Fine-Grained Fault Tolerance For Resilient pVM-based Virtual Machine Monitors

Djob Mvondo*, Alain Tchana†, Renaud Lachaize*,
Daniel Hagimont‡, Noël De Palma*

*University of Grenoble Alpes, † ENS Lyon, ‡ IRIT

DSN'20, 01 July 2020

Context

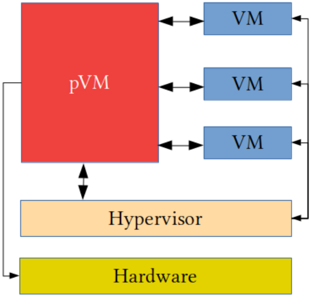
System virtualization is crucial in nowadays datacenters due to :

- Efficient physical resource usage
- Apps scalability improvement
- Fault tolerance increased for running apps
- etc ...

Context

Most virtualization systems (e.g., Xen or Hyper-V) rely on a particular VM, refered as **privileged VM (pVM)** to :

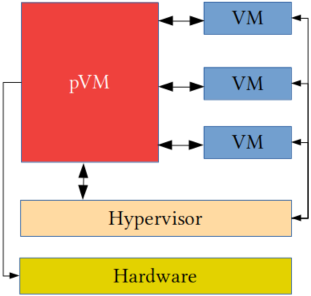
- VM management tasks
- Multiplexing I/O devices to VMs
- Hosting monitoring tools e.g., OpenStack Nova Compute



Problem Statement

Delegating those tasks to a VM raises many concerns :

- **pVM Resilience**
  Single point of failure. In case of the pVM's crash:
  - ~~Connect to physical server~~
  - ~~Manage user VMs~~
  - ~~Network applications~~



Related Work

- **Full Replication**[1].
  Replicate virtualized components across the datacenter.
  - Resource consuming
  - Synchronization across the different replicas

[1] Nutanix: https://nutanixbible.com

[2] Colp et al. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. SOSP'11

Related Work

- **Full Replication**[1].
  Replicate virtualized components across the datacenter.
  - Resource consuming
  - Synchronization across the different replicas
- **Disaggregation + periodic reboot**[2].
  Break the pVM to smaller blocks to reduce surface attacks and periodically reboot the blocks to recover from a faulted state (corrupted, hanged, etc ...)
  - Reduce security flaws
  - Huge overhead for latency-sensitive apps.

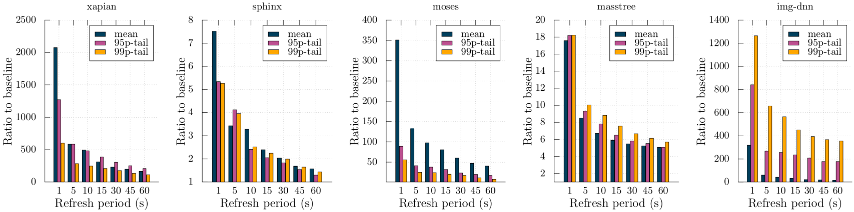
[1] Nutanix: `https://nutanixbible.com`

[2] Colp et al. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. SOSP'11

Related Work - II

For apps in the Tailbench suite[3], periodic reboot results up to:

- 5x-2000x degradation for the mean latency
- 5x-1300x for the 95th percentile, and
- 5x-1200x for the 99th percentile.



[3]H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," IISWC'16

Our work - PpVMM

PpVMM (Phoenix pVM VMM) relies on three design principles :

- **Disaggregation.**
  Split the pVM into independent blocks

Our work - PpVMM

PpVMM (Phoenix pVM VMM) relies on three design principles :

- **Disaggregation.**
  Split the pVM into independent blocks
- **Specialization.**
  For each independent block, use specialized fault detection + recovery techniques

Our work - PpVMM

PpVMM (Phoenix pVM VMM) relies on three design principles :

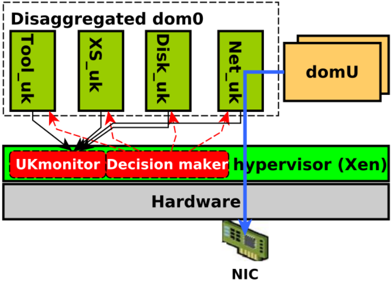
- **Disaggregation.**
  Split the pVM into independent blocks
- **Specialization.**
  For each independent block, use specialized fault detection + recovery techniques
- **Pro-activity.**
  Trigger recovery mechanism as soon as possible when the fault occurs

PpVMM applied on **Xen**

Applied on the `Xen hypervisor`, the pVM (dom0) splits in several independent blocks.

- **Stateful.**

  xs_uk: stores configuration data which can be queried
- **Stateless.**
  - **Net_uk** provides NIC access to VMs
  - **Disk_uk** provides block devices access (e.g hard drive) to VMs
  - **Tool_uk** hosts the Xen toolstack (xl)



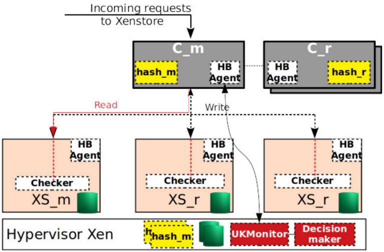
PpVMM applied on **Xen** - xs_uk

Must prevail against:

- **Unavailability.** Due to lack of resources or a crash
- **Database corruption.** Due to hardware faults e.g., bit flipping or software bugs.

PpVMM applied on **Xen** - xs_uk

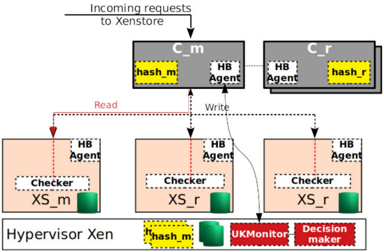
- **Unavailability**
  - We maintain via a third party consensus framework (**etcd**), a set of Xenstore replicas[4].
  - The **xs client libray** interacts with an etcd instance which sends the request to the leader Xenstore. Then forward the request to the others for state synchronization.



[4] Replicas are maintained on the same server

PpVMM applied on **Xen** - xs_uk

- **Database corruption**
  - A `sanity check` module (C_m) checks the integrity of a Xenstore for each request via a hash based mechanism. In case of problem, the given replica is corrected by replaying database logs from a clean replica.



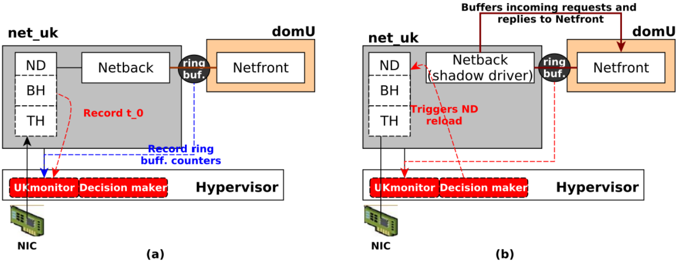
PpVMM applied on **Xen** - net_uk

PpVMM detects the unavailability of the net_uk at two levels:

- `Fine-Grained`: A fault/crash of the network card driver
- `Coarse-Grained` : A fault/crash of the entire net_uk

PpVMM applied on **Xen** - net_uk

Rely on `shared ring buffers counters` monitoring to detect network card failure. During recovery, a shadow driver buffers incoming requests which are forwarded to the real driver at recovery.

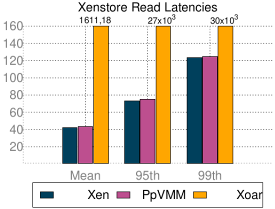


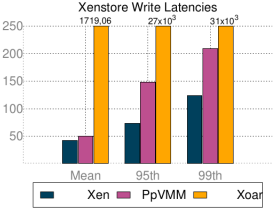
PpVMM applied on **Xen** - Evaluation

- **Testbed**.
  - 48-core PowerEdge R185 machine with AMD Opteron 6344 processors and 64 GB of memory.
  - Xen 4.10.0 on Ubuntu 12.04 LTS and Linux 5.0.8.
  - The NIC is NetXtreme II BCM5709 Gigabit Ethernet interface (bnx2 driver).
  - Apps from the Tailbench suite.
- **Overall performance**
  - PpVMM incurs **1-3%** overhead on IO applications

PpVMM applied on **Xen** - Evaluation II

- **xs_uk**.
  - **20.27%** overhead compared to vanilla Xen VM creation[5]. (≈ 900ms)
  - **1.54 ms** and **5.04 ms** for crash and data corruption detection times
  - **25.54ms** to recover from a faulty replica





[5] Xoar incurs up to 51.65%

PpVMM applied on **Xen** - Evaluation III

- **net_uk**.
  - **12.4 - 17.3%** overhead compared to vanilla Xen for mean latencies (Tailbench apps)[6].
  - **27.27 ms** to detect a fault with the NIC driver and **4.7 ms** recovery with no packet loss

	DT (ms)	**RT (s)**	**PL**
FG FT	27.27	4.7	0
CG FT	98.2	6.9	425, 866
TFD-Xen [4]	102.1	0.8	2379
Xoar [5]	52×10^3	6.9	1, 870, 921

`Please, check out the paper for more in-depth results.`

[6] Xoar incurs up to 1130.67%

Conclusion

- PpVMM relies on three design principles: `Disaggregation, specialization, and pro-activity`
- A functional prototype based on the Xen hypervisor
- Detection + recovery times better than state of the art approaches with minimal overall overhead (**1-3%**)
- Check out the prototype here : `https://github.com/r-vmm/R-VMM` (`We love stars, don't forget to drop one.`)

Conclusion

- PpVMM relies on three design principles: `Disaggregation, specialization, and pro-activity`
- A functional prototype based on the Xen hypervisor
- Detection + recovery times better than state of the art approaches with minimal overall overhead (**1-3%**)
- Check out the prototype here : `https://github.com/r-vmm/R-VMM` (`We love stars, don't forget to drop one.`)

Thank you for your attention !