# Tomcat Expert Series

# Performance Tuning

Filip Hanik
SpringSource
2009

# Agenda


- Performance Tuning Process
- Logging improvements
- TCP and HTTP
- Tuning your connectors
- Content delivery and caching
- Tuning the JVM

# The process

- Understand the system architecture
- Stabilise the system
- Set Performance Targets
  - Web applications are easy
  - Only one consideration – request/response time

# The process

- Measure current performance
- Identify the current bottleneck
  - Focus on one item at a time
- Fix the **root** cause
  - Easy to get side tracked

# The Process

- When possible, tune pre-production
  - Hard to profile in production
- Application tuning most important
  - 80% or more of request time is typically spent inside the application
- Tomcat tuning is fairly limited
  - Divided between JVM tuning and Tomcat connectors
  - Requires lower level of understanding

# Apache Tomcat in Production


- Out of the Box Tomcat
  - Tomcat is ready for production
- JVM settings must be applied
  - Default memory settings usually too small for most web applications
- Tuning is limited
  - So we can cover most of it

# Logging

- Tomcat logging is fairly good
  - Years of adjusting log levels pays off
  - Doesn't log what you don't need to see
- A few gotcha's with the default configuration
  - Catch all logger creates duplicate logs
    - Standard out – often piped to catalina.out
    - Log file on the file system
  - Synchronized logging
  - No overflow protection

# Logging

- Tomcat's logger
  - Rotated based on date
  - Implements a per-class-loader logger
    - Simply drop logging.properties into your web application and logging is configured
  - Synchronous logging
  - No file limit

# Logging

- Java Virtual Machine logger
  - Rotated based on size
  - One global configuration for entire JVM
  - Synchronous logging

# Logging

Remove duplicate logging (logging.properties)

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

Adjusted catch all logger

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

# Logging

- Overflow protection
  - 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
```

# 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

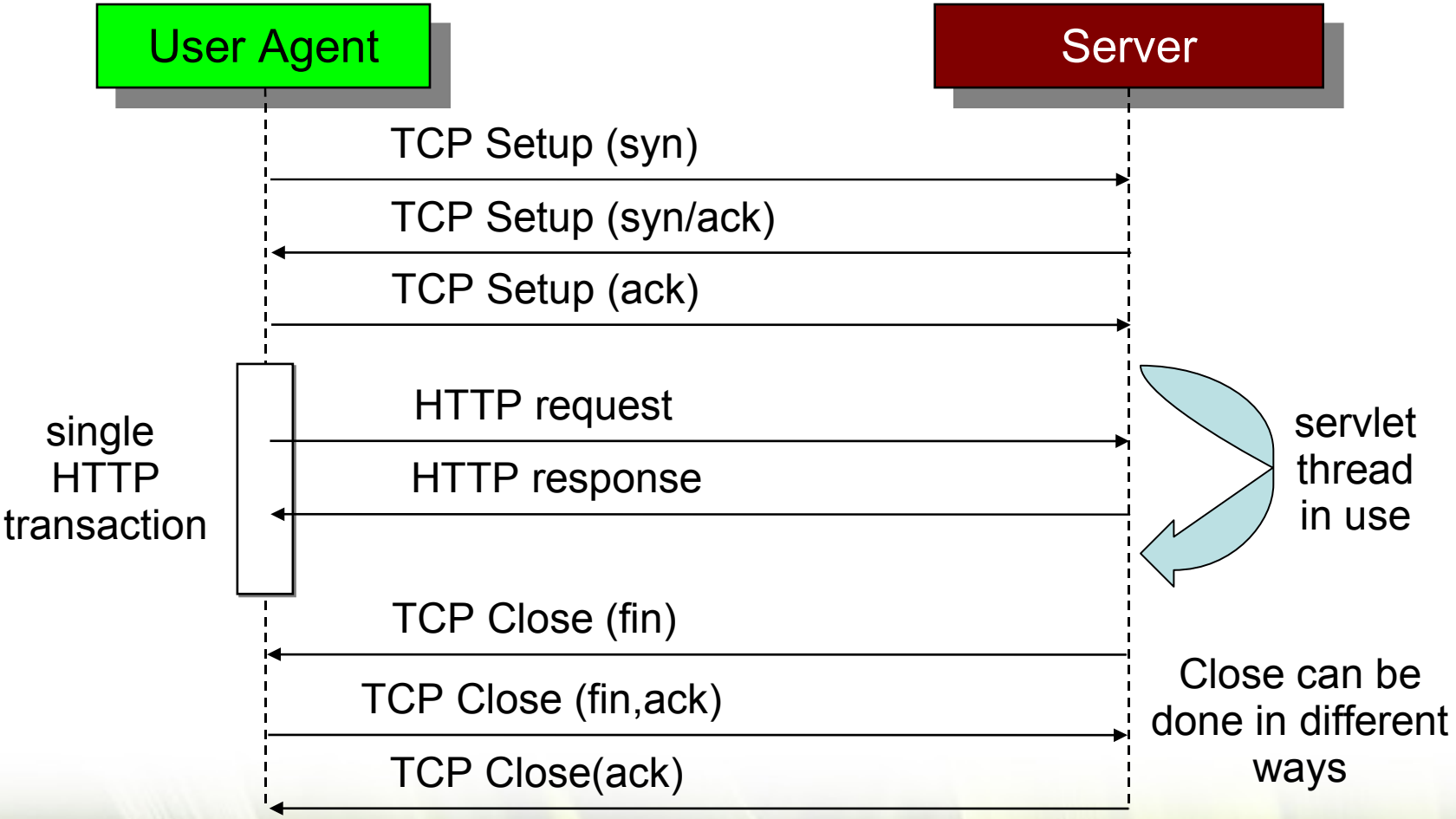
# TCP: Stateful protocol

- 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

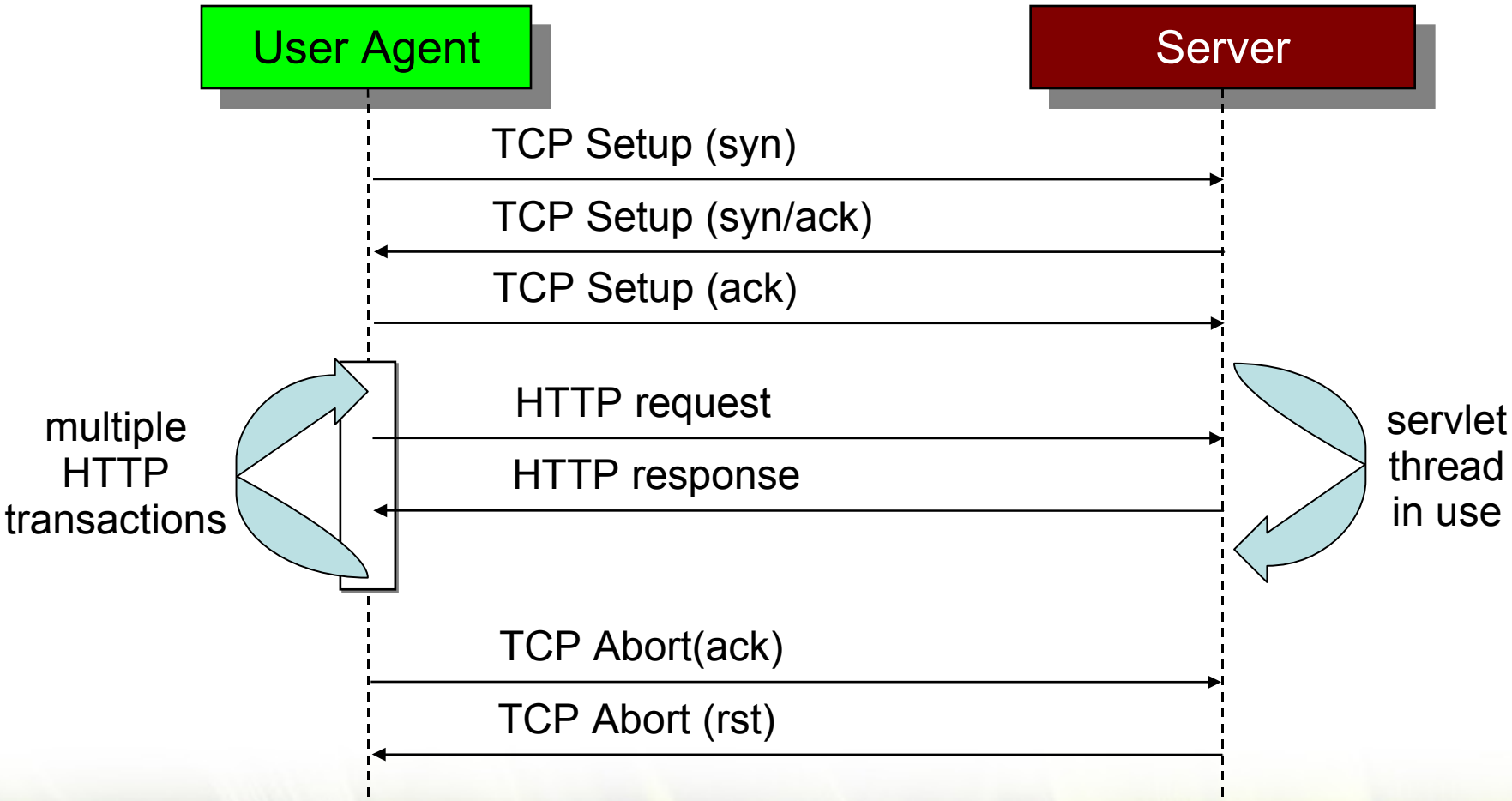
# TCP: Flow control

- 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
    - Average size of request
  - It's these patterns that decide how to tune our system

# HTTP

- Layer 7 in the OSI stack
- Stateless protocol

# 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
  - Socket buffers
- 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
    - 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
  - Keep alive is important
  - Lots of static content
  - Using Comet feature
  - Requires compilation of native library

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

# Which connector?

- Use NIO if:
  - Compiling APR is not an option
  - Keep alive is important
  - Using SSL
  - 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"
```

# Which connector?

spring source

| Comparison Chart | Java BIO | Java NIO | APR |
|---|---|---|---|
| Class | Http11Protocol | Http11NioProtocol | Http11AprProtocol |
| Version | 3.x+ | 6.x+ | 5.5.x+ |
| Polling | NO | YES | YES |
| Polling Size | N/A | Unlimited | Configurable |
| HTTP Req Read | Blocking | Non blocking | Blocking |
| HTTP Body Read | Blocking | Sim Blocking | Blocking |
| HTTP Write | Blocking | Sim Blocking | Blocking |
| SSL | JSSE | JSSE | OpenSSL |
| SSL Handshake | Blocking | Non blocking | Blocking |
| Max Connections | maxThreads | Unlimited | Configurable |

# 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"
```

# 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"

# 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
  - Fast dynamic content can piggy back on send file
    - Simply write to file, set request attribute and hand off to Tomcat's poller threads

# 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

# 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
  - Often these are not tuned, GC log will reveal

# 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:+CMSClassUnloadingEnabled
  - -XX:MaxGCPauseMillis=250
  - -XX:MaxGCMinorPauseMillis=100

# JVM Tuning

- Much bigger topic
- 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

# Questions...


and answers!