Debugging Complex Issues in Web Applications

Mark Thomas (he/him/his)

Staff Engineer

September 2021

Introductions

Mark Thomas

Tomcat committer since 2003

Tomcat PMC member since 2005

markt@apache.org

SpringSource / VMware / Pivotal / VMware since 2008

ASF, Eclipse, VMware

Disclaimer

This presentation may contain product features or functionality that are currently under development.

This overview of new technology represents no commitment from VMware to deliver these features in any generally available product.

Features are subject to change, and must not be included in contracts, purchase orders, or sales agreements of any kind.

Technical feasibility and market demand will affect final delivery.

Pricing and packaging for any new features/functionality/technology discussed or presented, have not been determined.

The information in this presentation is for informational purposes only and may not be incorporated into any contract. There is no commitment or obligation to deliver any items presented herein.

Agenda

- What do I mean by "Complex Issues"?
- Methodology
- Statistics
- Uses cases
  - Concurrent HTTP/2 bulk transfers
  - Terminating connections without a response
  - Response write does not complete
- Summary of techniques
- Questions

Background

Complex Issues

Subjective

Not 100% repeatable

- Less repeatable typically means harder to debug

Only occurs under load

Concurrency

Methodology

Issue needs to be sufficiently repeatable that this process is viable

Identify, at a high level, where the issue is occurring

Record state before

Record state after

Check the before and after states are consistent with expectations

Narrow down the focus

- Typically move down the stack
- Sometimes record state in more than two locations for speed

Aiming for binary “working as expected” / “not working as expected” test

Statistics

Keep notes

- What changed
- Test results

Need more samples than you think

- At least 20 tests
- At least 5 passes
- At least 5 failures

May have multiple root causes

- A fix, or the fix?

Use Case 1

Large, concurrent HTTP/2 responses

Original Report

Originated on Tomcat users mailing list

- 2021-06-16
- Trouble with HTTP/2 during bulk data transfer (server -> client)
- https://tomcat.markmail.org/thread/texcre345tmyn337

Well-written bug report

Multiple HTTP/2 steams on same connection are blocked indefinitely

Described the scenario in sufficient detail for a test case to be coded

- Writing large files
- Three or more concurrent streams

Described working configurations (HTTP/1.1) and non-working configurations (HTTP/2)

Original Report (cont.)

Described what they tried to vary (non-blocking, IOUtils)

Provided all relevant version numbers

Provided test case with source code

Provided some analysis

- HTTP/2 streams were waiting for a semaphore

The only thing they didn't mention was repeatability

They responded to this question in ~60 mins

Review relevant source code

HTTP/2 Connections are multiplexed

- Multiple streams trying to write
- Use a semaphore to ensure they write one at a time

HTTP/2 uses an internal async writes with a CompletionHandler

- If write can't complete, socket is added to the Poller
- An writeOperation field holds an OperationState instance that tracks the state of the async write

Narrow Focus of Investigation

Recreated within ~90 mins of original report

- Indicator of the quality of the original report
- Enables us to quickly include / exclude functionality

Rémy suggested disabling asyncIO

- This provided a workaround (~4 hours)
- Brief discussion on merits of using asyncIO

Confirmed NIO2 was unaffected

Identify Root Cause

Could see threads waiting for semaphore

- Semaphore released by Poller indicating ready for write

Start with code review

- Possible root cause – non-volatile interestOps flag
- Initial testing was positive, larger sample size ruled it out

Decide to debug Socket / Poller interactions

Debugging the Poller

Debug logs change the timing

- Issue a lot less repeatable

Need to change logging strategy

- Copy relevant information to local variables
- Log them AFTER the failure / event of interest
- Much less likely to affect timing

After a lot of debug logging

- Poller was working correctly

Debugging the Write Notification

Poller was signaling write was possible

- Trace the write notification
- OperationState was null so event wasn't processed

Why was OperationState null

- Code review
- Found (potential) root cause

Fix and Confirm

Applied fix

- Local testing confirmed fix (0 failures in 20 runs)

Explain fix on mailing list

- Other committers can check my reasoning

Check if same error exists elsewhere

- Read also affected

Explanation

For those that want to understand the problem

You'll need the code in front of you

The write

- https://github.com/apache/tomcat/blob/main/java/org/apache/tomcat/util/net/SocketWrapperBase.java#L1364

The associated completion handler

- https://github.com/apache/tomcat/blob/main/java/org/apache/tomcat/util/net/SocketWrapperBase.java#L1044

Code Walk-through

T1 obtains the write semaphore (L1366)

T1 creates an OperationState and sets writeOperation (L1390)

the async write for T1 completes and the completion handler is called

T1's completion handler releases the semaphore (L1046)

T2 obtains the write semaphore (L1366)

T2 creates an OperationState and sets writeOperation (L1390)

T1's completion handler clears writeOperation (L1050)

the async write for T2 does not complete and the socket is added to the Poller

The Poller signals the socket is ready for write

The Poller finds writeOperation is null so performs a normal dispatch for write

The async write times out as it never receives the notification from the Poller

Fix

The fix is to swap the clearing of writeOperation and the releasing of the semaphore

- https://github.com/apache/tomcat/commit/92b91857

Use Case 2

Connection drops before writing response

Original report

Originated on Tomcat users mailing list

- 2020-10-16
- Weirdest Tomcat Behaviour Ever
- https://tomcat.markmail.org/thread/bf6oz7ibxccvodd2

Well-written bug report

Very occasionally Tomcat does not send a response

The access log shows a response

No exceptions in the logs

Wireshark shows the GET request followed by a TCP FIN from Tomcat

All version information provided

Information Gathering

Asked various questions to try and eliminate features and/or possible failure modes

The response was small ~1k

- Small enough to be buffered entirely at various points in the network stack

Typical response time was 60ms

- Not going to be timeout related

The FIN was sent ~100µs after the request was received

- Further confirmation it isn't timeout related

The request is fully sent

- Not waiting for the rest of the request

User agent -> Firewall -> Nginx -> Tomcat

Information Gathering (cont.)

HTTP/1.0 request

- Rules out HTTP/2 code

Network traces obtained from both nginx and Tomcat

- Great to confirm behaviour
- Nothing that indicates a possible root cause

Application has unique request IDs that aid correlation across logs

Custom debug code tricky, but not impossible

Issue started in the last month or so

- No obvious changes

Systems are lightly loaded

User suggestion

Another user suggests using strace

- OP didn't see the suggestion
- It struck me as too low level at this point
- In hindsight, it might have saved some time

Configuration changes

Switching from BIO to NIO didn't fix it

- Issue not in the endpoint specific code
- JVM issue less likely

Timings suggest JSP is generating the response

- Adding %b to the access log confirms this

Happens with BIO so sendfile isn't a factor

No compression so GZIP isn't a factor

No obvious explanation

- Add custom debug logging to help narrow down search
- https://github.com/markt-asf/tomcat/tree/debug-7.0.72

Debug logging v1 to v2

Debug logging v1

- Response was written
- Socket was closed before this
- Correct objects were used

Debug logging v2

- Socket closed long before Tomcat tries to write
- Neither Tomcat nor the application are closing the socket

Debug logging v3 to v5

Debug logging v3

- Swallowed exception message "Bad file descriptor"
- Exception swallowed because it was assumed to be a dropped client connection
- Tomcat changed to log all such exceptions at debug

Possibly running out of file descriptors?

- No

Debug logging v4

- No other active connections between nginx and Tomcat when the issue occurs

Debug logging v5

- No indication of JRE mis-handling file descriptors

strace

strace shows that the socket close came from somewhere in the JRE

Try to correlate with thread dumps to identify where the close is occurring

Possibility it was database related

- False alarm

strace showed a native library incorrectly managing file descriptors associated with a fork

The native library closed a file descriptor twice

In same cases, that descriptor has already been re-used for the network connection

When this happened, the network connection was closed

Resolution

The vendor accepted the native library was at fault

- PDFTron

The vendor provided instructions to disable the use of the library that was triggering the issue

We recommended a switch to HTTP/1.1 for the nginx / Tomcat connection

- Fewer new connections, less chances of file descriptor reuse

Use Case 3

Response write does not complete

Original report

Originated on Spring Framework’s issue tracker

- 2021-01-23
- Response writing fails to complete with WebFlux on Tomcat
- https://github.com/spring-projects/spring-framework/issues/26434

Well-written bug report

Test case provided

Small number of failures under load

All version information provided

Multiple issues identified

Spring handling of failed flushes

Tomcat error handling

- Attempting to flush the error page content after an I/O write error overwrites original error
- IO errors in the first call to `onWritePossible()` were not triggering the error listener
- Tomcat assumed an `IOException` in `onWritePossible()` would be seen by the container (WebFlux swallowed them)

Fixed a Spring issue triggering a `IOException: Broken pipe`

Tomcat now calls error listener before internal error handling

And we end with an OS bug

- https://bugs.openjdk.java.net/browse/JDK-8263243
- https://bugs.launchpad.net/ubuntu/+source/linux/+bug/1924298

Techniques

Other debugging techniques

Logging / Wireshark

- use a 5 minute rolling window

Use ERROR level logs

Issues that depend on network latency

- simulate latency in the hypervisor

Choose you load generator carefully

Multiple physical machines

- Pull out network cable to simulate lost connection

Multiple platforms

- VM or bare metal seems to be less of an issue

Questions