Lessons in running libvirt at scale

Prerna Saxena

What we do @Nutanix:

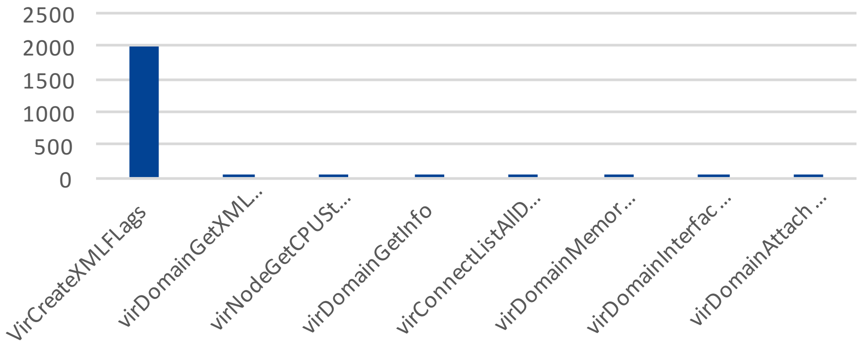
- Self healing, dynamically scaling, auto-balancing clusters.
- Proprietary orchestration layer to work with Libvirt, Qemu, KVM
- Designed for scale: must support 1000 VMs per box and 100s of concurrent ops.
- Near-immediate host failure detection based on libvirt keepalives.
- Automated VM failover in less than a minute.

How do I measure the efficacy of management layer?

- Reliability : Do all requests complete deterministically, without impacting vital functions ?

- Throughput : How many ops can be driven in a given time window.

RPC call stats in an idle environment



Bulk Power ONs:

- During bulk power ops, libvirtd would “lock up” for long intervals.
  - All client communication { RPC / Keepalives } ignored.
- Libvirt liveness tied to Host HA.
- False alarm : Perfectly healthy VMs force-evacuated.

Catastrophe !

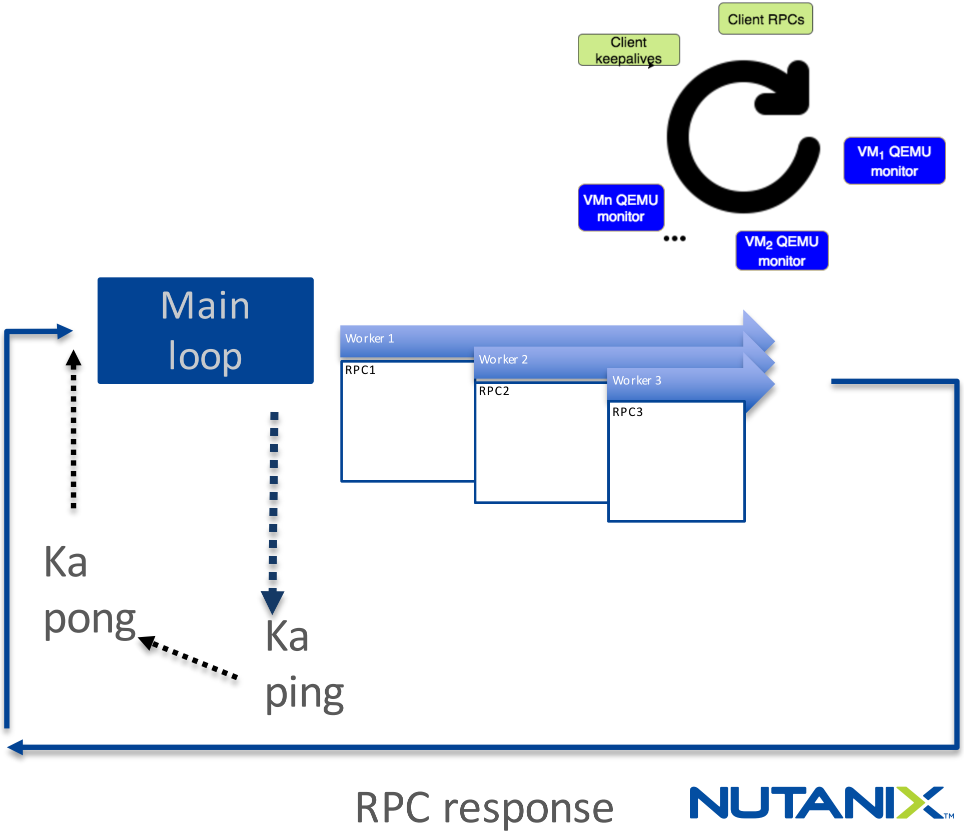
Analysis:

- Keepalives are enqueued right from the main loop, *each time epoll() detects a message could be sent.*[1]

[1] Reported: https://www.redhat.com/archives/libvir-list/2017-May/msg00016.html

Current Threading model (QEMU driver)

- Main epoll() loop listens on all sockets :
  - Client socket(s) for RPCs/ keepalives(Ka)
  - VM Monitor(s) for replies + events + hangup
- Client comunication:
  - Optimized keepalive handling.
  - All RPCs punted to threadpool.



Current threading model (contd)

- VM monitoring:
  - ‘reply’ messages wake up requisite worker thread (no overhead for main thread)
  - Hangups and async QMP events : *handlers run in main thread.*
    - All events need the per-VM lock.
    - Event races with existing RPC → lock contention holds down the main loop!

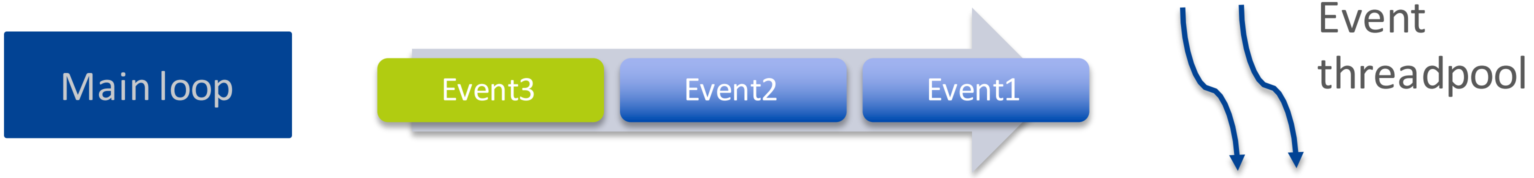
Solution #1:

- Introduce a dedicated thread(pool) in QEMU driver to handle async events.
  - Main thread just “posts” the event to this queue.
  - Events are picked up as they arrive, by a dedicated event handler thread.
  - Can be expanded to a event-threadpool based implementation.
  - Lock contention point moves from main thread to event handler thread.

Solution #1 (contd):

Drawback:

- Event processing can suffer latency.
  - Add more threads, burn more resources?
- Dedicated event worker may still contend with RPC(s)



Solution #2:

- Introduce a dedicated thread(pool) in QEMU driver to handle async events.
  - Main thread just “posts” the event to this queue.
  - Events are picked up by the worker thread(s) as they arrive, leaving the RPC threadpool free to pursue client RPCs.
  - Event threads only pick an event if the respective VM lock is available.
  - Additionally, each RPC worker also “drains” queued event(s) just before giving up the lock.

Limitation:

- Increases time accounted for in RPC context:
  - Other worker threads may time out ?
  - Non-deterministic client behaviour: same RPC may take varying times depending on size of event queue.
- Prioritizing Shutdown/hangup handling:
  - “standard” event worker vs priority worker?
- Blurred lines: When RPCs and events merge.

Returning to assessment mode : Throughput validation

Re-evaluating the current threading model:

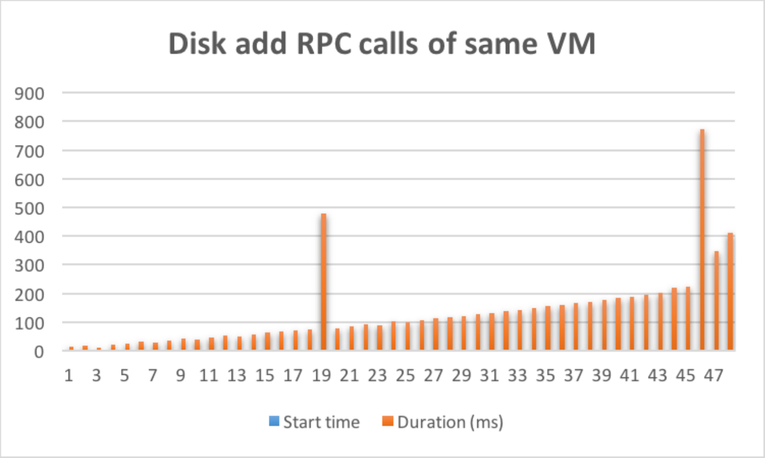
- Worker threads running RPCs for the same VM contend for the same lock
  - Only *one* of all makes forward progress.
  - Unfair delay for non-contending RPCs stuck behind in the queue.

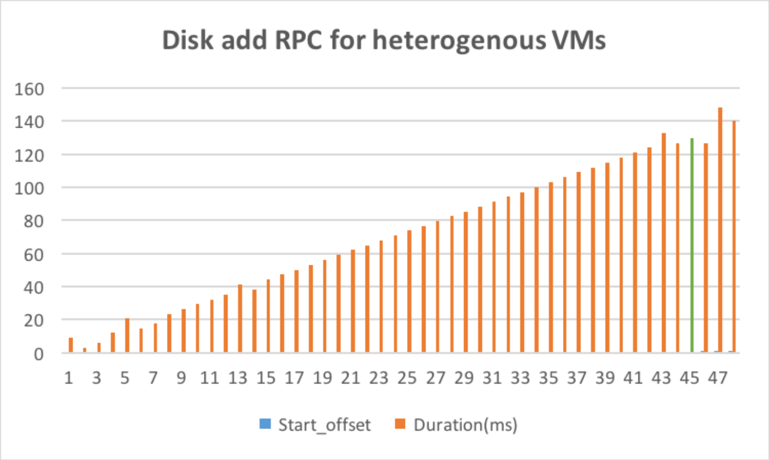
Evaluating libvirt throughput:

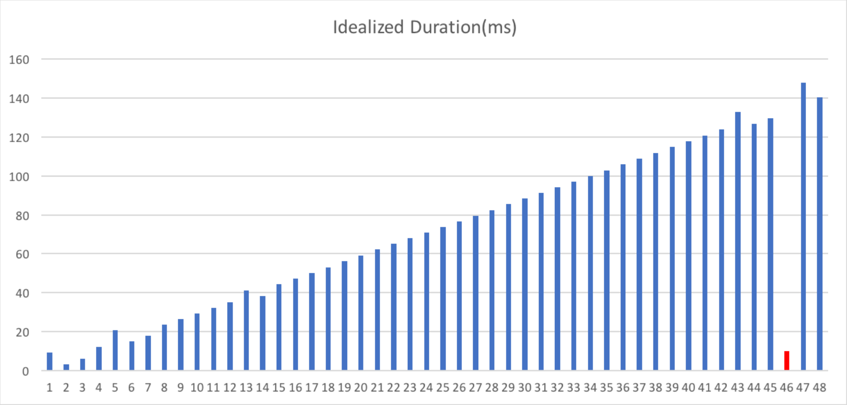
Test setup:

- Libvirtd configured with min, max workers of {2,10}
- Multithreaded C program used to drive >3x RPCs :
  - *for the same VM.*
    - As expected, RPCs exhibit sequential execution.
  - *Behavior with heterogenous VM RPCs*

Results:


Disk add RPC calls of same VM
900
800
700
600
500
400
300
200
100
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
🟦 Start time
🟧 Duration (ms)


Disk add RPC for heterogenous VMs
160
140
120
100
80
60
40
20
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
🟦 Start_offset
🟧 Duration(ms)


Idealized Duration(ms)
160
140
120
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48

Analysis:

- Impact:
  - Adversely hits a multi-tenant environment

Proposed solution :

Can we carve out per-VM queues ?

- Better throughput.
- Easier to bake in event infrastructure.

Other good-to-have(s), while we discuss..

- Libvirt daemon interacts with client in perfect async fashion, using serial numbers to tie requests to responses.
- Client translates API calls into async messages, but sends them using `virNetClientSendWithReply()`
- An application linked to the client has to work with blocking calls ☹
- Might need changes to the current multi-threaded dispatch model..

Questions ??

Legal Statement

This presentation and the accompanying oral commentary may include express and implied forward-looking statements, including but not limited to statements concerning our business plans and objectives, product features and technology that are under development or in process and capabilities of such product features and technology, our plans to introduce product features in future releases, the implementation of our products on additional hardware platforms, strategic partnerships that are in process, product performance, competitive position, industry environment, and potential market opportunities. These forward-looking statements are not historical facts, and instead are based on our current expectations, estimates, opinions and beliefs. The accuracy of such forward-looking statements depends upon future events, and involves risks, uncertainties and other factors beyond our control that may cause these statements to be inaccurate and cause our actual results, performance or achievements to differ materially and adversely from those anticipated or implied by such statements, including, among others: failure to develop, or unexpected difficulties or delays in developing, new product features or technology on a timely or cost-effective basis; delays in or lack of customer or market acceptance of our new product features or technology; the failure of our software to interoperate on different hardware platforms; failure to form, or delays in the formation of, new strategic partnerships and the possibility that we may not receive anticipated results from forming such strategic partnerships; the introduction, or acceleration of adoption of, competing solutions, including public cloud infrastructure; a shift in industry or competitive dynamics or customer demand; and other risks detailed in our Form 10-Q for the fiscal quarter ended April 30, 2017, filed with the Securities and Exchange Commission. These forward- looking statements speak only as of the date of this presentation and, except as required by law, we assume no obligation to update forward- looking statements to reflect actual results or subsequent events or circumstances. Any future product or roadmap information is intended to outline general product directions, and is not a commitment, promise or legal obligation for Nutanix to deliver any material, code, or functionality. This information should not be used when making a purchasing decision. Further, note that Nutanix has made no determination as to if separate fees will be charged for any future product enhancements or functionality which may ultimately be made available. Nutanix may, in its own discretion, choose to charge separate fees for the delivery of any product enhancements or functionality which are ultimately made available.Certain information contained in this presentation and the accompanying oral commentary may relate to or be based on studies, publications, surveys and other data obtained from third-party sources and our own internal estimates and research. While we believe these third-party studies, publications, surveys and other data are reliable as of the date of this presentation, they have not independently verified, and we make no representation as to the adequacy, fairness, accuracy, or completeness of any information obtained from third-party sources.

NUTANIX™