Developing cloud infrastructure from scratch: the tale of an ISP

Andrey Korolyov

Performix LLC

22 October 2013

FLOPS

Architecture planning

- Scalable to infinity and beyond
- Fault-tolerant (not just words for a salesman)
- Easy to build...or not so easy
- Cheap commodity hardware
- Avoid modular design because we are not building OpenStack
- Things that are tolerable in the private cloud completely unacceptable in the public one

The beginning: no expertise but lots of enthusiasm

Ceph for storage – just because every other distributed FS was far worse for our needs

Flat networking with L2 filtering using nwfilter mechanism - keep all logic inside libvirt

KVM virtualization because of rich featureset

Monitoring via collectd because many components already has its support

So seems we just need to write UI to build our cloud...

The beginning: after testing all together(mid 2012)

Ceph(Argonaut) has very promising benchmarks but was unusable due to I/O freezes during rebalance
RBD driver has a lot of leaks

Libvirt has no OVS support and a lot of hard-to catch leaks

Eventually first version of orchestrator was launched

VM management

All operations with VMs are passed through one local agent instance to ease task of syncronizing different kind of requests

VMs in cgroups: pinning and accounting
Mechanism for eliminate leak effects via delayed OOM, simular to the [a]
Statistics goes over preprocessing and squeezing on node agent

[a] https://lwn.net/Articles/317814/

VM management, part 2

Guest agent necessary for approximately half of tasks

Emulator upgrades can be done via migration

Automatic migration to meet guaranteed shareable resource needs

Rack-based management granularity

A lot of features: implement, look and throw it out

Pinning with exact match to the host topology

Custom TCP congestion protocol

Memory automatic shrinking/extending based on statistics

Online CPU hot-add/hot-remove

NUMA topology passthrough

Performance versus needs

Most kind of regular workloads does not require strict NUMA tuning for large performance improvements
Using THP is the same, boost is quite low

IDE driver supports discard operations but its bandwidth limited by about 70Mb/s, so virtio is only an option

BTRFS shows awesome benchmark resuls as a Ceph backend(zero-cost snapshot operations and workloads with ton of small operations) but ultimately unstable

Performance versus needs

Most kind of regular workloads does not require strict NUMA tuning for large performance improvements
Using THP is the same, boost is quite low

IDE driver supports discard operations but its bandwidth limited by about 70Mb/s, so virtio is only an option

BTRFS shows awesome benchmark resuls as a Ceph backend(zero-cost snapshot operations and workloads with ton of small operations) but ultimately unstable

So all these things did not went to our production.

Fine tuning

With a lot of VMs CFS scheduler is preferred
Scheduler granularity needs to be tuned to reduce # of context switches

For large VMs XBZRLE shoud be used for migration

Building on top of instabilities(end of 2012)

XFS was quite unstable under CPU-intensive workloads

Ceph daemons leaks in many ways and should be restarted eventually

Nested per-thread cgroups for QEMU may hang the system

OpenVSwitch can be brought down by relatively simple attack

Memory hot-add: ACPI hotplug – buggy but works

Reserved kernel memory takes a lot when high and low limits are different by an order of magnitude, so ballooning can be used only with samepage merging
Regular KSM does not have sufficiently fine tuning and UKSM has a memory allocation issues
None of us was able to fix memory compaction issues, so we had switched to ACPI hotplug driver
Out of tree, requires some additional work to play with [a]

[a] https://github.com/vliaskov/qemu-kvm/

```
<qemu:commandline>
  <qemu:arg value='-dimm'/>
  <qemu:arg value='id=dimm0,size=256M,populated=on'/>
 ...
  <qemu:arg value='id=dimm62,size=256M,populated=off'/>
</qemu:commandline>
```

Ceph

Ceph brings in a lot of awesome improvements in every release, as a Linux itself
And as with Linux some work required to make it rocket fast

Caches at every level
Fast and low-latency links and precise time sync
Rebalancing should be avoided as much as possible
Combine storage and compute roles but avoid resource exhaustion
Fighting with client latencies all the time

Ceph, part 2

Rotating media by itself is not enough to give proper latency ceilings to lot of read requests so large cache is required
DM-Cache is only one solution which is pluggable, reliable and presented in the Linux tree in same time
Cache helped us to put 99% of read latencies behind 10ms dramatically decreasing from 100ms with only HW controller cache

Operation priorities are most important thing
With 10x scale scope of problem priorities was completely changed
A lot of tunables should be changed
The less percentage of data one node holds, the lesser downtime (on peering) will be in a case of failure

VM scalability

CPU and RAM cannot be scaled over one node without RDMA and for very specific NUMA-aware tasks
Passing variable NUMA topology to VM is impossible to date so Very Large VMs are not in place

VM can be easily cloned and new instance with rewritten network settings can be spinned of (under a balancer)

Rebalance for unshareable/exhausted resources can be done using migration, but even live migration is visible for end-user; waiting for RDMA migration to come into mainline

Selecting networking platform

Putting efforts on RSocket for Ceph

Storage network may remain flat

Switch hardware for networking should run opensource firmware
Today a lot of options on the market:
Arista, HP, NEC, Cisco, Juniper, even Mellanox... none of them offers opensource platform
We chose Pica8 platform because it is managed by OVS and can be modified easily

Networking gets an upgrade

In the standalone OVS we need to build GRE mesh or put VLAN on interface to make private networking work properly
Lots of broadcast in the public network segment
Traffic pattern analysis for preventing abuse/spam can be hardly applied to the regular networking

OpenFlow 1.0+ fulfill all those needs, so we made necessary preparations and switched at once

OpenFlow

FloodLight was selected to minimize integration complexity

All OpenFlow logic comes from existing VM-VM and MAC-VM relation

Future extensions like overlay networking can built with very small effort

Still not finished multiple controller syncronization

Software storage and sotware networking

Due to nature of Ceph VMs can be migrated and placed everywhere using snapshot+flatten (and hopefully something easier in future)

Any VMs may be grouped easily
Amount of overlay networking rules need to be built reduced dramatically up to IDC scale
Only filtering and forwarding tasks are remains on rack scale for OF

Current limitations in Linux

No automatic ballooning

No reliable online memory shrinking mechanism outside ballooning

THP are not working with virtual memory overcommit

Online partition resizing came only in the 2012, so almost no distros have support out-of-box for this

Future work

Advanced traffic engineering on top of fat-tree ethernet segment using Pica8 3295 as ToR and 3922 for second level

Extending Ceph experience(RSockets and driver policies improvements)

Autoballooning “zero-failure” mechanism implementation

Any questions?