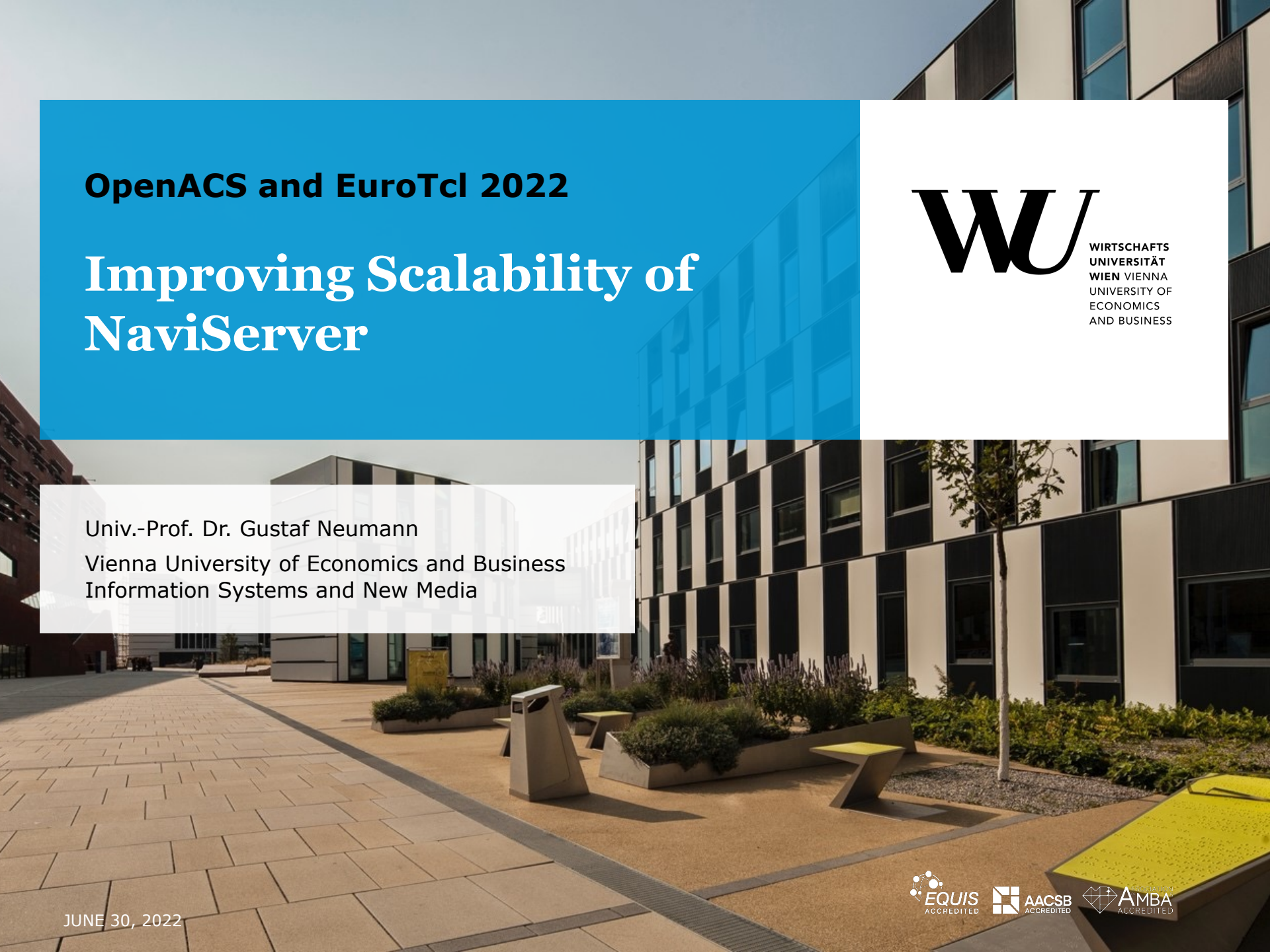


## OpenACS and EuroTcl 2022

# Improving Scalability of NaviServer

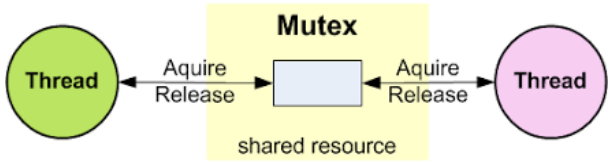

Univ.-Prof. Dr. Gustaf Neumann

Vienna University of Economics and Business
Information Systems and New Media


JUNE 30, 2022

# Overview

- NaviServer brief history
- Scalability
  - Improve Locking
  - Improve behaviour on overload situations
  - Help to identify bottlenecks
  - Bandwidth management




- More strict input handling
  - Induced by vulnerability scanning

acunetix
Web Vulnerability Scanner

- Misc
  - Expose internals to Tcl
  - Unit support for API and configuration (e.g. 1s, 100µs, … 1KB, 1.5MB, …)
  - Crypto support (OCSP Stapling, SNI, SCRYPT, SCRAM, HMAC, 20+ digest algorithms, …)
  - Experiments with HTTP/2 support

OpenSSL
Cryptography and SSL/TLS Toolkit

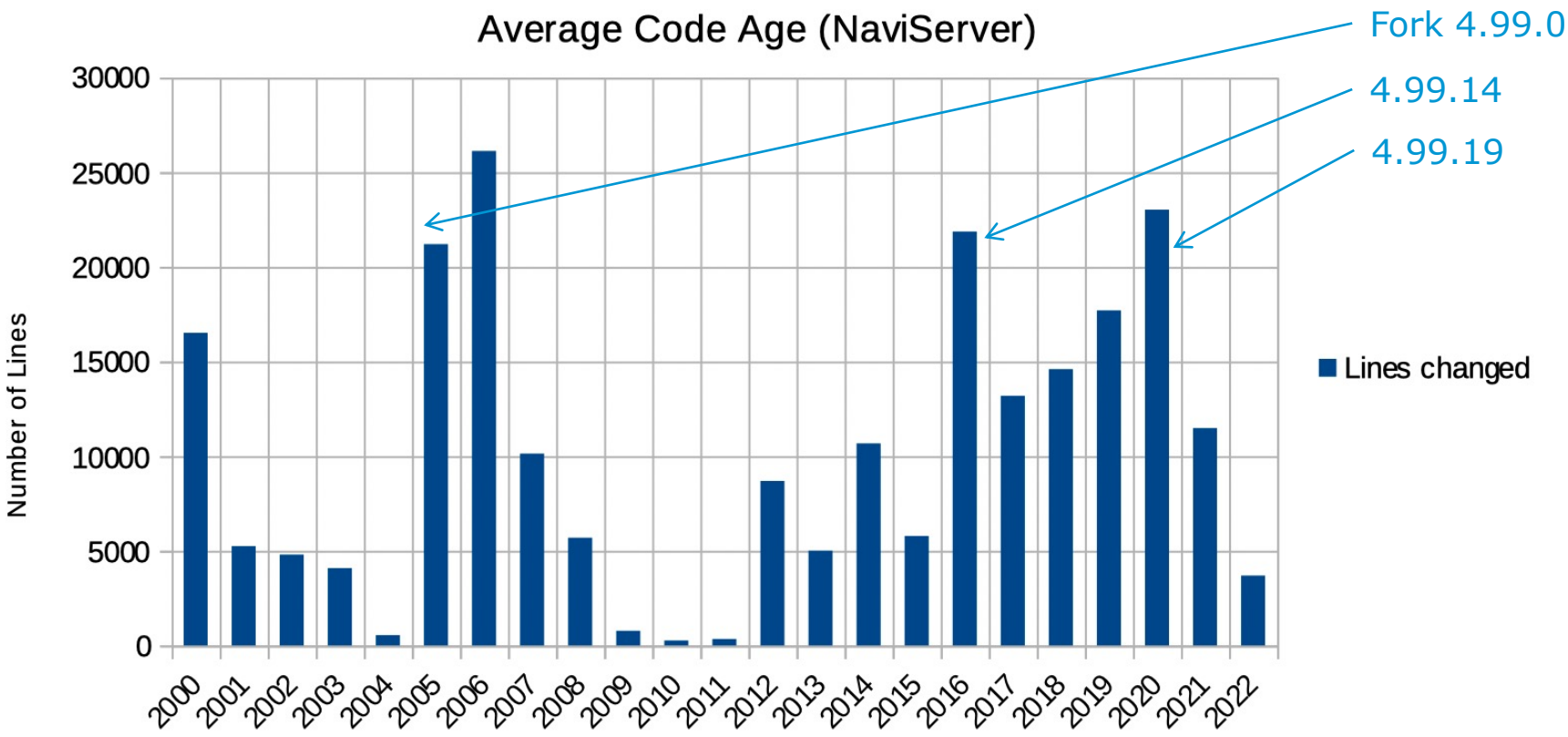
# History of NaviServer

- NaviServer brief history
  - 1991 Closed-source product, developed by company “NaviSoft”, used by AOL as AOLserver
  - 1999 AOLserver open source
  - 2005 fork of AOLserver 4.10 -> NaviServer 4.99.0 (original name)
  - 2016: integration of OpenSSL support directly in NaviServer, IPv6



- NaviServer: multi-protocol server
  - HTTP, UDP, SMTP, LDAP, DNS, COAP, IMAP, …
- Code Overview
  - 494 unique files
  - 140K lines of code (+58K lines comment + empty lines)
  - C: 73%, Tcl: 15%
  - >1900 tests in regression test

# Code Age analysis for NaviServer
## (based on git-blame statistics)



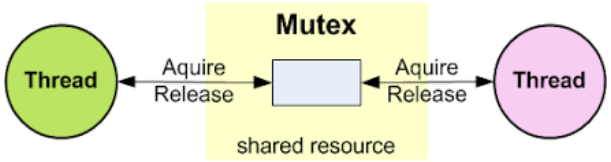

History: 22 years in repository
Average age per line: 10 years
Oldest lines: Copyright lines from AOL

Tcl ~15y

# Bottlenecks based on mutex locks: Example from OpenACS 5.9.1 (2019)

WU WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Symptom: cores are under-utilized




**Total locks: 1.83G, total requests 2.62M, locks per req 700, avg lock time 5.76μ, lock time per req 4.03m, max req per sec 248.12**
**(except: interp, jobThreadPool, ns:sched, tcljob:jobs, tcljob:ns_eval_q:production)**

| Name | ID | Locks | Busy | Contention | Total Lock | Avg Lock | Total Wait | Max Wait | Locks/Req | Pot.Locks/sec | Pot.Reqs/sec |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ns:cache:util_memoize | 1209 | 110.71M | 237.51K | 0.2145 | 5.14Ks | 46.42μs | 2.91Ks | 274.88ms | 42.27 | 21.54K | 509.63 |
| ns:cache:xo_site_nodes | 1204 | 127.85M | 266.04K | 0.2081 | 647.28s | 5.06μs | 147.62s | 312.55ms | 48.81 | 197.52K | 4.05K |
| ns:cache:db_cache_pool | 1210 | 5.33M | 6.23K | 0.1170 | 519.08s | 97.42μs | 84.37s | 99.88ms | 2.03 | 10.26K | 5.05K |
| ns:cache:xotcl_object_type_cache | 1215 | 27.72M | 28.04K | 0.1011 | 507.2s | 18.3μs | 226.16s | 150.96ms | 10.58 | 54.66K | 5.16K |
| nsv:138:production | 78 | 199.99M | 238.76K | 0.1194 | 352.42s | 1.76μs | 22.08s | 1.05s | 76.36 | 567.48K | 7.43K |
| tcljob:xocal_queue | 1249 | 29.62M | 0 | 0.0000 | 316.13s | 10.67μs | 0s | 0s | 11.31 | 93.69K | 8.29K |
| nsv:169:production | 47 | 237.75M | 177.15K | 0.0745 | 274.19s | 1.15μs | 3.2s | 67.14ms | 90.77 | 867.08K | 9.55K |

Statistics from nsstats.tcl

- Avg Locks/request: 700 (OpenACS 5.10: openacs.org: 144)
- Max reqs/sec: 248 (OpenACS 5.10: openacs.org: 4.8K)
- For single cache:
  - Max Wait: 275ms, Locks/req: 42, max req/s: 509

# Cache statistics
## from OpenACS 5.9.1 (2019)

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

- Hot caches: up to 47 cache hits per request

| Cache | Max | Current | Utilization | Entries | Avg Size | Flushes | Hits | Hits/Req | Reuse | Misses | Hit Rate | Expired | Pruned | Commit | Rollback | Saved/Hit | Saved/Req |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| xo_site_nodes | 2000000 | 1999970 | 100.00% | 20242 | 98 | 592857 | 6657880 | 47.1223 | 3289 | 1124789 | 98.00% | 0 | 489245 | 0 | 0 | 181.62µs | 8.56ms |
| util_memoize | 3000000 | 2721870 | 90.73% | 59290 | 45 | 277667 | 51432766 | 36.4024 | 867 | 5263317 | 90.00% | 4201504 | 287099 | 0 | 0 | 308.44µs | 11.23ms |
| xotcl_object_type_cache | 6000000 | 4558006 | 75.97% | 338123 | 13 | 12665 | 11909302 | 8.4290 | 35 | 1578033 | 88.00% | 0 | 0 | 0 | 0 | 977.21µs | 8.24ms |
| xotcl_object_cache | 500000000 | 394775157 | 78.96% | 197606 | 1997 | 132730 | 8736833 | 6.1836 | 44 | 505598 | 94.00% | 0 | 0 | 0 | 0 | 12.37ms | 76.49ms |
| xocal_sess | 200000000 | 24789342 | 12.39% | 64692 | 383 | 0 | 4322867 | 3.0596 | 67 | 1928218 | 69.00% | 0 | 0 | 0 | 0 | 1.45µs | 4.43µs |
| db_cache_pool | 40000000 | 687172 | 1.72% | 72086 | 9 | 8150 | 2696250 | 1.9083 | 37 | 93518 | 96.00% | 0 | 0 | 0 | 0 | 962.82µs | 1.84ms |

- Most important caches:
  Avg savings per request per cache up to 76ms

| Cache | Max | Current | Utilization | Entries | Avg Size | Flushes | Hits | Hits/Req | Reuse | Misses | Hit Rate | Expired | Pruned | Commit | Rollback | Saved/Hit | Saved/Req |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| xotcl_object_cache | 500000000 | 394410806 | 78.88% | 197408 | 1997 | 132589 | 8724970 | 6.1878 | 44 | 505091 | 94.00% | 0 | 0 | 0 | 0 | 12.37ms | 76.51ms |
| util_memoize | 3000000 | 2633448 | 87.78% | 60746 | 43 | 276082 | 51367972 | 36.4306 | 846 | 5249226 | 90.00% | 4188745 | 287099 | 0 | 0 | 307.4µs | 11.2ms |
| xo_site_nodes | 2000000 | 1999867 | 99.99% | 18645 | 107 | 592708 | 66450023 | 47.1270 | 3564 | 1121385 | 98.00% | 0 | 487697 | 0 | 0 | 181.03µs | 8.53ms |
| xotcl_object_type_cache | 6000000 | 4552430 | 75.87% | 337714 | 13 | 12656 | 11891251 | 8.4334 | 35 | 1575414 | 88.00% | 0 | 0 | 0 | 0 | 976.93µs | 8.24ms |
| gb_schema | 2000000 | 1134670 | 56.73% | 10816 | 104 | 399 | 2383674 | 1.6905 | 220 | 11801 | 99.00% | 0 | 0 | 0 | 0 | 1.99ms | 3.37ms |
| db_cache_pool | 40000000 | 686642 | 1.72% | 71951 | 9 | 8130 | 2691642 | 1.9089 | 37 | 93352 | 96.00% | 0 | 0 | 0 | 0 | 962.67µs | 1.84ms |
| lms_favorite | 2000000 | 102124 | 5.11% | 12774 | 7 | 42597 | 360125 | 0.2554 | 28 | 55396 | 86.00% | 0 | 0 | 0 | 0 | 2.87ms | 732.13µs |

EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION AMBA ACCREDITED

# Hottest Cache entries from OpenACS 5.9.1

WU WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

- Hot cache entries in util_memorize cache:

**50 most frequently used entries from cache 'util_memoize'**

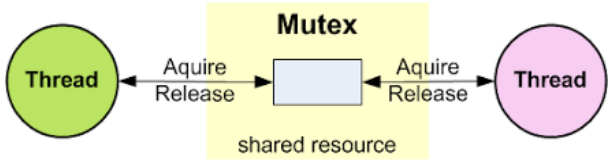
| Key | Size | Hits | Expire |
|---|---|---|---|
| lang::system::get_locales_not_cached | 17 | 6039005 | -1 |
| apm_package_installed_p_not_cached ref-timezones | 1 | 1291383 | -1 |
| lc_time_fmt_compile {%a, %d. %B %Y %H:%M} de_DE | 241 | 1009904 | -1 |
| ad_acs_version_no_cache | 5 | 841339 | -1 |
| acs_lookup_magic_object_no_cache security_context_root | 2 | 746132 | -1 |
| plpgsql_utility::get_function_args content_item__get_content_type | 12 | 466429 | -1 |
| package_plsql_args -object_name get_content_type content_item | 7 | 466429 | -1 |
| package_function_p -object_name get_content_type content_item | 1 | 466429 | -1 |

## High reuse is good and bad:

- Best savings
- Potential candidate for more scalable caching forms

# Improve scalability of caching (1/2)






- Reduce locking time
  - Bad on large caches (e.g. 300K+ entries):
    Operations iterating over every item (e.g. wild-card operations)

```
ns_cache names …
util_memoize_flush_pattern …
```

  - Reducing size of cache reduces locking time, when such operations are used
- Reduce number of locking operations
  - Use per-thread or per-request caches (lock-free)
  - Use more fine-granular mutexes (split caches, cache partitioning)

# Improve scalability of caching (2/2)

WU WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

- Split caches and cache partitioning
  - Advantages:
    - Different caches use different locks, a lock on a specialized cache does not block operations on the `util_memoize` cache
    - Less irrelevant data is processed, when wild-card operations are applied to caches
  - Use different caches for different purposes
    e.g. separate `permission_cache` in OpenACS 5.10*
  - Cache partitioning:
    use different caches based on key values (OpenACS 5.10 supports different partitioning strategies as config options)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| xotcl_object_type_cache-1 | 30000 | 18125 | 60.42% | 1738 | 10 | 0 | 18248 |
| xotcl_object_type_cache-0 | 30000 | 11529 | 38.43% | 980 | 11 | 0 | 3849 |

# Cache Transactions (1/2)

- Motivation:

```
db_transaction {
   # Create service contracts
   auth::authentication::create_contract
   auth::password::create_contract
   auth::registration::create_contract
   auth::get_doc::create_contract
   # . . .
)
```


- Potential problems:
  - What happens with cached values created from API-calls, when transaction fails (e.g. in third API call)
  - Consequence: cache poisoning
  - Breaking isolation
  - Misbehavior is hard to debug

# Cache Transactions (2/2)

WU WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

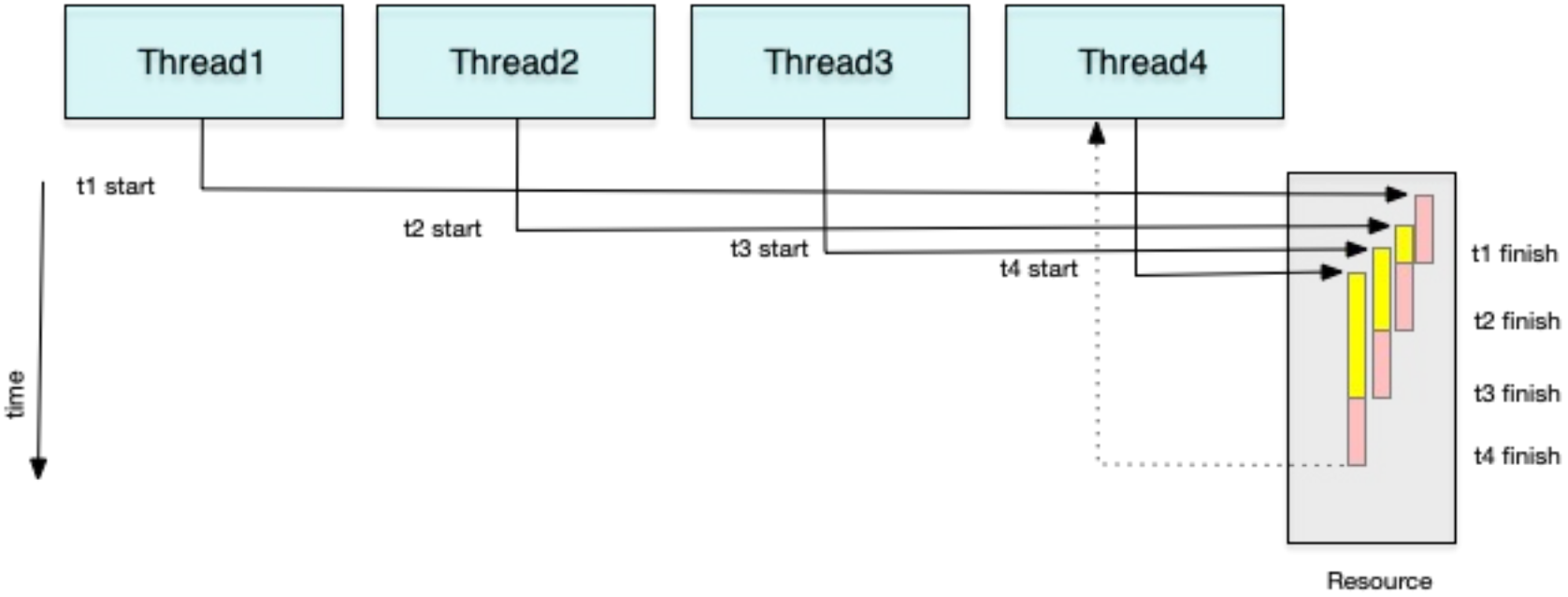
- NaviServer API support (NaviServer 4.99.16)

```
ns_cache_transaction_begin
ns_cache_transaction_commit
ns_cache_transaction_rollback
```

- Integrated with OpenACS 5.10*:
  - Automatically performed in `db_transaction` and xo* counterparts
  - Rollback statistics in nsstats:

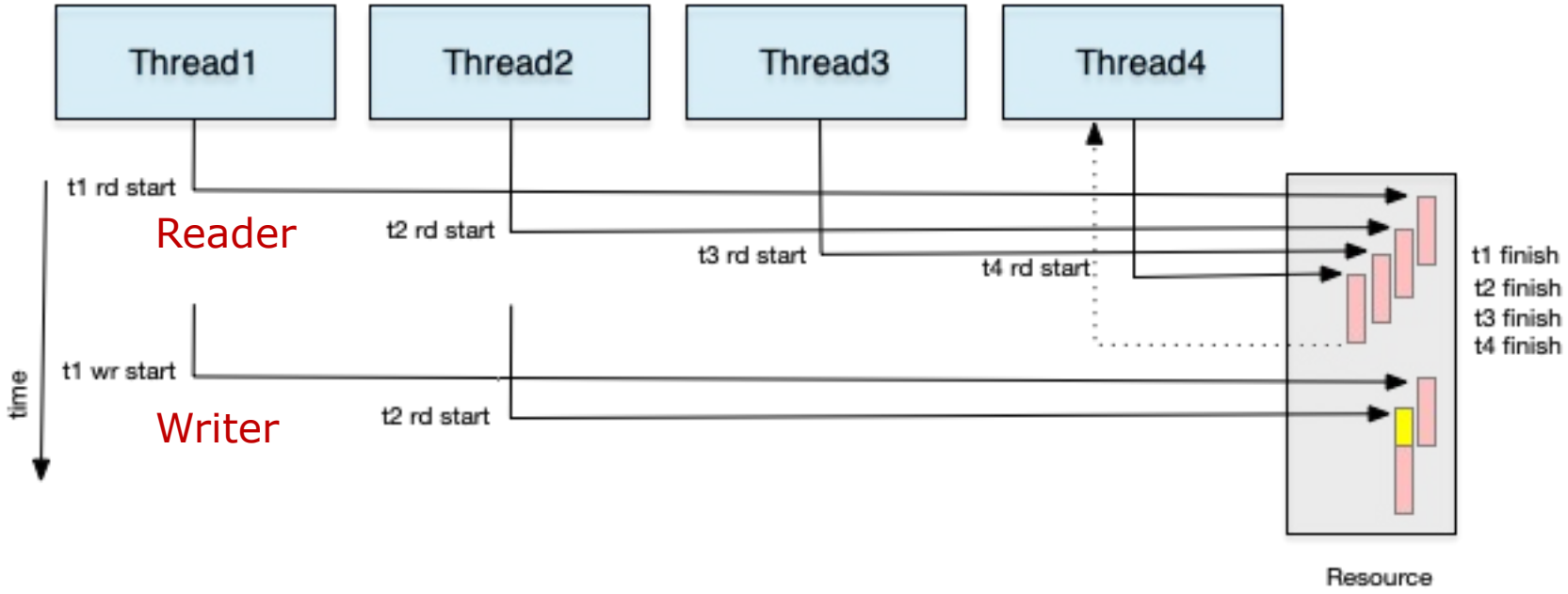
| Cache | Max | Current | Utilization | Entries | Avg Size | Flushes | Hits | Hits/Req | Reuse | Misses | Hit Rate | Expired | Pruned | Commit | Rollback |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tlf_lrn_pretty_link_cache-4 | 20000 | 17689 | 88.44% | 271 | 65 | 2811 | 2119853 | 0.7216 | 7822 | 666165 | 76.09% | 0 | 170475 | 161786 | 0 |
| xotcl_object_cache-1 | 250000000 | 249982370 | 99.99% | 40241 | 6212 | 199631 | 20666286 | 7.0351 | 514 | 1731223 | 92.27% | 0 | 1061653 | 114045 | 2 |
| xotcl_object_cache-0 | 250000000 | 249964500 | 99.99% | 38296 | 6527 | 200654 | 25472052 | 8.6710 | 665 | 1735624 | 93.62% | 0 | 1125791 | 113257 | 0 |
| xotcl_object_type_cache-0 | 7500000 | 7499985 | 100.00% | 525119 | 14 | 389 | 13037609 | 4.4382 | 25 | 2660310 | 83.05% | 0 | 404031 | 4510 | 1 |
| xotcl_object_type_cache-1 | 7500000 | 7499986 | 100.00% | 511154 | 14 | 231 | 13054981 | 4.4441 | 26 | 2782677 | 82.43% | 0 | 325878 | 3605 | 1 |

# Multiple Threads Accessing a Mutex-protected Resource



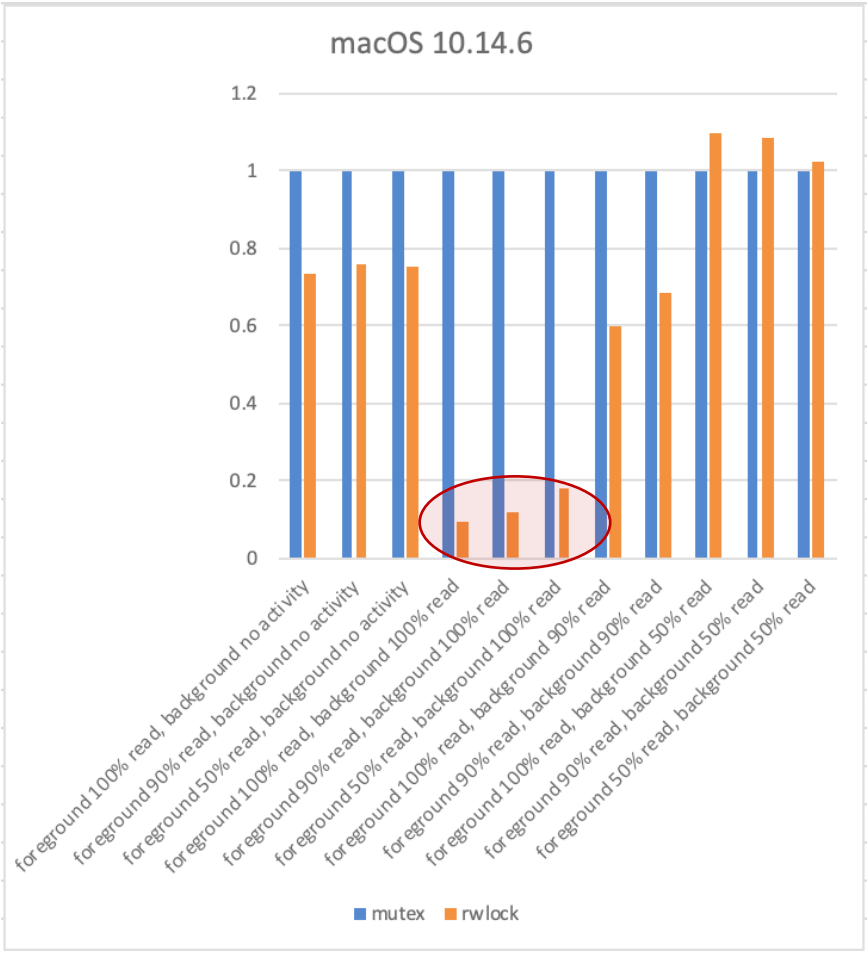

- When mutex (e.g. t2) tries to get a lock on an already locked resource (e.g. locked by t1), the mutex has to wait until this lock is finished.
- When multiple threads try locks: high contention, increasing wait times
- Waiting time can pile up

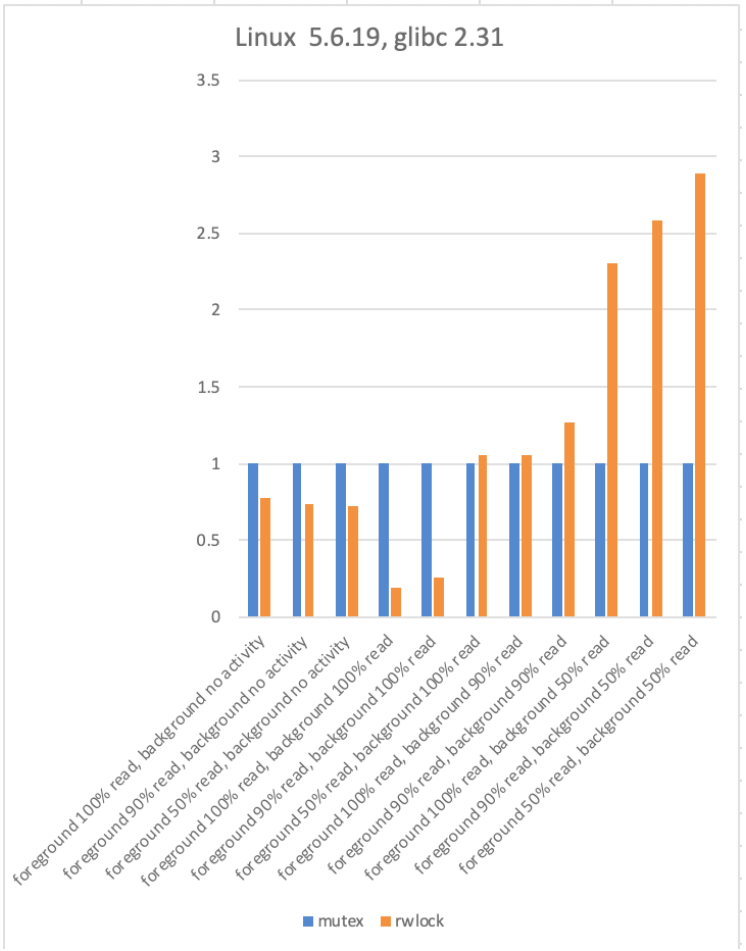
# Read/Write Locks




- Distinction in API between "Reader" and "Writer" of a resource
- Multiple concurrent readers are allowed (t*rd) without waiting => improved scalability
- A writer request has to wait until the currently active readers are finished, behaves then like a mutex, a next reader (or writer) has to wait, until writer has finished.
- On write operations a Write-Lock of a RW-Lock is more expensive than a mutex.
- RW-locks are better, when there are substantially more reader than writer requests

# Performance comparison with different read/write load patterns




Best performance increase with many concurrent read operations

# Busy locks with Mutex vs. RWLocks on nsv

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

With mutex (after 24h)

| | locks | busy |
|---|---|---|
| nsv:3:openacs.org | 4.71M | 1.3K |
| nsv:6:openacs.org | 4.88M | 1.03K |
| nsv:2:openacs.org | 3.37M | 784 |
| nsv:7:openacs.org | 9.11M | 755 |
| nsv:5:openacs.org | 2.88M | 460 |

With rwlocks (after 24h)

| | | |
|---|---|---|
| nsv:7:openacs.org | 7.22M | 0 |
| nsv:6:openacs.org | 3.92M | 143 |
| nsv:3:openacs.org | 3.31M | 1 |
| nsv:2:openacs.org | 2.23M | 16 |
| nsv:5:openacs.org | 2.16M | 0 |

Substantially reduced busy operations.

# Most nsv operations on OpenACS instances are read operations

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

| Name | ID | Locks | Busy | Contention | Total Lock | Avg Lock | Total Wait | Max Wait | Locks/Req | Pot.Locks/sec | Pot.Reqs/sec | Read | Write | Write % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| nsv:135:live | 77 | 10.06M | 0 | 0.0000% | 4.2ms | 0s | 0s | 0s | 35.05 | 2.41G | 68.77M | 10.05M | 10.71K | 0.11% |
| ns:driver:async-writer:queue | 2232 | 5.66M | 100.1K | 1.7696% | 1.59s | 280ns | 272.62ms | 9.82ms | 19.71 | 3.57M | 180.96K | | | |
| rw:adp:tags:live | 214 | 5.5M | 0 | 0.0000% | 0s | 0s | 0s | 0s | 19.18 | 500.39G | 26.09G | 5.5M | 44 | 0.00% |
| ns:rw:urlspace:4:live | 216 | 5.43M | 62 | 0.0011% | 268.8ms | 50ns | 183µs | 0s | 18.91 | 20.19M | 1.07M | 5.39M | 34.95K | 0.64% |
| ns:cache:misc_cache-1 | 2080 | 5.36M | 1.13K | 0.0211% | 4.67s | 870ns | 6.68ms | 578µs | 18.69 | 1.15M | 61.43K | | | |
| nsv:56:live | 156 | 5.34M | 1 | 0.0000% | 17.3ms | 0s | 1µs | 0s | 18.61 | 308.17M | 16.56M | 5.3M | 44.69K | 0.84% |
| nsv:91:live | 121 | 4.63M | 5 | 0.0001% | 17.5ms | 0s | 3µs | 0s | 16.14 | 265.37M | 16.44M | 4.62M | 9.99K | 0.22% |
| nsv:132:live | 80 | 3.77M | 0 | 0.0000% | 1.2ms | 0s | 0s | 0s | 13.13 | 3.05G | 232M | 3.77M | 3K | 0.08% |
| nsv:134:live | 78 | 3.53M | 174 | 0.0049% | 868.2ms | 250ns | 920µs | 0s | 12.31 | 4.07M | 330.54K | 3.36M | 171.51K | 4.86% |
| nsv:131:live | 81 | 2.92M | 0 | 0.0000% | 22.4ms | 10ns | 0s | 0s | 10.17 | 130.11M | 12.79M | 2.85M | 65.59K | 2.25% |
| ns:cache:site_nodes_cache-0 | 2066 | 2.51M | 1.02K | 0.0407% | 4.39s | 1.75µs | 29.31ms | 8.98ms | 8.74 | 571.62K | 65.42K | | | |
| ns:cache:site_nodes_cache-1 | 2067 | 2.51M | 805 | 0.0321% | 4.57s | 1.82µs | 16.78ms | 2.61ms | 8.73 | 548.23K | 62.8K | | | |

Statistics from nsstats.tcl

Write percentage usually very little!

RWLocks used for:

- nsv (shared variables)
- URLspace (trie for managing URLs, using path segments)
- Connection channels

# Numbers every developer should know

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

```
    114    ns time {dict get {a 1 b 2 c 3} b} 100000
    156    ns set x 1; time {set x} 100000
    160    ns time {set x 1} 100000
    163    ns set x 1; time {info exists x} 100000
    204    ns time {ns_quotehtml "hello world"} 100000
    209    ns time {ns_trim {hello world}} 100000
    210    ns set x 1; time {expr {$x + $x}} 100000
    212    ns nsv_set foo x 1; time {nsv_get foo x} 100000
    235    ns proc foo {x} {return $x}; time {foo 1} 100000
    269    ns time {info commands ::db_string} 100000
    288    ns time {array set x {a 1 b 2 c 3}} 100000
    291    ns time {ns_cache_eval ns:memoize 1 {set x 1}} 100000
    303    ns nx::Class create Foo {:public method bar {} {return 0};:create ::foo1}; time {::foo1 bar} 100000
    305    ns time {nsv_set foo x 1} 100000
    360    ns time {ns_sha1 foo} 100000
    513    ns ns_urlspace set -key foo1 /*.adp A; time {ns_urlspace get -key foo1 /static/test.adp} 100000
    821    ns time {nx::Object create ::o} 100000
  28668    ns time {md5::md5 foo} 100000
  30338    ns time {sha1::sha1 foo} 100000
  78894    ns time {xo::dc get_value dbqd..qn {select title from acs_objects where object_id=179}} 100000
  86132    ns time {db_string dbqd..qn {select title from acs_objects where object_id=179}} 100000
 152654    ns time {set F [open /tmp/nix w]; puts $F x; close $F} 10000
2562594    ns time {exec ls /} 1000
```

`nsv get` 25% slower than `info exists`
`nsv set` similar to `array set`
`ns_cache read` between above (0.3µs)
`ns_urlspace get` is cache *2
… but 100x faster then DB
… REDIS cache time ~1ms

== 2.6 ms

`ns_udp roundtrip: 1ms`

EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION AMBA ACCREDITED

# Atomic nsv operations (1/2)

- Motivation

```
if {![nsv_exists ARRAY KEY]} {
    #
    # Danger Zone
    #
    nsv_set ARRAY KEY DEFVALUE]
 }
# . . .

set oldCmds [nsv_get ARRAY KEY]
#
# Danger Zone
#
nsv_set ARRAY $newCmds
```

- Race conditions!
  - What happens if similar other code is executed concurrently in a different thread?
  - Consequence: unreliable code
  - Hard to debug

# Atomic nsv operations (2/2)



- Obtain (old) value from an nsv ARRAY and set it to a new value (Similar to `GETSET` in REDIS)

```
set foo [nsv_set -reset ARRAY KEY NEWVALUE]
```


- Obtain a value from an nsv ARRAY and unset it (no new value is provided)

```
set foo [nsv_set -reset ARRAY KEY]
```

- Set a default value for an nsv ARRAY (Similar to `SETNX` in REDIS)

```
nsv_set -default ARRAY KEY DEFAULTVALUE
```

- Atomic dict operations (Similar to "dict" in Tcl)

```
nsv_dict get|set|exists|… ARRAY KEY …
```

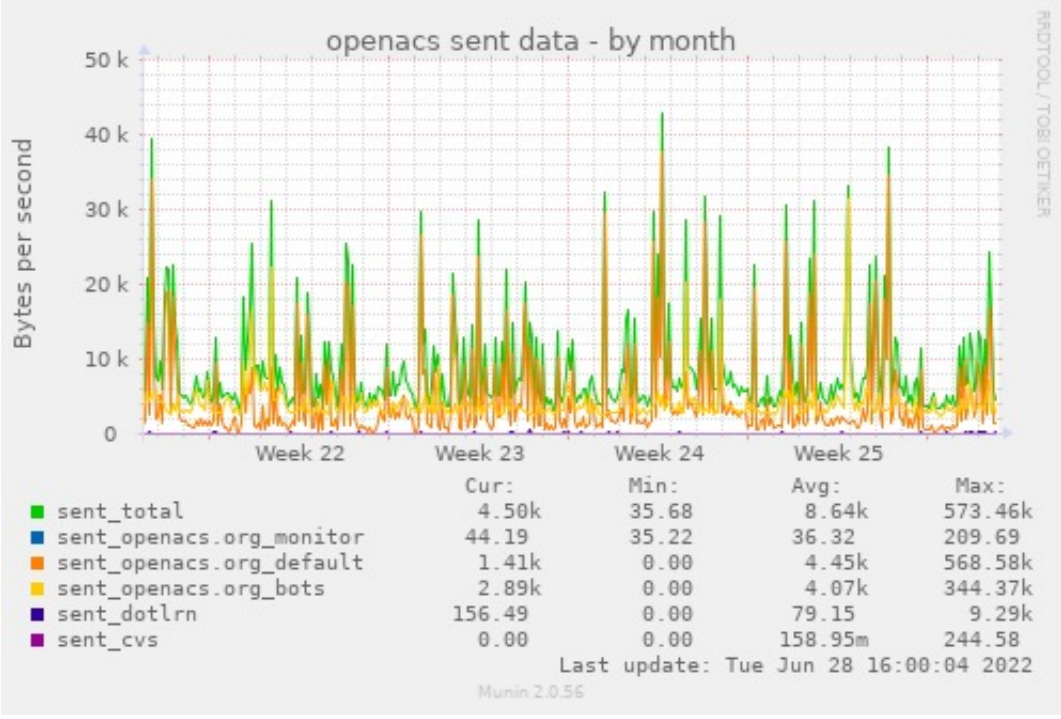
# More Scalability improvements

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

## Some more selected scalability improvements

- All low level API calls in NaviServer became asynchronous
- Configurable behavior for request overruns (optional sending 503 on certain pools)
- Support of multiple driver threads (every driver thread can listen on multiple IP addresses and ports)
- Works with >1024 concurrent open connections (COVID tested)
- Logging commands with high latency

- Bandwidth management
  - Motivation: high number of bot-requests on public sites
  - Bot-requests can be assigned to special request queues
  - These queues can be configured
    - with max transmission rates
    - few connection threads
  - Bandwidth metering of running requests



# Other NaviServer improvements

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

- More strict input handling
  - ns_parseurl, ns_parsehostport
  - Dealing with invalid UTF-8
  - Fallback charsets

acunetix
Web Vulnerability Scanner

```
Log entry: … invalid UTF-8: 'xx|\xe6b|yy'
```

- Improve handling of external services
  - Increasingly more requests depend on external resources (e.g. cloud services)
  - Log-files for outgoing HTTP/HTTPS requests

- Misc

```
ns_http … -timeout 1.5ms …
```

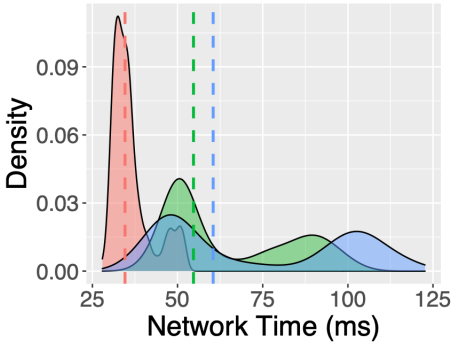
  - Tcl API for URLspace (with context filters 4.99.19)
  - Unit support for API and configuration (e.g. 1s, 100µs, … 1KB, 1.5MB, …)
  - Same collating support for Tcl as in PostgreSQL
  - Crypto support (OCSP Stapling, SNI, SCRYPT, SCRAM, key management EC, MD, HMAC, 20+ digest algorithms) through integration with OpenSSL
  - More NaviServer modules: UDP support, COAP, letsencrypt, revproxy, nsshell…

# Experiment: HTTP/2 for NaviServer

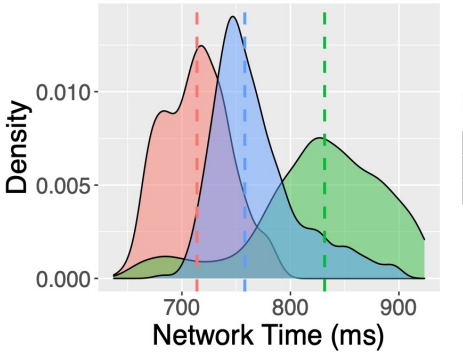
## Master Thesis of Philip Minić:

- Prototype version of NaviServer with HTTP/2 support
- Better performance than Apache and nginx (synthetic data, chromium)

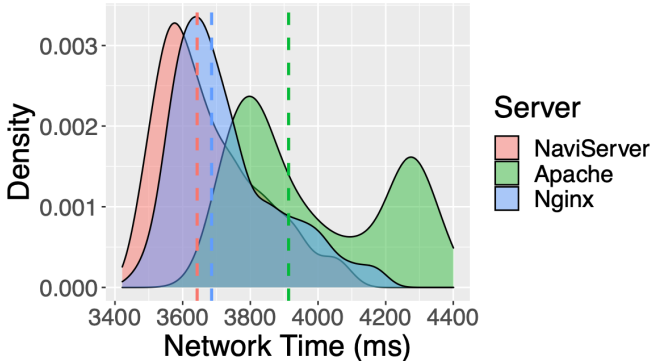
Small files




Medium files




Large files




## Not done yet:

- Integration with http-client, existing API
- Pull requests for OpenSSL with HTTP/3 (QUIC) maybe included in OpenSSL 3.1

# Summary

- Substantially improved concurrency
  - Cache partitioning
  - RWLocks instead of Mutex locks
  - …. Better performance means better use of existing cores
- Development bases on real-world necessities
  - Mostly OpenACS based large scale applications for us (running on year average ~100 threads, requests, background jobs, video streaming, etc. in latency sensitive applications)
  - Many development based on vulnerability scanners
  - Others have very different usage patterns
    - Zoran: windows systems, enterprise backup
    - John Buckman: coffee machine
      ….
  - Questions?

VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

**Institute for Information Systems and New Media**
Welthandelsplatz 1, 1020 Vienna, Austria

**UNIV.PROF.  DR. Gustaf Neumann**

T +43-1-313 36-4671
Gustaf.neumann@wu.ac.at
www.wu.ac.at