Theme: Shanghai library FOLIO project

Time: May 31, 2022 07:00pm (EST) / June 1, 2022 07:00am (GMT+8)

Attendees:
Martin Tran (Team Lead, Performance Task Force of FOLIO, EBSCO)
Vincent Bareau (Enterprise Architect, EBSCO)
Gang Zhou (Project manager, Shanghai library)
Sha Jiang (Technical Director, Jiatu)
Lucy Liu (Product Owner, Folio China)

Notes:

1.We are testing circulation modules in the SHL's environment and are trying to add stress to the system. When there are 10 queries per second, the response time is about one second per query, which is acceptable. However, when there are 20 queries per second, the average response time is about 3-4 seconds per query. The longest time can be over 10 seconds. How can we reduce the response time? (Sha Jiang)

Martin:

- Do a capacity test and slowly increase the number of users every 2 or 5 minutes. That way, you can see when exactly things start to fall apart. Maybe there is an inflection point somewhere between 20-30 users. And if we can know which point that is, we can look at other graphs to see what is causing the problem. Maybe it's the modules CPU, modules like mod-inventory-storage or mod-inventory that get stuck at that point, or it's the DB that gets overwhelmed. With KR I/O, you can set the ramp up time of the users to be 5 minutes per user. For 30 users, you will need at least 150 minutes to run the test. For each user, you can be sure there's no spikes within the 5 minutes. So do a manual or automated capacity test so that you can look at the graphs carefully as you gradually add more users, so you can see where in the system, database, modules or even instance begin to pile up. Maybe Okapi is a choking point, too. So we need to know when it happens.
- Something to clarify:
  - What is the number of concurrent users tested?
  - Do you have the general JMeter graph which shows the rate of call in checkout and checkin?
  - Do you call the checkout workflow APIs or just the POST /checkout-by-barcode API call?

Sha Jiang: Answers to the questions:

- 30 users.
- No.
- Through many APIs.

Martin: We need to see the module graph for CPU. If you do incremental addition of users, we would see each relevant module in the system, i.e., okapi, circulation-storage, circulation-inventory. Even inventory-storage, mod-authtoken, mod-permission may play a role as well. We can see how they behave as we add more users. If one module CPU starts to take off compared to the other ones, that may be the problem. Then we can dig deeper to see whether or not that module is touched to the database. If the database is the problem, you can turn on the slow query filter in the database so that it can catch the slow queries.

Sha Jiang: We found no slow queries in the database.

Martin: If so, was the database CPU usage pretty constant, as expected?

Sha Jiang: We have the database monitor in the document. We can see a peak in the CPU usage for DB services. But the highest CPU usage was just 30 percent. So it's normal.

Martin:

- Yes. And you have network bandwidth. Maybe the network card can't process that much data.
- These are all 30-user tests. Do you have the view of the 20 users test? We need to compare the 20-user test vs. the 30-user test. Or, add the users incrementally to see the changes in the modules.

Sha Jiang: Will do.

To do: Shang Jiang will do another test as Martin suggested and share a new report. The discussions will continue on Slack. Another meeting will be scheduled if needed.

2.If we need to keep the indexes that we added on our own, how can we reduce the response time? (Sha Jiang)

Martin: If index is really the problem, maybe redesign?

Vince: The first thing is to decide if the indexes are problematic or not. Remove the index to see if it's getting better. If yes, you may need to look at solutions. There are ways to optimize indexes. Sometimes you make a composite index. Sometimes you need to break it into multiple indexes.

Martin: Another way to troubleshoot the problem. In KR I/O or in JMeter, you will get a list of the API calls in the workflow. You can sort them by response time. As you run tests between 20 users and 30 users, you can also narrow down which API is becoming much slower. That's another way to troubleshoot.