Server-Side Thread Pooling for Client-Server TLS

Problem

GEODE-6150 was created in December, 2018. Mario Ivanac started working on it mid-2020. Here are the PRs:

- #5270 created 6/17/2020 (closed)
- #6063 created 2/27/2021 (closed)
- #6101 created 3/8/2021 (closed)
- #6156 created 3/18/2021 OPEN

The ticket centers on the max-threads configuration property. By default, Geode client-server communications start a new server-side thread for each new client connection. That server-side logic uses “old I/O” i.e. blocking I/O where each server-side task gets its own dedicated thread and blocks on socket read (and write.) But when max-threads is specified, a separate code path is used to sever the tight coupling between server-side tasks and threads. In that case “new I/O” is used. A Selector is used to share a pool of threads among a potentially much larger set of server-side tasks.

The problem the bug aims to fix, is that TLS/SSL doesn’t work with max-threads. (a fact that was only in January, 2021 resolved in the gfsh documentation with GEODE-8796.) Enhancing Geode to respect the max-threads configuration parameter will enable users to secure client-server communication with TLS, and limit the number of server-side threads required to service those clients.

Anti-Goals

(none)

Solution

Before the changes described in this RFC and GEODE-6150, there were two main server-side code paths for client-server communication, implementing three of four possible configuration combinations. Here are those four configurations:

Client-Server max-threads + TLS Configurations

1. No TLS, no max-threads PATH #1
2. No TLS, max-threads PATH #2
3. TLS, no-max-threads PATH #1
4. TLS, max-threads (NEW! to be added by this RFC and GEODE-6150)

A question arises in adding support for configuration (4): shall we introduce a brand new (third) code path, or can we modify PATH #2 to accommodate the new code path. The PR #6063 (created 2/27/2021 OPEN) implements the decision to modify PATH #2 (call the modified path: PATH #2a) to use an NioFilter with essentially a null cipher (the NioPlainEngine) for case (2) and a non-null cipher (NioSslEngine) for case (4), the new case.

This solution strikes an appropriate balance between code reuse, maintainability, performance, and risk of regression (functional, performance). In choosing this approach we will modify the implementation of case (2) by putting the NioPlainEngine in that path.

By modifying the code serving case (2) to support both that case and case (4) we run the risk of breaking functionality (and performance) for users already relying on case (2). See below for mitigations.

Current status is that we have 2 stable PRs (all test are passing):

- #6063 - solution implements the decision to modify PATH #2 (call the modified path: PATH #2a) to use an NioFilter with essentially a null cipher for case (2) and a non-null cipher (NioSslEngine) for case (4), the new case.
- #6156 - solution implements the decision to modify PATH #2 to use an NioFilter with essentially (the NioPlainEngine) for case (2) and a (NioSslEngine) for case (4), the new case.

Next step could be performance tests, to see which alternative is better.

Changes and Additions to Public Interfaces
Performance Impact

There is a potential to negatively impact performance of existing functionality (case (2) above.) This is a critical performance path in Geode. As risk mitigation we intend to use various performance tests in https://github.com/apache/geode-benchmarks to ensure that the enhancement does not cause a performance regression.

The test topology will be: 1 locator, 2 cache servers, and 1 client JVM.

The machine specs are: Amazon EC2 c5_18_xlarge which has 72 vCPU, 144GB RAM, 25Gb networking.

Geode max-threads will be set to (4 times the number of cores) on each of the two cache servers.

The benchmark will be configured for NO TLS/SSL.

The benchmark tuning options -PwithGc, -PwithHeap, -PwithThreads, -PwithWarmup, -PwithDuration are also TBD.

We'll run all combinations of these benchmarks:

{ Partitioned, Replicated } x { Get, Put, PutAll } x { (default: Portfolio object), Long, Bytes }

These benchmarks are TBD:

- PartitionedIndexedQueryBenchmark
- PartitionedNonIndexedQueryBenchmark

We'll also run TBD proprietary client-server performance tests and verify that we haven't regressed.

Backwards Compatibility and Upgrade Path

(this feature introduces no backward compatibility or upgrade issues)

Functional Testing

The full Geode test suite will be run to ensure no regressions are introduced in case (2).

We'll also run TBD proprietary client-server functional tests and verify that we haven't regressed.

These tests will be updated to add max-threads variants:

TLS-enabled tests:

- either enhance SSLNoClientAuthDUnitTest to have a max-threads variant or introduce a new similar test with max-threads > 0
- (others TBD)
- Maybe one or all of the various SNI acceptance tests: SingleServerSNIAcceptanceTest, DualServerSNIAcceptanceTest, ClientSNIAcceptanceTest, ClientSNIProxyAcceptanceTest

Prior Art

Without the proposed solution, Geode users wishing to use TLS to secure client-server communication cannot limit the number of server-side threads (via max-threads) used to service client requests.

FAQ

Answers to questions you’ve commonly been asked after requesting comments for this proposal.

Errata

What are minor adjustments that had to be made to the proposal since it was approved?