Vhost: Sharing is better

Bandan Das

Eyal Moscovici

Partly sponsored by:

What's it about ?

- **Paravirtualization: Shared Responsibilities**
- **Vhost: How much can we stretch ?**
- **Design Ideas: Parallelization**
- **Design Ideas: Consolidation**
- **Vhost: ELVIS**
- **Upstreaming**
- **Results**
- **Wrap up and Questions**

Shared Responsibilities

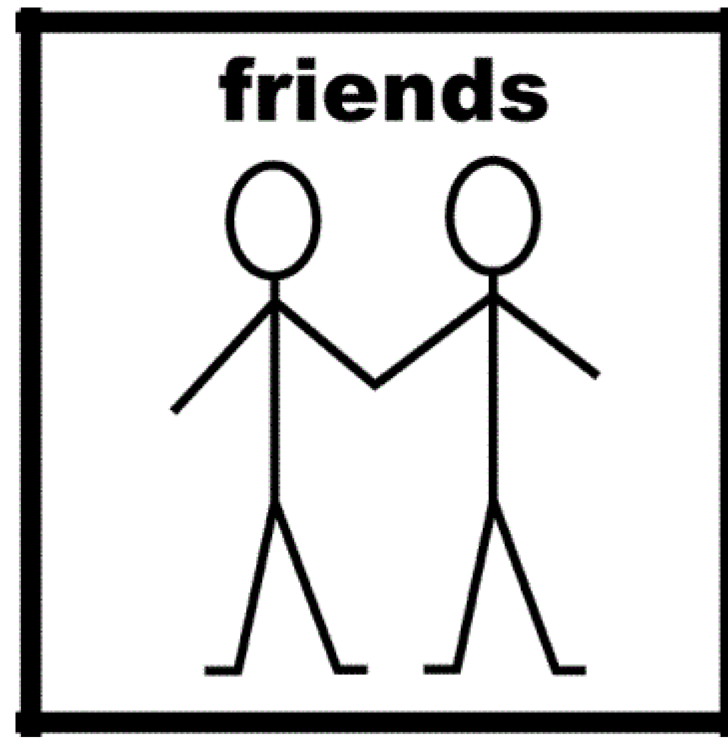
- From Virtualization to Paravirtualization
- Virtio – Host/Guest co-ordination

  - Standardized backend/frontend drivers
- Advantages

  - Host still has ultimate control (compared to hardware device assignment)

  - Security, Fault tolerance, SDN, file-based images, replication, snapshots, VM migration
- Disadvantages

  - Scalability Limitations



Shared Responsibilities

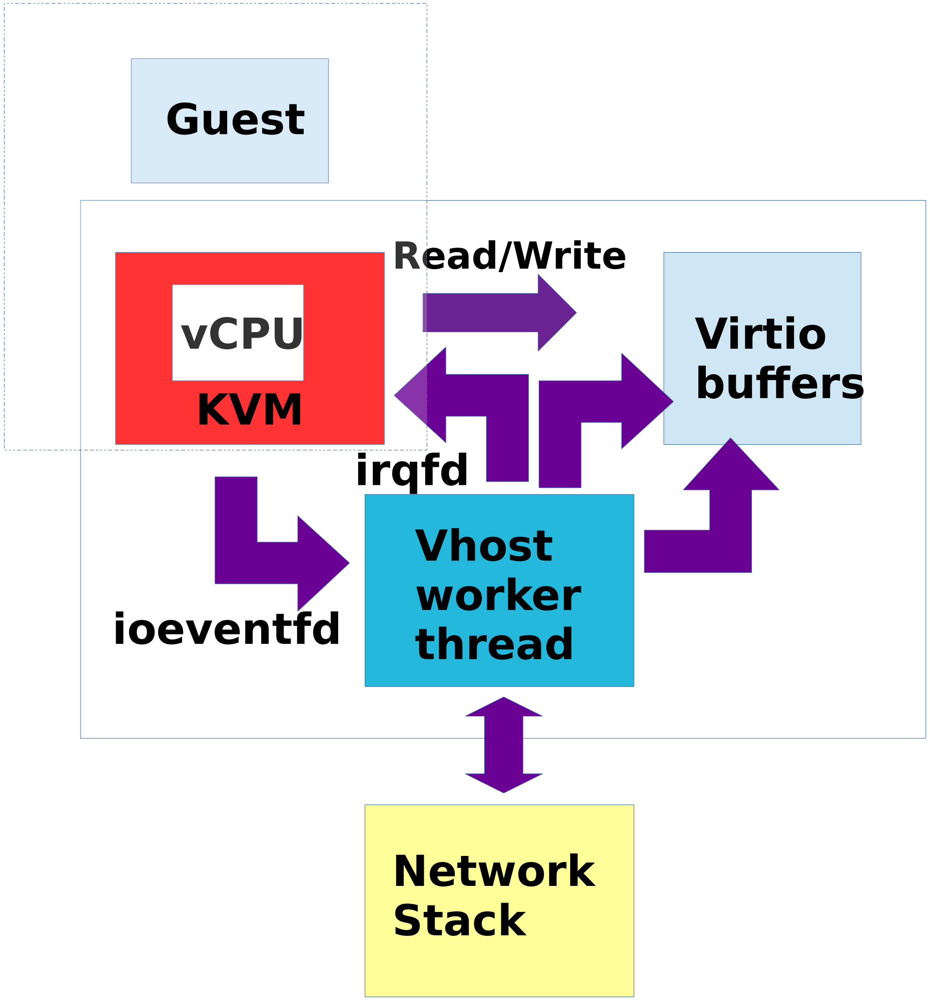
- Vhost kernel

  - Let's move things into the kernel (almost!)

  - Better userspace/kernel API

  - Avoids system calls, improves performance

  - And comes with all the advantages of virtio



How much can we stretch ?

- One worker thread per virtqueue pair
- More guests = more worker threads

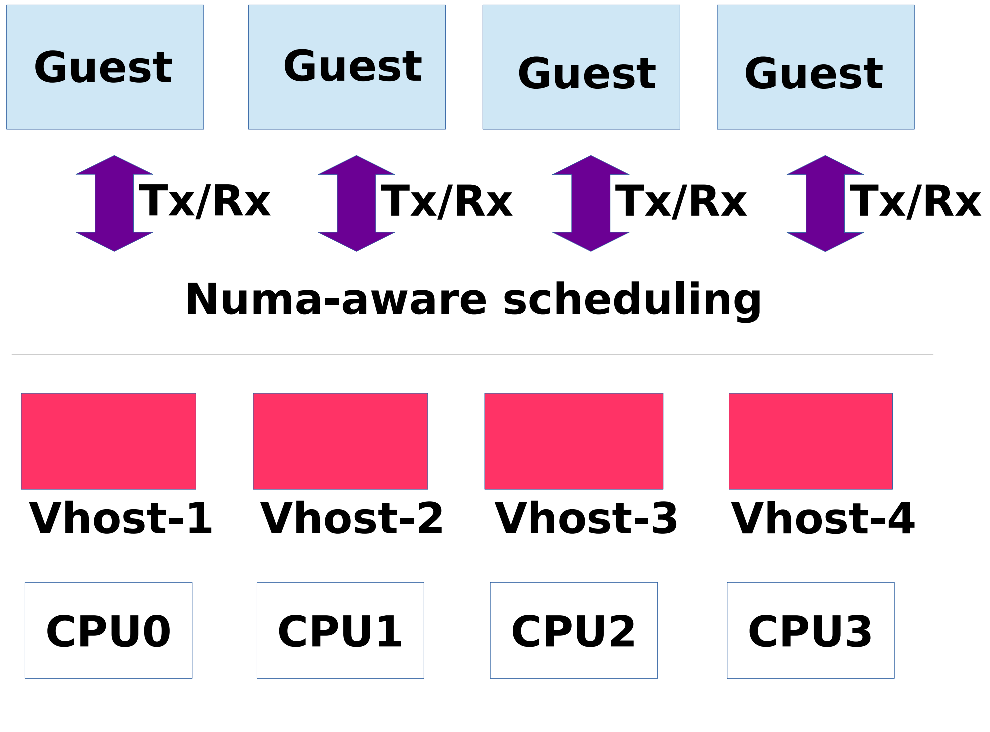
  - But is it necessary ?

  - Can a worker share responsibilities ?
- Performance will improve (or at least stay the same)

  - Main objective: Scalable performance
- No userspace modifications should be necessary

Parallelization (Pronunciation Challenge)

- A worker thread running on every CPU core.
- Guest/Thread mapping is decoupled.
- Guest serviced by a free worker thread with NUMA locality
- Presented by Shirley Ma at LPC 2012



Parallelization

- But....

    - Do we really need “always-on” threads ?

        - is it enough to create threads on demand ?

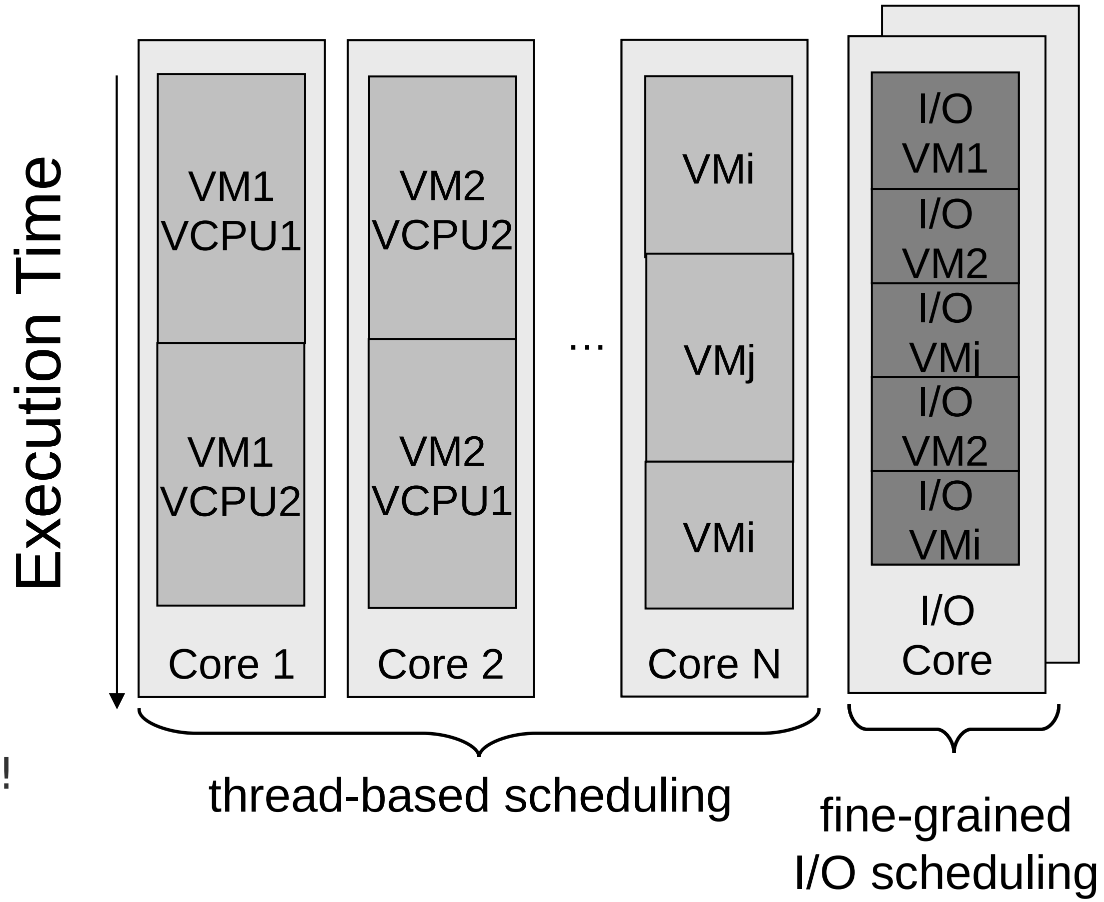
    - Scheduling more complicated when number of guests increase ?

        - Why not share a thread among multiple devices ?

Consolidation - ELVIS (Not the singer)

Presented by Abel Gordon at KVM Forum 2013

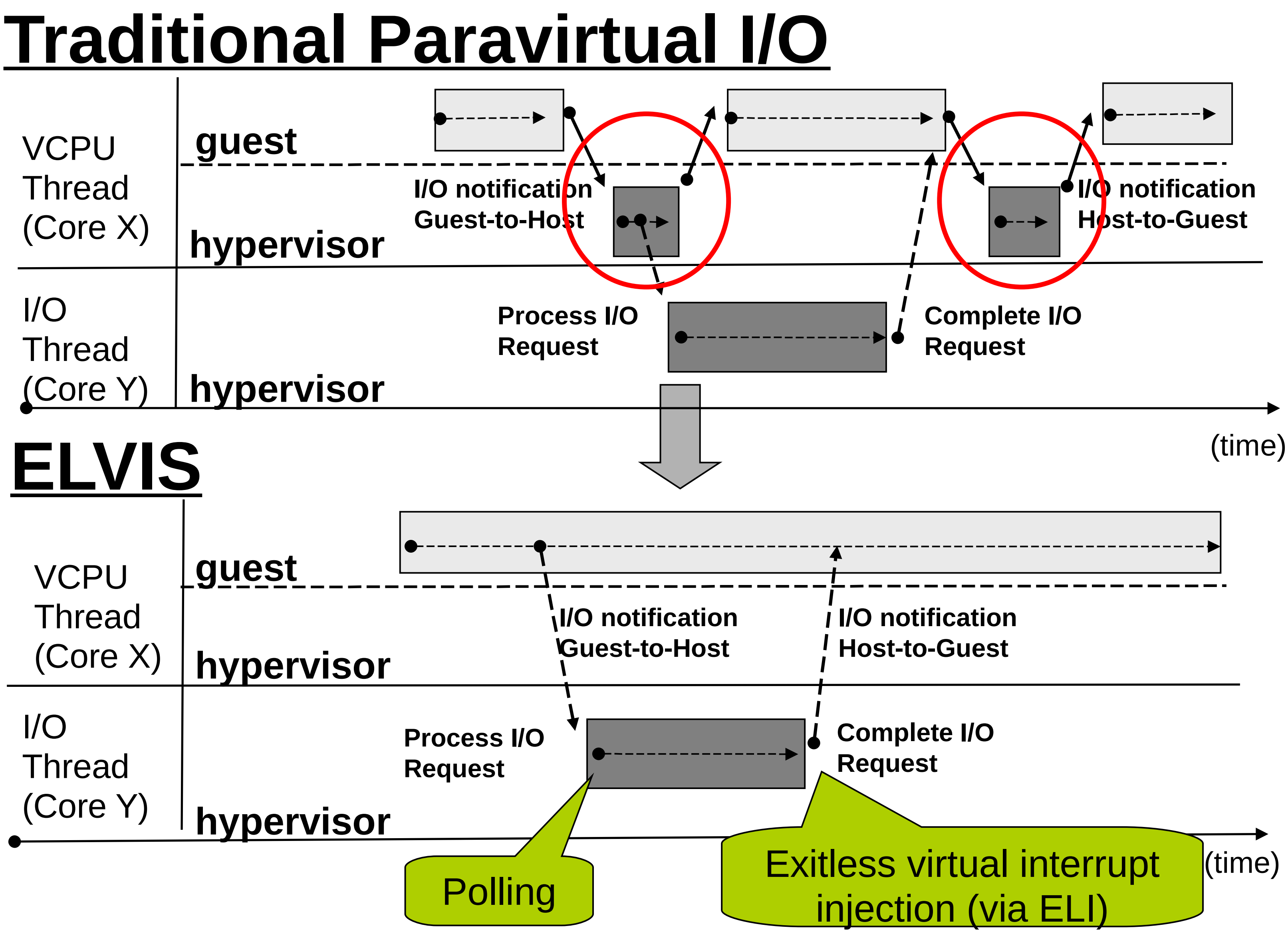
- Divide the cores in the system into two group: VM cores and I/O cores.
- A vhost thread servicing multiple I/O devices from different guest
  - has a dedicated CPU core
  - A user configurable parameter determines how many.
  - A dedicated I/O scheduler on the vhost thread
- Posted interrupts and polling included!



ELVIS Polling Thread

- Single thread in a dedicated core monitors the activity of each queue (VMs I/O)
- Balance between queues based on the I/O activity
  - Decide which queue should be processed and for how long
  - Balance between throughput and latency
- No process/thread context switches for I/O
- Exitless communication (in the next slides)

ELVIS Polling Thread



ELVIS Exitless communication

- Implemented software posted interrupt based on ELI (Exitless interupts)

  - ELI will be very hard to upstream

- Possible replacements

  - KVM PV EOI introduced by Michael S. Tsirkin

  - INTEL VT-d Posted-interrupts (PI) which may be leveraged

Upstreaming..

- A lot of new ideas!
- First Step

  - Stabilize a next generation vhost design.
- The plan:

  - Introduce a shared vhost design and run benchmarks with different configurations

    - RFC posted upstream

    - Initial test results favorable
- Later enhancements can be introduced gradually...

Cgroups **(Buzzwords, JK ;))**

- Initial approach

  - Add a function to search all cgroups in all hierarchies for the new process.

  - Even a single mismatch => create a new vhost worker.

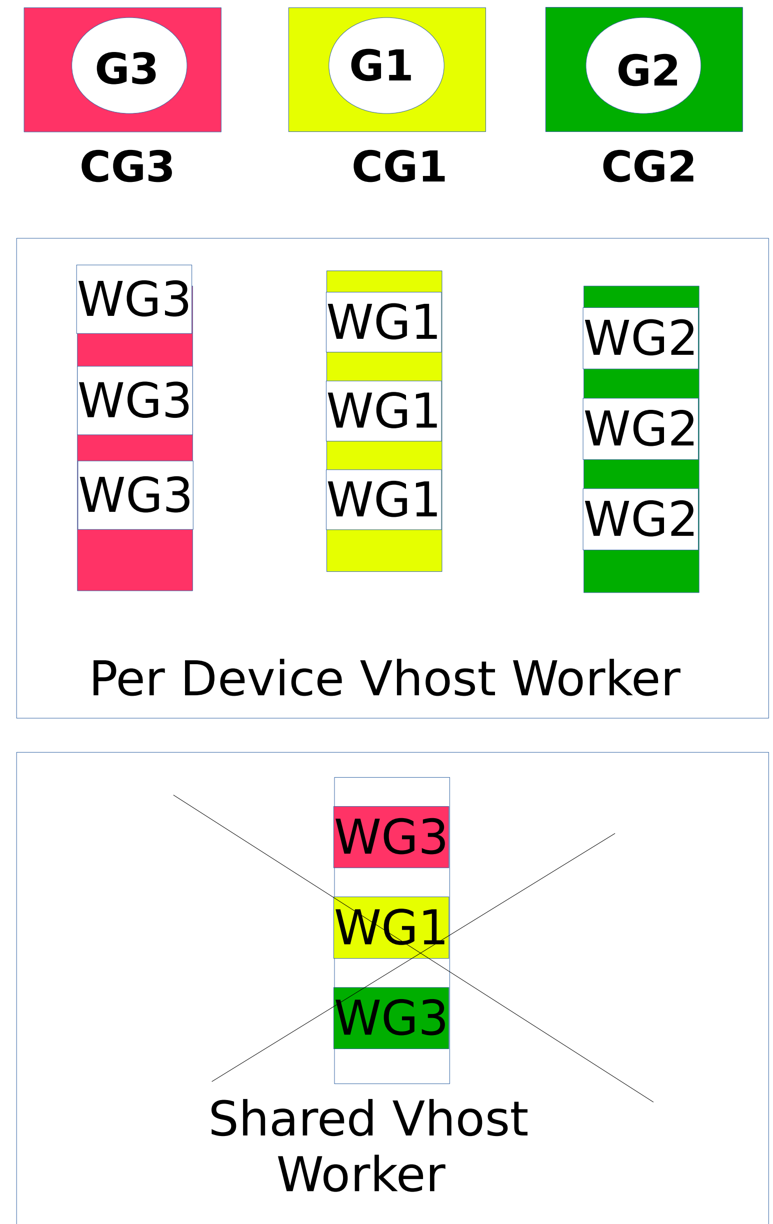
- But..

  - What happens when a VM process is migrated to a different cgroup ?

  - Can we optimize the cgroup search ?

  - What happens if use polling?

  - Rethink cgroups integration ?



Cgroups and polling

- Can a vhost polling thread poll guests with missmatching cgoups?

  - Yes, but it will require the polling thread to take into account cgroup state of the guest.

- Probably requires a deeper integration of vhost and cgroups

Workqueues (cmwq) (Even more sharing!)

- Can we use concurrency managed workqueues ?
- NUMA awareness comes free!
- But wait, what about cgroups ?

  - No cgroups support (at least yet, WIP)
- Less code to manage, less bugs.
- Cons-

  - Minimal control once work enters the workqueue

  - Again, no cgroups support :(

Results

- ELVIS results

  - A little old but significant

  - Includes testing for Exit Less Interrupts, Polling

    - Valuable data for future work

- Setup

  - Linux Kernel 3.1

  - IBM System x3550 M4, two 8-cores sockets of Intel Xeon E5-2660, 2.2 GHz, 56GB RAM

  and with an Intel x520 dual port 10Gbps

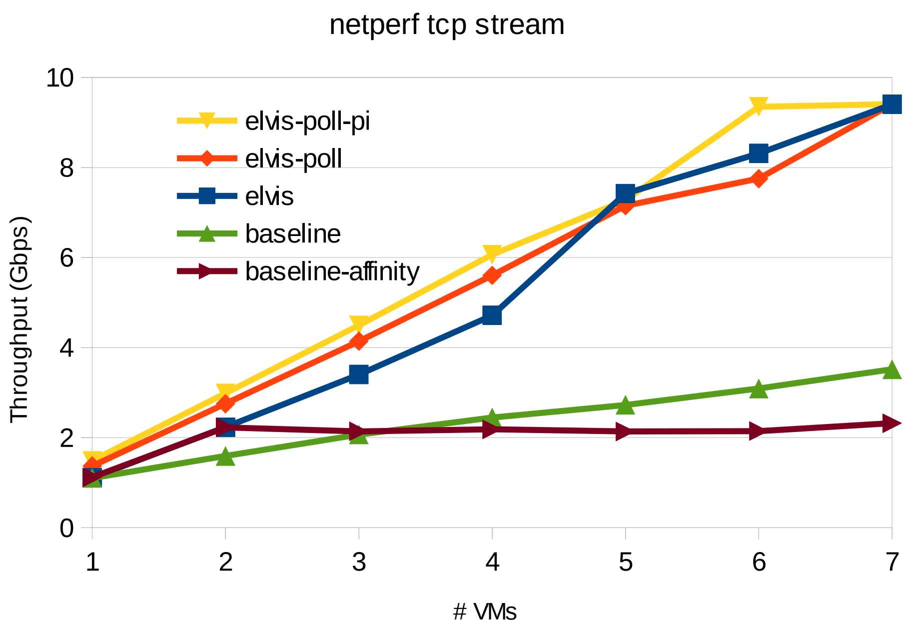
  - QEMU 0.14

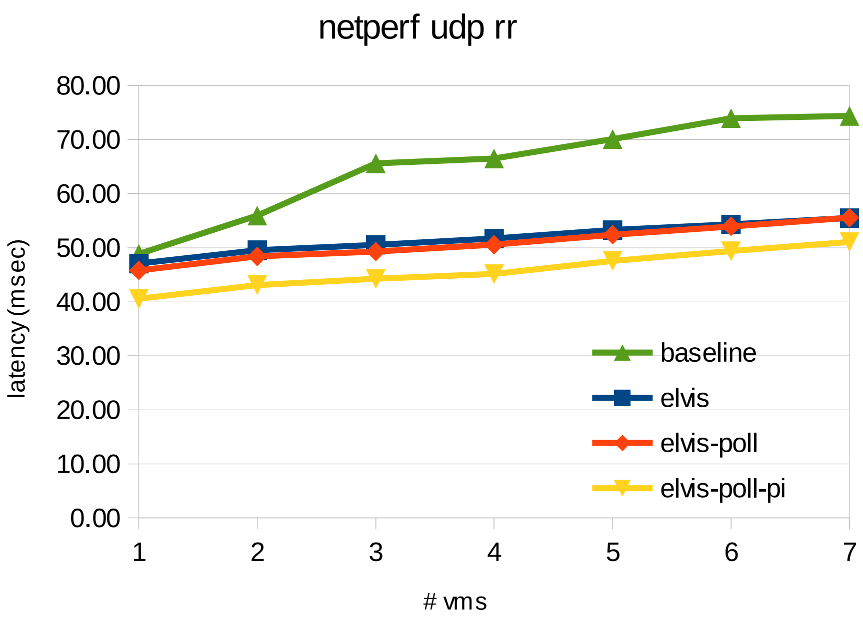
- Results showing the performance impact of the different components of ELVIS

  - Throughput: Netperf TCP stream w. 64 byte messages

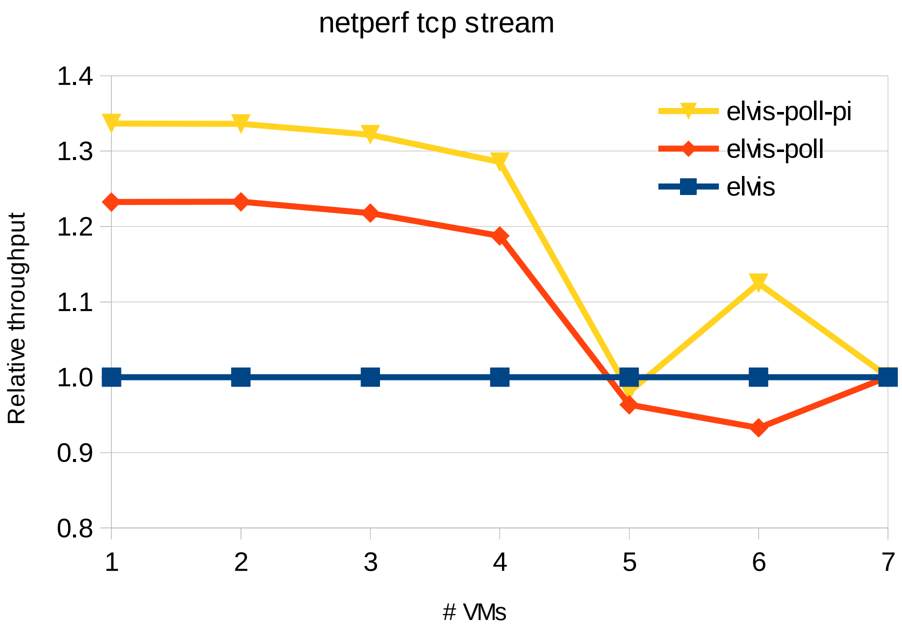
  - Latency: Netperf UDP RR

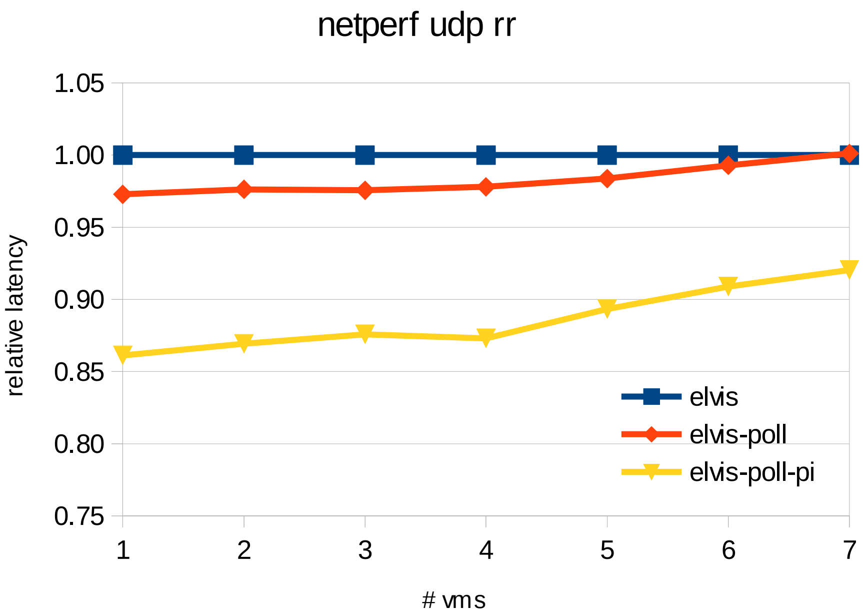
Results – Performance (Netperf)





Results – Components of ELVIS

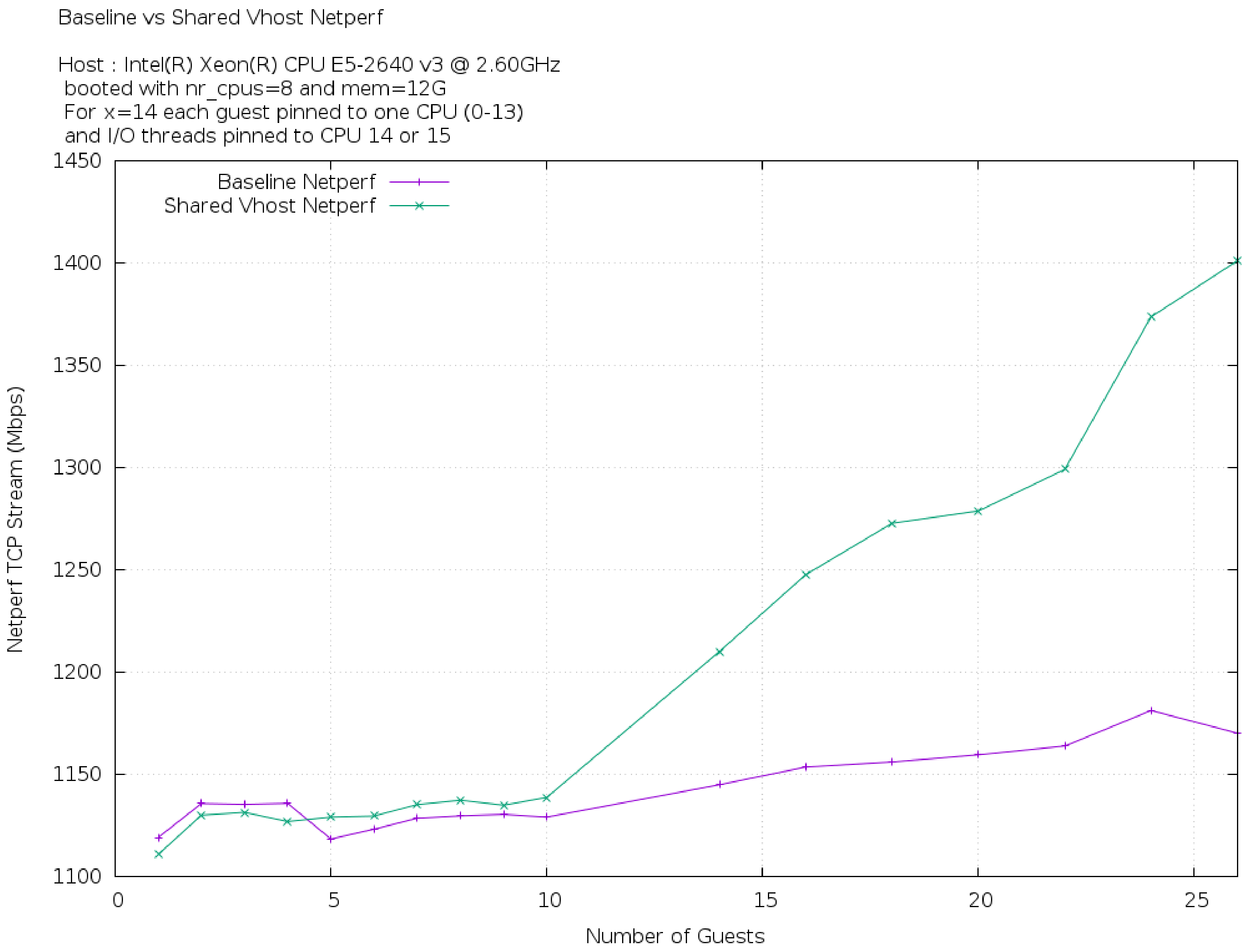




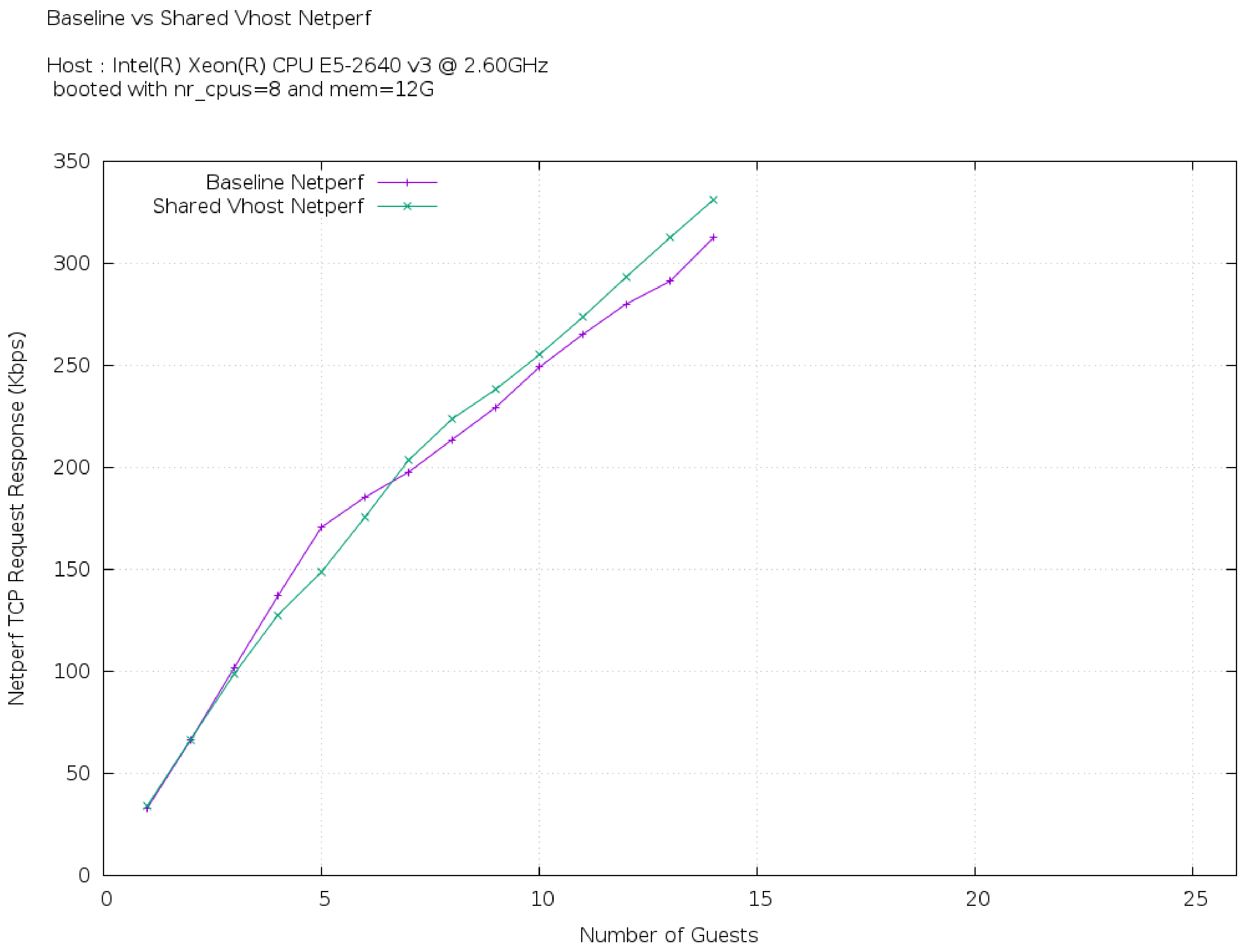
Even more Results

- New results with RFC patches
    - Two systems with Xeon E5-2640 v3
    - Point to point network connection
    - Netperf TCP throughput (STREAM & MAERTS)
    - Netperf TCP Request Response

Results


Baseline vs Shared Vhost Netperf
Host : Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
booted with nr_cpus=8 and mem=12G
For x=14 each guest pinned to one CPU (0-13)
and I/O threads pinned to CPU 14 or 15
Baseline Netperf
Shared Vhost Netperf
Netperf TCP Stream (Mbps)
Number of Guests
1450
1400
1350
1300
1250
1200
1150
1100
0
5
10
15
20
25

Results


Baseline vs Shared Vhost Netperf
Host : Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
booted with nr_cpus=8 and mem=12G
Baseline Netperf
Shared Vhost Netperf
Netperf TCP Request Response (Kbps)
Number of Guests
0
50
100
150
200
250
300
350
0
5
10
15
20
25

So, ship it ?!

- Not yet :)
- Slowly making progress towards a acceptable solution
- Scope for a lot of interesting work

Questions/Comments/Suggestions ?

Backup

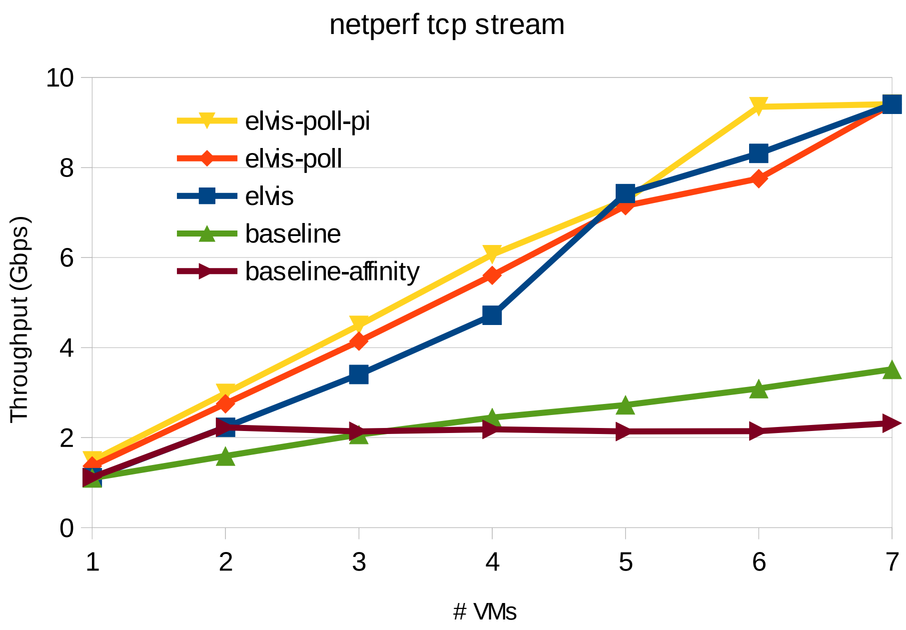
ELVIS missing piece

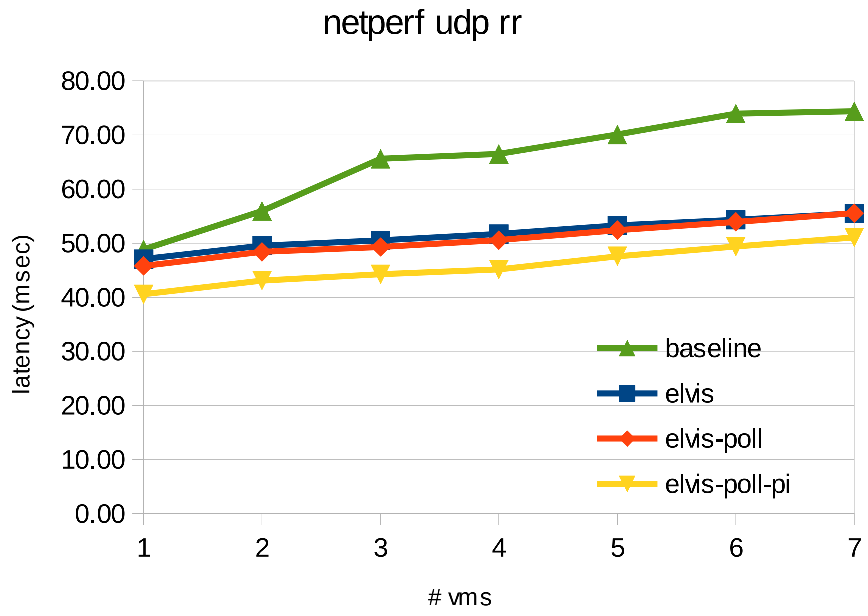
- Polling on the physical NIC

  - It may be possible to use low-latency Ethernet device polling introduced in kernel 3.11

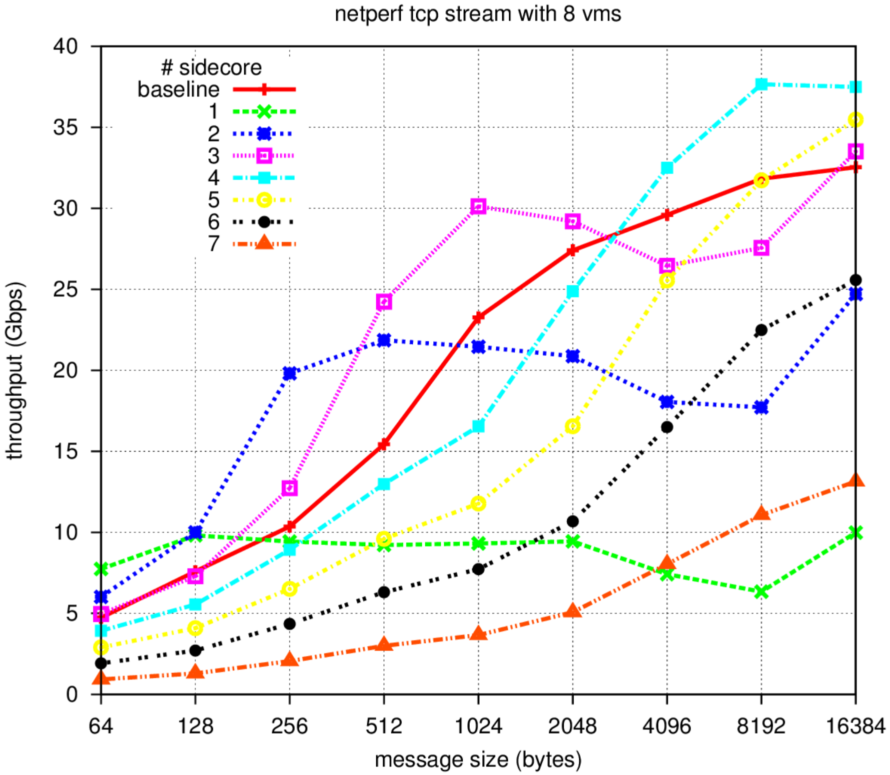
    * I have an ELVIS version polling the physical NIC that is not using this patch

Results – Performance (Netperf)



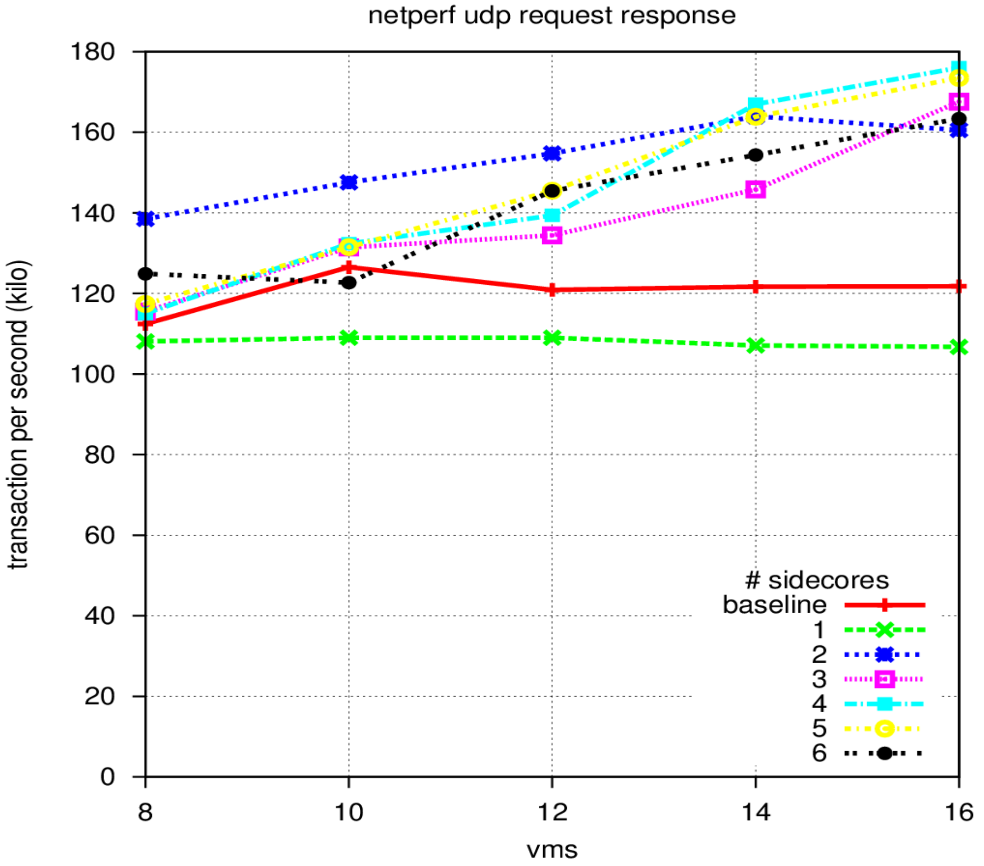


Results – Performance (Netperf)



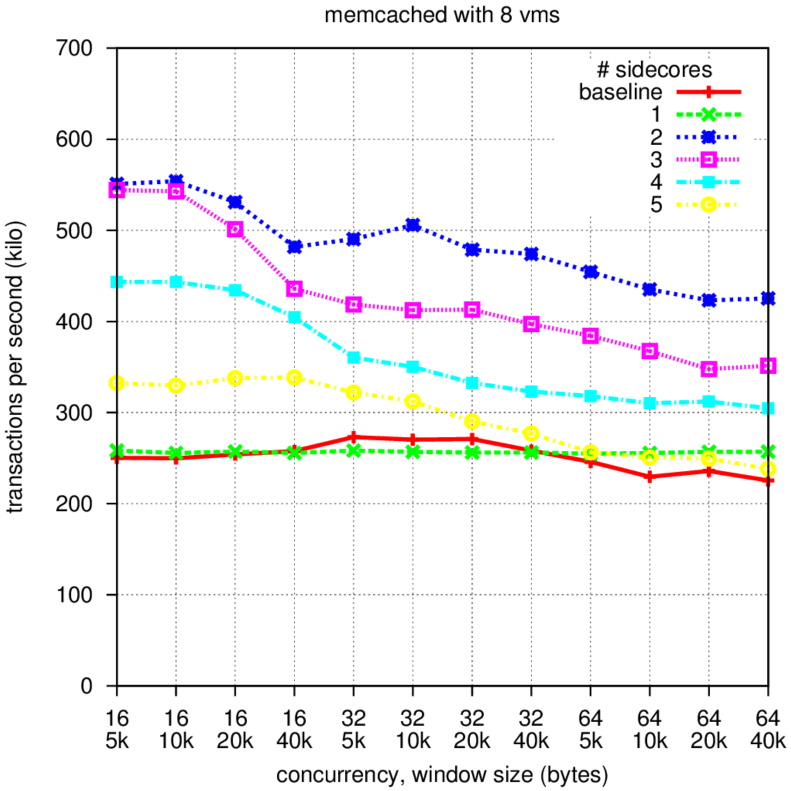
- Different message sizes require different number of IO cores
- Using sidecores is beneficial in a wide range of message sizes
- The number of VMs “doesn't matter” for throughput

Results – Performance (Netperf TCP RR)



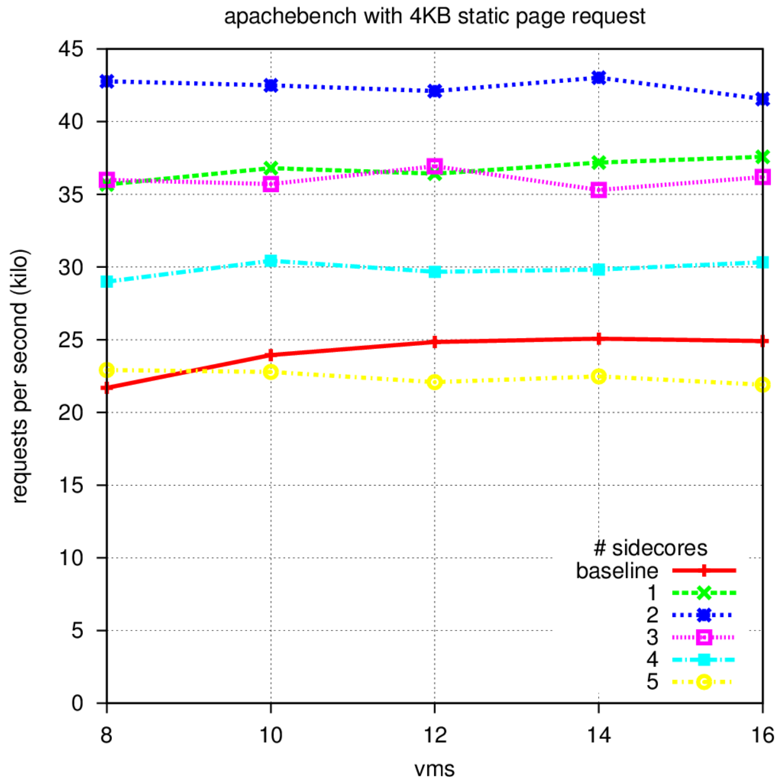
- One I/O side core is not enough, two is needed
- sidecore performs up to x1.5 better then Baseline

Results – Performance
(memcached)



- One I/O side core is not enough, two is needed
- sidecore performs up to > x2 better then Baseline

Results – Performance
(apachebench)



- One I/O side core is not enough, two is needed
- sidecore performs up to x2 better then Baseline