Performance review of the ArchivesSpace application

Prepared for Yale University by Yves Trudeau, Percona inc.

Changelog:

- 2015/08/20: Document creation

People involved:

- Steve DiSorbo, Yale University
- Robert Rice, Yale University
- Yves Trudeau, Percona

Contents

1 Summary

Yale university is implementing an opensource solution, ArchivesSpace, to manage archives and is experiencing serious performance issues, likely due to the scale of the deployment. The main goals of this engagement are to identify the issues, see if they can be fixed and if not, provide pointers to the developers to improve the application code.

The main issue found is the number of queries sent to the database. Over a single thread, queries are serialized and the total network round trip time on a tcp connection between the application and the database server must be added to the strict database query execution time. When this is taken into account, the number of queries needed and a tcp latency of 5 ms explains the problem.

2 Description of the issues

I have worked mainly on 3 issues that can be called "get resources", "view resource" and "export". I'll review each of them in more details in the following sections. There are likely many other slow operations but these are sufficient to illustrate the problem.

2.1 Get resources

This issue can be observed when someone open a session in the application, select the repository "BRBL", click on "Browse" and then "Resources". The slow application is the one going to:

```
http://testaspace.library.yale.edu:8080/resources
```

The call takes between 5 and 8s to complete. During the call, the jruby process is using one core at 100%, the database is under very little load but it does process more than 3000 queries, all being pretty fast but over a single thread, the latency of the network adds up quickly. Over a single connection on a remote host, it is hard to do more than 200 queries/s.

2.2 View resource

Once we have the list of resources, pick one and click on "view". Here, the slow application call is one similar to:

```
http://testaspace.library.yale.edu:8080/resources/1407?inline=true&resource_id=1409
```

This call takes around 6s to complete. Strangely, the application is doing two identical calls. Again, the pattern is similar to the previous issue, the application is using more than one core for the duration of the operation. The database is almost idle, but it did handled more the 4000 queries. The latency of the network is propably responsible for most of the time.

2.3 Export EAD

This issue is special. Once you have viewed a resource, you can export it. Click on "Export" and than, "Download EAD". The download is handled in the background but for resource_id 1409, it takes 7 minutes to complete. During the operation, there are a staggering 110k queries sent to the database. Clearly something is wrong in the application logic.

3 Potential explanation

3.1 Java garbage collector

The jruby instance has 8GB of heap defined. That's quite a lot and Java may have a hard time handling so much memory. I tried to lower to 1GB and saw no difference. Also, the issue would be sporadic. This is likely not the problem.

3.2 Solr

During each process, at least a query to Solr is issued. If the Solr response time is long, that will degrade the performance. Although it has a large standard deviation, most of the time Solr answer queries very quickly and yet, the issues are always present.

3.3 MySQL

MySQL processes many queries for each of the issues but the queries are all pretty simple and efficient. Most of the queries are executed in less than 0.5 ms, a time smaller to the network roundtrip time across the tcp stacks on both servers. The query cache was a potential source of

issues since it can induce locking and it has been disabled. No impacts have been observed. Some keys have been added but for queries related to the Solr index updater and again, no impacts have been observed. The CPU usage of MySQL is barely at 10% of a cpu core during problematic calls.

3.4 Javascript

The javascript could be processing too much but observation haven't shown any high load on that side.

3.5 Solr index updates

The Solr are updated frequently, every few minutes in fact. Some of the queries used by the updated were not efficient, they have been fixed but with no impact on the issues.

3.6 Application

If you consider that a few milliseconds are needed to prepare a query in jruby, send it to database server, get an answer and decode the result, then, the main issue is the number of queries done by the application. The overall roundtrip time is more than the ping time and gets worse if there's a significant amount of traffic on the network. Just consider the "Export" issue. With 110k serialized queries at 5 ms per query, we have more than 9 minutes. In fact, the number of queries, seems to be a clear indication of the time an operation will take. Here's a hping tcp roundtrip output over a GbE network between 2 physical hosts followed by a regular icmp ping:

```
root@serveur-yves:~# hping3  10.2.2.1 -S -p 80
HPING 10.2.2.1 (br0 10.2.2.1): S set, 40 headers + 0 data bytes
len=44 ip=10.2.2.1 ttl=64 DF id=0 sport=80 flags=SA seq=0 win=14600 rtt=4.4 ms
len=44 ip=10.2.2.1 ttl=64 DF id=0 sport=80 flags=SA seq=1 win=14600 rtt=4.3 ms
len=44 ip=10.2.2.1 ttl=64 DF id=0 sport=80 flags=SA seq=2 win=14600 rtt=4.3 ms
^C
--- 10.2.2.1 hping statistic ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 4.3/4.3/4.4 ms

root@serveur-yves:~# ping 10.2.2.1
PING 10.2.2.1 (10.2.2.1) 56(84) bytes of data.
64 bytes from 10.2.2.1: icmp_seq=1 ttl=64 time=0.148 ms
64 bytes from 10.2.2.1: icmp_seq=2 ttl=64 time=0.124 ms
64 bytes from 10.2.2.1: icmp_seq=3 ttl=64 time=0.141 ms
^C
--- 10.2.2.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.124/0.137/0.148/0.016 ms
```

As one can quickly noticed, the tcp round trip is about 30 times slower.

4 Possible fixes

4.1 Caching

Many of the queries are directed to almost static entities. Here's two example queries that ran numerous times during a single "Get resources" call (over about 6s).

```
Called 68 times: SELECT * FROM `enumeration_value` WHERE (`id` IN (882));
Called 12 times: SELECT * FROM `repository` WHERE (`repo_code` = '_archivesspace') LIMIT 1;
```

These results or the objects they represents should be cached or made persistent in the application, see here for more details:

```
https://github.com/jruby/jruby/wiki/Persistence
```

4.2 Object instance factory

Here's another common pattern, list of objects or values are obtained using multiple point queries. Below is an example from a "Get resources" call, remember the page calls the method twice, hence they come in pairs.

```
SELECT DISTINCT `role_id` FROM `linked_agents_rlshp` WHERE (`agent_person_id` = 11834);
SELECT DISTINCT `role_id` FROM `linked_agents_rlshp` WHERE (`agent_person_id` = 11834);
SELECT DISTINCT `role_id` FROM `linked_agents_rlshp` WHERE (`agent_person_id` = 11836);
SELECT DISTINCT `role_id` FROM `linked_agents_rlshp` WHERE (`agent_person_id` = 11836);
SELECT DISTINCT `role_id` FROM `linked_agents_rlshp` WHERE (`agent_person_id` = 11840);
SELECT DISTINCT `role_id` FROM `linked_agents_rlshp` WHERE (`agent_person_id` = 11840);
SELECT DISTINCT `role_id` FROM `linked_agents_rlshp` WHERE (`agent_person_id` = 12073);
SELECT DISTINCT `role_id` FROM `linked_agents_rlshp` WHERE (`agent_person_id` = 12073);
... 68 more
```

Clearly, this is way too many round trips, this could either be replaced by a single query returning multiple rows or the whole logic needs to be revised.

4.3 Multiple threads

If the number of queries cannot be lowered, maybe the number of threads can be increased so that more queries run in parallel, mitigating the network latency. JRuby supports multithreading from the JVM so that could be implemented easily. This solution is far from ideal though since it doesn't fix the bad behaviors and offer a rather limited scalability.

4.4 Cohosting

Cohosting the database with the application saves a significant amount of the round trip time but, this is how production is currently setup at Yale and, although performance is indeed better, there are still numerous complains about performance.

5 MySQL database changes

While looking at the databases and the queries, I made some change to the schema:

```
create index idx_repo_id_sysmtime on top_container (repo_id,system_mtime);
create index idx_ts on deleted_records (timestamp);
create index idx_repo_id_sysmtime on archival_object (repo_id,system_mtime);
create index idx_repo_id_sysmtime on event (repo_id,system_mtime);
drop index repo_id on event;
create index idx_repo_id_sysmtime on resource (repo_id,system_mtime);
create index idx_repo_id_sysmtime on accession (repo_id,system_mtime);
create index idx_repo_id_root_record_id on archival_object (repo_id,root_record_id);
```

But most/all of these are used by the Indexer so the gains on the web side are not there. The query cache was enable on the database and although the hit ratio was good, the update of the session table were often colliding on the query cache mutex. In the my.cnf file, I modified:

```
query_cache_type=0  # was 1
query_cache_size=0  # was 1024M
```