COLO: COarse-grain LOck-stepping Virtual Machine for Non-stop Service

Eddie Dong, Yunhong Jiang

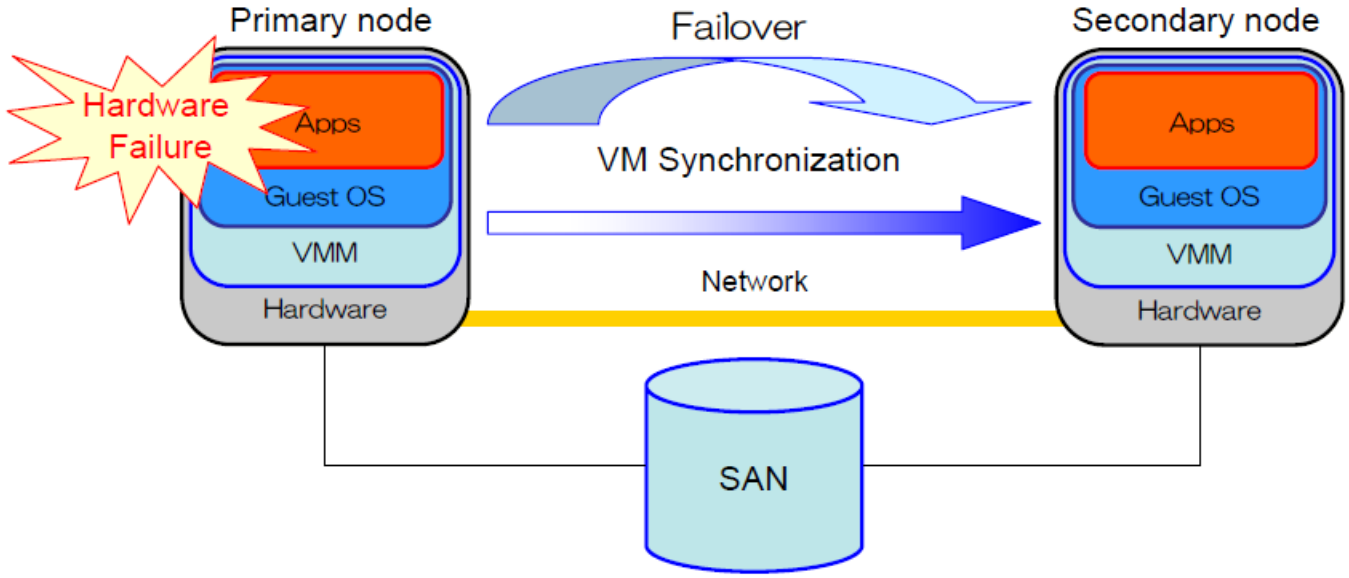
Legal Disclaimer

Agenda

- Background
- COarse-grain LOck-stepping
- Performance Optimization
- Evaluation
- Summary

Non-Stop Service with VM Replication

- Typical Non-stop Service Requires
  - Expensive hardware for redundancy
  - Extensive software customization
- VM Replication: Cheap Application-agnostic Solution



Existing VM Replication Approaches

- Replication Per Instruction: Lock-stepping
  - Execute in parallel for deterministic instructions
  - Lock and step for un-deterministic instructions
- Replication Per Epoch: Continuous Checkpoint
  - Secondary VM is synchronized with Primary VM per epoch
  - Output is buffered within an epoch

Problems

- Lock-stepping
  - Excessive replication overhead
    - memory access in an MP-guest is un-deterministic
- Continuous Checkpoint
  - Extra network latency
  - Excessive VM checkpoint overhead

Agenda

- Background
- COarse-grain LOck-stepping
- Performance Optimization
- Evaluation
- Summary

Why COarse-grain LOck-stepping (COLO)

- VM Replication is an overly strong condition
  - Why we care about the VM state ?
    - The client care about response only
  - Can the control failover *without* "precise VM state replication"?
- Coarse-grain lock-stepping VMs
  - Secondary VM is a replica, as if it can generate same response with primary so far
    - Be able to failover without service stop

Non-stop service focus on server response, not internal machine state!

How COLO Works

- Response Model for C/S System

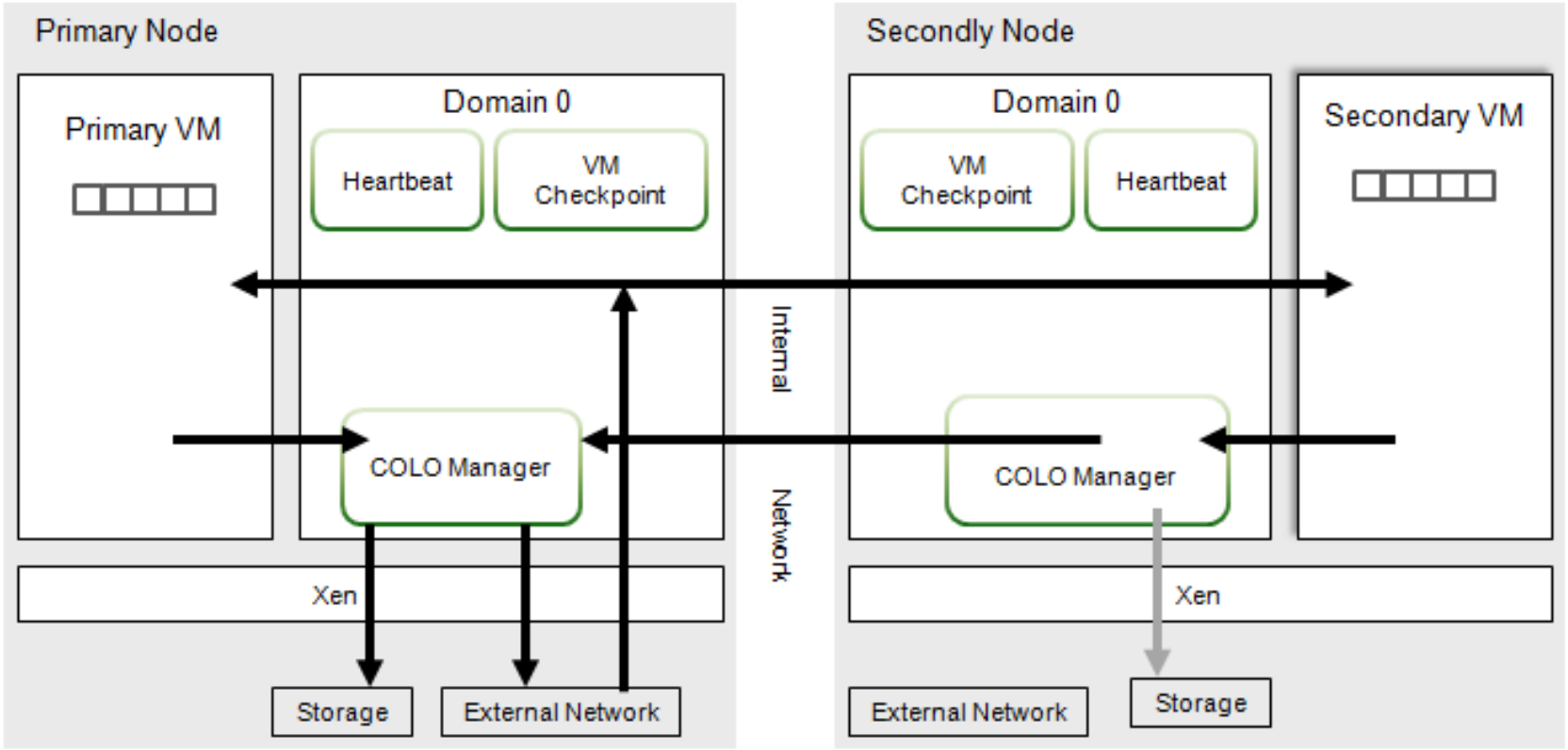
$$R_n = g_n(r_0, r_1, r_2, \dots, r_n, u_0, \dots, u_m)$$

  - r_i & u_i are the request and the execution result of an un-deterministic instruction
  - Each response packet from the equation is a semantics response
- Successfully failover at k^{th} packet if

$$C = \{R_1^P, \dots, R_k^P, R_{k+1}^S, \dots\} \qquad \forall i \leq k, R_i^S = R_i^P$$

  (C is the packet series the client received)

Architecture of COLO



COarse-grain LOck-stepping Virtual Machine for Non-stop Service

Why Better

- Comparing with Continuous VM checkpoint
  - No buffering-introduced latency
  - Less checkpoint frequency
    - On demand vs. periodic
- Comparing with lock-stepping
  - Eliminate excessive overhead of un-deterministic instruction execution due to MP-guest memory access

Agenda

- Background
- COarse-grain LOck-stepping
- Performance Optimization
- Evaluation
- Summary

Performance Challenges

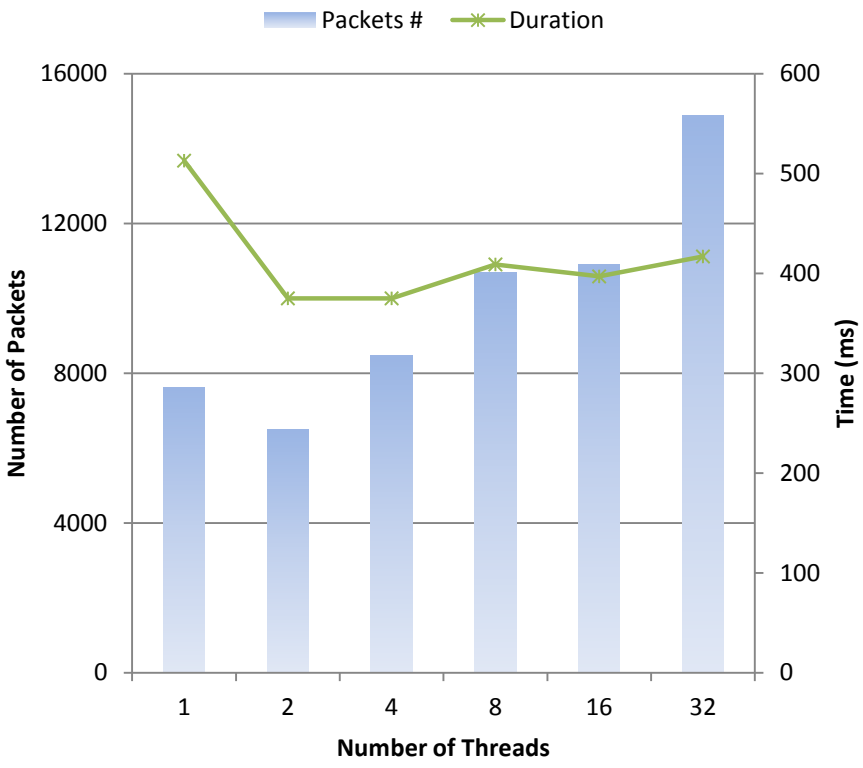
- Frequency of Checkpoint
  - Highly dependent on the *Output Similarity*, or *Response Similarity*
    - Key Focus is TCP packet!
- Cost of Checkpoint
  - Xen/Remus uses passive-checkpoint
    - Secondary VM is not resumed until failover → Slow path
  - COLO implements active-checkpoint
    - Secondary VM resumes frequently

Improving Response Similarity

- Minor Modification to Guest TCP/IP Stack
  - Coarse Grain Time Stamp
  - Highly-deterministic ACK mechanism
  - Coarse Grain Notification Window Size
  - Per-Connection Comparison

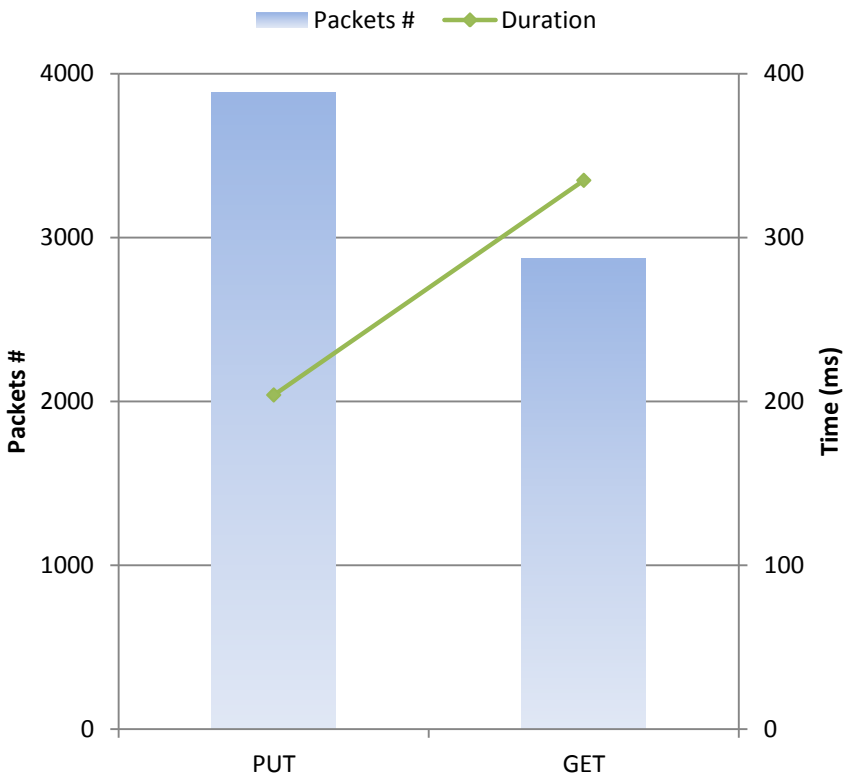
Similarities after Optimization

- Web Server



*Run Web Bench in Client

- FTP Server

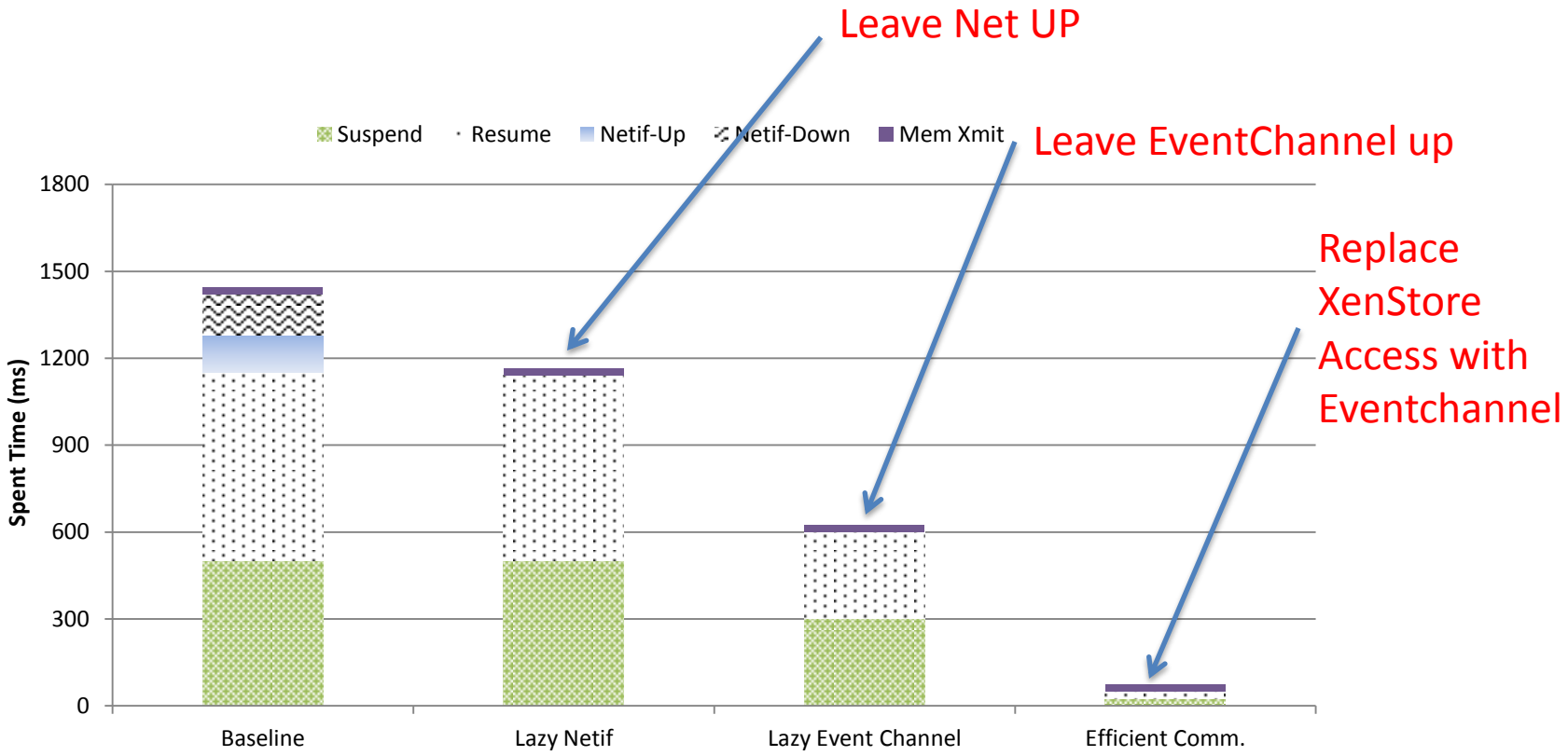


For more complete information about performance and benchmark results, visit Performance Test Disclosure

Reducing the Cost of Active-checkpoint

- Lazy Device State Update
  - Lazy network interface up/down
  - Lazy event channel up/down
- Fast Path Communication

Checkpoint Cost with Optimizations



Final cost: 74ms/checkpoint: (1/3 on page transmission, 2/3 on suspend/resume)

For more complete information about performance and benchmark results, visit Performance Test Disclosure

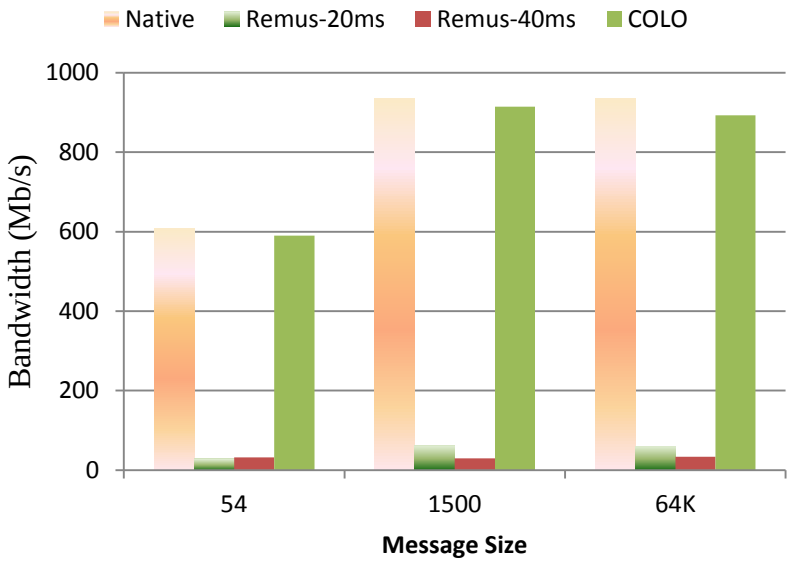
Agenda

- Background
- COarse-grain LOck-stepping
- Performance Optimization
- Evaluation
- Summary

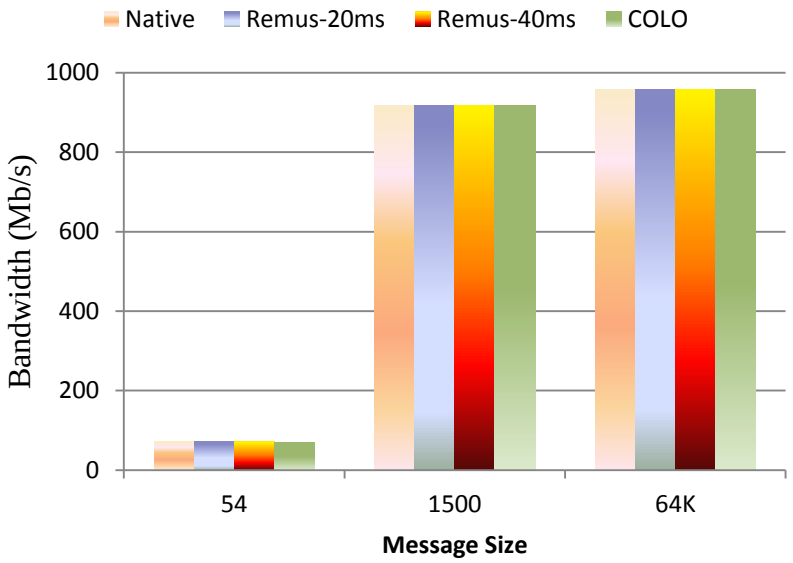
Configurations

- Hardware
  - Intel® Core™ i7 platform, a 2.8 GHz quad-core processor
  - 2GB RAM
  - Intel® 82576 1Gbps NIC * 2 (internal & external)
- Software
  - Xen 4.1
  - Domain 0: RHEL5U5
  - Guest: 32-bit BusyBox 1.20.0, Linux kernel 2.6.32
    - 256MB RAM and uses a ramdisk for storage

Bandwidth of NetPerf



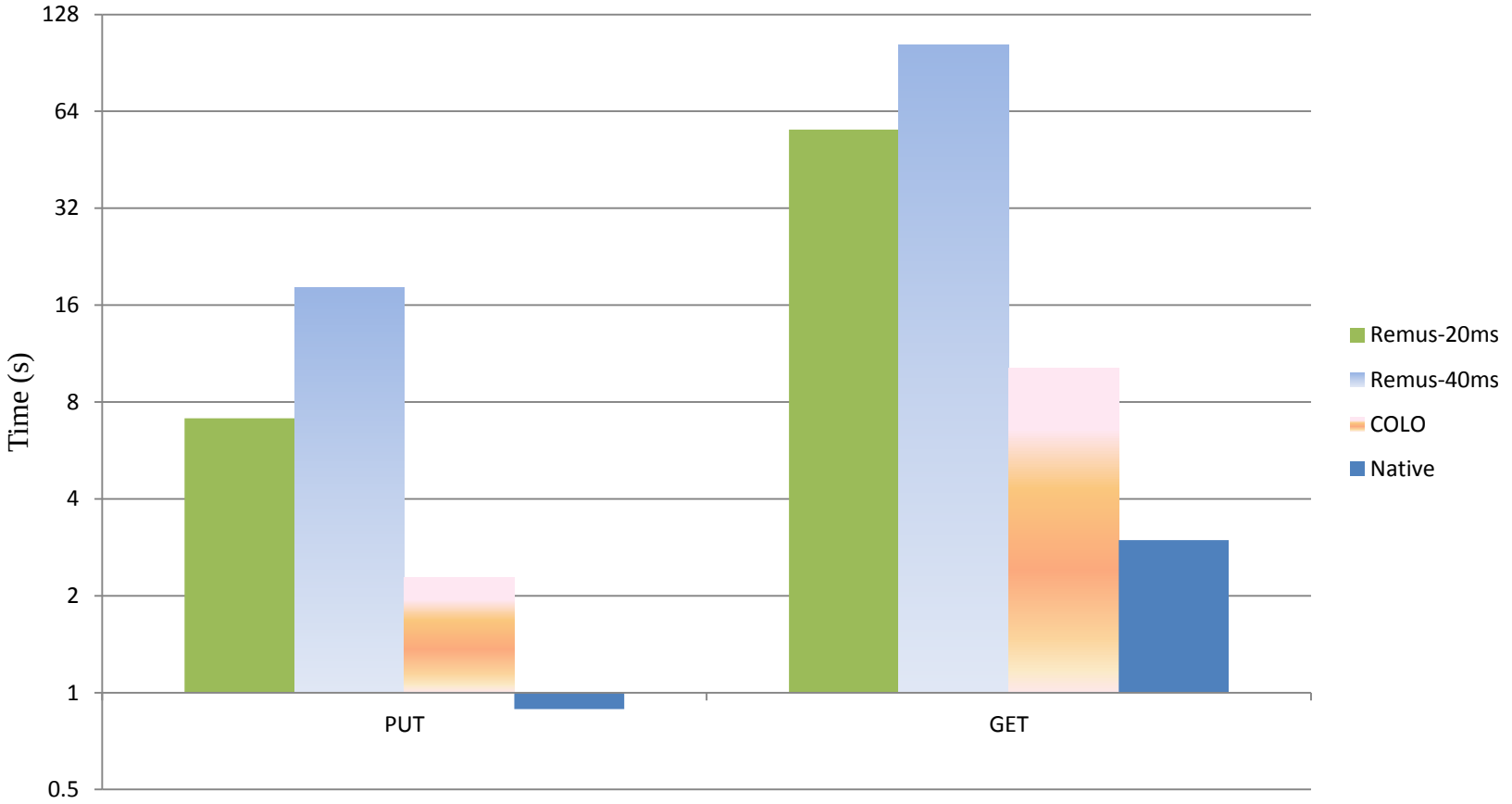
TCP



UDP

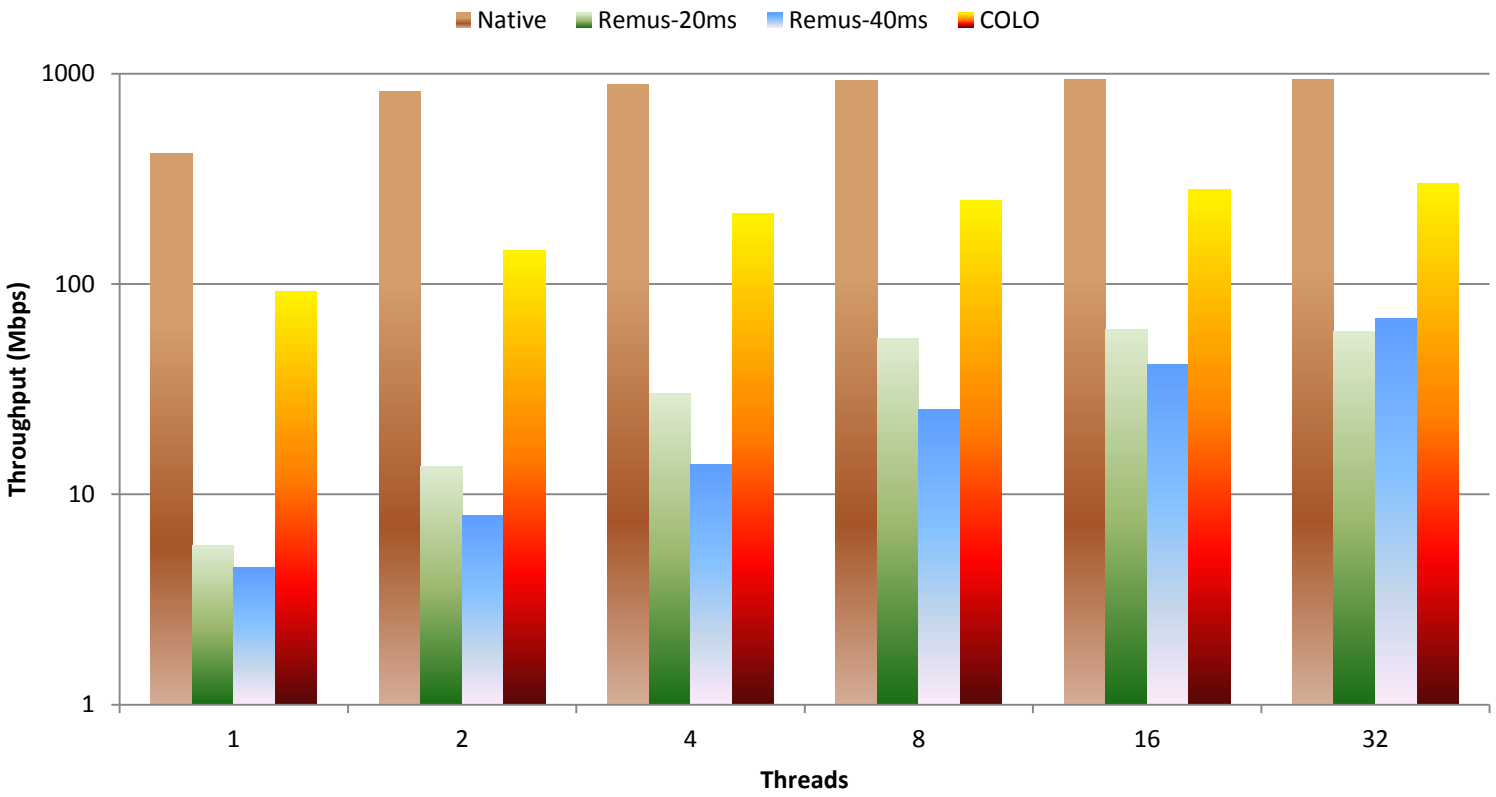
For more complete information about performance and benchmark results, visit Performance Test Disclosure

FTP Server



For more complete information about performance and benchmark results, visit Performance Test Disclosure

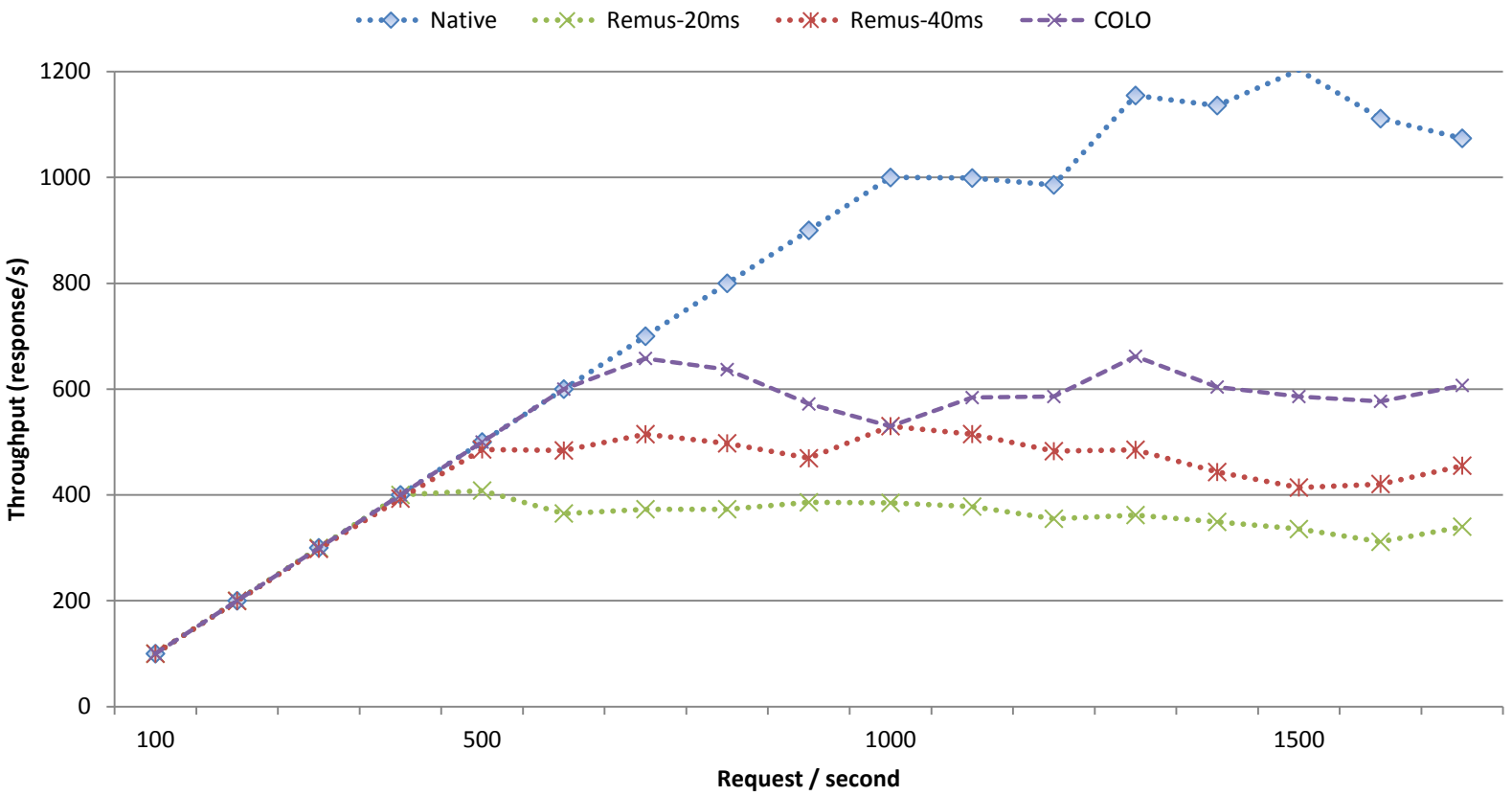
Web Server - Concurrency



Run Web Bench in Client

For more complete information about performance and benchmark results, visit Performance Test Disclosure

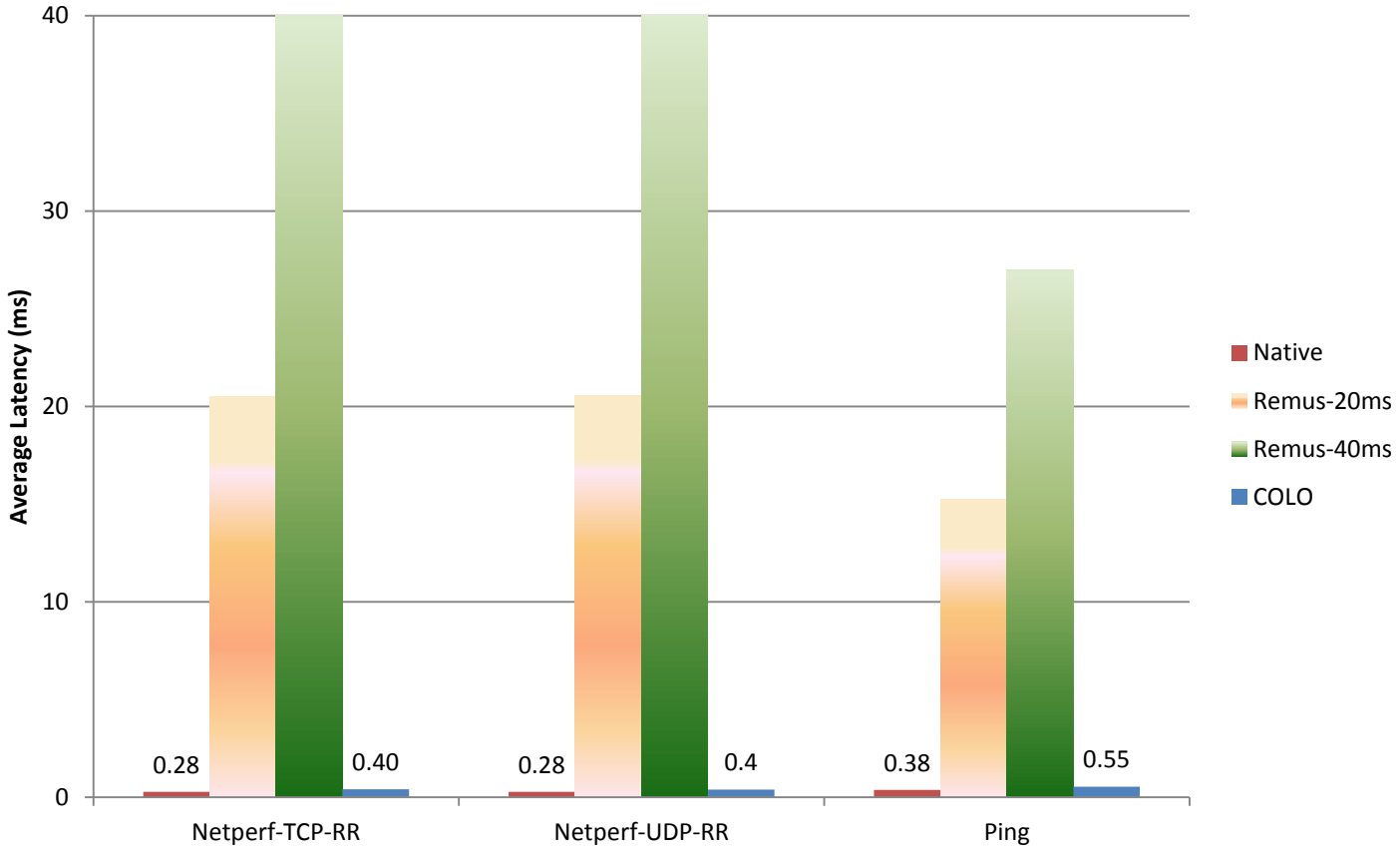
Web Server - Throughput



Run httperf in Client

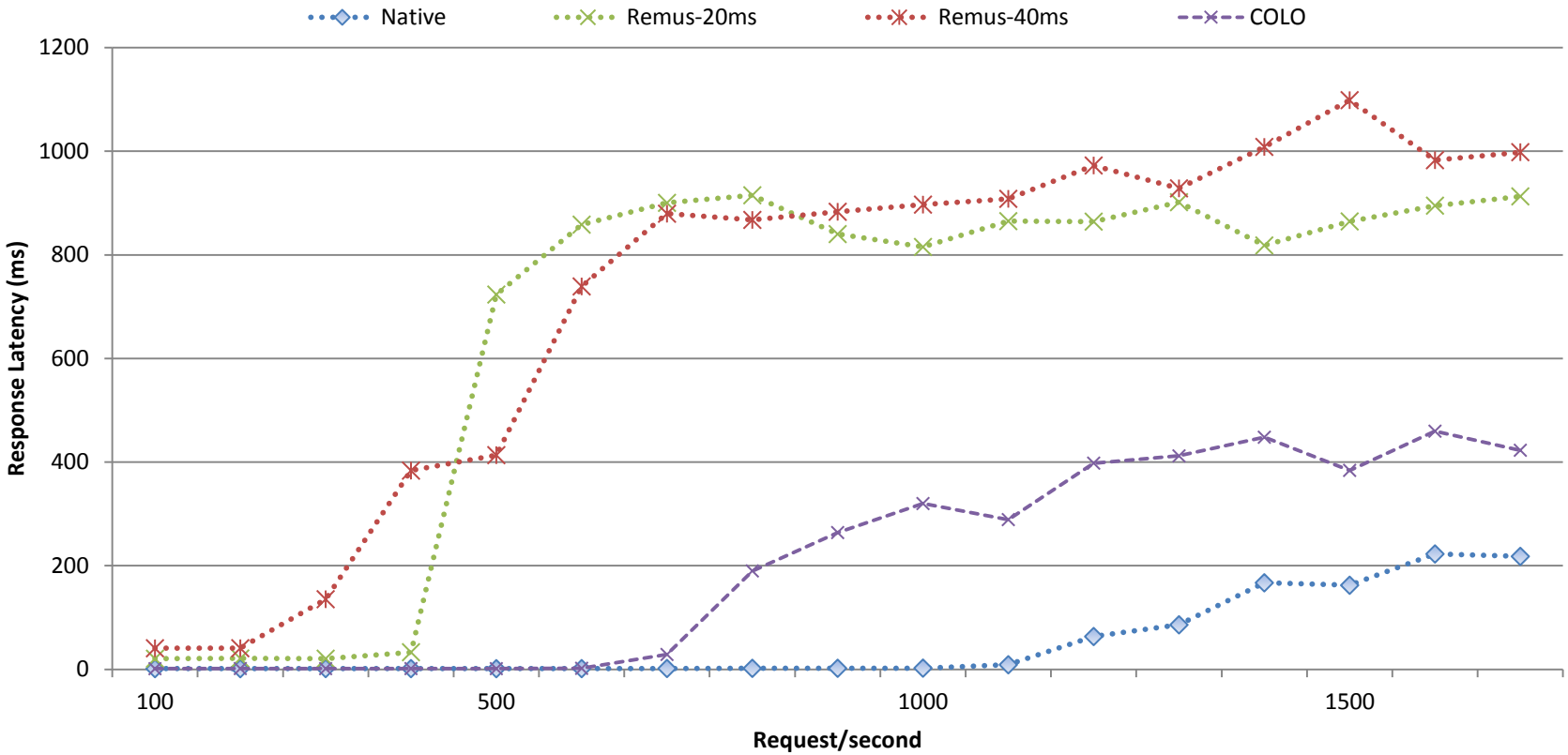
For more complete information about performance and benchmark results, visit Performance Test Disclosure

Latency in Netperf/Ping



For more complete information about performance and benchmark results, visit Performance Test Disclosure

Web Server - Latency



Run httperf in Client

For more complete information about performance and benchmark results, visit Performance Test Disclosure

Agenda

- Background
- COarse-grain LOck-stepping
- Performance Optimization
- Evaluation
- Summary

Summary

- COLO is an ideal Application-agnostic Solution for Non-stop service
  - Web server: 67% of native performance
  - CPU, memory and netperf: near-native performance
- Next steps:
  - Merge into Xen
  - More optimizations


intel®
Software