Apache Tomcat Tuning for Production

Filip Hanik & Mark Thomas
SpringSource
September 2008

About SpringSource

- Created, developed and leads Spring
- Rod Johnson, CEO and father of Spring
- Deep Apache involvement
- Provide full commercial support for Apache Tomcat
- Offer a host of production ready Apache and Spring-based products
- Offices worldwide – US, UK, France, Netherlands, Germany, Australia and Canada

About the presenters

- Apache Tomcat Committers for 5+ years
- Apache Software Foundation members
- SpringSource technologists
  - Apache development team
  - Senior Software Engineers and Consultants
- Performance, troubleshooting and security experts

Tomcat versions

- Focused on Tomcat 6.0.x
- 5.5.x – bugs and security
- 4.1.x – occasional bugs and security
- 3.x, 4.0.x, 5.0.x are unsupported
- 7.x is on the horizon with some initial discussions on the mailing list

Agenda

- Review of part 1
- Logging
- The role of TCP and HTTP in performance
- Load balancing
- Tuning connectors for your network
  - Timeouts are critical
- Content delivery & caching
- Tuning the JVM
- Q&A

Review of Part 1: The process

- Understand the system architecture
- Stabilise the system
- Set performance target
- Measure current performance
- Identify the current bottleneck
- Fix the **root cause** of the bottleneck
- Repeat until you meet the target

Review of Part 1: Availability

- Clustering
- Multiple Tomcat instances
- Load balancer
  - httpd, IIS, hardware, etc.
- How stateful is your applciation?
- How seamless do you want fail over to be?

Review of Part 1: Reliability

- Redeployment can cause memory leaks
  - Include redeployment in your testing
- Design for scalability
  - Include clustering in your testing
- In highly concurrent environments turn off KeepAlive

Apache Tomcat in Production

- Applications usually account for >80% of request processing time
- Recall the tuning process
- Focus your efforts on the bottlenecks
- Don't try and guess the bottleneck

Apache Tomcat in Production

- Out Of The Box Tomcat
  - Tomcat ready for production
- No JVM settings applied
- Tuning is fairly limited
  - So lets get to it

Apache Tomcat in Production

- What will we tune?
  - Adjust logging
  - Tune connectors
  - Tune content cache
  - JVM settings

Production Logging

- Tomcat’s logging is OK
- But one must understand what it does
  - Catch-all logger goes to stdout and file
  - No overflow protection
  - Synchronized logging

Production Logging

- Remove duplicate logging (logging.properties)
  - Default logs to stdout and file

```
.handlers = 1catalina.org.apache.juli.FileHandler,
            java.util.logging.ConsoleHandler
```

  - Remove logging to stdout – no rotation

```
.handlers = 1catalina.org.apache.juli.FileHandler
```

Production Logging

- Overflow protection (logging.properties)
  - Use if your logs have the risk of overflowing

```
handlers = 1catalina.org.apache.juli.FileHandler,…
```

  - The logger name contains class file
  - Swap in custom implementation (or JDK)

```
handlers = 1catalina.java.util.logging.FileHandler,…
```

  - Tomcat accepts a custom class name

Production Logging

- Overflow protection (logging.properties)
  - Size based rotation using JVM logger

```
handlers = 1catalina.java.util.logging.FileHandler,…
```

  - No more than 5x20mb files

```
1catalina.java.util.logging.FileHandler.pattern =
    ${catalina.base}/logs/catalina.%g.log
1catalina.java.util.logging.FileHandler.limit = 20000000
1catalina.java.util.logging.FileHandler.count = 5
```

Production Logging

- Synchronous logging (logging.properties)
  - Can become a bottleneck
  - We don't want disk IO to become the limiting factor

```
handlers = 1catalina.my.custom.AsyncFileHandler,…
```

  - Asynchronous file handlers must be able to
    - Limit log queue in size
    - Fall back to sync logging or
    - Throw away overflow log records to avoid out of memory error

Apache Tomcat in Production

- Tuning Tomcat connectors
  - server.xml
  - <Connector>
- To properly tune one must
  - Understand the TCP protocol
  - Understand how the CPU works
  - Understand load balancers and their algorithms

TCP

- Layer 4 in the OSI stack
- Session based
- TCP stack implementation keeps state
- Flow control
- Delivery guarantee

TCP: Session based

- Setup and break down handshakes
- Client response time
  - Handshakes add to HTTP transaction time
  - HTTP keep alive improves client response time
  - HTTP keep alive takes up server resources

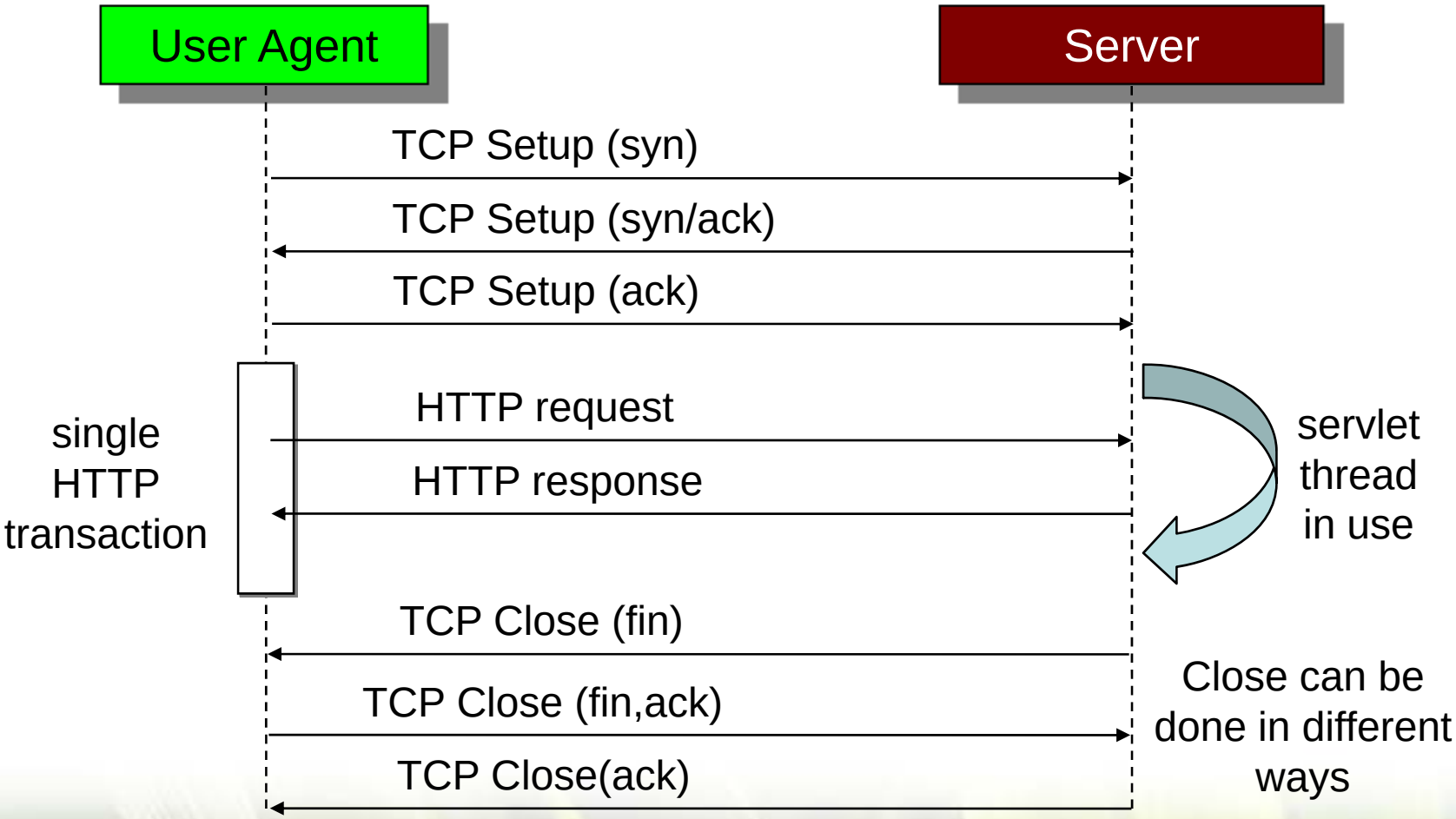
TCP: Stateful protocol

- Each connection represented by
  - source address
  - source port
  - destination address
  - destination port
- This is all the information a layer 4 load balancer has for load balancing

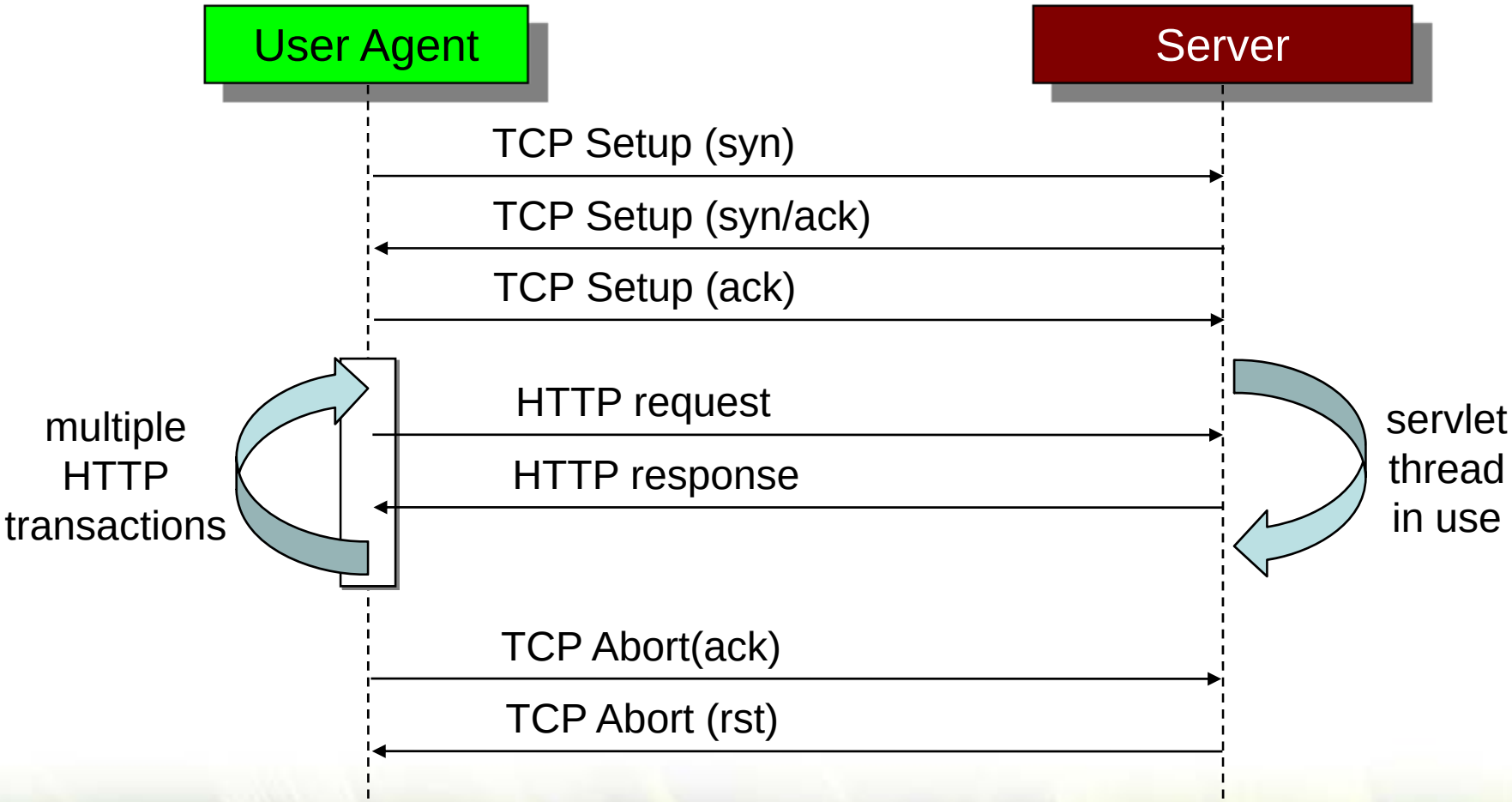
TCP: Flow control

- Prevents buffer overflow and lost data
- Server must adjust write speed to client's ability to read
- Servlet specification is blocking IO
  - Utilize a thread for the entire HTTP transaction
  - For static content, Tomcat offers SEND_FILE with APR and NIO connectors

TCP: No Keep-Alive



TCP: Keep-Alive



TCP: Summary

- How does TCP affect our system?
  - Traffic patterns
    - High concurrency/short requests
    - Low concurrency/long requests
    - Static content
    - Dynamic content
    - Combination
    - Variations
  - It’s these patterns that decide how to tune our system

HTTP

- Layer 7 in the OSI stack
- Stateless protocol
- Not tied to TCP
- Can be leveraged by load balancers

HTTPS

- HTTP over SSL
- Expensive handshake
  - Keep alive makes a big difference
- Encryption hides HTTP from routing devices
- For any appliances, such as LB, this means
  - Fallback to layer 4

Load Balancing: TCP/HTTP

- TCP
  - Based on destination address/port
  - Connection centric – 1:1
  - Can lead to uneven loads
- HTTP
  - Based on HTTP headers
  - Can reuse server connections
  - Can drain sessions

Load Balancing: Algorithms

- Load balancing
  - Connection limits
  - Reusing connections
  - Traffic shaping
- Load balancing algorithm drive Tomcat configuration choices

Apache Tomcat: HTTP/S

- Our tuning options
  - Threads
  - Keep alive requests
  - TCP Backlog (acceptCount)
  - connectionTimeout
- Different connectors
  - BIO – Blocking Java connector, default
  - APR – Uses native C code for IO
  - NIO – Non blocking Java connectors

Apache Tomcat: HTTP/S

- Disclaimer
  - Tuning options are meant for working and high performing applications
  - Options will not fix bad application behavior
  - If application is not tuned
    - Effects of tuning will not be visible
    - Situation can worsen

Which connector?

- Use BIO if:
  - Stability is the highest priority
    - APR and NIO are more recent
  - Most content is dynamic
  - Keep alive is not a determining factor

protocol="org.apache.coyote.http11.Http11Protocol"

Which connector?

- Use APR if:
  - SSL is terminated at Tomcat
  - Platforms are Linux or Windows
  - Keep alive is important
  - Lots of static content
  - Using Comet feature

```
protocol="org.apache.coyote.http11.Http11AprProtocol"
```

Which connector?

- Use NIO if:
  - Compiling APR is not an option
  - SSL is terminated at Tomcat
  - Keep alive is important
  - Lots of static content
  - Using Comet features

```
protocol="org.apache.coyote.http11.Http11NioProtocol"
```

Which connector?

- If uncertain:
  - Use BIO connector
  - Most mature code, both in Tomcat and JVM
  - Will not break down
  - Auto tune feature to disable keep alive
    - When hitting 75% if maxThreads in connection count

```
protocol="org.apache.coyote.http11.Http11Protocol"
```

```
protocol="HTTP/1.1" <!--same as above-->
```

Which connector?

- What about AJP?:
  - We don’t recommend it
  - Doesn’t perform better than HTTP
  - You lose a lot of troubleshooting and configuration abilities using it

```
protocol=“AJP/1.3”
```

Tuning threads

- maxThreads
  - Typical range 200-800
  - Maximum nr of concurrent requests
  - For BIO, max nr of open/active connections
  - Good starting value 400

Tuning threads

- maxThreads="400"
  - Decrease if you see heavy CPU usage
    - Application might be CPU bound instead of IO bound
    - Find out what is causing CPU usage
  - Increase if you don't see much CPU usage
    - Applications could be synchronized -> no gain
    - Take into account other resources, such as database connections

Tuning keep alive

- maxKeepAliveRequests
  - Typical values 1, 100-200
  - Represents the number of requests Tomcat will handle on a TCP connection
  - Set to 1 disables keep alive
  - connectionTimeout/keepAliveTimeout controls the timeout in between requests

Tuning keep alive

- maxKeepAliveRequests
  - Set to 1 if
    - Very high concurrency
    - Not using SSL in Tomcat
    - Using layer 4 load balancer
    - Using BIO connector
  - Set to >1 if
    - Using SSL or low concurrency
    - Layer 7 load balancer with advanced features
    - Using APR or NIO connector
  - BIO connector automatically disables keep alive for high connection counts

Tuning TCP backlog

- acceptCount
  - Typical ranges 50-300
  - Represents nr of additional connections the OS should accept on your behalf
  - Useful when Tomcat can’t accept connections fast enough

Tuning TCP backlog

- acceptCount="100"
  - Increase if
    - Very high concurrency (nr of connections)
    - Connections getting rejected during peak traffic
    - Keep alive should be off
  - Decrease if
    - Keep alive is on
    - Connections getting accepted but never serviced

Tuning timeouts

- connectionTimeout
  - Values from 2000-60000
  - Represents the SO_TIMEOUT value
  - Essentially, max time between TCP packets during a blocking read or write
  - Critical to a stable system
  - Also used for keep alive timeout

Tuning timeouts

- connectionTimeout="3000"
  - Increase if
    - Working with slow clients (dial up)
    - Using a layer 7 load balancer with connection limit/pool and keep alive on
  - Decrease if
    - Need faster timeouts
  - Default value of 20,000 (20secs) is too high for a web server

Content Delivery

- Dynamic content
  - No caching done
  - Tomcat has to deliver it blocking mode
  - Worker thread is not released until all content has been delivered
  - Changing connector wont change behavior

Content Delivery

- Static content
  - Size based cache, default 10mb
  - BIO - Tomcat has to deliver it blocking mode
  - NIO/APR
    - Tomcat can use SEND_FILE
    - Release worker thread, deliver the content using a background thread when the client is ready to receive
    - Increases concurrency

Content Delivery

- Configured in <Context> element
- 40MB cache (default 10MB)
- cache revalidation every 60 seconds (default 5 seconds)
- caching enabled (default true)

```
<Context
   cacheMaxSize=”40960”
   cacheTTL=”60000”
   cachingAllowed=”true”>
</Context>
```

JVM Tuning

- Key parameters for JVM tuning
  - Memory
  - Garbage collection
- They are not independent

JVM Tuning: The ideal

- Short lived objects never reach the Old Generation
- Short lived objects cleaned up by short minor garbage collections
- Long lived objects promoted to Old Generation
- Long lived objects cleaned up by (rare) full garbage collection

JVM Tuning: Memory

- -Xms/-Xmx
  - Used to define size of Java heap
  - Aim to set as low as possible
  - Setting too high can cause wasted memory and long GC cycles
- -XX:NewSize/-XX:NewRatio
  - Set to 25-33% of total Java heap
  - Setting too high or too low leads to inefficient GC

JVM Tuning: GC

- GC pauses the application
  - Regardless of GC algorithm
- Pause can range from milliseconds to seconds
- The pause will impact your response time
  - How much does this matter?

JVM Tuning: GC

- -XX:MaxGCPauseMillis
  - Set GC pause time goal
  - More frequent GC
  - Shorter pauses
  - Goal is for major collections
- -XX:MaxGCMinorPauseMillis
  - Applies to young generation

JVM Tuning: Try it out

- GC Settings – JDK 1.5 and 1.6
  - -XX:+UseConcMarkSweepGC
  - -XX:+CMSIncrementalMode
  - -XX:+CMSIncrementalPacing
  - -XX:CMSIncrementalDutyCycleMin=0
  - -XX:CMSIncrementalDutyCycle=10
  - -XX:+UseParNewGC
  - -XX:+CMSPermGenSweepingEnabled
  - -XX:MaxGCPauseMillis=250
  - -XX:MaxGCMinorPauseMillis=100

JVM Tuning

- Much bigger topic
  - Watch for future webinar
- Same tuning rules apply
  - Doesn't compensate for bad, slow or poorly written applications
- Sun JDK options

http://blogs.sun.com/watt/resource/jvm-options-list.html

SpringSource Products and Services

- SpringSource Support
- SpringSource ERS
  - Commercial distribution of Apache Tomcat, HTTPd
- SpringSource Application Platform
  - SpringSource Enterprise
  - SpringSource dm Server

Apache Tomcat Support

- Enterprise level commercial support
  - Guaranteed SLA's
  - Guaranteed bug fixes
  - Security notifications and patches
- Consulting
  - Troubleshooting
  - Training
  - Security, performance and best practices reviews

Enterprise Support Options

Platinum Plan
Times: 24 x 7 x 365
Method: phone or web

Severity	Response	Workaround	Permanent Correction
Level 1	1 Hour	72 Hours	Next Release
Level 2	4 Hours	5 Business Days	Next Release
Level 3	1 Business Day	Next Release	Next Release

Gold Plan
Times: 6AM – 6PM local time, weekdays, excluding national holidays
Method: phone or web

Severity	Response	Workaround	Permanent Correction
Level 1	4 Hours	72 Hours	Next Release
Level 2	1 Business Day	5 Business Days	Next Release
Level 3	2 Business Days	Next Release	Next Release

Silver Plan
Times: 6AM – 6PM local time, weekdays, excluding national holidays
Method: web

Severity	Response	Workaround	Permanent Correction
Level 1	1 Business Day	None	Next Release
Level 2	2 Business Days	None	Next Release
Level 3	4 Business Days	None	Next Release

SpringOne Americas 2008

Dec 1-4, 2008
Hollywood, FL

- World's Biggest Conference on Spring & Tomcat
- Keynotes by Rod Johnson, Adrian Colyer
- Over 70 sessions in 5 technical tracks
- Technical Sessions, Case Studies, Best Practices
- Extensive sessions on production configuration of Tomcat in large scale enterprise systems
- Register now for special discounts

www.springone.com

Thank You for Attending

Find out more at:

http://tomcat.apache.org

SpringSource Enterprise support:

http://springsource.com/support

insidesales@springsource.com

+1 800-444-1935

Questions?

Filip Hanik

filip.hanik@springsource.com

Mark Thomas

mark.thomas@springsource.com

http://springsource.com