End-to-End Java Security Performance Enhancements for Oracle SPARC Servers

Performance engineering for a revenue product

Luyang Wang, Pallab Bhattacharya, *Yao-Min Chen*, Shrinivas Joshi and James Cheng

ORACLE®

 |

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE

Motivation

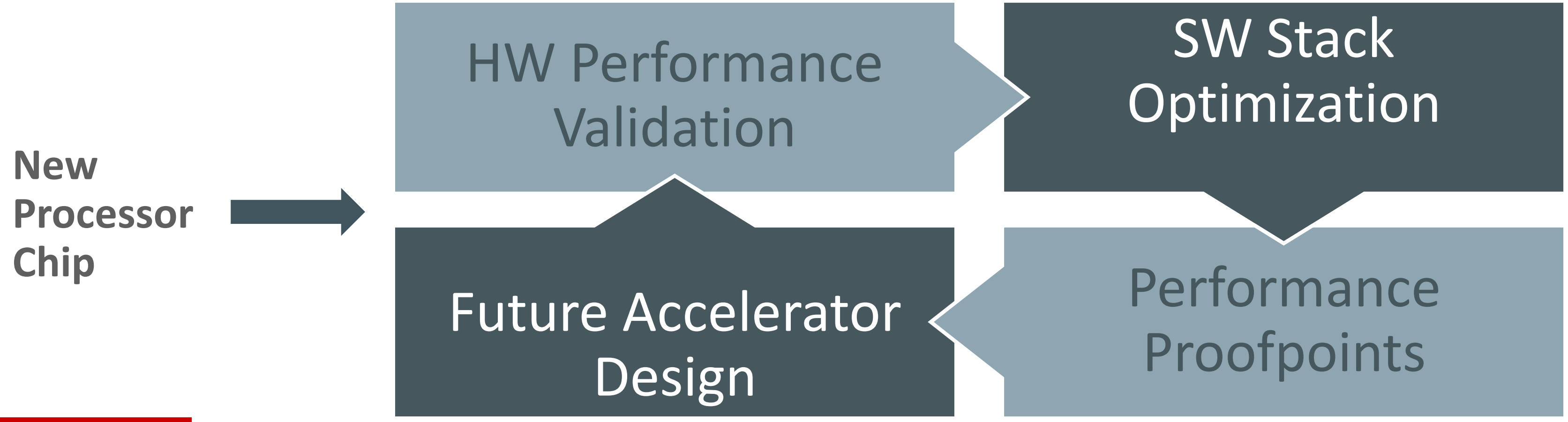
How the project was "funded"

- Escalations from financial/banking customers
  - Application performance issues using PKCS#11 and Java on SPARC processors
- SPARC built-in crypto acceleration might not be fully leveraged
  - Due to application/SW level performance bottlenecks
- Need an in-house environment to root-cause customer issues
  - Genesis of the ***End-to-End Java Security (EEJS)*** workload
- Vertical optimization of software stack
  - Optimizing App + App server + run time system + OS

Project Goals

"Life cycle of performance engineering"

- Fully leveraging HW acceleration of today
- Optimizing software stack
- Generating performance proof points
- Guiding the accelerator design of tomorrow



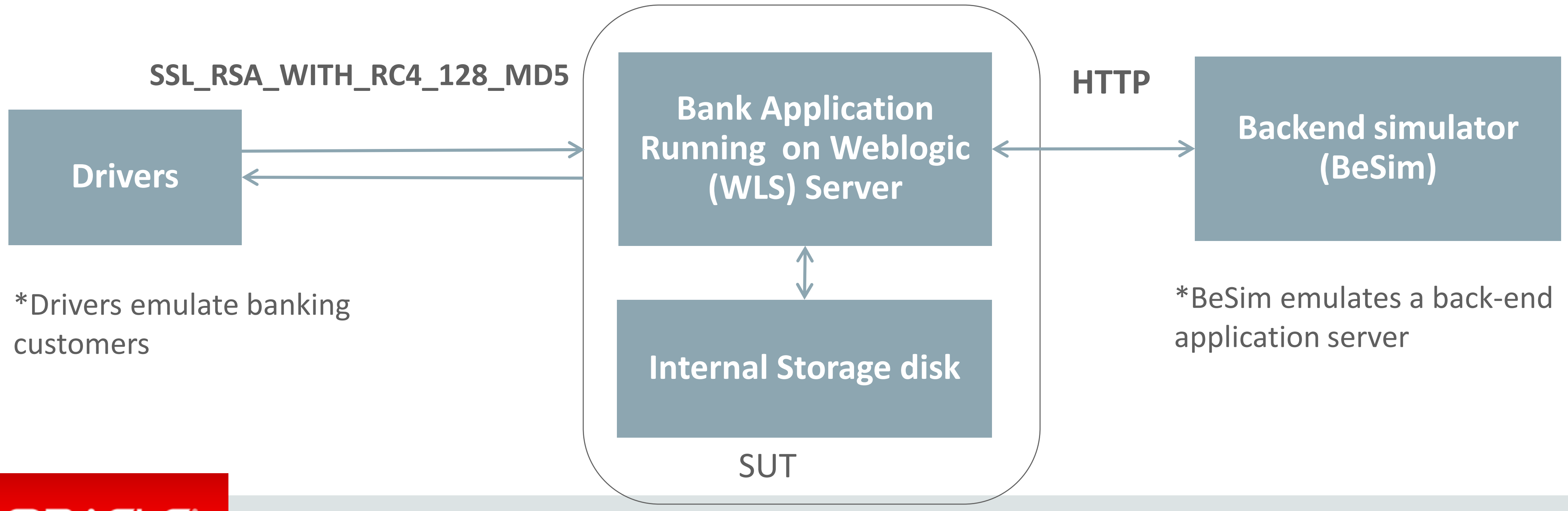
Outline

1. Motivation and Goals
2. End-to-End Java Security (EEJS) Workload
3. Crypto Accelerator Performance Validation
4. Software Stack Performance Optimization
5. Conclusion and Future Work
6. Q&A

ORACLE

SPECweb2005 Banking workload

- Captured online banking activities
- Design was focused on the Web transaction types, banking business logic, transaction payload sizes and the relevant security cipher specification
- Used RSA1024_RC4_MD5/SSL3 cipher suite



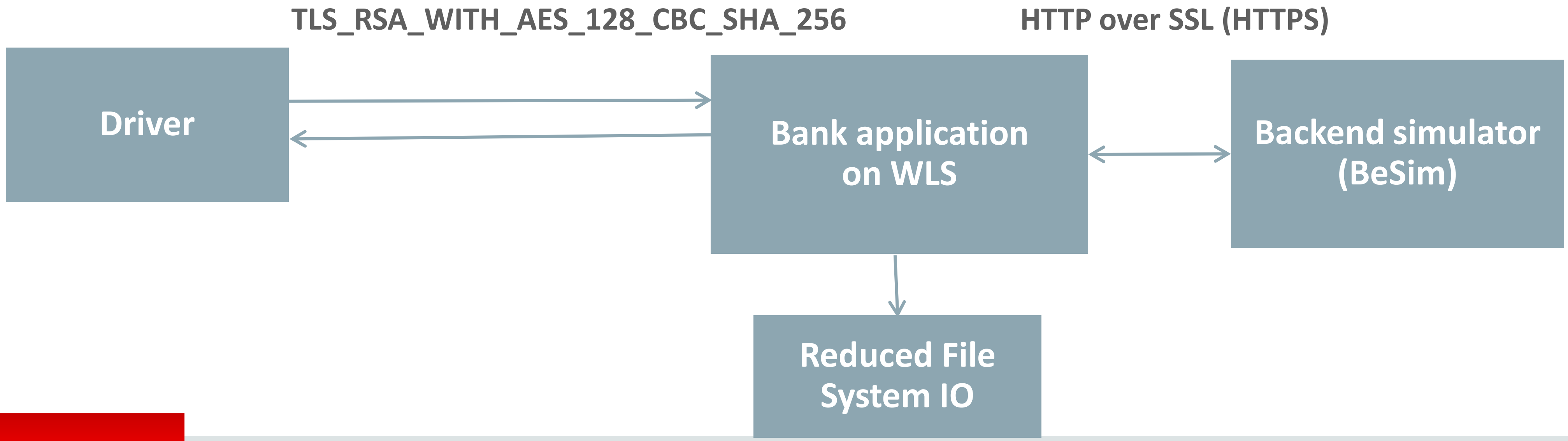
*Drivers emulate banking customers

*BeSim emulates a back-end application server

 |

EEJS Workload Overview
Modernization of SPECweb2005 Banking Workload

- Modern Ciphers
  - AES replaces RC4 and SHA replaces MD5
- Secured backend
  - HTTP over SSL (HTTPS) between WLS and BeSim



EEJS: Performance Metrics

- *Simultaneous user sessions* while meeting the following QoS requirements
  - 95% responses should be returned within TIME_GOOD (2s)
  - 99% responses should be returned within TIME_TOLERABLE (4s)
- With a fixed number of simultaneous user sessions, one can record:
  - Average response time
  - User and system CPU utilizations
  - Percentage of responses that meet TIME_GOOD
  - Percentage of responses that meet TIME_TOLERABLE

ORACLE

Outline

1. Motivation
2. End-to-End Java Security (EEJS) Workload
3. Crypto Accelerator Performance Validation
4. Software Stack Performance Optimization
5. Conclusion and Future Work
6. Q&A

ORACLE

 |

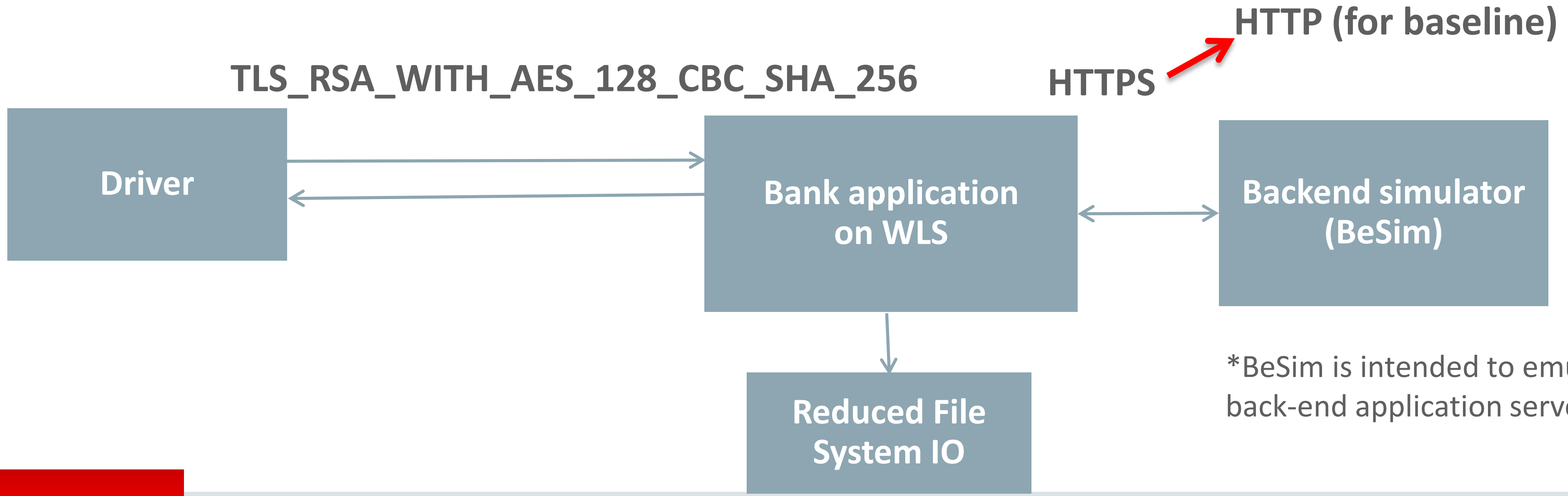
Experiment Setup

- SPARC T7-1
  - 4.13GHz CPU frequency, 480 GB DRAM
  - Single socket of 32 cores; only 4 cores were used for the validation
- JDK 8u40
  - JVM flag: -Xms16g -Xmx16g -Xmn8g -XX:+PrintGCTimeStamps -XX:+PrintGCDetails
- Weblogic 12.2.1
- Solaris 11U3 build 22
- 10Gbps Ethernet private network
- Ramp up 180s, warm up 300s, steady run 600s, ramp down 180s

ORACLE

EEJS: Initial Attempt

- Secured backend exposed application bottlenecks
  - Optimizations are still in progress
- For performance validation purpose, plain text for the backend



*BeSim is intended to emulate a back-end application server

Crypto Implementations in Java

Intro to Cryptographic Service Providers (CSPs)

- A CSP provides concrete implementation of the JDK Security API
- Multiple implementations may exist for the same crypto algorithm
  - *PKCS#11* – legacy implementation of the security API
  - *OracleUcrypto* – implementation using new SPARC crypto instructions, through JNI
  - *Sun JCE* provider with AES Instrinsics (no JNI)
  - *Sun* provider with SHA Intriniscs (no JNI)

ORACLE

CSP Configurations Evaluated

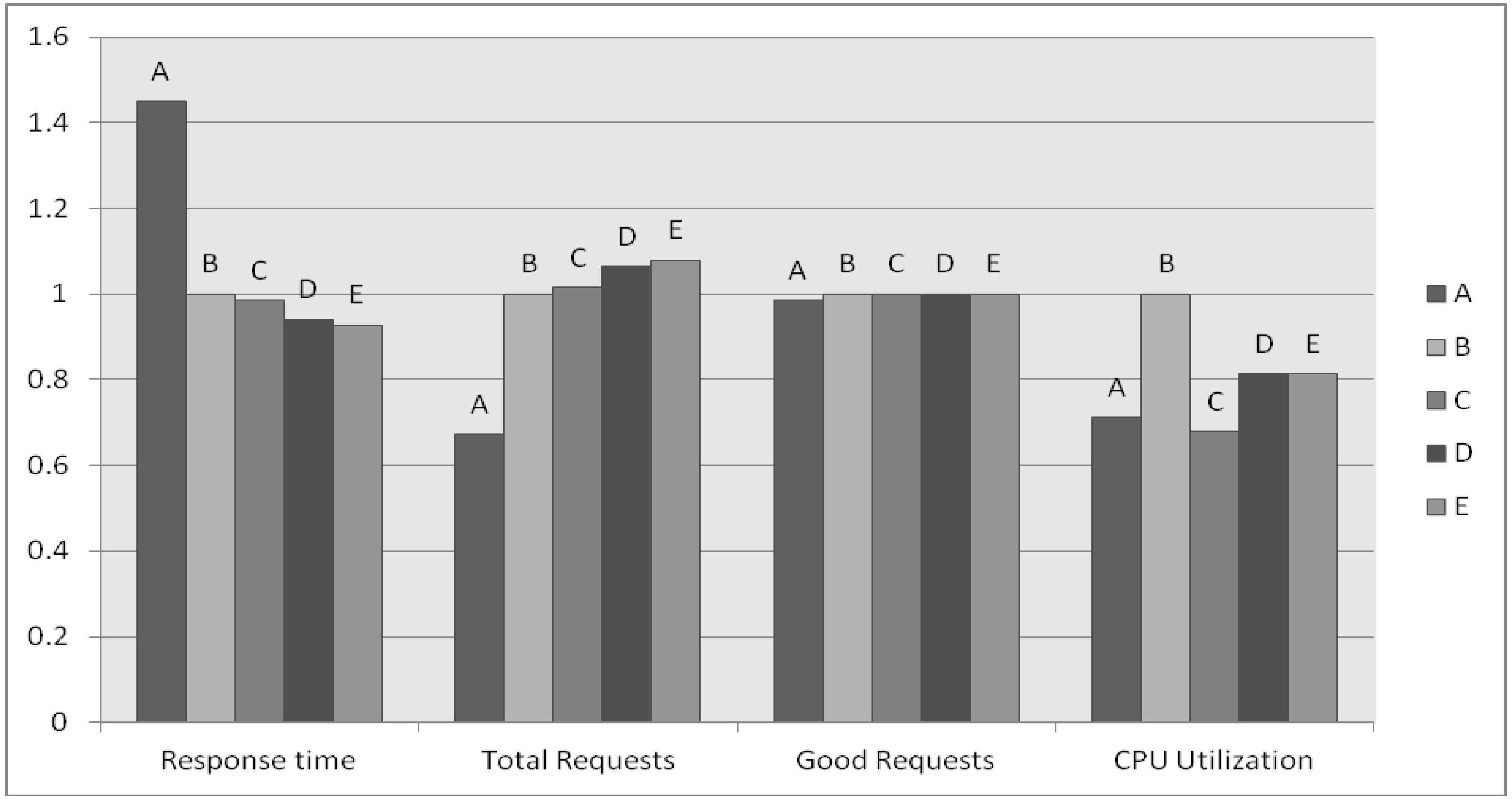
The Lego blocks of the *TLS_RSA_WITH_AES_128_CBC_SHA_256* cipher suite

CSP Configuration	A	B	C	D	E
RSA Encryption and Decryption	SunPKCS11	OracleUcrypto	SunPKCS11	OracleUcrypto	OracleUcrypto
RSA Key Generation	SunPKCS11	SunPKCS11	SunPKCS11	SunPKCS11	SunJCE
AES	SunPKCS11	OracleUcrypto	SunJCE with AES intrinsics	SunJCE with AES intrinsics	SunJCE with AES intrinsics
SHA	SunPKCS11	OracleUcrypto	SUN with SHA intrinsics	SUN with SHA intrinsics	SUN with SHA intrinsics

ORACLE

CSP Configuration Comparisons

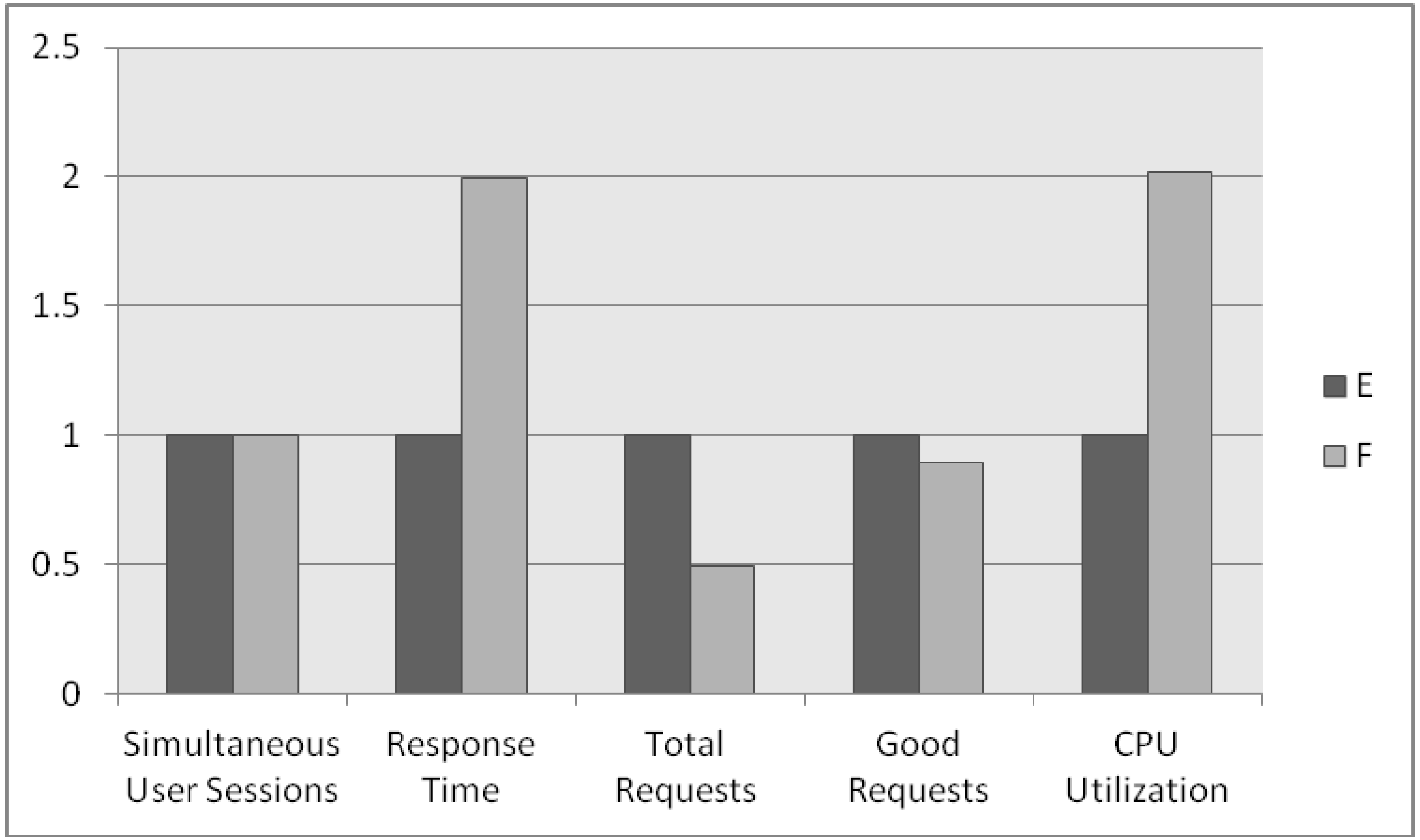
Configuration E is best and will be the default for SPARC



 |

Evaluating Performance Gain from HW Acceleration

Half the response time and twice the throughput at half the CPU



Configuration F: software-only by disabling code path to accelerator

 |

Outline

1 Motivation

2 End-to-End Java Security (EEJS) Workload

3 Crypto Accelerator Performance Validation

4 Software Stack Performance Optimization

5 Conclusion and Future Work

6 Q&A

ORACLE

 |

IO Bottleneck Optimization

- With JDK 8u40, high system CPU time (around 70%) on the SUT
  - *Oracle Studio* profiles helped root cause to the *read()* system call
  - Upon SSL connection creation, a new *TrustManager* was initialized by reading the *cacerts* file
  - From the truss tool, found that *cacerts* was read by one byte at a time
- Tracked by an OpenJDK bug (JDK-8129634)
  - Resolved as a byproduct of a relevant bug JDK-8062552
  - The fix for JDK-8062552 wraps the *DataInputStream* representing the *cacerts* file in a *BufferedInputStream* object
- System CPU time drops to around 20%

Removal of Redundant Operations

- *TrustManager* instances repeatedly read the *cacerts* file and create the *KeyStore* instance and loads the trusted certificates
- Only one *KeyStore* instance needs to be created and cached
- The *cacerts* file should only be read when there is a modification
- Tracked by JDK-8129988
- Implemented Proof-of-Concept code
- 9.5% improvement in average response time

Elimination of Hot Locks

- *SecureRandom.nextBytes(byte[] bytes)*
  - With jstack, noticed major synchronization overhead on this synchronized method
    - Most server threads are blocked in this method
  - Observed for both Sun and PKCS11 secure random providers
    - NativePRNG, SHA1PRNG
- OpenJDK bug JDK-8098581
  - Now fixed in JDK 9
  - Buffered bytes read from */dev/random* file
  - Using more fine-grained lock
- The average response time improved by additional 19%

ORACLE

SW Optimization: a Summary

- 1.6X simultaneous user sessions
- 34.6% improvement in average response time
- Many issues are in progress:
  - OS bugs on threads blocking on destroying an object
  - Out of memory when enabling a slightly different cipher suite
  - Static certificate cache limits scalability
  - And several others

ORACLE

Outline

1 Motivation

2 End-to-End Java Security (EEJS) Workload

3 Crypto Accelerator Performance Validation

4 Software Stack Performance Optimization

5 Conclusion and Future Work

6 Q&A

ORACLE

Conclusion

- A case study of performance engineering in practice
  - Engineered a workload to suit our need
  - Identified the best CSP configuration for HW performance validation
  - Generated performance proof points
  - Used profiling tools to identify software optimization opportunities
  - Tracked them as bugs and produced PoC implementations
  - Provided feedback and participated in future SPARC processor design

ORACLE

Future Work

- Make best CSP configuration the default
- Backport critical performance issues to deployed JDK versions
- Continue performance optimizations using EEJS
- Validate the next hardware iteration

ORACLE

Questions?

- Feel free to contact yaomin.chen@oracle.com
- Check out Software in Silicon Cloud https://swisdev.oracle.com/

ORACLE