Runtime Infrastructure - Feature #6673

rework appserver agent approach to use a shared queue instead of a check-in/check-out registry

08/15/2022 08:12 AM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #6424: fill gaps in Java Open Client repla...	**WIP**

History

#2 - 08/15/2022 08:27 AM - Greg Shah

In #6672, the simple test case shows significant performance degradation between the single-user and multi-user cases. Recent discussions and review of the architecture of the appserver agents has highlighted a possible cause.

The current approach (see Appserver Support is designed with an AgentPool which is a highly synchronized registry of available appserver agents. Each appserver request (which may come from legacy REST/SOAP/webhandler OR from an appserver client) will block attempting to obtain an agent from that registry. In simple single user mode, this will not show any issues. In multi-user mode I suspect this will become a bottleneck. At the least it is unnecessarily complex and implements a facility which requires more task switching and more processing than is needed.

I think a faster, simpler and more scaleable approach is to implement a thread-safe queue approach. Appserver requests can be posted into a thread-safe queue and each agent will simply block reading the queue. Posting into the queue would return a Future which will be used to track/return the result to the caller. For the cases where an agent becomes bound to the particular caller, we should create a unique queue that the caller will use until the agent becomes unbound. Bound agents only listen to their own queue and unbound agents listen to a shared queue. Otherwise the processing is identical.

#5 - 09/29/2022 04:25 PM - Greg Shah

I'd like to take a look at this case. I plan to instrument this code and measure the cost of various approaches and at various levels of concurrent requests.

I could use the POC/customer application code to test this, but it would be easier if I had something standalone that I could use to test with. Do we have anything I can use as a starting point?

#7 - 05/01/2024 01:33 AM - Galya B

- Related to Feature #6424: fill gaps in Java Open Client replacement (appserver client support) added