KVM Tuning @ eBay

Li, Chengyuan / Jiang, Xu / Li, Simon / Murthy, Ashok

August 2013

Agenda

- Project Background

- Overview
  - KVM Architecture and ESX Architecture
  - Performance Tuning Methodology
  - Benchmark Configuration
  - Profiling and Tracing Tools

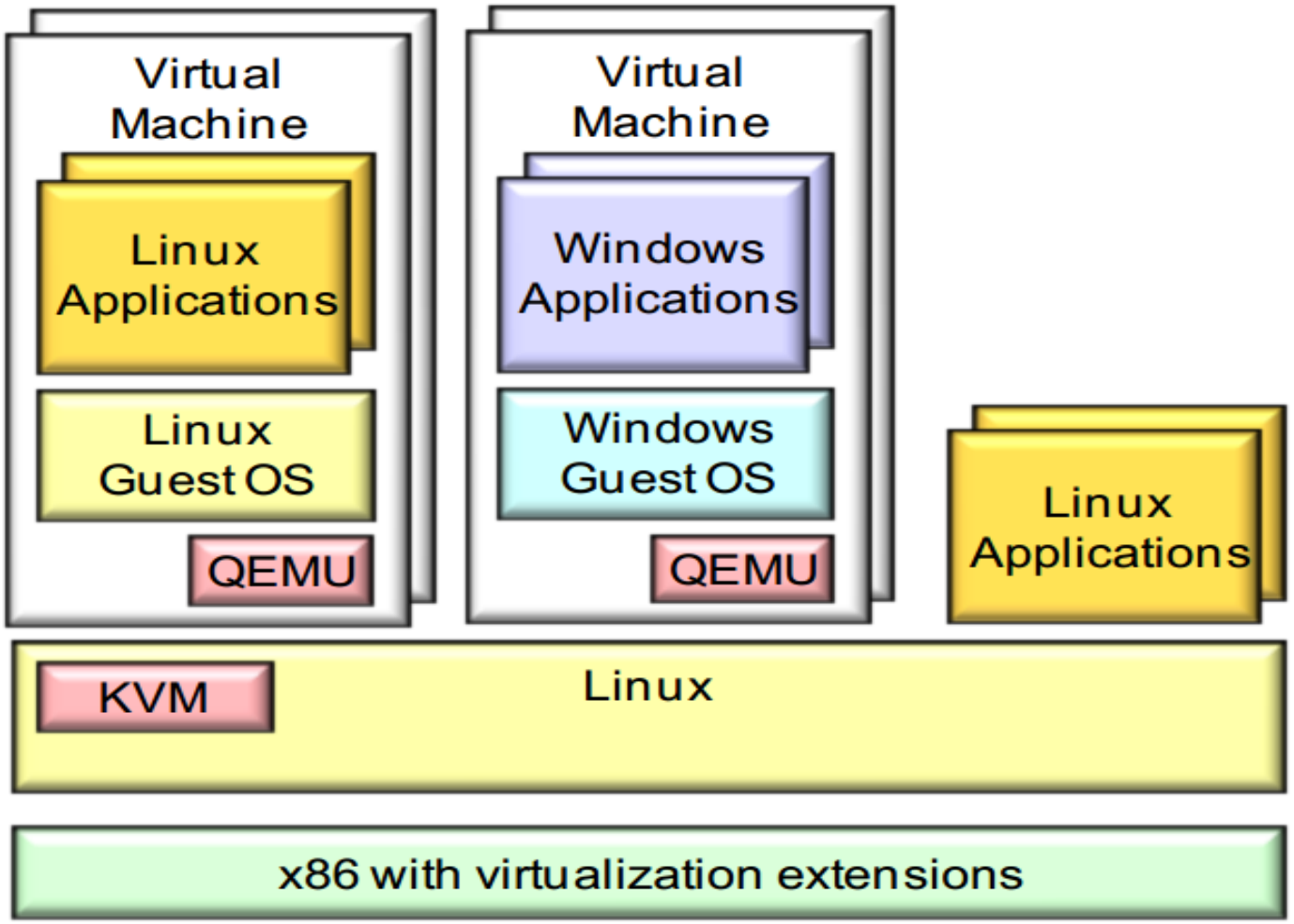
- KVM Tuning
  - KVM Performance
  - KVM Scalability
  - KVM Over-Subscription.
  - Conclusion

- Case Study.

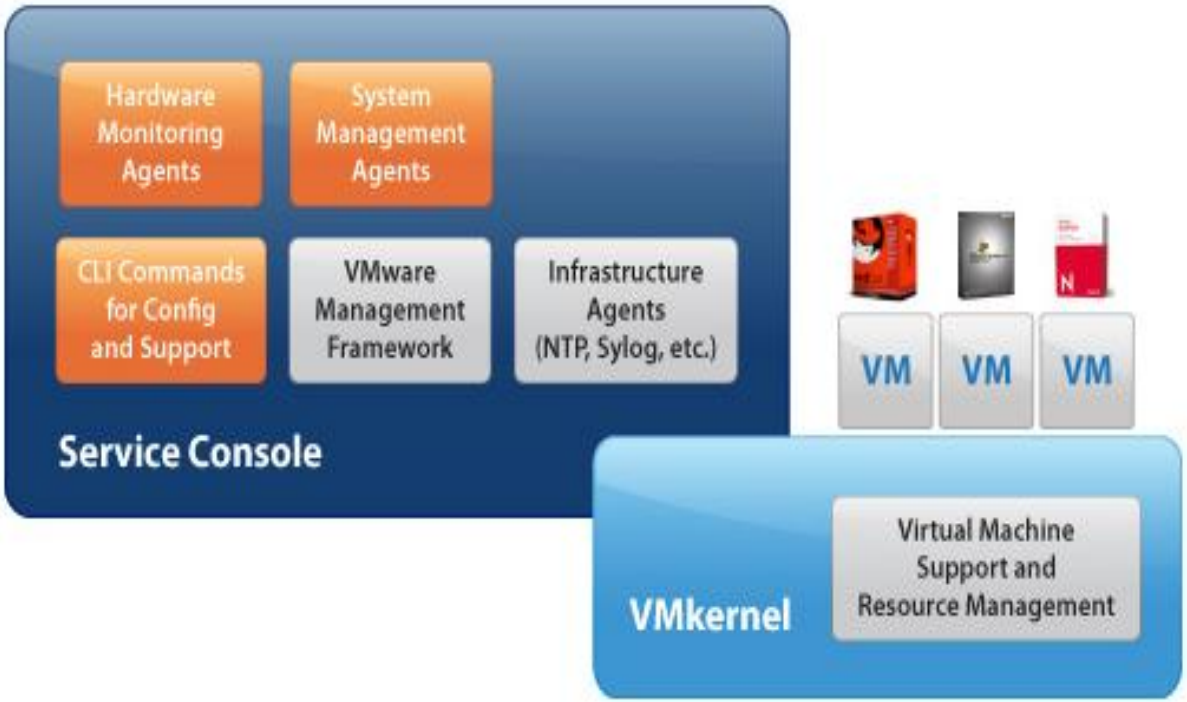
Project Background

- Production environment is moving to OpenStratus, KVM based.
- Find capacity conversion ratio between ESX and KVM.
- Tuning KVM performance, make it same as or better than ESX.

KVM Architecture



ESX Architecture



Performance Tuning Methodology

- Benchmark ***ESX*** as baseline, optimize ***KVM*** to beat ***ESX***

- Micro Benchmark
  - stream (memory)
  - dbench, hdparm (disk)
  - compress-gzip, openssl (cpu)
  - iperf, netperf (network)
  - apache (system)
  - lmbecnch (OS internals)

- Macro Benchmark
  - eBay's frontend application

Benchmark Configuration

- ***Guest VM***
  - *vHW: 4 Cores, 12G Mem, 80G Disk.*
  - *SW: Ubuntu 12.04.2 LTS, kernel 3.2.0-38-virtual,*

- ***KVM BM***
  - *HW (P1G1): HP ProLiant SL170s G6, Intel Xeon X5675 12 Cores/24 Threads 3.07GHz, 72G Mem, 600G Disk*
  - *SW: Ubuntu 12.04 LTS, kernel 3.2.0-24-generic, libvirt-bin 0.9.8-2ubuntu17,*
-
- ***ESX BM***
  - *HW (P1G1): HP ProLiant SL170s G6, Intel Xeon X5675 12 Cores/24 Threads 3.07GHz, 72G Mem, 600G Disk*
  - *SW: VMware ESX 4.1*

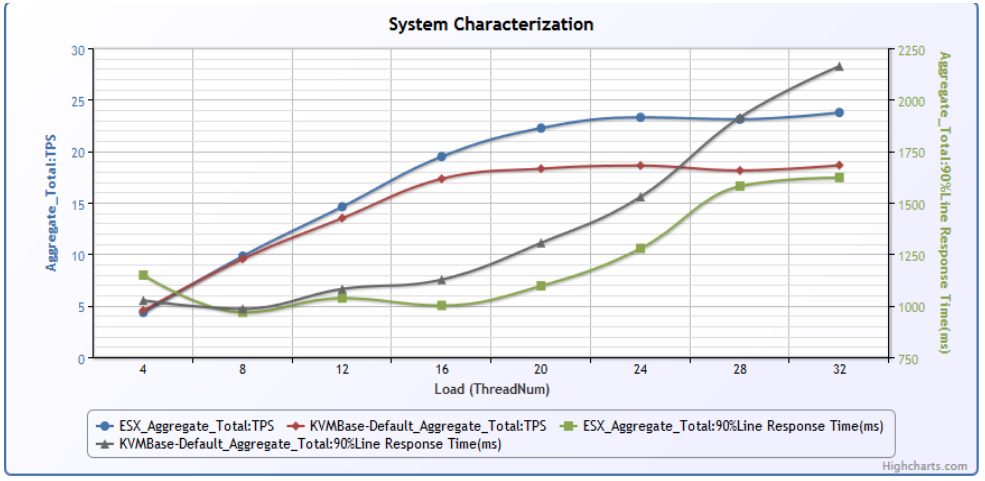
Profiling and Tracing Tools

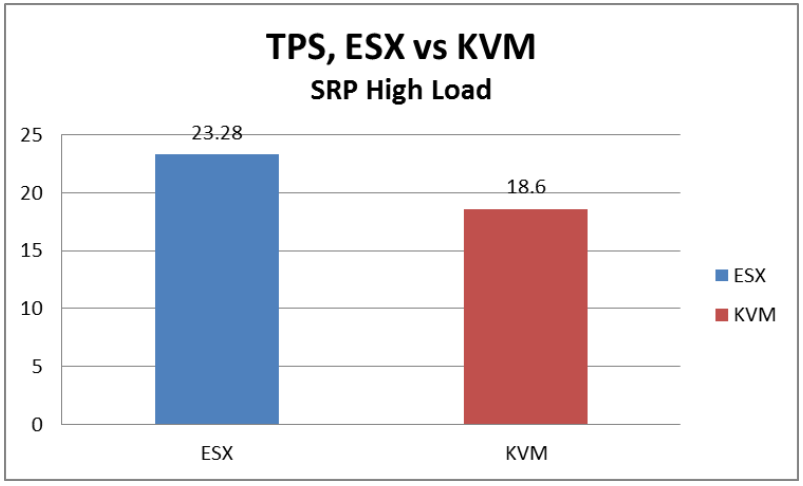
- Profiling and Tracing Tools
  - perf
  - ftrace
  - turbostat/powertop
  - vmstat
  - itop
  - /proc and /sys
  - systemtap
  - JMeter perfmon plugin

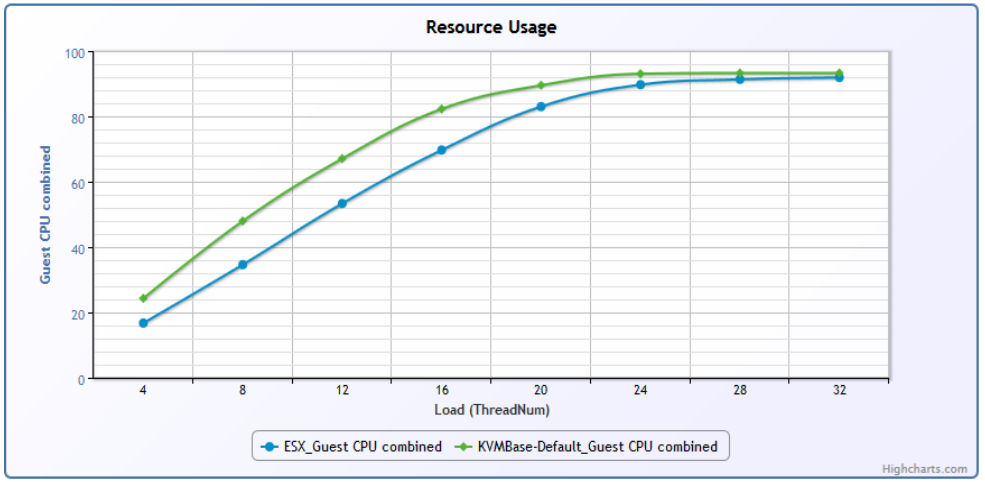
KVM Performance - Baseline

- Baseline, ***KVM*** vs ***ESX***
  - Without tuning,
    KVM performance is poorer than ESX..

	Normal Load (8 clients)			High Load (24 clients)		
	ESX	Base	improvement	ESX	Base	improvement
TPS	9.82	9.54	-2.85%	23.28	18.6	-20.10%
Response Time	967	985	-1.86%	1278	1529	-19.64%
VM Total CPU	34.66	47.99	-38.46%	89.7	93.04	-3.72%



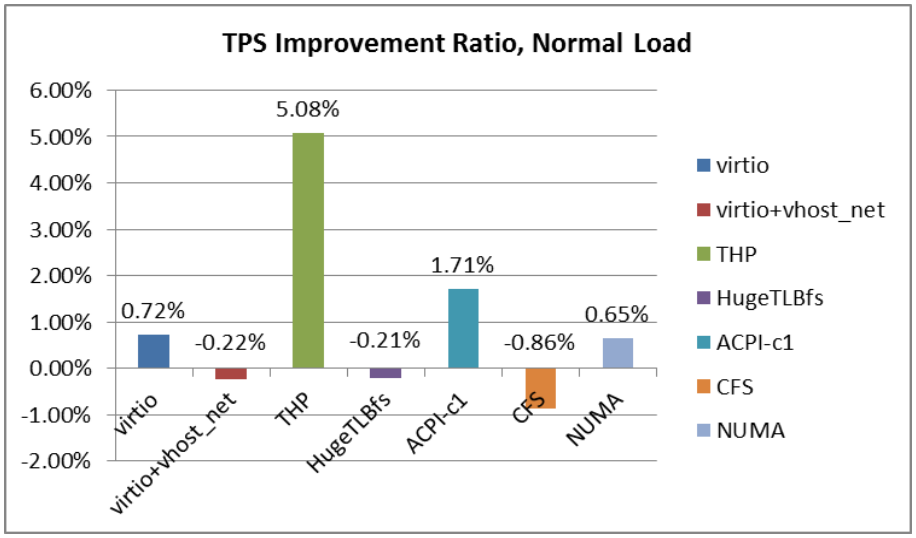




KVM Performance – Optimization List

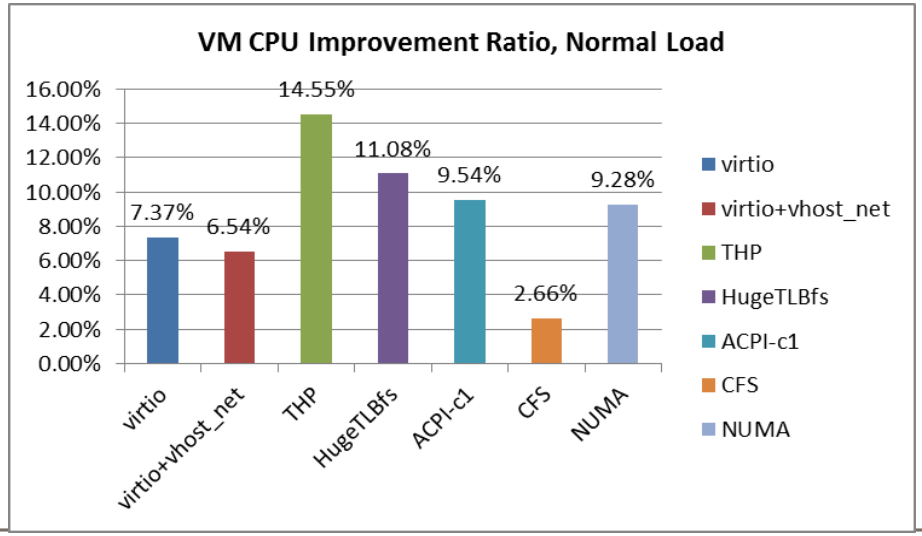
Optimization	Rational
virtio	Para-virtualizaiton driver, reduce vmexit events
virtio+vhost_net	Putting the device backend into host kernel instead of userspace. Reduce context switches.
THP (Transparent Huge Pages)	2MB page size instead of default 4KB page size, reduce pagefaults and TLB misses. Dynamically allocation.
HugeTLBfs	2MB page size instead of default 4KB page size, reduce pagefaults and TLB misses. Memory statically pre-allocation.
ACPI c1 and performance governor	ACPI in c1 (idle) and p0 (active) state. Reduce latency when entering/exiting idle state and CPU is in maximum frequence if it's active.
CFS parameter (Completely Fair Scheduler)	Increase the time slice of process scheduler. Reduce the context switch overhead.
NUMA pinned	Pin each VM's memory and cpu in same NUMA node. To avoid remote memory access overhead.

KVM Performance – Optimizations, Normal Load



***THP** has 5.08% improvement as highest, other tunings almost don't affect TPS in the normal load.*

*The cpu usage decreases in all the tunings except CFS case, **THP** improve the cpu usage most, it's 14.55%.*

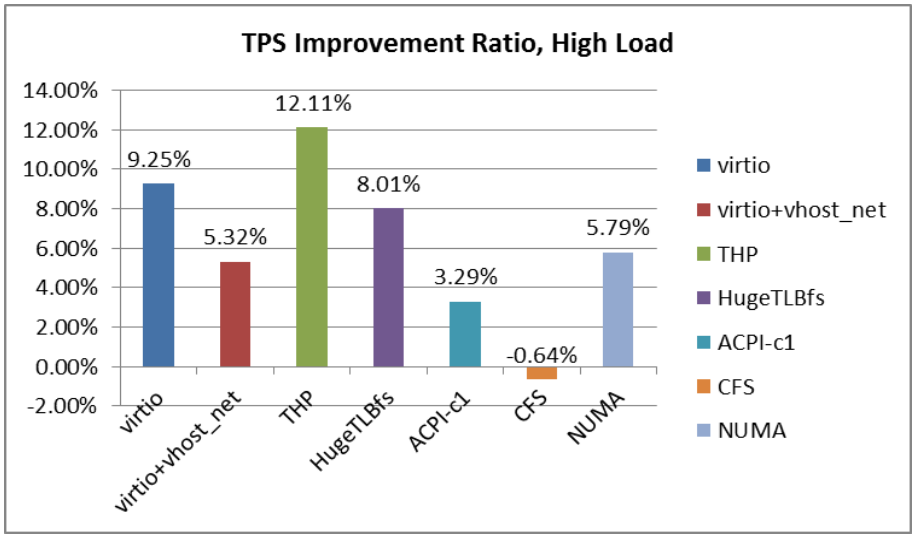


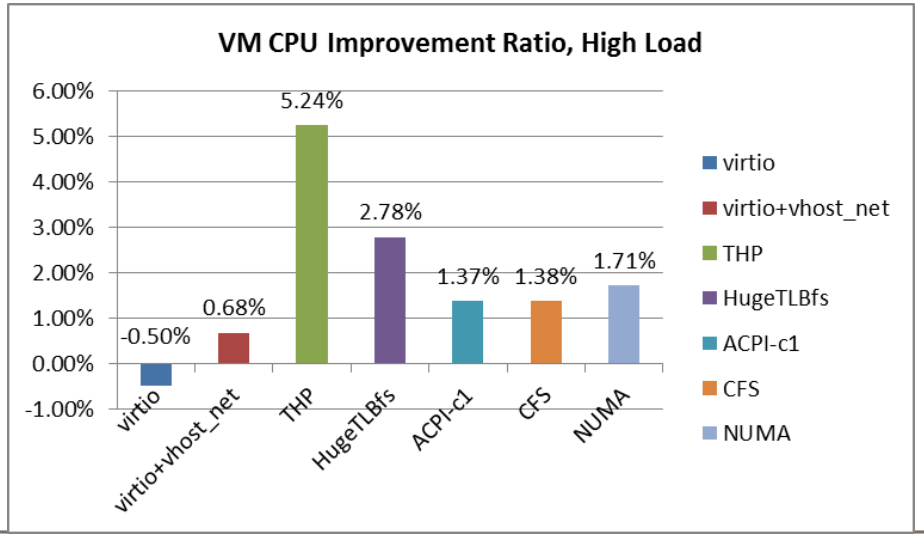
KVM Performance – Optimizations, High Load

Only CFS tuning doesn't improve the TPS,

the other tunings improve the TPS ranging from 3% to 12%,

THP improves the most, and it's 12.11%.





KVM Performance – Optimizations, Selection

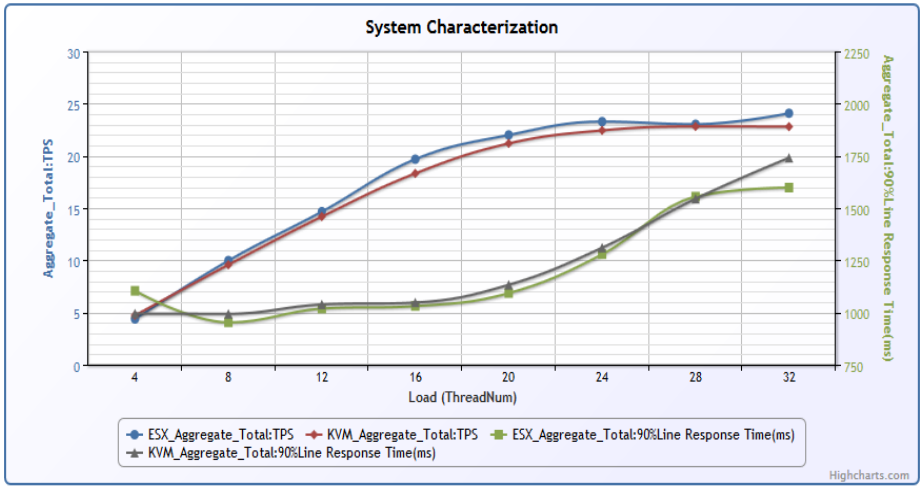
	Performance Improvement	Deployment	Others	Selected
virtio	Medium	Easy	N/A	Yes
virtio+vhost_net	Medium	Easy	N/A	Yes
THP	High	Easy	N/A	Yes
HugeTLBfs	High	Difficult	Can't do over-subscription	No
ACPI c1 and performance governor	Low	Easy	The power consumption may increase	?
CFS parameter	None	Easy	N/A	No
NUMA pinned	Medium	Difficult	N/A	?

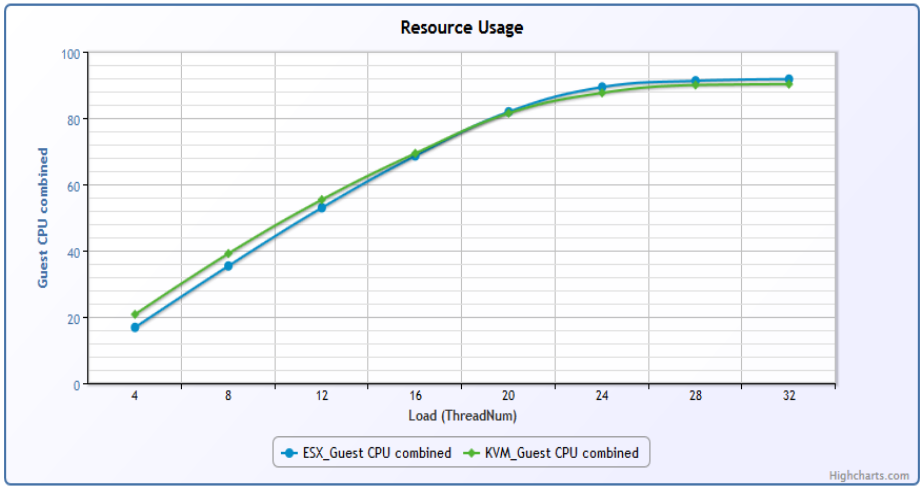
KVM Performance – After Tuning

After tuning, KVM vs ESX

- ***virtio+vhost_net+THP***

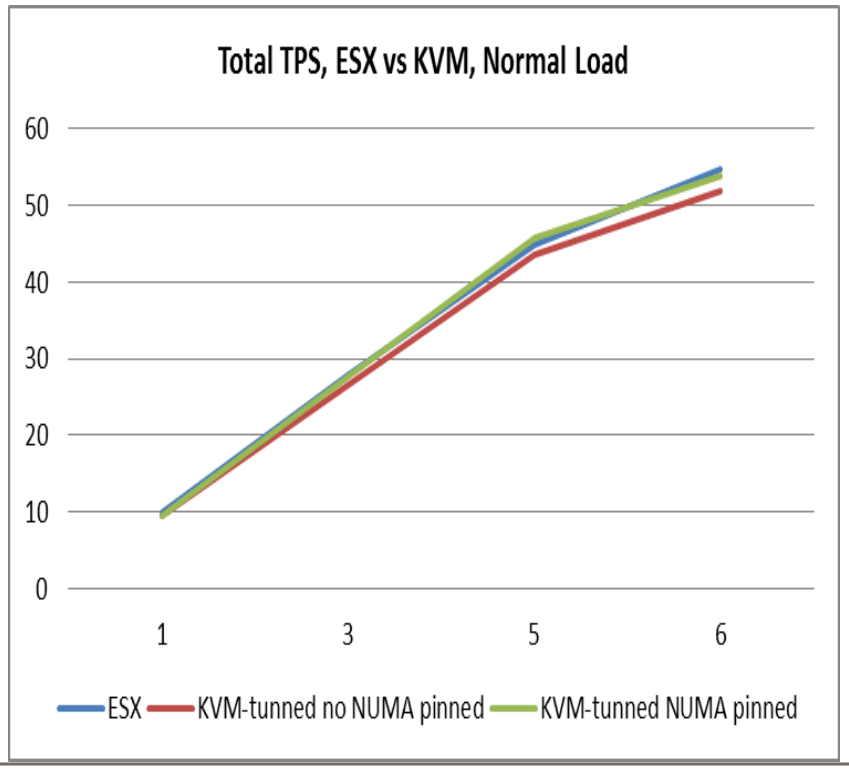
	Normal Load (8 clients)			High Load (24 clients)		
	ESX	virtio + vhost-net+THP	improvement	ESX	virtio + vhost-net+THP	improvement
TPS	9.98	9.58	-4.01%	23.28	22.44	-3.61%
Response Time	953	994	-4.30%	1279	1311	-2.50%
VM Total CPU	35.4	39.13	-10.54%	89.36	87.61	1.96%

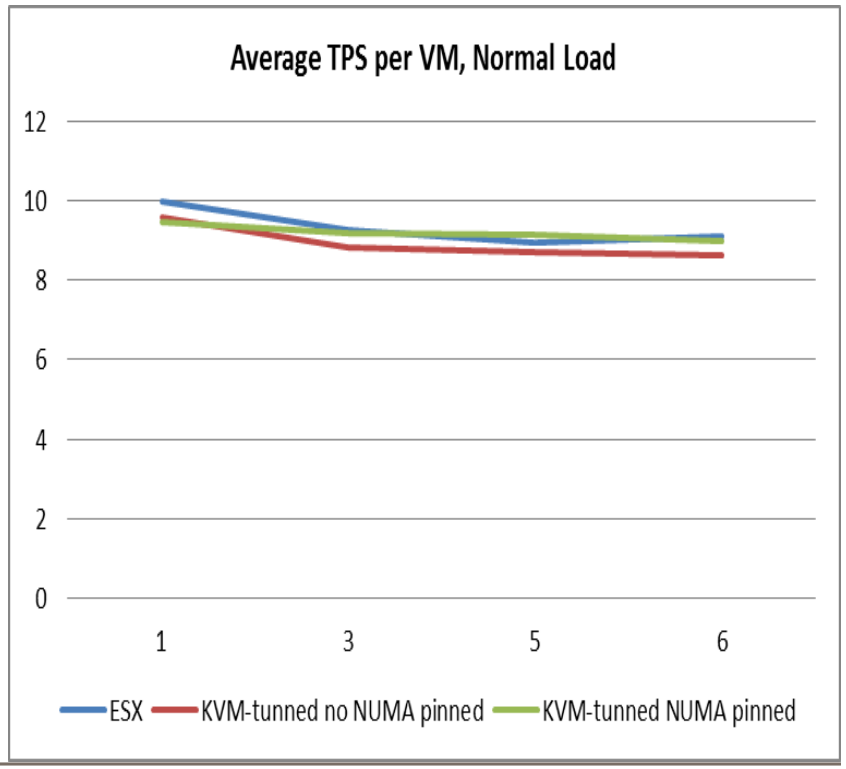




KVM Scalability – Normal Load

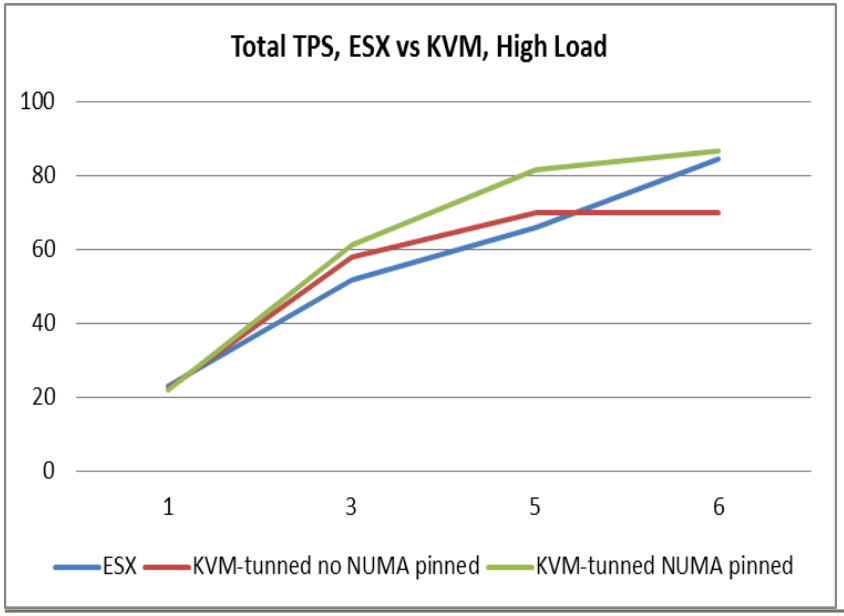
- Total TPS scale well from 1 VM to 6 VMs
- *Average TPS of 6 VMs is 95% of average TPS of 1 VM*

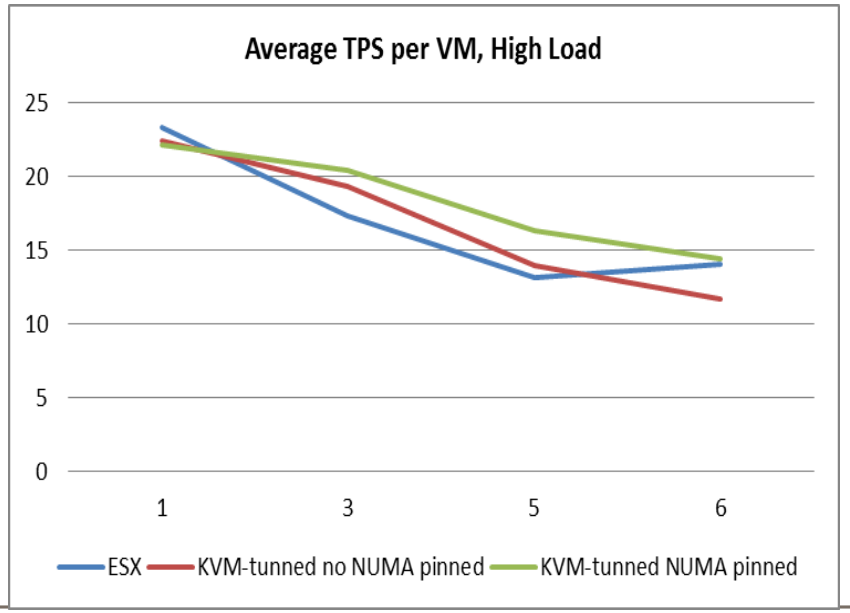




KVM Scalability – High Load

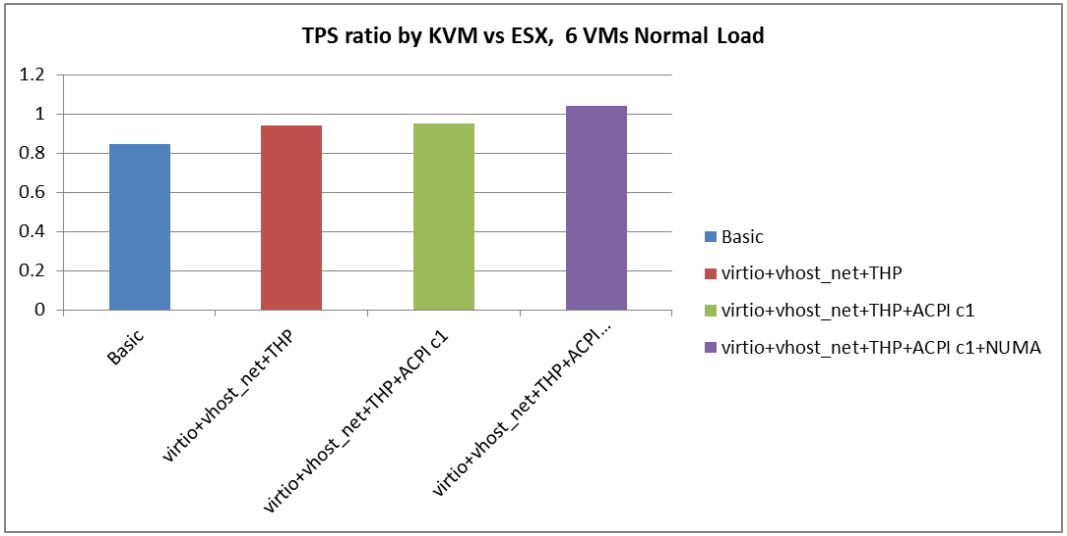
- *Total TPS in KVM scales well from 1 VM to 5 VMs, but it stops increasing from 5 VMs to 6 VMs if no NUMA pinned.*
- After NUMA pinned, ***KVM reaches 86.64 TPS on 6 VMs, while ESX is 84.4 on 6VMs***
- *Average TPS of 6 VMs is 65% of average TPS of 1 VM, both ESX and KVM are the same.*

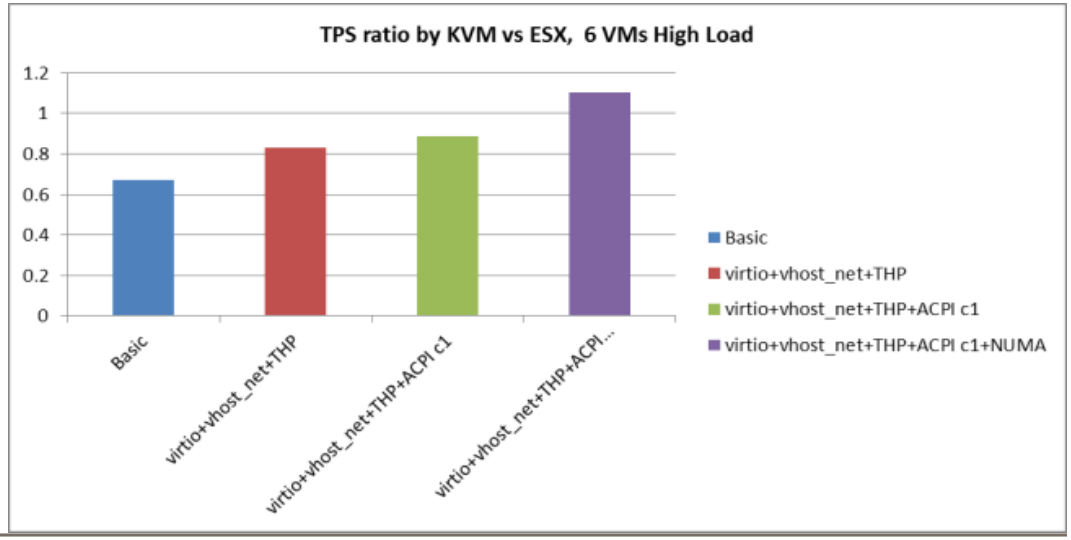




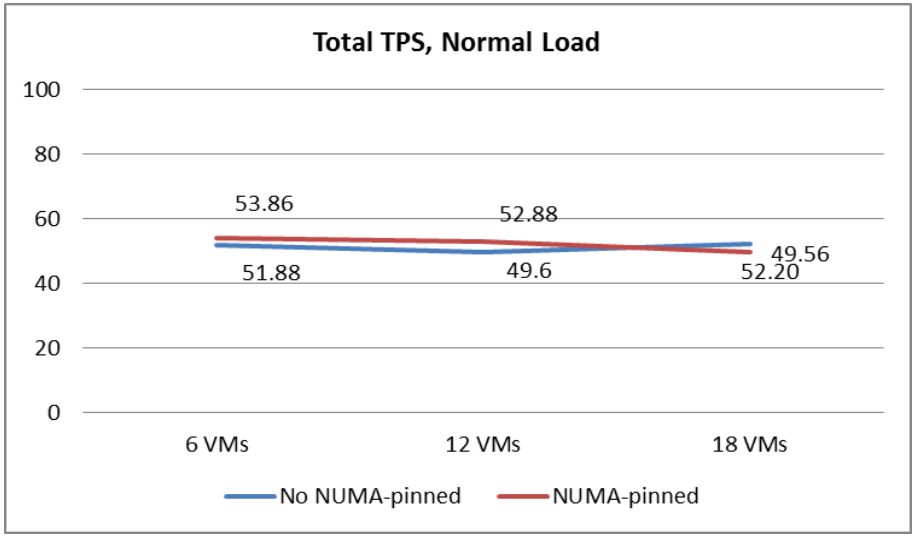
KVM Scalability – Optimizations Comparison

- *Default tunned parameters (**virtio+vhost_net+THP**), improves TPS 23.88% in high load compared to no tunned KVM*
- ***ACPI-c1** improves the performance in high load, compared to default tunned parameters, it's about **6.8%**,*
- ***NUMA pinned** improvement is big in high load, compared to default tunned parameters, it's **32.9%**.*





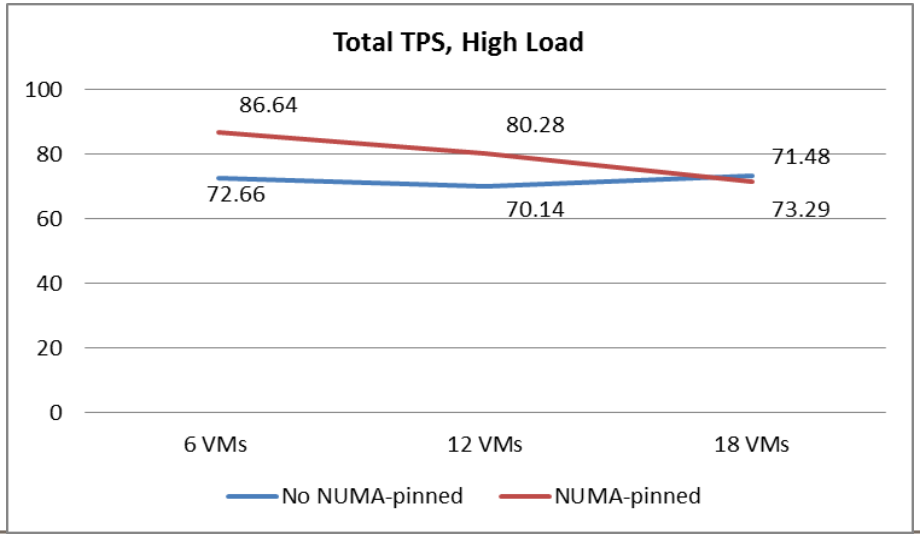
KVM Over-Subscription – Add comments



In the normal load, the total TPS has no much decreasing from 6 VMs to 18 VMs

In the high load,

- *NUMA pinned, decreases from 86.64 to 71.48 linear*
- *No NUMA pinned, no much decreasing from 6 VMs to 18 VMs.*



Conclusion

- Four optimizations for KVM
  - ***virtio + vhost_net (production ready)***
  - ***THP (production ready)***
  - ***ACPI c1 + performance governor (power consuming trade-off)***
  - ***NUMA pinned, both memory and cpu (works for automation)***

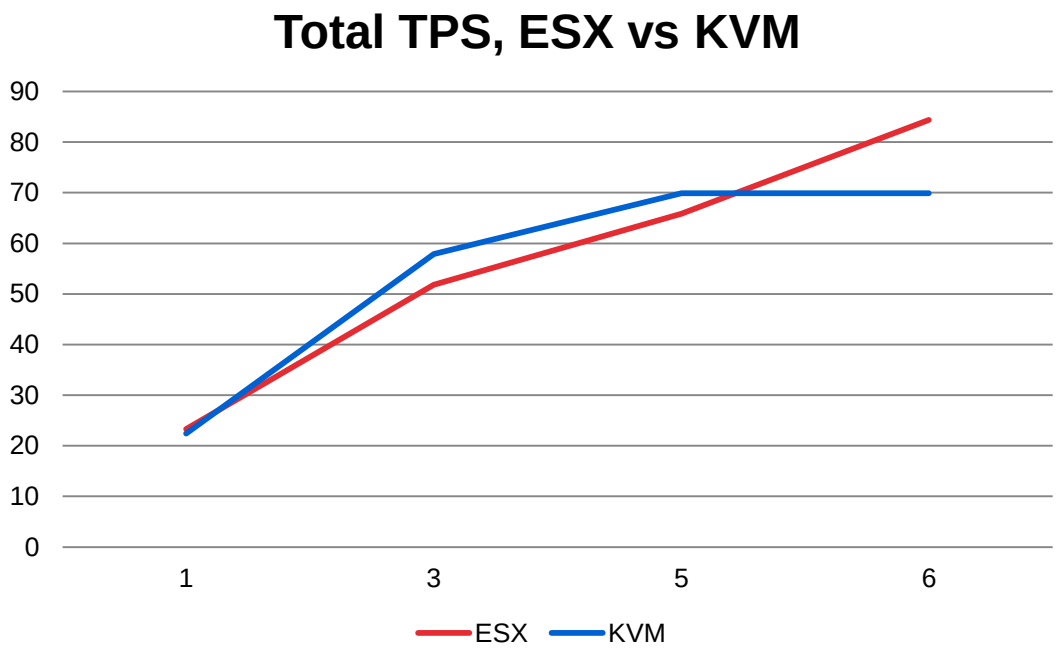
- Interestingly, we find similar optimization in ESX
  - *Paravirtualization vmxnet3 driver in ESX guest.*
  - *Use large page by default*
  - *Always put cpu in C0/C1 and P0 state (i.e. esxtop can show it)*
  - *Use NUMA based scheduler by default*

How to go production?

- ***virtio + vhost_net + THP***
  - These parameters can be easily deployed.
  - 24* 7 stability testing has been done, no issue.
- ***NUMA pinned***
  - Two solutions
    - NUMA aware VM allocation.
      - Changes in the openstack are required.
    - Dynamically NUMA pinned.
      - numad from fedora doesn't work well, e.g. 6 VMs running on two NUMA nodes, numad finally pins 5 VMs on one node, and 1 VM on the other node, not balance!
- ***ACPI c1 + performance governor***
  - Power consumption evaluation is required.
  - ESX enable this feature by default.

Case Study – Scalability Bottleneck

- Why KVM doesn't scale well from 5 VMs to 6 VMs ?



Perf Count Results

5 VMs			6 VMs		
600,674,847,582 cpu-cycles	# 0.000 GHz	[10.62%]	692,718,541,673 cpu-cycles	# 0.000 GHz	[10.85%]
615,503,104,135 stalled-cycles-frontend	# 102.47% frontend cycles idle	[10.49%]	608,402,870,552 stalled-cycles-frontend	# 87.83% frontend cycles idle	[8.44%]
415,977,152,918 stalled-cycles-backend	# 69.25% backend cycles idle	[8.30%]	390,644,519,988 stalled-cycles-backend	# 56.39% backend cycles idle	[9.46%]
261,838,732,463 instructions	# 0.44 insns per cycle		270,342,212,588 instructions	# 0.39 insns per cycle	
6,013,152,007 cache-references		[12.31%]	6,706,244,580 cache-references		[10.23%]
1,245,216,505 cache-misses	# 20.708 % of all cache refs	[12.79%]	1,377,626,391 cache-misses	# 20.542 % of all cache refs	[12.93%]
55,224,401,996 branch-instructions		[13.37%]	56,373,524,752 branch-instructions		[13.63%]
1,563,430,755 branch-misses	# 2.83% of all branches	[13.22%]	1,667,324,339 branch-misses	# 2.96% of all branches	[13.66%]
26,185,450,970 bus-cycles		[10.84%]	29,902,243,624 bus-cycles		[10.92%]
71,994,856,719 L1-dcache-loads		[10.93%]	74,226,974,437 L1-dcache-loads		[10.96%]
5,318,280,044 L1-dcache-load-misses	# 7.39% of all L1-dcache hits	[10.82%]	5,949,543,073 L1-dcache-load-misses	# 8.02% of all L1-dcache hits	[11.00%]
41,802,010,590 L1-dcache-stores		[10.61%]	40,993,696,889 L1-dcache-stores		[10.96%]
550,945,510 L1-dcache-store-misses		[10.47%]	579,368,117 L1-dcache-store-misses		[10.95%]
710,714,939 L1-dcache-prefetches		[10.62%]	938,448,475 L1-dcache-prefetches		[10.84%]
118,317,042 L1-dcache-prefetch-misses		[10.80%]	441,090,493 L1-dcache-prefetch-misses		[10.76%]
128,992,208,310 L1-icache-loads		[11.09%]	138,105,761,040 L1-icache-loads		[10.74%]
13,200,826,782 L1-icache-load-misses	# 10.23% of all L1-icache hits	[10.64%]	13,994,515,317 L1-icache-load-misses	# 10.13% of all L1-icache hits	[10.76%]
<not supported> L1-icache-prefetches			<not supported> L1-icache-prefetches		
<not supported> L1-icache-prefetch-misses			<not supported> L1-icache-prefetch-misses		
2,217,951,113 LLC-loads		[9.74%]	2,580,211,389 LLC-loads		[9.57%]
559,980,796 LLC-load-misses	# 25.25% of all LL-cache hits	[8.77%]	640,175,891 LLC-load-misses	# 24.81% of all LL-cache hits	[8.70%]
1,087,229,989 LLC-stores		[4.61%]	1,121,217,327 LLC-stores		[4.58%]
344,202,583 LLC-store-misses		[4.59%]	347,490,170 LLC-store-misses		[4.65%]
15,943,704 LLC-prefetches		[4.95%]	21,710,308 LLC-prefetches		[4.54%]
3,999,115 LLC-prefetch-misses		[4.68%]	1,698,376 LLC-prefetch-misses		[4.49%]
70,170,631,112 dTLB-loads		[7.65%]	76,523,572,586 dTLB-loads		[8.63%]
154,300,600 dTLB-load-misses	# 0.22% of all dTLB cache hits	[10.19%]	177,636,788 dTLB-load-misses	# 0.23% of all dTLB cache hits	[10.74%]
40,463,258,043 dTLB-stores		[10.77%]	42,187,863,788 dTLB-stores		[10.70%]
66,203,222 dTLB-store-misses		[10.37%]	74,800,488 dTLB-store-misses		[10.72%]
<not supported> dTLB-prefetches			<not supported> dTLB-prefetches		
<not supported> dTLB-prefetch-misses			<not supported> dTLB-prefetch-misses		
268,700,185,261 iTLB-loads		[10.52%]	273,306,970,448 iTLB-loads		[10.75%]
93,299,244 iTLB-load-misses	# 0.03% of all iTLB cache hits	[10.99%]	111,550,779 iTLB-load-misses	# 0.04% of all iTLB cache hits	[10.76%]
53,840,783,134 branch-loads		[10.91%]	56,981,506,698 branch-loads		[10.78%]
36,328,117,191 branch-load-misses		[11.04%]	39,122,505,017 branch-load-misses		[10.73%]
530,502,769 node-loads		[10.20%]	649,138,677 node-loads		[9.51%]
259,772,734 node-load-misses		[9.06%]	305,209,399 node-load-misses		[8.61%]
314,771,743 node-stores		[4.96%]	358,859,789 node-stores		[7.11%]
165,928,138 node-store-misses		[4.78%]	183,918,526 node-store-misses		[5.98%]
14,317,705 node-prefetches		[6.83%]	19,340,103 node-prefetches		[4.61%]
6,929,845 node-prefetch-misses		[6.48%]	6,727,434 node-prefetch-misses		[4.66%]

Root Cause?

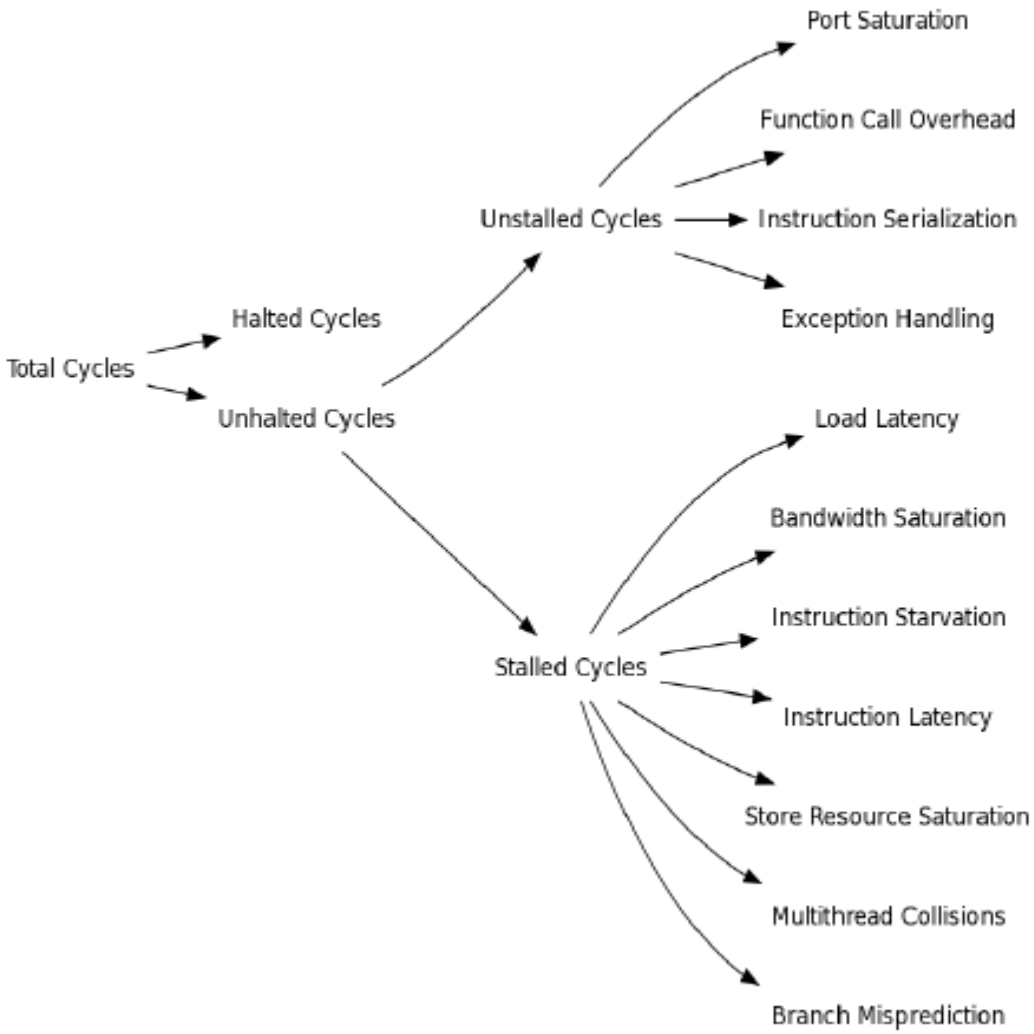
From 5 VMs to 6VMs

- ***CPU** usage increases from 82% to 97%, but **TPS** only increases from 70 to 72.*

- ***What's CPU usage? What's the TPS?***
  - *CPU = Cycles (600B -> 690B)*
  - *CPU <> Instruction*
  - *Instruction == TPS (261B -> 2760B)*

- *The IPC(Instructions Per Cycle) reduces from 0.44 to 0.39, so the **instruction latency increases about 10%***

Cycle Accounting



Memory Bottleneck?

- ***Cycle Accounting***
  - *pipeline execution (same for 5 VM and 6 VM)*
  - *data access (L1/L2/LLC/Memory)*

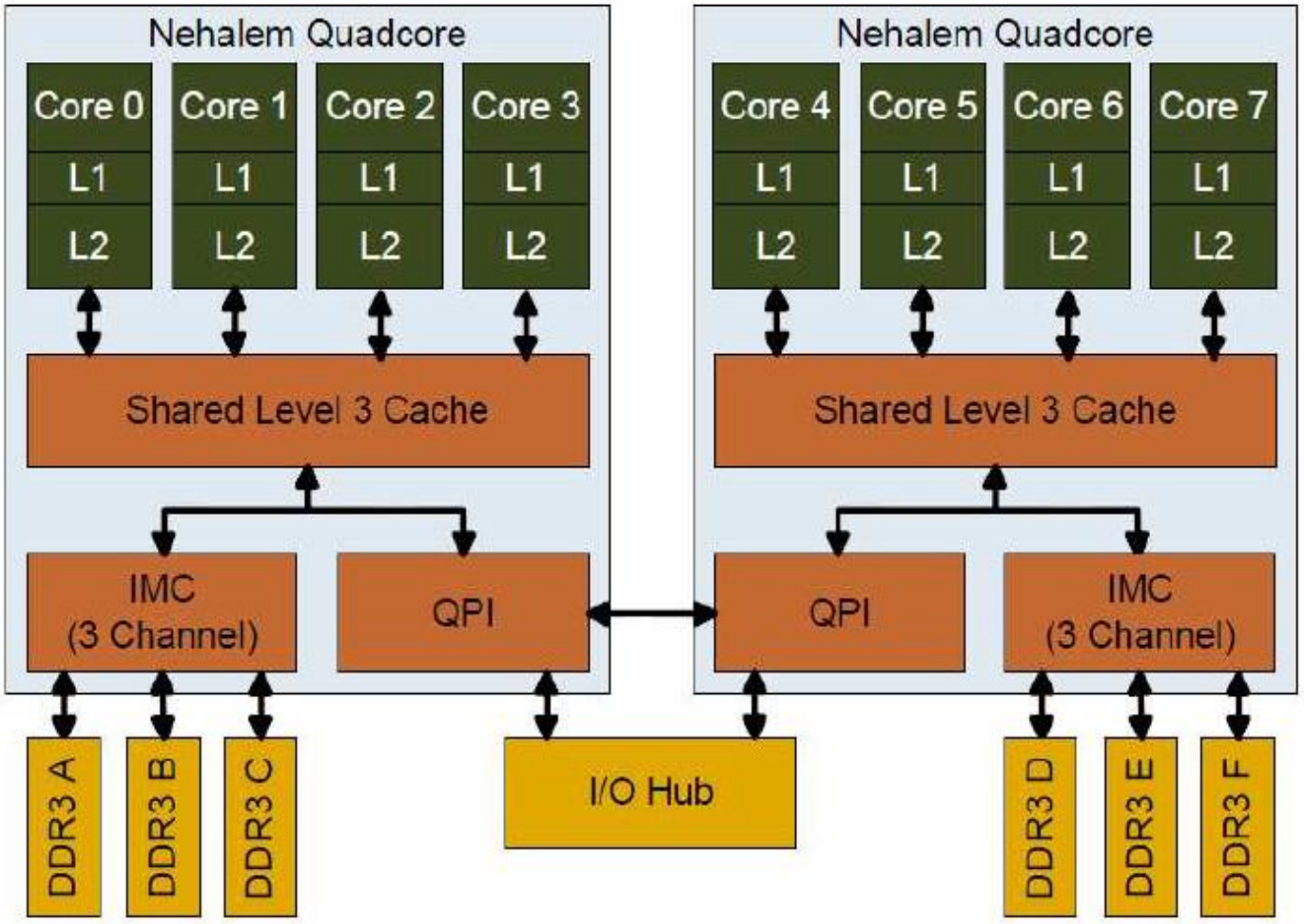
- *Instruction branch misses ratio is almost same.*

- *Cache misses ratio and TLB misses ratio are almost same.*

- *The LLC (Last Level Cache) misses show memory accesses are huge.*

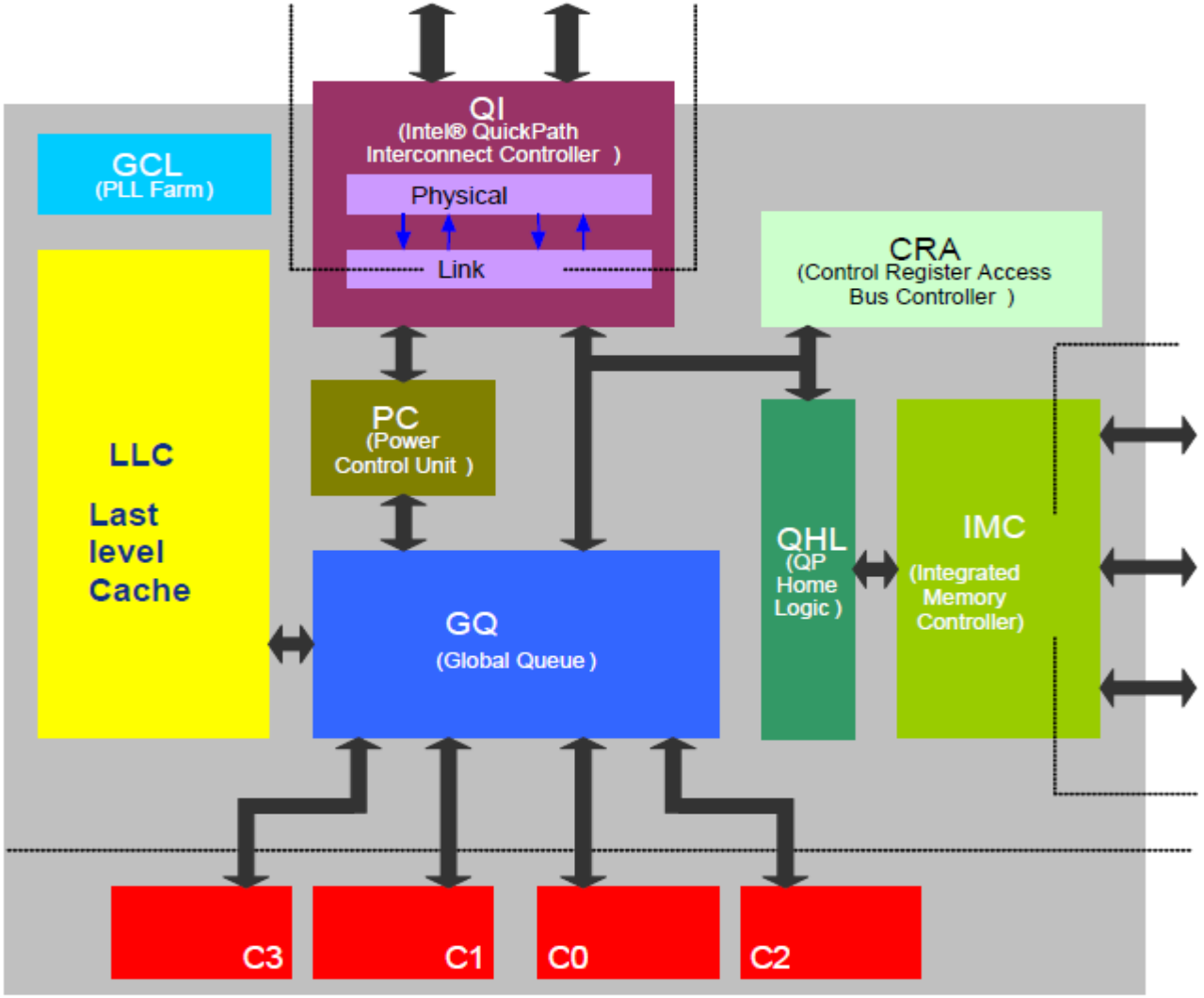
NUMA

- NUMA ?
  - Local Dram ~60 ns
  - Remote Dram ~100 ns



NUMA

- One NUMA socket internal



NUMA Pinned

- Solution_1: Memory and CPU pinned when application starts.

```
#pin to numa node 0
echo 0 > /sys/fs/cgroup/cpuset/libvirt/qemu/cpuset.mems
echo '0-5,12-17' > /sys/fs/cgroup/cpuset/libvirt/qemu/cpuset.cpus
```

- Solution_2: Do memory and CPU migration after application starts.

```
echo 1 > /sys/fs/cgroup/cpuset/libvirt/qemu/instance-00000001/cpuset.memory_migrate
echo 1 > /sys/fs/cgroup/cpuset/libvirt/qemu/instance-00000001/cpuset.mems
echo '6-11,18-23' > /sys/fs/cgroup/cpuset/libvirt/qemu/instance-00000001/cpuset.cpus
```

NUMA Pin Issue – Can't pin memory on Node 1

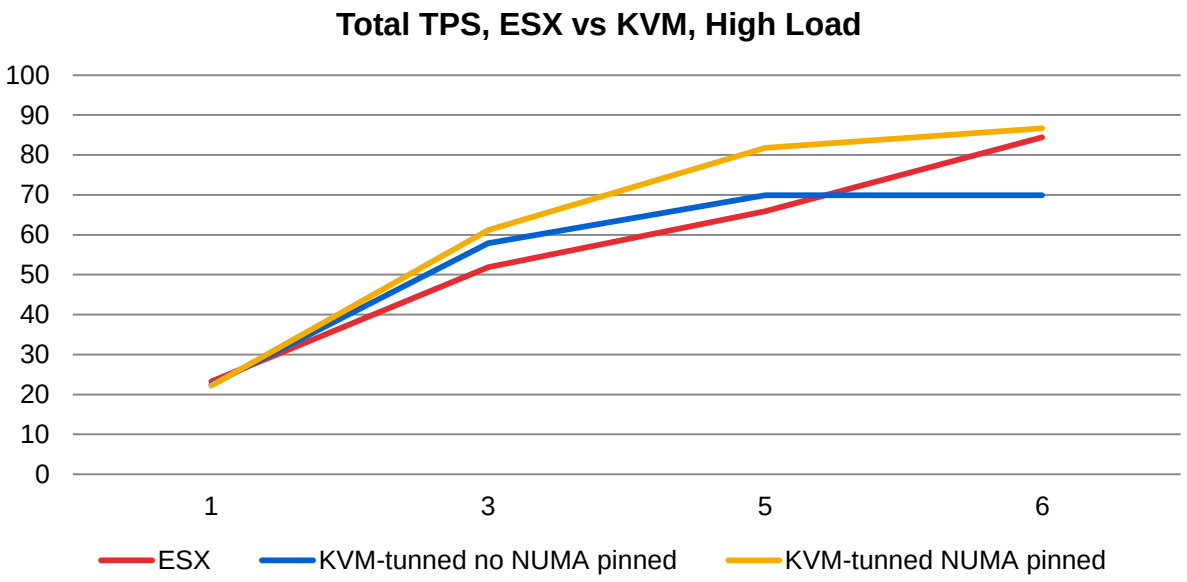
– Static pin Node 1 fails when VM starts.

```
static int alloc_mmu_pages(struct kvm_vcpu *vcpu)
{
	struct page *page;
	int i;

	ASSERT(vcpu);

	/*
	 * When emulating 32-bit mode, cr3 is only 32 bits even on x86_64.
	 * Therefore we need to allocate shadow page tables in the first
	 * 4GB of memory, which happens to fit the DMA32 zone.
	 */
	page = alloc_page(GFP_KERNEL | __GFP_DMA32);
	if (!page)
		return -ENOMEM;

	vcpu->arch.mmu.pae_root = page_address(page);
	for (i = 0; i < 4; ++i)
		vcpu->arch.mmu.pae_root[i] = INVALID_PAGE;

	return 0;
}
```

```
<...>-10082 [019] .... 3844269.833127: kvm_vm_ioctl <-do_vfs_ioctl
<...>-10082 [019] .... 3844269.833130: kvm_vm_ioctl_create_vcpu <-kvm_vm_ioctl
<...>-10082 [019] .... 3844269.833130: kvm_arch_vcpu_create <-kvm_vm_ioctl_create_vcpu
<...>-10082 [019] .... 3844269.833136: kvm_vcpu_init <-vmx_create_vcpu
<...>-10082 [019] .... 3844269.833137: kvm_async_pf_vcpu_init <-kvm_vcpu_init
<...>-10082 [019] .... 3844269.833140: kvm_arch_vcpu_init <-kvm_vcpu_init
<...>-10082 [019] .... 3844269.833142: kvm_set_tsc_khz <-kvm_arch_vcpu_init
<...>-10082 [019] .... 3844269.833142: kvm_get_time_scale <-kvm_set_tsc_khz
<...>-10082 [019] .... 3844269.833144: kvm_mmu_create <-kvm_arch_vcpu_init
<...>-10079 [003] .... 3844269.834109: kvm_vm_release <-__fput
<...>-10079 [003] .... 3844269.834110: kvm_irqfd_release <-kvm_vm_release
<...>-10079 [003] .... 3844269.834112: kvm_put_kvm <-kvm_vm_release
<...>-10079 [003] .... 3844269.834112: kvm_destroy_vm <-kvm_put_kvm
```

```
[    0.000000] Zone ranges:
[    0.000000]   DMA      [mem 0x00010000-0x00ffffff]
[    0.000000]   DMA32    [mem 0x01000000-0xffffffff]
[    0.000000]   Normal   [mem 0x100000000-0x84fffffff]

[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x00010000-0x00098fff]
[    0.000000]   node   0: [mem 0x00100000-0x1febdfff]
[    0.000000]   node   0: [mem 0x20000000-0x2fffbfff]
[    0.000000]   node   0: [mem 0x30000000-0xaa771fff]
[    0.000000]   node   0: [mem 0xac804000-0xad7fffff]
[    0.000000]   node   0: [mem 0x100000000-0x44fffffff]
[    0.000000]   node   1: [mem 0x450000000-0x84fffffff]
```

After NUMA pinned



Intel PMU Comparison before & after tuning

- MEM_UNCORE_RETIRED.REMOTE_DRAM (0x53100f) ratio is 1:56 between pinned and unpinned

```
No Numa-pinned Performance counter stats for 'sleep 10':

   710,300,844,074 cpu-cycles                #    0.000 GHz
   601,007,044,016 stalled-cycles-frontend   #   84.61% frontend cycles idle
   372,803,725,261 stalled-cycles-backend    #   52.49% backend  cycles idle
   296,127,867,513 instructions              #    0.42  insns per cycle
                                             #    2.03  stalled cycles per insn
     6,858,123,583 cache-references
     1,585,636,726 cache-misses              #   23.121 % of all cache refs
    61,376,658,572 branch-instructions
     1,802,040,216 branch-misses             #    2.94% of all branches
    31,271,097,043 bus-cycles
    80,943,596,922 L1-dcache-loads
     5,341,950,160 L1-dcache-load-misses     #    6.60% of all L1-dcache hits
    44,358,111,568 L1-dcache-stores
       608,072,296 L1-dcache-store-misses
       794,026,829 L1-dcache-prefetches
         7,361,351 L1-dcache-prefetch-misses
   145,590,259,259 L1-icache-loads
    15,304,886,658 L1-icache-load-misses     #   10.51% of all L1-icache hits
     2,525,352,250 LLC-loads
       688,604,119 LLC-load-misses           #   27.27% of all LL-cache hits
     1,204,800,833 LLC-stores
       380,492,549 LLC-store-misses
         7,556,897 LLC-prefetches
           663,407 LLC-prefetch-misses
    80,509,262,108 dTLB-loads
       148,526,881 dTLB-load-misses          #    0.18% of all dTLB cache hits
    44,609,701,571 dTLB-stores
        28,337,484 dTLB-store-misses
   300,817,920,244 iTLB-loads
       134,841,532 iTLB-load-misses          #    0.04% of all iTLB cache hits
    63,957,272,046 branch-loads
    43,207,637,719 branch-load-misses
       687,082,109 node-loads
       294,998,432 node-load-misses
       374,095,449 node-stores
       175,030,254 node-store-misses
         6,757,341 node-prefetches
         1,358,813 node-prefetch-misses
        14,861,045 r53040f
       154,346,970 r53100f
         7,371,858 r53020f
         7,164,363 r53020f
       138,124,898 r53080f
        24,407,070 r5308cb
       963,879,223 r5304cb
       396,244,931 r5310cb
                 0 r53100b
    44,581,886,699 r53020b
    79,583,659,011 r53010b

      10.003649078 seconds time elapsed
```

```
NUMA-pinned Performance counter stats for 'sleep 10':

   702,793,990,912 cpu-cycles                #    0.000 GHz
   582,820,983,528 stalled-cycles-frontend   #   82.93% frontend cycles idle
   330,817,312,412 stalled-cycles-backend    #   47.07% backend  cycles idle
   333,332,208,267 instructions              #    0.47  insns per cycle
                                             #    1.75  stalled cycles per insn
     7,793,833,628 cache-references
     1,473,277,512 cache-misses              #   18.903 % of all cache refs
    71,063,584,7875branch-instructions
     2,057,857,907 branch-misses             #    2.90% of all branches
    30,723,034,503 bus-cycles
    94,298,592,838 L1-dcache-loads
     6,237,336,719 L1-dcache-load-misses     #    6.61% of all L1-dcache hits
    52,666,704,926 L1-dcache-stores
       706,950,453 L1-dcache-store-misses
       898,565,700 L1-dcache-prefetches
       112,644,423 L1-dcache-prefetch-misses
   170,698,317,207 L1-icache-loads
    17,242,329,600 L1-icache-load-misses     #   10.10% of all L1-icache hits
     2,962,469,184 LLC-loads
       728,263,821 LLC-load-misses           #   24.58% of all LL-cache hits
     1,356,817,874 LLC-stores
       413,489,944 LLC-store-misses
        13,055,107 LLC-prefetches
         1,802,562 LLC-prefetch-misses
    93,013,196,431 dTLB-loads
       155,778,812 dTLB-load-misses          #    0.17% of all dTLB cache hits
    51,890,194,280 dTLB-stores
        35,113,185 dTLB-store-misses
   341,798,407,437 iTLB-loads
       142,470,920 iTLB-load-misses          #    0.04% of all iTLB cache hits
    72,329,010,899 branch-loads
    49,004,414,433 branch-load-misses
       682,473,220 node-loads
        15,735,233 node-load-misses
       419,633,676 node-stores
         9,708,701 node-store-misses
         4,643,988 node-prefetches
         1,725,835 node-prefetch-misses
         1,120,571 r53040f
         2,757,424 r53100f
        19,035,954 r53020f
        19,723,650 r53020f
       278,483,141 r53080f
        64,287,092 r5308cb
     1,176,521,053 r5304cb
       372,932,212 r5310cb
                 0 r53100b
    50,621,769,118 r53020b
    90,557,129,592 r53010b

      10.001123354 seconds time elapsed
```

Thanks!