artdaq - Bug #21591

Run stop/start intermittently fails at protoDUNE (stale Request messages)

12/19/2018 12:14 PM - Kurt Biery

Status:	Closed	**Start date:**	12/19/2018
Priority:	Normal	**Due date:**	
Assignee:	Kurt Biery	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:	artdaq v3_04_00		
Experiment:	-	**Co-Assignees:**	Eric Flumerfelt

Description

In order to debug run stop/start issues at protoDUNE, I ran with the Timing system, a couple of RCEs, and 4-6 EventBuilders. Most of the time, the second, third, etc. run in the same DAQ session would work fine, but after a while, one of the runs would invariably fail to produce the expected amount of data on disk, and would not appear to have data flowing through the system. Sometimes it took 6 or 7 runs within the same DAQ session to see the failure.

History

#1 - 12/19/2018 12:17 PM - Kurt Biery

The first clue in the TRACE logs was the following:

- in the runs that failed to produce data flowing through the system, the RequestReceiver threads in the BoardReaders would exit immediately after the beginning of the run.

It wasn't clear what was causing the RequestReceiver threads to exit early, but based on the suggestion in an existing bugfix branch (bugfix/RoutingTokensPersistBetweenRuns) that certain types of messages might persist between runs, I added a run number field to the RequestPacket and added checking of the run number to the receiver. I also added the request mode to the 'Setting request mode in Message Header' TRACE message in RequestSender. All of this showed that there were stale EndOfRun request messages received from the EBs immediately after Start (begin-run) in the runs that had problems.

(Aside: I created a git branch for managing the software for this testing (kurt_validation_tests_12Dec, based on the for_dune-artdaq branch), and I merged the feature/RequestLoggingImprovements and feature/RequestReceiver_AdditionalCleanupInGetAndClearRequests into this new branch.)

Discarding request messages with the wrong run number did not completely fix the problem.

To debug the issue further, I enabled extra TRACE statements related to request sending and receiving:

- tonM -N *RequestReceiver 11
- tonM -N *RequestReceiver 10
- tonM -N *RequestReceiver 20
- tonM -N *RequestSender 4
- tonM -N *RequestSender 12

The extra TRACE statements showed the following:

- whenever the problem occurred, in the run that immediately preceded the problematic run, the RequestReceiver::stopRequestReceiverThread() method was **not** called
- in the preceding run, the timestamps associated with the Timing system BoardReader receiving and processing the Stop (end-run) message were noticeably different than those for the other BoardReaders (say ~2 seconds apart)

The following egrep statement was helpful in illustrating this behavior in the TRACE logs:

- egrep 'Joining requestThread|Starting Request Reception Thread|Ending Request Thread|Starting run|Stopping run|request listen socket|wrong run number|Setting mode flag in Message Header to EndOfRun'

Since protoDUNE was shutting down for winter break, I looked into reproducing this problem on the mu2edaq cluster. I was able to do that with the mediumsystem_with_routing_master sample_config and the following hacks to daqinterface.py:

```
index 0689978..e10c55b 100755
--- a/rc/control/daqinterface.py
+++ b/rc/control/daqinterface.py
@@ -172,7 +172,7 @@ class DAQInterface(Component):
                 else:
                     return False
             else:
-                if int(self.port) < int(other.port):
+                if self.label < other.label:
```

```
                    return True
                return False
```

```
@@ -1032,6 +1032,8 @@ class DAQInterface(Component):
                         t = Thread(target=process_command, args=(self, i_procinfo, command))
                         threads.append(t)
                         t.start()
+                        if "component01" in procinfo.label and command == "Stop":
+                            sleep(7)

                 for thread in threads:
                     thread.join()
```

The first hack was to sort the components by label, rather than port, in the procinfos list. The second one was to sleep for some amount of time after sending the Stop command to 'component01'.

The command that I used to run the demo was the following:

- sh ./run_demo.sh --config mediumsystem_with_routing_master --bootfile `pwd`/artdaq-utilities-daqinterface/simple_test_config/mediumsystem_with_routing_master/boot.txt --comps component01 component02 component03 component04 component05 component06 component07 component08 component09 component10 --runduration 40 --partition=3 --no_om --runs 2

Starting over on mu2edaq01, I ran a series of tests in which I captured selected entries in the TRACE buffer at with various code changes.

The first two TRACE buffers are with just one code change (the addition of the request mode to the RequestSender TRACE message), and with a 1-second and a 7-second sleeps in DAQInterface. The first of these was basically to capture a TRACE that showed normal behavior, and both of the two runs in that test had the expected data file size on disk. For the two runs in the test with a 7-second delay, the second data file did not have the expected size (much smaller).

- WithOneSecondDelayForComponent01StopInDAQInterface.trace (runs 87 and 88)
- WithSevenSecondDelayForComponent01StopInDAQInterface.trace (runs 89 and 90)

[biery@mu2edaq01 data]$ pwd
/scratch/biery/data

~~rw-r--r--~~ 1 biery mu2e 2561610 Dec 18 16:25 artdaqdemo_r000090_sr01_1_dl1.root
~~rw-r--r--~~ 1 biery mu2e 3972843296 Dec 18 16:24 artdaqdemo_r000089_sr01_1_dl1.root
~~rw-r--r--~~ 1 biery mu2e 3990867392 Dec 18 16:19 artdaqdemo_r000088_sr01_1_dl1.root
~~rw-r--r--~~ 1 biery mu2e 3990867392 Dec 18 16:18 artdaqdemo_r000087_sr01_1_dl1.root

The next test was with run number tracking added to Request Messages.

- WithRunNumberAddedToRequestMessage.trace (runs 91 and 92)
- WithRunNumberAddedToRequestMessage-Verbose.trace (runs 91 and 92)

~~rw-r--r--~~ 1 biery mu2e 2616152 Dec 18 16:45 artdaqdemo_r000092_sr01_1_dl1.root
~~rw-r--r--~~ 1 biery mu2e 3945823495 Dec 18 16:44 artdaqdemo_r000091_sr01_1_dl1.root

The next test was with checking for, and dropping, RequestMessages with the wrong run number.

- WithRequestMessageRunNumberChecking.trace (runs 95 and 96)

~~rw-r--r--~~ 1 biery mu2e 2425281 Dec 19 09:31 artdaqdemo_r000096_sr01_1_dl1.root
~~rw-r--r--~~ 1 biery mu2e 3774676164 Dec 19 09:30 artdaqdemo_r000095_sr01_1_dl1.root

This was git commit 9a4760ae9dc70f788e7cc20c4312cc52275fdc1c.

As mentioned earlier, the root causes of this issue were:

- one of the BoardReaders is told to Stop the run noticeably earlier than the other Bfs
- it does so, and sends its EndOfData fragments to the EventBuilders
- the EBs send RequestMessages with an EndOfRun type to **all** of the BRs. This causes the RequestReceiver Thread to exit, both in the BR that have received the Stop message, and those that haven't.
- when the BRs that haven't received the Stop message yet do receive the Stop message, the RequestReceiver::stopRequestReceiverThread() method is **not** called by CommandableFragmentGenerator::StopCmd() because it only does so if the RequestReceiver::isRunning() method returns TRUE (which it doesn't because the thread has already exited)
- this means that the Request Socket in the RequestReceiver is not closed and other variable resets do not happen
- at the beginning of the next run, the old socket is still used and the stale messages are processed
- the more important problem is the lack of resetting of variables, though. For example, highest_seen_request is not set to zero, so there are a log of complaints about requests that have already been processed (in the second run)

Looking at RequestReceiver, there appeared to be a better object-oriented naming (design), and that is to avoid exposing the internal implementation details of a thread being used to receive requests. The following translation appeared to be useful:

- stopRequestReceiverThread —> stopRequestReception
- startRequestReceiverThread -> startRequestReception

These name changes, coupled with the removal of tests of whether the Request Receiver Thread is running before deciding whether to call them, fixed the problem, and seems like a better design.

- WithRequestReceptionNamingChanges.trace (runs 97 and 98)

~~rw-r--r~~ 1 biery mu2e 3900779833 Dec 19 11:54 artdaqdemo_r000098_sr01_1_dl1.root
~~rw-r--r~~ 1 biery mu2e 3882771988 Dec 19 11:53 artdaqdemo_r000097_sr01_1_dl1.root

This was git commit 6e1b1d6ca08edbf3c83e1d9b6f149bd49908c677.

#2 - 12/19/2018 12:20 PM - Kurt Biery

- File WithOneSecondDelayForComponent01StopInDAQInterface.trace added

- File WithSevenSecondDelayForComponent01StopInDAQInterface.trace added

- File WithRunNumberAddedToRequestMessage.trace added

- File WithRunNumberAddedToRequestMessage-Verbose.trace added

- File WithRequestMessageRunNumberChecking.trace added

- File WithRequestReceptionNamingChanges.trace added

#3 - 12/19/2018 12:29 PM - Kurt Biery

- Status changed from New to Resolved

- Assignee set to Kurt Biery

#4 - 12/19/2018 02:23 PM - Eric Flumerfelt

Notes from code review:

```
@ -118,7 +118,11 @ namespace artdaq                        {
out[in.first] = in.second;
}
+                       if(requests_.size())
+               highest_seen_request_ = requests_.rbegin()->first;
+                       out_of_order_requests_.clear();
requests_.clear();
+                       request_timing_.clear();
return out;
}
```

The if statement here should probably have its body wrapped in braces for clarity, or have the body on the same line as the if statement.

Is there any need to also track subrun information in requests?

#5 - 01/07/2019 09:53 AM - Kurt Biery

I made the suggested change to the assignment of highest_request_seen, within the IF statement.

As to whether we should track subrun numbers in requests... I can certainly imagine that being useful at some point, but I want to wait until we have clear direction from one or more experiments on how subruns should be handled before making any changes here.

To provide a more descriptive name for the branch that holds these changes, I create branch bugfix/RequestMessagesPersistBetweenRuns in the artdaq repository for this Issue. I'll update the 'repository branches' Wiki page to reflect this.

#6 - 01/10/2019 02:03 PM - Kurt Biery

A test of this code at protoDUNE was successful. With APA4 in partition 2, 9 MB events, 40 Hz, I was able to start and stop 10+ runs in a single DAQ session, including ones with no triggers.

With older code, the first run after one with no triggers would exhibit problems.

Something that I haven't been able to demonstrate yet is the early existing of the Request Thread (before the Stop command arrives). That might be worth doing still,

- (tshowt | egrep -i 'Joining requestThread|Ending Request Thread')

#7 - 01/22/2019 03:34 PM - Kurt Biery

I observed the early existing of the Request Thread with this code at protoDUNE and successful subsequent runs on 1/16/2019 (runs 6398, 6399, and 6400).

#8 - 02/18/2019 10:58 AM - Eric Flumerfelt

- Status changed from Resolved to Reviewed

- Co-Assignees Eric Flumerfelt added

Code review looks good.

Was able to perform differential testing to show the issue exists in develop and is resolved by this branch.

Merging into develop.

#9 - 03/06/2019 11:30 AM - Eric Flumerfelt

- Target version set to artdaq v3_04_00

- Status changed from Reviewed to Closed

Files

WithOneSecondDelayForComponent01StopInDAQInterface.trace	30.2 KB	12/19/2018	Kurt Biery
WithSevenSecondDelayForComponent01StopInDAQInterface.trace	22.2 KB	12/19/2018	Kurt Biery
WithRunNumberAddedToRequestMessage.trace	22.2 KB	12/19/2018	Kurt Biery
WithRunNumberAddedToRequestMessage-Verbose.trace	906 KB	12/19/2018	Kurt Biery
WithRequestMessageRunNumberChecking.trace	8.8 MB	12/19/2018	Kurt Biery
WithRequestReceptionNamingChanges.trace	25.4 KB	12/19/2018	Kurt Biery