**OAI-PMH data harvesting [LOTUS]**

- Overview
- Environment
- Summary
- Observations
- Test flow
- Test Results
  - Test 1 (PTF data set)
  - Test 2 (Bugfest data set)
  - Test 3 (Bugfest data set/5 Users)
  - Test 4 (Bugfest data set, 35 connections)
  - Test 5 (Bugfest data set, 35 connections, 2 users)
  - Test 6 (417 branch)
  - Test 7 (New modules versions)
- mod-oai-pmh capacity

**Overview**

The purpose of these set of tests is to measure performance of Lotus release and to find possible issues, bottlenecks. PERF-231, PERF-233 OAI-PMH Lotus release -performance testing IN PROGRESS

**Environment**

- mod-oai-pmh:3.7.0
- edge-oai-pmh:2.4.2
- mod-source-record-manager:3.3.0
- mod-source-record-storage:5.3.0
- mod-inventory-storage:23.0.0
- okapi:4.13.0

**Summary**

- Average response time per request with resumption token 0.852s (compared to KIWI's 0.874s, (+36%).)
- No memory leaks or unexpected CPU spikes
- Issue with long response time due to absents of underlying records for instances that may lead to timeouts is still exists MODOAIPMH-390
- Incremental calls performed - 35,667 (PTF data set). Test failed due to timeout.
- Incremental calls performed - 99,477 (Bugfest data set 1 user and 20 DB connections)*.
- Incremental calls performed - 43,968 (Bugfest data set 1 user 35 DB connections)*
- Incremental calls performed - 68,350 (Bugfest data set 5 users and 20 DB connections)*

* Note: Bugfest dataset was used because it has more SRS records than PTF's dataset.

**Observations**

1. Two identical tests may have different number of calls from edge-oai-pmh because mod-oai-pmh retrieves instances from the database in random order and in case if there are no underlying records for instances it will call DB one more time. While this process occurs in random order it may cause different number of calls to harvest same amount of data.
2. A risk of client timeouts can occur if the dataset is missing a lot of underlying SRS records exists
3. There is large number of timeouts while data transferring from inventory-storage to oai-pmh which can lead to data loss (and also may cause different number of requests from client side to complete a job).
4. All tests ended as expected and without the errors on client side. And it looks like Lotus is more stable than Kiwi.
5. Test with 35 DB_MAXPOOLSIZE is less stable due to more database timeouts occurring on mod-oai-pmh, as the result ±40% records was missed.
6. For PERF-233 - Getting issue details... we performed 5 concurrent harvests, which seemed to be working as expected (and each harvest worked on a separate process, having a different request_id). However due to the high load initially there were lots of DB connection timeouts on mod-oai-pmh, as a result the number of instances got transferred were less than expected. Below is an example of the numbers of instances that got transferred from mod-inventory-storage to oai-pmh database for each of the concurrent harvests (Number of instances expected in a transfer is 8158821 (verified by one user)).

<table>
<thead>
<tr>
<th>Harvest Request ID</th>
<th>Total Instances Transferred</th>
<th>Percentage of expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0abdab4c-2efd-484d-99dc-c7d642c7e974</td>
<td>6,488,821</td>
<td>79%</td>
</tr>
</tbody>
</table>
Test flow

Test consist of few calls:

Initial call that was performed only once

/oai/records?verb=ListRecords&metadataPrefix=marc21_withholdings&apikey=[APIKey]

Subsequent harvesting calls:

/oai/records?verb=ListRecords&apikey=[APIKey]&resumptionToken=[resumptionToken]

These calls were performed repeatedly, harvesting 100 records each time until there is no more data in [tenant]_mod_oai_pmh.instances table to harvest. [resumptionToken] was set to 100, returning in initial call response and in each harvesting call until there is no more records to harvest. When all data has being harvested - resumptionToken will not return with the response.

Test Results

Test 1 (PTF data set)

Duration 6 hr 24 m

Calls performed - 35 667

Average response time 0.642s
Test 2 (Bugfest data set)

Calls performed: 99,477

*Part before Vusers spike contain 68,791 calls

duration ±20 h

Average response time 0.852s
Average CPU usage is not higher than 30-60% except few spikes at the beginning (may be caused by repointing DB and modules restart)
There is no visible memory leaks during 20 hr of a test. Average grow of Memory usage during test is 5-10%.

RDS CPU usage are not high on regular flow. However there's spikes up to 70% during data transferring.

Test 3 (Bugfest data set/ 5Users)

Calls performed: 68 350
Duration ±3 hr 42 m
Average response time 0.831
Timeouts found 6 790 X 5000 = 33 950 000 (data lost)
Test 4 (Bugfest data set, 35 connections)

Calls performed: 43,968

Duration ±9 hr

Average response time: 796 ms

Timeouts found 752 × 5000 = 3,760,000 records missed.

Response times are more or less on the same rate without sudden spikes and gaps
Service CPU usage has spikes at the beginning of a test while data transferring process is in progress.

No memory leaks defined. Small growing of memory usage during test is expected.
DB CPU usage has a spike at the beginning of the test due to data transferring process progress.

Test 5 (Bugfest data set, 35 connections, 2 users)

Calls performed - 70 827;
Duration 10 hr 14 min
Average response time: 895 ms

Timeouts found 1861 X 5000 = 9 305 000 records missed.
Test 6 (417 branch)

Calls performed - 81 560;
Duration 14 hr 28 min
Average response time: 636 ms

Response times are predictable with only few spikes during test duration.

CPU level is strait without spikes. Spikes at the beginning of the test caused by modules restarting before test.
All EC2 instances has less than 20% CPU usage.
Test 7 (New modules versions)

Versions:

- mod-oai-pmh: 3.7.0 → 3.8.0
- edge-oai-pmh: 2.4.2
- mod-source-record-manager: 3.3.0 → 3.3.8
- mod-source-record-storage: 5.3.0 → 5.3.3
- mod-inventory-storage: 23.0.0 → 23.0.5

Calls performed - 81,544;

Duration 14 hr 28 min;
Average response time: 635 ms

Service CPU utilisation is predictable. Spikes at the beginning and end of a test was caused by switching DB to bugfest like.

Note: Here mod-oai-pmh memory usage did grow up, however it didn’t come down after test and few days later as well.
mod-oai-pmh capacity

- even with two parallel threads oai-pmh can't handle harvests;
- oai-pmh failed time after time with two parallel savings UUIDs;

Symptoms:
with two parallel harvests we can see high response times for call with resumption token. It was able to make only 346 calls per thread.

inside the logs we see:

```
java.lang.OutOfMemoryError: Java heap space
Dumping heap to /usr/ms/mod-oai-pmh.hprof ...
2022-07-05T16:05:05,708 WARN [vertx-blocked-thread-checker] BlockedThreadChecker Thread Thread[vert.x-eventloop-thread-0,5,main] has been blocked for 3302 ms, time limit is 2000 ms
```

trace:
java.lang.OutOfMemoryError: Java heap space
at java.util.Arrays.copyOf(Arrays.java:3689) ~[?:?]
at java.util.ArrayDeque.grow(ArrayDeque.java:150) ~[?:?]
at java.util.ArrayDeque.addLast(ArrayDeque.java:308) ~[?:?]
at java.util.ArrayDeque.add(ArrayDeque.java:495) ~[?:?]
as io.vertx.core.parsetools.impl.JsonParserImpl.checkTokens(JsonParserImpl.java:240) ~[ms.jar:?]
at org.folio.oaipmh.processors.OaiPmhJsonParser.handle(OaiPmhJsonParser.java:31) ~[ms.jar:?]
at io.vertx.core.parsetools.impl.JsonParserImpl.handle(JsonParserImpl.java:151) ~[ms.jar:?]
at io.vertx.ext.web.codec.impl.JsonStreamBodyCodec$1.write(JsonStreamBodyCodec.java:37) ~[ms.jar:?]
at java.util.Arrays.copyOf(Arrays.java:3689) ~[?:?]
at io.vertx.ext.web.codec.impl.JsonStreamBodyCodec$1.write(JsonStreamBodyCodec.java:22) ~[ms.jar:?]
at io.vertx.ext.web.codec.impl.JsonStreamBodyCodec$1.write(JsonStreamBodyCodec.java:60) ~[ms.jar:?]
as io.vertx.core.strems.impl.PipeImpl.lambda$to$1(PipeImpl.java:81) ~[ms.jar:?]
at io.vertx.core.impl.AbstractContext.dispatch(AbstractContext.java:96) ~[ms.jar:?]
at io.vertx.core.http.impl.HttpEventHandler.handleChunk(HttpEventHandler.java:51) ~[ms.jar:?]
at io.vertx.core.http.impl.HttpClientResponseImpl.handleChunk(HttpClientResponseImpl.java:239) ~[ms.jar:?]
at io.vertx.core.http.impl.HttpClientRequestBase$$Lambda$390/0x000000084054f840.handle(Unknown Source) ~[?:?]
at io.vertx.core.http.impl.Http1xClientConnection$StreamImpl$$Lambda$381/0x000000084054b440.handle(Unknown Source) ~[?:?]
at io.vertx.core.http.impl.Http1xClientConnection$StreamImpl.lambda$new$0(Http1xClientConnection.java:380) ~[ms.jar:?]
at io.vertx.core.http.impl.Http1xClientConnection$StreamImpl$$Lambda$381/0x000000084054b440.handle(Unknown Source) ~[?:?]
at io.vertx.core.http.impl.Http1xClientConnection$StreamImpl.lambda$handleEvent$1(Http1xClientConnection.java:583) ~[ms.jar:?]
at io.vertx.core.http.impl.Http1xClientConnection$StreamImpl.lambda$handleHttpMessage$1(Http1xClientConnection.java:682) ~[ms.jar:?]
at io.vertx.core.http.impl.Http1xClientConnection.handleHttpMessage(Http1xClientConnection.java:682) ~[ms.jar:?]
at io.vertx.core.http.impl.Http1xClientConnection.handleMessage(Http1xClientConnection.java:649) ~[ms.jar:?]
at io.vertx.core.net.impl.ConnectionBase$1.run(ConnectionBase.java:155) ~[ms.jar:?]
at io.netty.channel.AbstractChannelHandlerContext$1.run(AbstractChannelHandlerContext.java:139) ~[ms.jar:?]

After that- a lot of this errors appears in logs:

OaiPmhJsonParser Error position at error part of json is 907958256
2022-07-05T16:06:38,867 ERROR [vert.x-eventloop-thread-1] OaiPmhJsonParser: Error position at error part of json is 907958256
2022-07-05T16:06:38,867 ERROR [vert.x-eventloop-thread-1] OaiPmhJsonParser Still have 13 undecoded bytes, should not call 'feedInput'
at [Source: UNKNOWN; line: 1, column: 907958352]
2022-07-05T16:06:38,867 ERROR [vert.x-eventloop-thread-1] OaiPmhJsonParser: Error position at error part of json is 907958352

Then finally container failed. We have heap dump created automatically and can share it additionally

heap dump leak suspects:
**Problem Suspect 1**

One instance of "java.lang.Object[]" loaded by "<system class loader>" occupies 1,028,188,552 (71.82%) bytes. The instance is referenced by io.vertx.core.impl.VertxThread @ 0xa721b4c0 vert.x-eventloop-thread-2, loaded by "jdk.internal.loader.ClassLoaders$AppClassLoader @ 0xa61f80b8". The memory is accumulated in one instance of "java.lang.Object[]", loaded by "<system class loader>", which occupies 1,028,188,552 (71.82%) bytes.

**Keywords**
java.lang.Object[]
jdk.internal.loader.ClassLoaders$AppClassLoader @ 0xa61f80b8

**Problem Suspect 2**

One instance of "org.folio.oapmh.processors.OaiPmhJsonParser" loaded by "jdk.internal.loader.ClassLoaders$AppClassLoader @ 0xa61f80b8" occupies 377,402,520 (26.36%) bytes. The memory is accumulated in one instance of "java.lang.Object[]", loaded by "<system class loader>", which occupies 377,395,504 (26.36%) bytes.

**Keywords**
org.folio.oapmh.processors.OaiPmhJsonParser
jdk.internal.loader.ClassLoaders$AppClassLoader @ 0xa61f80b8
java.lang.Object[]