Old: Flat Sequential Multicast in Linux Kernel

• robust against including 2nd hyperthread

Tree-based multicast:
• mostly linear scaling
• dynamic receiver groups — maintenance cost for multicast trees
• in practice: just sequential propagation with linear scaling

New: On-the-fly Binomial Tree Multicast

• send IPIs to all receivers
• perform the task locally
• decrement counter for pending acks from children
• if zero: propagate to ancestors via shared mem.

Example for Tree-based Multicast

1) CPU 0
   parent: 1
   weight: 7
   ack: 8

2) CPU 2
   parent: 1
   weight: 3

3) CPU 1
   parent: 0
   weight: 4

4) CPU 1
   parent: 0
   weight: 7

5) CPU 4
   parent: 2
   weight: 1

6) CPU 2
   parent: 1
   weight: 1

7) CPU 6
   parent: 0
   weight: 3

8) CPU 1
   parent: 0
   weight: 1

9) CPU 3
   parent: 1
   weight: 2

Recursive propagation based on [1]
• unit set of receivers is empty:
  • remove one as child
  • send message + IPI to child
  • remove one as child
• perform the task locally

Acknowledgement based on [2]
• use on-the-fly tree for software tree combining:
  • decrement counter for pending acks from children
  • if zero: propagate to ancestors via shared mem.

Latency vs. Receivers

Flat variants and implement TLB shutdown:
• mostly linear scaling
• drastic latency increase when including 2nd hyperthread

Tree-based multicast:
• roughly logarithmic scaling
• robust agains including 2nd hyperthread
• crossing NUMA domains: no notable effect

Impact on uRCU Throughput

uRCU reader throughput speedup with varying frequency of write operations.
Normalized to the throughput of standard implementation.
• major increase in maximal frequency of synchronize_rzu
• minor impact (< 1.25%) on reader throughput up to 10kHz
• decrease < 10% at maximal frequency of standard implementation
• decrease < 15% at maximal frequency for tree-based multicast

Example for Tree-based Multicast

• send IPIs to all receivers
• perform the task locally
• decrement counter for pending acks from children
• if zero: propagate to ancestors via shared mem.

Latency vs. Concurrency

Flat variants:
• flat variants scale linearly with the number of concurrent multicasts
• only initiator spends time for sending the multicast messages
• concurrent multicasts are perfectly parallelized

Tree-based multicast:
• visible impact and jitter
• part of the effort for propagation is delegated to receivers
• concurrent multicasts compete on these helpers

Conclusions

• logarithmic scaling over cores works
• latency improvement for membarrier syscall
• little impact on throughput
• also usable for "faster" TLB shutdown
• searching for real-world applications:
  • many threads in shared address space
  • multiple threads waiting for completion or waiting anyway

Goal: Low Latency on Multicores

• Linux membarrier syscall uses multicast to force barriers
• used by user-space Read-Copy-Update library
• multiscard dominates latency on multicore systems
• tree topologies → logarithmic scaling with number of cores
• dynamic receiver groups — maintenance cost for multicast trees
• in practice: just sequential propagation with linear scaling

Evaluation @ HPI Future SOC Lab

• 4-sockets, 72 physical cores, 144 logical cores (hyperthreads)
• Intel Xeon CPU E7-8890 v3 processor
• Ubuntu 16.04
• Linux kernel version 4.16.7 extended with new multicast implementation
• modified kernel is evaluated bare-metal to exclude noise from hypervisors

This work was supported by the German Research Foundation (DFG) under project no. 244611324.