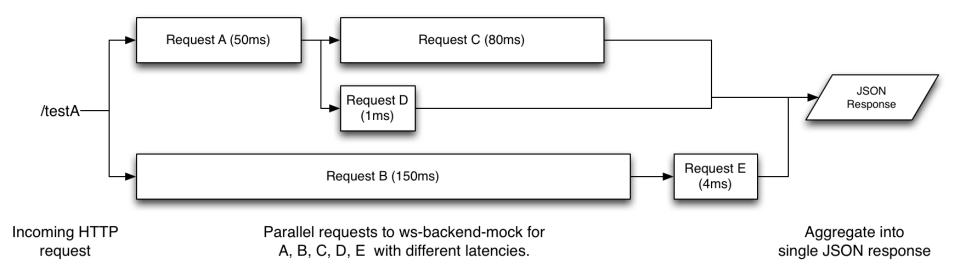
# **RxNetty vs Tomcat** Performance Results

Brendan Gregg; Performance and Reliability Engineering Nitesh Kant, Ben Christensen; Edge Engineering updated: Apr 2015

### **Results based on**

- The "Hello Netflix" benchmark (wsperflab)
- Tomcat
- RxNetty
- physical PC
  - Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz: 4 cores, 1 thread per core
- OpenJDK 8
  - with frame pointer patch
- Plus testing in other environments

### **Hello Netflix**



### **RxNetty vs Tomcat performance**

In a variety of tests, RxNetty has been faster than Tomcat. This study covers:

- 1. What specifically is faster?
- 2. By how much?
- 3. Why?

### 1. What specifically is faster?

### 1. What specifically is faster?

#### • CPU consumption per request

- RxNetty consumes less CPU than Tomcat
- This also means that a given server (with fixed CPU capacity) can deliver a higher maximum rate of requests per second

### • Latency under load

• Under high load, RxNetty has a lower latency distribution than Tomcat

### 2. By how much?

## 2. By how much?

The following 5 graphs show performance vs load (clients)

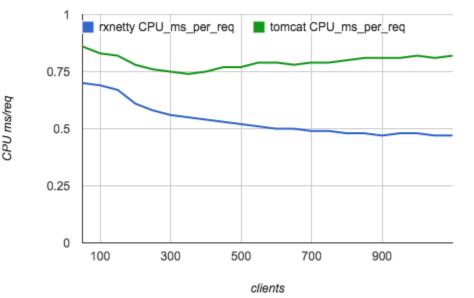
- 1. CPU consumption per request
- 2. CPU resource usage vs load
- 3. Request rate
- 4. Request average latency
- 5. Request maximum latency

Bear in mind these results are for this environment, and this workload

## 2.1. CPU Consumption Per Request

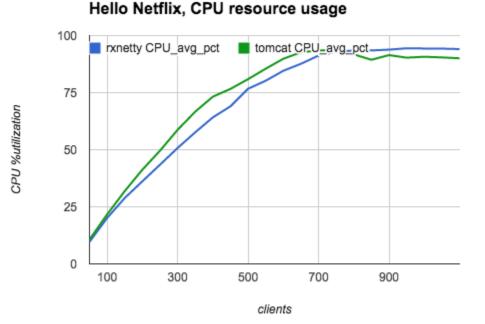
- RxNetty has generally lower CPU consumption per request (over 40% lower)
- RxNetty keeps getting faster under load, whereas Tomcat keeps getting slower





### 2.2. CPU Resource Usage vs Load

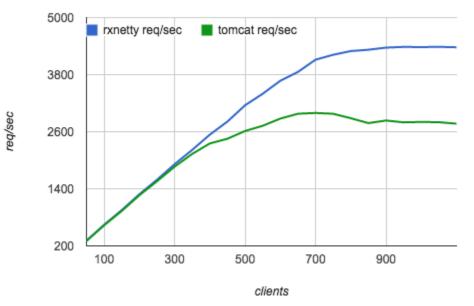
 Load testing drove the server's CPUs to near 100% for both frameworks



### 2.3. Request Rate

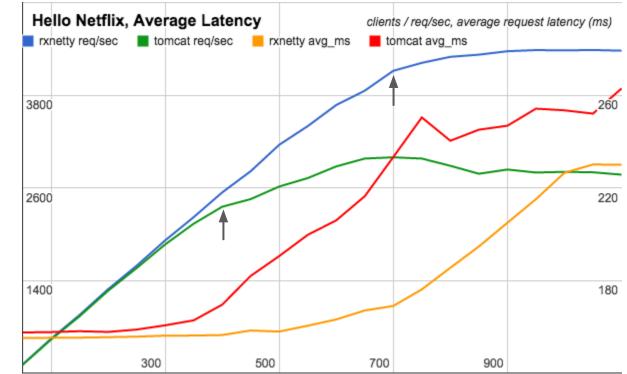
- RxNetty achieved a 46% higher request rate
- This is mostly due to the lower CPU consumption per request

#### Hello Netflix, Request Rate



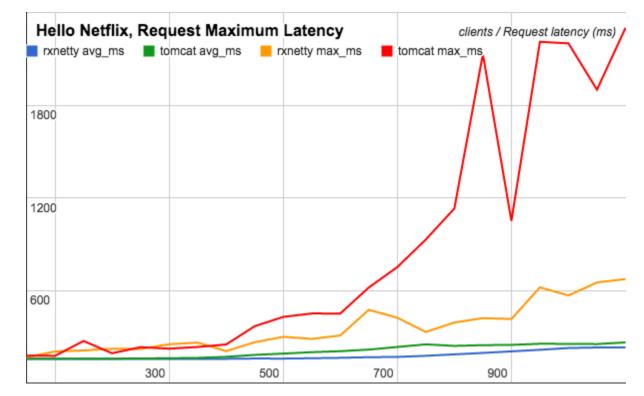
## 2.4. Request Average Latency

- Average latency increases past the req/sec knee point (when CPU begins to be saturated)
- RxNetty's latency breakdown happens with much higher load



### 2.5. Request Maximum Latency

 The degradation in maximum latency for Tomcat is much more severe





# 3. Why?

### 1. CPU consumption per request

- RxNetty is lower due to its framework code and lower object allocation rate, which in turn reduces GC overheads
- RxNetty also trends lower due to its event loop architecture, which reduces thread migrations under load, which improves CPU cache warmth and memory locality, which improves CPU Instructions Per Cycle (IPC), which lowers CPU cycle consumption per request
- 2. Lower latencies under load
  - Tomcat has higher latencies under load due to its thread pool architecture, which involves thread pool locks (and lock contention) and thread migrations to service load

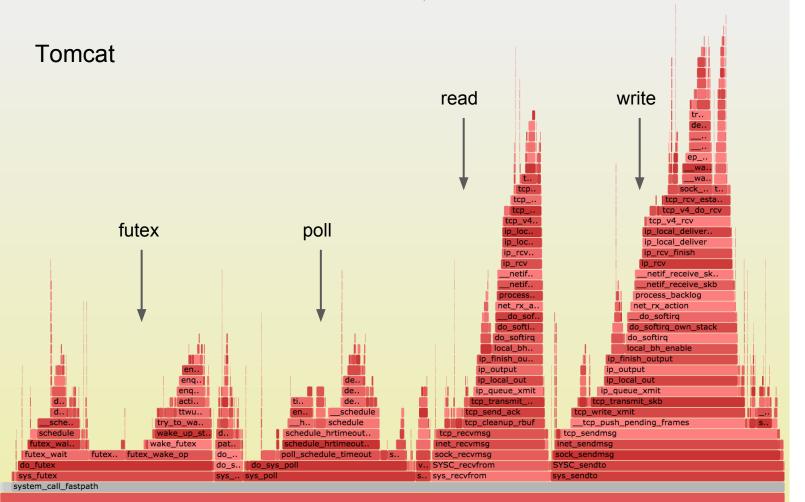
## **3.1. CPU Consumption Per Request**

Studied using:

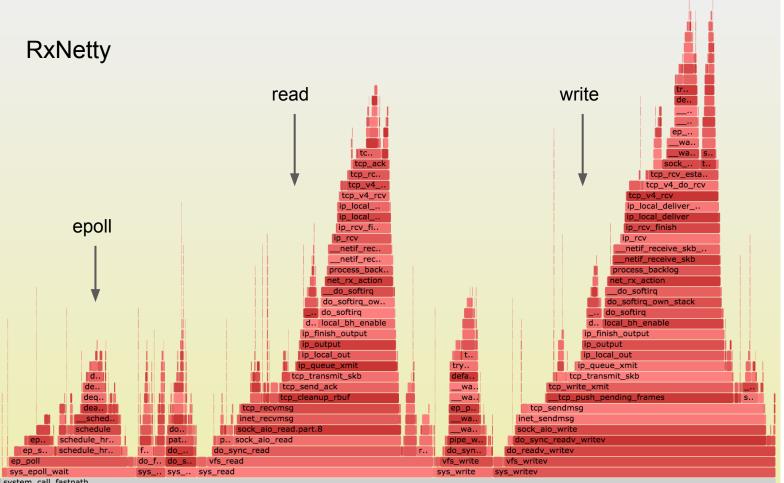
- 1. Kernel CPU flame graphs
- 2. User CPU flame graphs
- 3. Migration rates
- 4. Last Level Cache (LLC) Loads & IPC
- 5. IPC & CPU per request

### 3.1.1. Kernel CPU Flame Graphs

Kernel CPU Flame Graph: Tomcat



Kernel CPU Flame Graph: rxNetty



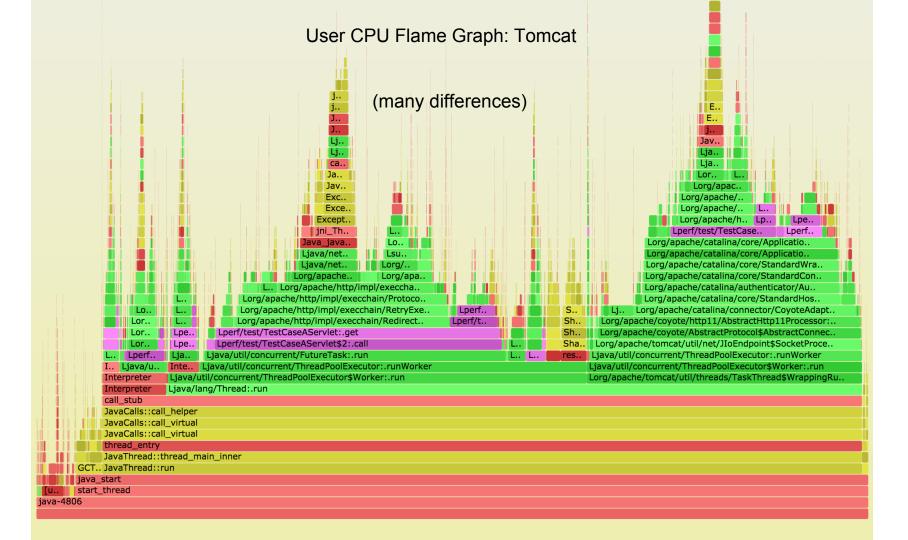
system\_call\_fastpath

### **3.1.1. Kernel CPU Time Differences**

CPU system time delta per request: 0.07 ms

- Tomcat futex(), for thread pool management (0.05 ms)
- Tomcat poll() vs RxNetty epoll() (0.02 ms extra)

### **3.1.2. User CPU Flame Graphs**



User CPU Flame Graph: RxNetty	L L L L L L L L L L	L L L L L L L L
Lio/netty/channel/nio/NioEventLoop:.processSelectedKey Lio/netty/channel/nio/NioEventLoop:.run		Lio/n Lio/netty/util/concurrent/Sin
Interpreter		
Interpreter		
Interpreter call stub		
JavaCalls::call_helper		
JavaCalls::call_virtual		
JavaCalls::call_virtual		
C. thread_entry		
G. JavaThread::thread_main_inner		
java start		
start_thread		n an
java-17644		

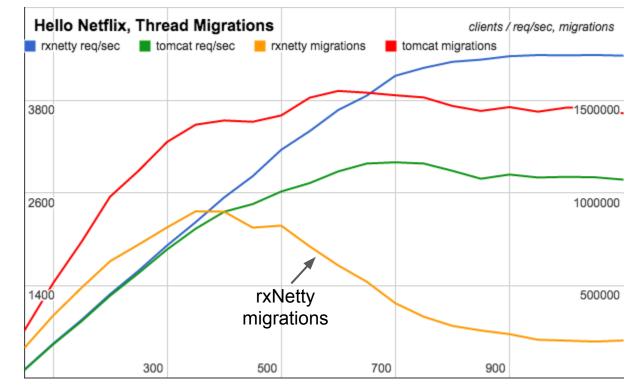
### **3.1.2. User CPU Time Differences**

CPU user time delta per request: 0.14 ms Differences include:

- Extra GC time in Tomcat
- Framework code differences
- Socket read library
- Tomcat thread pool calls

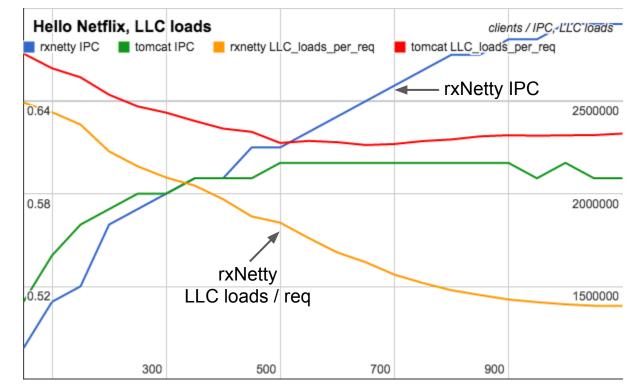
## **3.1.3. Thread Migrations**

- As load increases, RxNetty begins to experience *lower* thread migrations
- There is enough queued work for event loop threads to keep servicing requests without switching



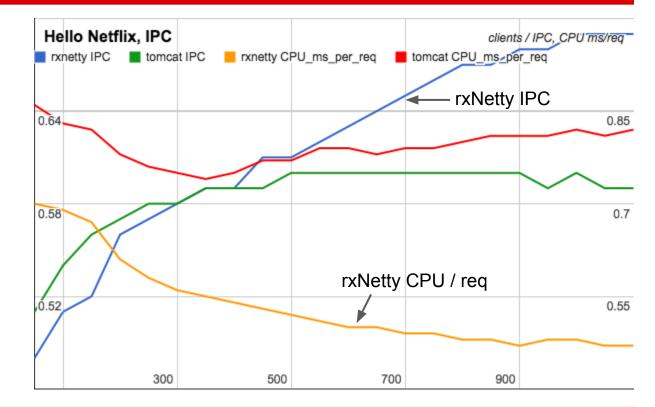
### 3.1.4. LLC Loads & IPC

 ... The reduction in thread migrations keeps threads on-CPU, which keeps caches warm, reducing LLC loads, and improving IPC



### 3.1.5. IPC & CPU Per Request

 … A higher IPC leads to lower CPU usage per request



### **3.2. Lower Latencies Under Load**

Studied using:

- 1. Migration rates (previous graph)
- 2. Context-switch flame graphs
- 3. Chain graphs

### **3.2.2. Context Switch Flame Graphs**

- These identify the cause of context switches, and blocking events.
  - They do not quantify the magnitude of off-CPU time; these are for identification of targets for further study
- Tomcat has additional futex context switches from thread pool management

							sched sched sched		
		 S S S		ThreadPoo Executor lo		schedule	schedule schedule schedule_hrtimeout schedule_hrtimeout		sche do_s sche Sys_p poll syst do_s [unk syst Java syst Ljava [unk Lorg/ sched
	s s po. d	d s sy		↓ ↓		schedule schedule_hrtimeout schedule_hrtimeout poll_schedule_timeout do_sys_poll	poll_schedule_time do_sys_poll sys_poll system_call_fastpath [unknown]		Ljava. Lsun/ fute   Ljava. Lorg/ fute   Lorg/ Lorg/ do_fu   Lorg/apache/h sys_f   Lorg/apache/h syst
	s sch sy. fut [	Lj		schedule schedule futex_wait_queue_me		sys_poll system_call_fastpath [unknown]	Java_java_net_Sock Ljava/net/SocketInp Lorg/apache/http/im	sc	Lorg/apache/h pthr Lorg/apache/h Unsa Lperf/test/Tes Lsun
	fut Ja. do <mark>Lj.</mark>	. Lo . L		futex_wait do_futex	futex_wa futex_wait	Java_java_net_Socke Ljava/net/SocketInpu	Lorg/apache/http/im Lsun/reflect/Genera	fute f fute f	Lorg/apache/catalina/c
	sys 📕 Lo.	. Lo Lo rg/a	L L	sys_futex system_call_fastpath pthread_cond_wait@@GLIB	sys_futex	Ljava/net/SocketInpu Lorg/apache/http/imp Lorg/apache/http/impl/e	Lorg/apache/http/pr Lorg/apache/http/pr xecchain/MainClientE	do_f sys syst	Lorg/apache/catalina/c Lorg/apache/catalina/c Lorg/apache/catalina/a
	Lsu 🗾 Lor	rg/a rg/a rg/a	Lp.,	Unsafe_Park Lsun/misc/Unsafe:.park Ljava/util/concurrent/locks/	Lorg/apache/http/impl/e	rg/apache/http/impl/execchain/ProtocolExec rg/apache/http/impl/execchain/RetryExec: rg/apache/http/impl/execchain/RedirectExec		Lorg/apache/catalina/c Lorg/apache/catalina/c Lorg/apache/coyote/htt	
	Lja Lperf/te Lja Ljava/util/concurrent/Linked Lperf/tes Lperf/test/TestCaseAServlet:.get Ljava/util/concurrent/ThreadPoolExecutor:.runWorker					Ljav Loi	Lorg/apache/coyote/Abs rg/apache/tomcat/util/ne		
	Interpreter Ljava/lang/Thread:.run call_stub							concurrent/ThreadPoolEx	
JavaCalls::call_helper JavaCalls::call_virtual JavaCalls::call_virtual JavaCalls::call_virtual									
S sch	thread_entr JavaThread: JavaThread: java_start	::thread	_main_	inner					
java-4806	start_thread								

#### Context Switch Flame Graph: RxNetty

	(epoll)	
	▼	S sc schsched
	hedule	sc schsched fut fut f schedule
sched		fut fut f futex_w s
	dule_hrtimeout_range_clock	do do futex_w f
	dule_hrtimeout_range	sy sys s do_futex f
ep_po		sy sys s sys_futex d
	epoll_wait	l. pth system s
	em_call_fastpath	ObjectMo pthread s
	nown]	ObjectMo Unsafe p
	/nio/ch/EPollArrayWrapper:.epollWait	SharedRu L Lsun/mi U
	etty/channel/nio/NioEventLoop:.select	_complet L Ljava/uti L
	etty/channel/nio/NioEventLoop:.run	L Ljava/util/c
Interp		Ljava/util/co
Interp		
Interp		
call_st		
	Calls::call_helper	
	Calls::call virtual	
	Calls::call_virtual	
	d_entry	
	Tread::thread_main_inner	
sche sch sc sc JavaT	hread::run	
sysr sysr sy sy java_s	start	
	thread	
java-13868		
all		

### 3.2.3. Chain Graphs

- These quantify the magnitude of off-CPU (blocking) time, and show the chain of wakeup stacks that the blocked thread was waiting on
  - x-axis: blocked time
  - y-axis: blocked stack, then wakeup stacks

J Java	JavaCalls::call_virtual	do_futex	do_futex	do_f sys_futex	Ljava/util/concurrent/T		
	JavaCalls::call_virtual	sys_futex	sys_futex	sys system	Ljava/lang/Thread:.run		
Chain Cranhy Tomast	thread_entry	system_call_fast	system_call	syst pthread	call_stub		
Chain Graph: Tomcat	JavaThread::thread_main_inner	pthread_cond_sig	pthread_cond	pthr Unsafe	JavaCalls::call_helper		
-	JavaThread::run	Unsafe_Unpark	Unsafe_Unpark	Unsa Lsun/mi	JavaCalls::call_virtual		
server: java-11516	java_start	Lsun/misc/Unsafe	Lsun/misc/Un	Lsun Ljava/u	JavaCalls::call_virtual		
hadkand inva 10000	. start_thread	Ljava/util/concu	Ljava/util/c	Ljav Ljava/u	thread_entry		
backend: java-18008	. java-18008	Ljava/util/concu	Ljava/util/co	Ljava Lorg/ap L	JavaThread::thread_main_inner		
-	woke-up-by	Ljava/util/concur.	Ljava/util/con	Ljava/util/con L	JavaThread::run		
WOKE	eschedule	Ljava/lang/Thread:	Ljava/lang/Thre	Ljava/lang/Thr L	java_start		
S <b>S.</b> . <b>S</b>	schedule	call_stub	call_stub	call_stub c	start_thread		
s sche		JavaCalls::call_helper	JavaCalls::call	JavaCalls::cal J	java-11516		
s fute.	. schedule_hrtimeout_range	JavaCalls::call_virt	JavaCalls::call	JavaCalls::cal J	woke-up-by		
p fute.	. poll_schedule_timeout	JavaCalls::call_virt	JavaCalls::call	JavaCalls::cal J	schedule		
d do_f.	do_sys_poll	thread_entry	thread_entry	thread_entry t	schedule		
S Sys_	sys_poll	JavaThread::thread_m	JavaThread::thre	JavaThread::th J	futex_wait_queue_me		
s syst.	. system_call_fastpath	JavaThread::run	JavaThread::run	JavaThread::run J	futex_wait		
[ <b>22</b> pthr.	. [unknown]	java_start	java_start	java_start j l	do_futex		
J Unsa	Java_java_net_SocketInputSt eam_s	start_thread	start_thread	start_thread s l	sys_futex		
L English and Lsun	Ljava/net/SocketInputStream .sock	java-11516	java-11516	java-11516 🗼 j 🛽	system_call_fastpath		
Ljav.	. Lorg/apache/http/impl/io/SessionI	woke-up-by	woke-up-by	woke-up-by w I	pthread_cond_wait@@GLIBC_2		
Lorg.	Lorg/apache/http/impl/io/Abstract	schedule	schedule	schedule	Unsafe_Park		
L Lorg.	Lsun/reflect/GeneratedMethocAcces	schedule	schedule	schedule s l	p Lsun/misc/Unsafe:.park		
L Lorg/	/ Lorg/apache/http/protocol/HttpReq	futex_wait_queue_me	futex_wait_queue	futex_wait_qui f	Lorg/apache/catalina/core/Applic		
L Lorg/	/ Lorg/apache/http/protocol/HttpReq	futex_wait	futex_wait	futex_wait f L	Lorg/apache/catalina/core/Applic		
Lo Lo. L. L. Lorg/a	pache/http/impl/execchain/MainClent	do_futex	do_futex	do_futex d L	Lorg/apache/catalina/core/Standa		
Lo Lo. L. L. L. Lorg/a	pache/http/impl/execchain/Protoc IEx	sys_futex	sys_futex	sys_futex s l	.org/apache/catalina/core/Standa		
Lo Lo. L. L. L. Lorg/a	pache/http/impl/execchain/RetryEkec:	system_call_fastpath	system_call_fast.	system_call_fa . s I	Lorg/apache/catalina/authenticat		
Lo Lo. L. L. L. Lorg/a	pache/http/impl/execchain/Redire tEx	pthread_cond_wait@@G.	pthread_cond_wai	pthread_cond_t p 1	.org/apache/catalina/core/Standa		
Lo., Lo. L. L. L. Lorg/a	pache/http/impl/client/CloseableH tp	Unsafe_Park	Unsafe_Park	Unsafe_Park U L	Lorg/apache/catalina/connector/C		
Lp., L., L., Lperf/t	est/TestCaseAServlet\$2:.call	Lsun/misc/Unsafe:.par	Lsun/misc/Unsate	Lsun/misc/Unsa L L	org/apache/coyote/http11/Abstrac		
Ljava Lja Lja Ljava/u	util/concurrent/FutureTask:.run		Ljava/util/concu	Ljava/util/con. L. L	org/apache/coyote/AbstractProtoc		
Ljava/ Ljav Inte Ljava/u	util/concurrent/ThreadPoolExecutor:.runWorke	er		Ljava/util/con. Lorg/	/apache/tomcat/util/net/JIoEndpo		
Interp Inte Ljava/util/conc	current/ThreadPoolExecutor\$Worker:.run			Ljava/util/concurrent/Th	readPoolExecutor:.runWorker		
Interp Ljava/lang/Thread:.r							
call_stub	Normal blocking path:		mcat blocke	ed on 📃 🗖			
JavaCalls::call_helper							
JavaCalls::call_virtual	JavaCalls::call_virtual server thread waits on		itself: thread pool locks				
JavaCalls::call_virtual		100					
thread_entry	backend network I/O						
JavaThread_main_inne							
JavaThread::run			4				
java_start							
start_thread							
java-11516	java-11516						

### Reasoning

- On a system with more CPUs (than 4), Tomcat will perform even worse, due to the earlier effects.
- For applications which consume more CPU, the benefits of an architecture change diminish.



