Dax Co-work 2019-09-04 Meeting notes

Date

04 Sep 2019

Attendees

- Igor Gaponenko
- Kenny Lo
- John Gates
- Andy Hanushevsky
- Unknown User (npease)
- Fritz Mueller
- Andy Salnikov
- Fabrice Jammes
- Unknown User (cbanek)

Goals

Discussion items

Time	Item	Who	Notes
	Promethus Demo	Fabrice	
	Query ID		Some queries don't get listed in the query history because they fail before the query id gets generated (parse errors fall into this category). Can we 1) generate earlier? 2) Replace with UUID? ...it's possible, not too hard, to allocate query ids earlier & record failed parse queries in the query history. There are some strong feelings about UUIDs vs. sequential integer IDs. This choice is somewhat orthogonal to generating the query ID earlier. We can't totally get rid of the need for an identifier that's used early between the proxy and the top level of the czar plugin. TBD what to do, Fritz Mueller will make an executive decision at some point.
	Qserv logs not getting into Elastic Search		It looks like there's a networking problem. We're hoping a restart will fix things. If not, we'll contact NCSA and have them jiggle the handle. UPDATE: Unknown User (cbanek) was able to get things working again. Some changes had been made to the system configuration and who did it was not immediately obvious. She may follow up on that.

	An effect of changes in the **Logger** service configuration on the performance and resource utilization of Qserv *czar*	Igor	The study was triggered by the following observations on Qserv *czar* in **PDAC** (**lsp-int** at **NCSA**):
			
			• the log file */qserv/log/mysql-proxy-lua.log* of the service grows quite rapidly over time. Depending on how heavily the service is used the file could grow as fast as **100 GB / 24 hours**.
			• the XRootD/SSI service (tagged as **xrdssi.msgs** in the log files) was found responsible for nearly **50%** of all logged messages. These messages were posted at the **DEBUG** level.
			• when Qsev *czar* is processing queries it's aggregate CPU consumption almost never exceeds **400%** (roughly - 4 cores). Note that the machine is equipped with x2 CPUs (14 core per each CPU, or 28 cores in total).
			• when *czar* is put under a heavy load (a typical scenario: launching a few *shared scan* or alike queries within a minute) lists of queued tasks on the worker nodes grow quite slowly, as if Qserv *czar* is unable to push chunk-specific queries to workers. This prevents Qserv workers to get into the most favorable mode an process multiple queries in the *shared scan* regime.
			
			This let to a theory that Qserv **Logger** wishing *czar* could be one of the potential bottlenecks limiting the performance of the service. To test this theory an alternative configuration of the **Logger** at Qserv *master* container was attempted. The configuration is now passed into the container (at its start time) as the following Docker *volume:*
			
			`% docker inspect qserv`
			`...`
			`"Mounts": [`
			`{`
			`"Type": "bind",`
			`"Source": "/qserv/config/log4cxx.czar.properties",`
			`"Destination": "/qserv/stack/stack/current/Linux64/qserv/tickets.DM-19156-g007a958c02/share/qserv/configuration/templates/etc/log4cxx.czar.properties",`
			`"Mode": "ro",`
			`"RW": false,`
			`"Propagation": "rprivate"`
			`},`
			`...`
			
			The first set of tests conducted within the effort was aiming at testing if disabling messages from **xrdssi.msgs** would have any affect on the performance of the service. During the test Qserv *czar's* **Logger** was reconfigured to increase the default threshold of the logger from **DEBUG** to **ERROR**. Unfortunately, the test didn't show any improvements (apart from the expected reduction of the log file payload by a factor of 2).
			
			The second test was of just an expatiation of the first one by suppressing **ALL** messages logged by Qserv *czar* below level *ERROR*. This **DID** result in a noticeable growth of the CPU consumption (by the *czar*) from **400%** up to **700 %** (which had never been seen earlier). It was also observed that the number of tasks at the worker queues was growing much faster (on the order of 10 or much higher).
			
			CONCLUSIONS (directions for further studies): the **Logger** is obviously one (though, not the only) of the bottlenecks within Qserv *czar* . It's not clear though, what exactly is happening, if there is an internal *mutex* (acting as a single point of congestion for Qserv threads), or if there is a general performance burden for using C++ **iostream** class (which is behind the implementation of the **Logger**) due to alleged dependency on **locale** (as speculated by Andy Hanushevsky).

Action items

- [ ]