        at org.apache.spark.storage.TimeTrackingOutputStream.write(TimeTrackingOutputStream.java:58)
        at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:82)
        at java.io.BufferedOutputStream.write(BufferedOutputStream.java:126)
```

- Complains about '/tmp' is full
- Controlled by 'spark.local.dir' parameter
  - Default is '/tmp'
  - Stores map output files and RDDs
- Two reasons '/tmp' is not an ideal place for Spark "scratch" space
  - '/tmp' usually is small and for OS
  - '/tmp' usually is a single disk, a potential IO bottleneck
- To fix, add the following line to '`spark-defaults.conf`' file:

```
spark.local.dir     /data/disk1/tmp,/data/disk2/tmp,/data/disk3/tmp,/data/disk4/tmp,…
```

Max result size exceeded

- Typical error

```
stream5/query_05_22_77.sql.out:Error: org.apache.spark.SparkException: Job aborted due to stage failure:
Total size of serialized results of 381610 tasks (5.0 GB) is bigger than spark.driver.maxResultSize (5.0
GB) (state=,code=0))
```

- Likely to occur with complex SQL on large data volumes
- Limit of total size of serialized results of all partitions for each Spark action (e.g., collect)
- Controlled by '`spark.driver.maxResultSize`' parameter
  - Default is 1G
  - Can be '0' or 'unlimited'
  - 'unlimited' will throw OOM on driver
- To fix, add the following line to '`spark-defaults.conf`' file:

```
spark.driver.maxResultSize     5g

** 5G is a learned value for Spark SQL running TPCDS queries at 1TB scale factors
```

Catalyst errors

- Typical error

```
stream7/query_07_24_48.sql.out:Error: org.apache.spark.sql.catalyst.errors.package$TreeNodeException:
execute, tree: at org.apache.spark.sql.execution.exchange.ShuffleExchange$$anonfun$doExecute
$1.apply(ShuffleExchange.scala:122)
        at org.apache.spark.sql.execution.exchange.ShuffleExchange$$anonfun$doExecute
$1.apply(ShuffleExchange.scala:113)
        at org.apache.spark.sql.catalyst.errors.package$.attachTree(package.scala:49)
        ... 96 more
Caused by: java.util.concurrent.TimeoutException: Futures timed out after [800 seconds]
        at scala.concurrent.impl.Promise$DefaultPromise.ready(Promise.scala:219)
        at scala.concurrent.impl.Promise$DefaultPromise.result(Promise.scala:223)
        at scala.concurrent.Await$$anonfun$result$1.apply(package.scala:190)
        at scala.concurrent.BlockContext$DefaultBlockContext$.blockOn(BlockContext.scala:53)
        at scala.concurrent.Await$.result(package.scala:190)
        at org.apache.spark.util.ThreadUtils$.awaitResult(ThreadUtils.scala:190)
        ... 208 more
```

- On surface appears to be Catalyst error (optimizer)
- Actually an internal Spark timeout error most likely to occur under concurrency

  `java.util.concurrent.TimeoutException: Futures timed out after [800 seconds]`
- Controlled by an unpublished Spark setting '`spark.sql.broadcastTimeout`' parameter
  - Default in source code shows 300 seconds
- To fix, add the following line to '`spark-defaults.conf`' file or as CLI --conf

```
spark.sql.broadcastTimeout    1200

**1200 is the longest running query in a SQL workload in our case.
```

Other timeouts

- Typical errors

```
16/07/09 01:14:18 ERROR spark.ContextCleaner: Error cleaning broadcast 28267
org.apache.spark.rpc.RpcTimeoutException: Futures timed out after [800 seconds]. This timeout is
controlled by spark.rpc.askTimeout
at org.apache.spark.rpc.RpcTimeout.org$apache$spark$rpc$RpcTimeout$
$createRpcTimeoutException(RpcTimeout.scala:48)
at org.apache.spark.rpc.RpcTimeout$$anonfun$addMessageIfTimeout$1.applyOrElse(RpcTimeout.scala:63)
at org.apache.spark.rpc.RpcTimeout$$anonfun$addMessageIfTimeout$1.applyOrElse(RpcTimeout.scala:59)
at scala.PartialFunction$OrElse.apply(PartialFunction.scala:167)
at org.apache.spark.rpc.RpcTimeout.awaitResult(RpcTimeout.scala:83)
at org.apache.spark.storage.BlockManagerMaster.removeBroadcast(BlockManagerMaster.scala:143)

And timeout exceptions related to the following:
spark.core.connection.ack.wait.timeout
spark.akka.timeout
spark.storage.blockManagerSlaveTimeoutMs
spark.shuffle.io.connectionTimeout
spark.rpc.askTimeout
spark.rpc.lookupTimeout
```

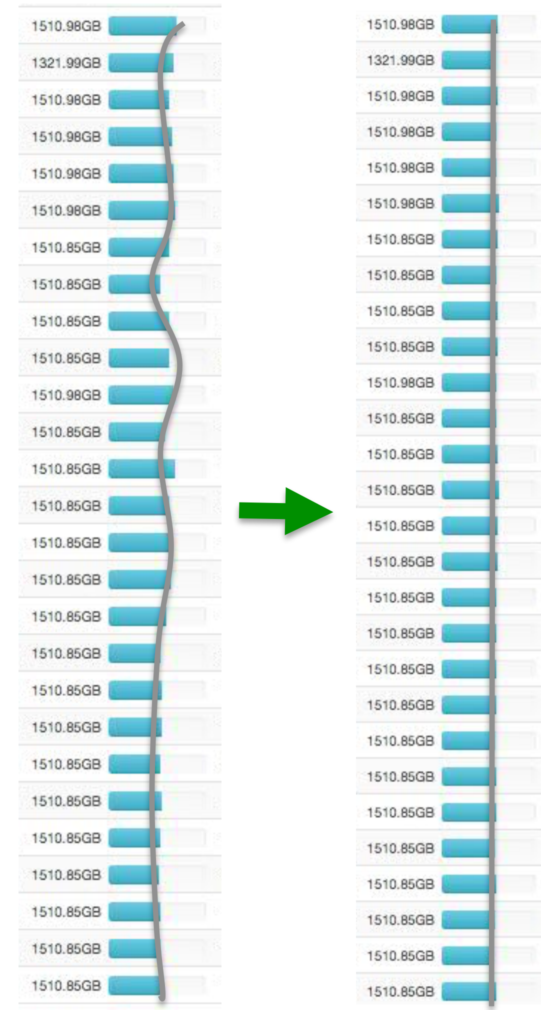
- Depending on system resource usage, any of the above can occur (e.g., no heartbeats)
- You can tune each individual setting OR use an "umbrella" timeout setting
- Controlled by '`spark.network.timeout`' parameter
  - Default is 120 seconds
  - Overrides all above timeout values
- To fix, add the following line to '`spark-defaults.conf`' file:

```
spark.network.timeout     700
```

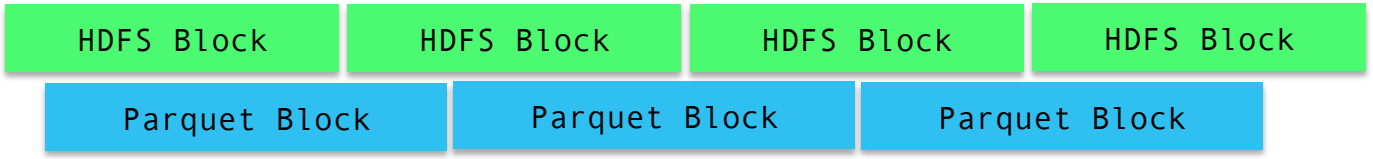
Out of space on a few data nodes …

- Unbalanced HDFS forces more IO over network
- Run command ‘`hdfs balancer`’ to start rebalancing
- `dfs.datanode.balance.bandwidthPerSec`
  - Default 6250000 or 6.25 MB/s network bandwidth
  - Increased to 6 GB/s on F1 to take advantage of fat pipe
- `dfs.datanode.balance.max.concurrent.moves`
  - Default is undefined
  - Add this setting in hdfs-site
  - Set to 500 concurrent threads
  - Example shows 5.4 TB/hour balancing rate

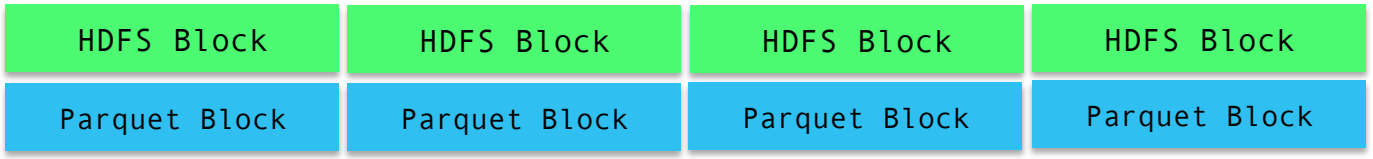
```
16/10/05 10:17:24 INFO balancer.Balancer: 0 over-utilized: []
16/10/05 10:17:24 INFO balancer.Balancer: 0 underutilized: []
The cluster is balanced. Exiting...
Oct 5, 2016 10:17:24 AM         337    19.71 TB   0 B   -1 B
Oct 5, 2016 10:17:24 AM  Balancing took 3.6939516666666665 hours
```



What block size to use in HDFS and in Parquet?

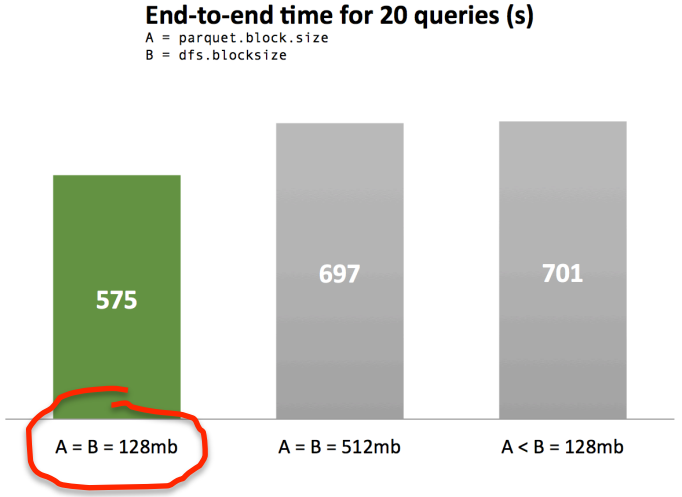


Remote reads occur when block boundaries cross
Slows down scan time
Prefer row group boundaries be at block boundaries



Take-away:

Keep block size for both at default (128MB)



In-flight capturing of executor thread & heap dumps

- Typically run as YARN containers across multiple nodes, e.g.,

```
yarn      355583  355580 91 09:15 ?        00:05:35 /usr/jdk64/java-1.8.0-
openjdk-1.8.0.45-28.b13.el6_6.x86_64/bin/java -server -XX:OnOutOfMemoryError=kill %p -Xms6144m -Xmx6144m -
Djava.io.tmpdir=/data6/hadoop/yarn/local/usercache/biadmin/appcache/application_1452558922304_0075/
container_1452558922304_0075_01_000020/tmp -Dspark.driver.port=3110 -Dspark.history.ui.port=18080 -
Dspark.yarn.app.container.log.dir=/data1/hadoop/yarn/log/application_1452558922304_0075/
container_1452558922304_0075_01_000020 org.apache.spark.executor.CoarseGrainedExecutorBackend --driver-url
akka.tcp://sparkDriver@9.30.104.154:3110/user/CoarseGrainedScheduler --executor-id 19 –hostname
node133.yxz.com --cores 1 --app-id application_1452558922304_0075 --user-class-path file:/data6/hadoop/
yarn/local/usercache/biadmin/appcache/application_1452558922304_0075/
container_1452558922304_0075_01_000020/__app__.jar
```

- OpenJDK has a set of tools for Java thread and heap dumps

```
jmap, jstack, jstat, jhat, etc.
```

- Typical location of OpenJDK tools for IBM Hadoop platform

```
/usr/jdk64/java-1.8.0-openjdk-1.8.0.45-28.b13.el6_6.x86_64/bin/
```

- To get a full thread dump

```
% jstack –l 355583 > /TestAutomation/results/twitter/javacore.355583.1
% jstack –l –F 355583 > /TestAutomation/results/twitter/javacore-hung.355583.1

Use –F to attach to a non-responsive JVM
```

- To get a full heap dump

```
% jmap -dump:live,format=b,file=/TestAutomation/results/dump.355583.2 355583
Dumping heap to /TestAutomation/results/sparkstreamtests/dump.355583.2 ...
Heap dump file created
```

Can't find core dumps even when Spark says there are ….

- Core dumps created by Spark jobs

```
16/11/14 16:45:05 WARN scheduler.TaskSetManager: Lost task 692.0 in stage 4.0 (TID 129 node12.xyz.com,
executor 824): ExecutorLostFailure (executor 824 exited caused by one of the running tasks) Reason:
Container marked as failed: container_e69_1479156026828_0006_01_000825 on host: node12.xyz.com. Exit status:
134. Diagnostics: Exception from container-launch.
Exit code: 134
Container id: container_e69_1479156026828_0006_01_000825
Exception message: /bin/bash: line 1: 3694385 Aborted                 (core dumped) /usr/jdk64/java-1.8.0-
openjdk-1.8.0.77-0.b03.el7_2.x86_64/bin/java -server -Xmx24576m -Djava.io.tmpdir=/data2/hadoop/yarn/local/
….ontainer.log.dir=/data5/hadoop/…container_e69_1479156026828_0006_01_000825/com.univocity_univocity-
parsers-1.5.1.jar > /data5/hadoop/yarn/log/application_1479156026828_0006/
container_e69_1479156026828_0006_01_000825/stdout 2> /data5/hadoop/yarn/log/application_1479156026828_0006/
container_e69_1479156026828_0006_01_000825/stderr

Stack trace: ExitCodeException exitCode=134: /bin/bash: line 1: 3694385 Aborted (core dumped) /usr/jdk64/
java-1.8.0-openjdk-1.8.0.77-0.b03.el7_2.x86_64/bin/java -server -Xmx24576m -Djava.io.tmpdir=/data2/hadoop/-…
container_e69_1479156026828_0006_01_000825/com.univocity_univocity-parsers-1.5.1.jar > /data5/hadoop/yarn/
log/application_1479156026828_0006/container_e69_1479156026828_0006_01_000825/stdout 2> /data5/hadoop/yarn/
log/application_1479156026828_0006/container_e69_147915602 0006_01_000825/stderr
```

- YARN settings for core dump file retention

```
yarn.nodemanager.delete.debug-delay-sec default is 0, files deleted right after application finishes
Set it to enough time to get to files and copy them for debugging
```

- Steps: 1. Find the hostname in the error log; 2. Find the local directory where 'stderr' resides; 3. Open the 'stderr', you will find lines similar to:

```
/data2/hadoop/yarn/local/usercache/spark/appcache/application_1479156026828_0006/
container_e69_1479156026828_0006_01_000825/hs_err_pid3694385.log
```

- and core dump files too!
- More on this setting https://hadoop.apache.org/docs/r2.7.3/hadoop-yarn/hadoop-yarn-common/yarn-default.xml