Metacat - Bug #7188

MNodeService.replicate() is failing

05/11/2017 07:19 AM - Chris Jones

Status:	Resolved	**Start date:**	05/11/2017
Priority:	Immediate	**Due date:**	
Assignee:	Jing Tao	**% Done:**	0%
Category:	metacat	**Estimated time:**	0.00 hour
Target version:	2.8.3	**Spent time:**	0.00 hour
Bugzilla-Id:			

Description

Laura Moyers reported that she is seeing many failed replication attempts in the Coordinating Node index. In particular, KNB, GOA, UIC, ARCTIC, mnUCSB1, and mnORC1 are all affected, and are all running Metacat.

After looking at catalina.out on the MNs, we're seeing errors in MNodeService.replicate():

```
20170508-06:59:14: [ERROR]: Error computing checksum on replica: mark/reset not supported [edu.ucsb.nceas.metacat.dataone.MNodeService]
```

Here's the number of requests and failures

```
host         requests        failures      failures_since
-----------------------------------------------------------
mn-orc-1     145             25            20170511-01:23:48
mn-ucsb-1    105             56            20170508-06:59:14
mn-unm-1     0               0             -
knb          71              28            20170509-16:57:53
uic          no log access
```

I'm pretty sure the failures represent 100% of the requests since the failures began, but we'd need to confirm this. Basically, MN replication looks to be entirely broken in Metacat.

The error reported above comes from line 866 of MNodeService.java, where the checksum of the bytes of the object from the source MN (to be replicated) is calculated. Once the checksum is calculated, we call object.reset() on the input stream so it can be read again when writing to disk. This is throwing the exception above.

So what's changed? The last changes regarding the InputStream was that Jing wrapped the calls in a try{ } finally { } block in order to ensure the input stream gets closed after use to prevent memory leaks. This doesn't seem like an issue at all, although the finally{ } block could have been used in the existing try { } block instead of having three levels of try nesting. This seems inconsequential though.

The other change is that d1_libclient_java is now using the Apache Commons IO AutoCloseInputStream. Looking at the documentation there, it seems to delegate to the underlying input stream implementation. We know that not all input streams support the mark() method and therefore can't be reset(), which is why we call markSupported() before attempting to calculate the checksum. So why is markSupported() succeeding, but then reset() is failing after reading the input stream? It seems like we need to track this down between the interaction of MNodeService and MultipartMNode.getReplica().

History

#1 - 05/11/2017 07:25 AM - Chris Jones

- Description updated

#2 - 05/11/2017 04:09 PM - Jing Tao

I debug the replicate method and found the class of input stream is org.apache.commons.io.input.AutoCloseInputStream.
The code which has the issue is here:

```
if (object.markSupported()) {
                Checksum givenChecksum = sysmeta.getChecksum();
```

```
                Checksum computedChecksum = null;
                try {
                    computedChecksum = ChecksumUtil.checksum(object, givenChecksum.getAlgorithm());
                    logMetacat.info("
MNodeService.replicate - after checksum, the class of object inputstream is "
+object.getClass().getCanonicalName()+". Does it support the reset method? The answer is "
+object.markSupported());
                    object.reset();

                } ...
               .....
       }
```

I found the input stream object supported the mark/reset first. But after the code ran the statement " computedChecksum = ChecksumUtil.checksum(object, givenChecksum.getAlgorithm());", the input stream doesn't support the mark/reset anymore!
The checksum method look like

```
 public static Checksum checksum(InputStream is, String checksumAlgorithm) throws NoSuchAlgorithmException,
IOException {
        byte[] buffer = new byte[1024];
        MessageDigest complete = MessageDigest.getInstance(checksumAlgorithm);
        int numRead;

        do {
            numRead = is.read(buffer);
            if (numRead > 0) {
                complete.update(buffer, 0, numRead);
            }
        } while (numRead != -1);

        // reset if it is supported
        //
        // mark is only supported on two Java supplied InputStreams
        // BufferedInputStream and ByteArrayInputStream
        // for BufferedInputStream, reset will only go the beginning of the
        // buffer (which if shorter than the stream itself, will result
        // in resetting to the middle of the stream )

        if (is.markSupported()) {
            is.reset();
        }

        String csStr = getHex(complete.digest());
        Checksum checksum = new Checksum();
        checksum.setValue(csStr);
        checksum.setAlgorithm(checksumAlgorithm);
        return checksum;
    }
```

The method read the input stream until the end in order to compute the checksum. According the java doc, the AutoCloseInputStream will act this way:
Proxy stream that closes and discards the underlying stream as soon as the end of input has been reached or when the stream is explicitly closed.

So the checksum method closes and discards the input stream and it can't be reset again.

#3 - 05/11/2017 04:12 PM - Jing Tao

Rob confirmed that we have been use the AutoCloseInputStream a while ago. And Metacat has been use the d1_libclient_java 2.3.0 and d1_common_java 2.3.0 for a while. So the issue should be there for a long time.

#4 - 05/11/2017 06:12 PM - Matt Jones

Bummer that its been there a while. Such a hugely serious bug as this should have been caught by a test. I suggest that your first step would be to modify the Metacat test suite to detect when replication is not working. Does edu.ucsb.nceas.metacat.replication.ReplicationTest detect a failure to replicate? If not, then that test should probably be extended to be sure replication is thoroughly tested and shows the error, which will help us avoid regressions like this. Probably some of the CN integration tests should be extended as well. Once the tests are working well, it would be good to make sure tests are all passing for metacat and other key infrastructure packages before new releases of libclient get approved. Did you run all of the test suites before the most recent release of metacat? Is Jenkins running the metacat test suite?

#5 - 05/11/2017 08:10 PM - Jing Tao

But from Chris' table, those failures have started recently (May, 2017). So I am confused.

Before releasing Metacat, I always ran the test suite and everything passed. Today I looked at the replicate test in the MNodeServiceTest class and

found it doesn't expect the test success :)

I tried to write the test again and found it was hard. Some certificate issues are hard to overcome but I will work on it.

By the way, the edu.ucsb.nceas.metacat.replication.ReplicationTest is for the Metacat replication (Metacat API) and it works well. This issue is about the Dataone replication.

#6 - 05/12/2017 11:41 AM - Jing Tao

Chris and I discussed the issue today. We propose the solution this way:
1. Since MN.replicate and D1NodeService.create have the duplicated code for checking checksum, the checksum code will be removed from the MN.replicate method. The MN.replicate will depend on D1NodeService.create for the checking.
2. In the D1NodeService, we will check the checksum after the object input stream is serialized into the disk instead of the checking of the input stream. This can avoid checking checksum can close the AutoCloseInputStream.
3. If the calculated checksum doesn't match the one in the given system metadata, Metacat will delete the object, records on the systemmetadata, identifier and et al tables. It will be a totally rollback.

Possible issue:
Since the checksum will be computed from the file in the disk, we worry if there is a performance issue.

#7 - 05/12/2017 12:17 PM - Matt Jones

It would be best if the stream only has to be read once to be serialized and have the checksum checked, so we don't introduce further performance delays on large objects. Maybe an inline pipe could be used to calculate the checksum while the stream is being serialized to disk? Rather than reading it twice. I think Java has some IO streams that might allow that. Not sure if it's possible, but it would be good to explore before a decision is made.

#8 - 05/16/2017 04:10 AM - Dave Vieglais

The purpose of the checksum is to ensure integrity of content written to the data store. Computing a checksum enroute to the store is arguably incorrect behavior since there is possibility that the object actually written to the store is corrupted. So it seem appropriate to compute the checksum once written to disk rather than enroute.

If computing checksum while writing the stream to disk is necessary behavior, then perhaps java.security.DigestOutputStream should be investigated.

#9 - 05/23/2017 03:37 PM - Jing Tao

- Status changed from New to Resolved

We use the DigestOutputStream to wrap the output stream which will write the bytes to disks. The checksum are computed during the write process. There is NO need to read the object twice - one for checksum and the other for writing bytes. Test the replicate method on both junit and integration test, it works.