Deterministic Container Resource Management in Derivative Clouds

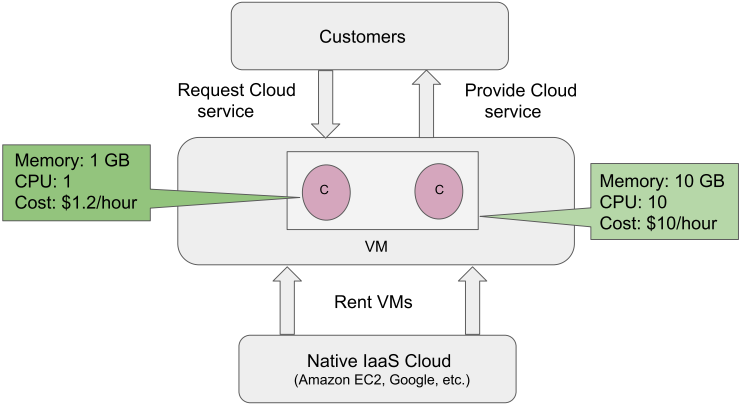
Chandra Prakash, Prashanth, Umesh Bellur, Purushottam Kulkarni

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

19^{th} April, 2018

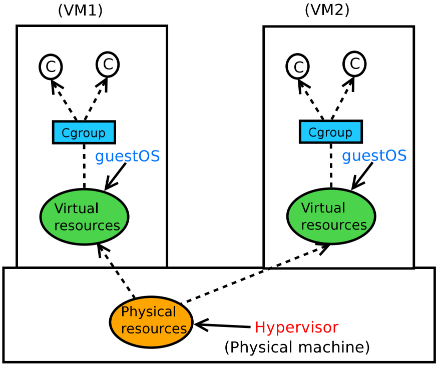
International Conference on Cloud Engineering

Derivative cloud



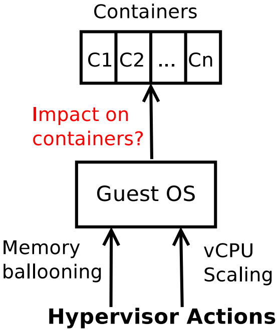
Dual control over resources

- Hypervisor and guest OS both control the same resources
- Hypervisor not aware of containers requirements



Hypervisor mechanism under consideration

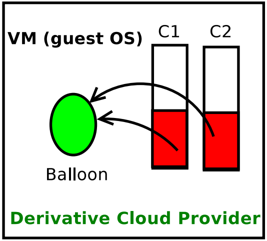
- **Ballooning** is used to achieve memory overcommitment[1]
- **vCPU scaling** is used to reduce scheduling overheads in over committed situation [2]



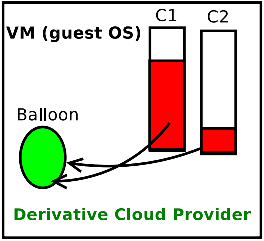
[1] C. A. Waldspurger, "Memory resource management in vmware esx server," ACM SIGOPS Operating Systems Review, vol. 36, no. SI, pp. 181-194, 2002.

[2] L. Cheng, J. Rao, and F. Lau, "vscale: automatic and efficient processor scaling for smp virtual machines," in Proceedings of the 11th European Conference on Computer Systems, ACM, 2016.

Undesirable effects due to ballooning



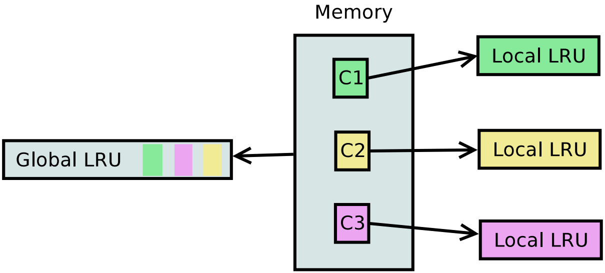
Happens



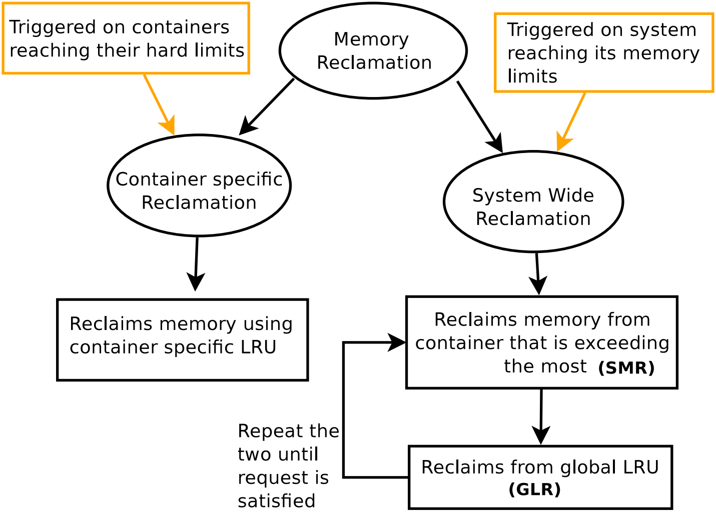
Desired

Existing memory reclamation in containers

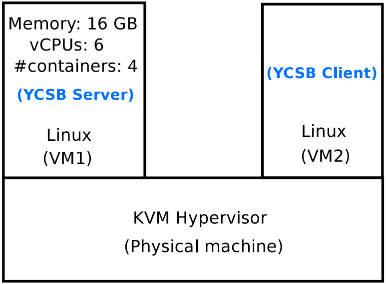
- Memory provisioning knobs: *Hard-Limit* and *Soft-Limit*
- ***exceed***: difference between memory usage and Soft-Limit
- **SMR** (Soft Memory Reclaimed): memory reclaimed from local LRU
- **GLR** (Global LRU Reclaimed): memory reclaimed from global LRU



Existing memory reclamation in containers



Impact of ballooning



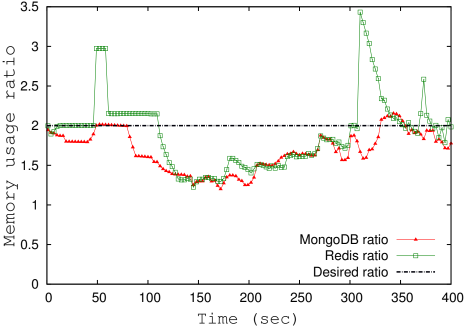
Set-up

- Memory reclamation rate: 2 GB every 30 seconds (generated from host after 100 seconds)

Impact of ballooning

Default configuration of four containers

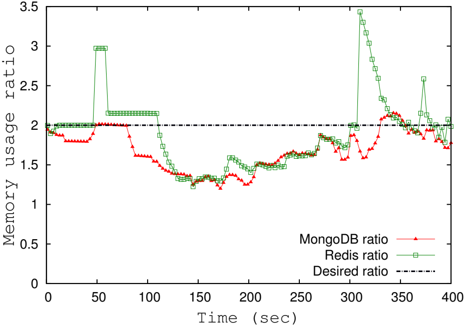
Container	Hard-limit(GB)	Soft-limit(GB)	Key size (# records)
Redis-Low	2	0.5	500K
Redis-High	4	1	1000K
Mongo-Low	2	0.5	500K
Mongo-High	4	1	1000K



Memory usage ratio

Impact of ballooning

- ***Existing knobs (limits) do not guarantee proportionate memory allocation during memory pressure situations***



Memory usage ratio

CPU provisioning Issues

Container configuration

Pin container to CPU --> cpuset.cpus
Maintain container cpu share --> cpu.share

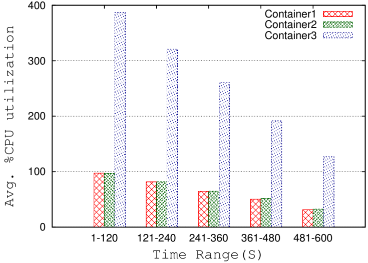
Hypervisor optimization

Reduce scheduling overhead --> Dynamic vCPU scaling

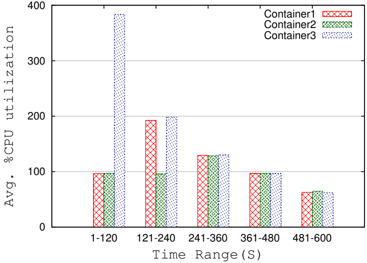
Hypervisor unaware of cpuset.cpus & cpu.share
Container requirement disturbed

Experimental setup

VM configuration	7 vCPUs and 8GB Memory
Number of containers inside VM	3
CPU allocation ratio	1:1:4
Benchmark	Sysbench
vCPU scaling down frequency	1 vCPU every 120s (vCPU1,2,3,&4)
vCPU mapping using cpuset.cpus	C1: vCPU1, C2: vCPU2, C3: vCPU3,4,5,&6



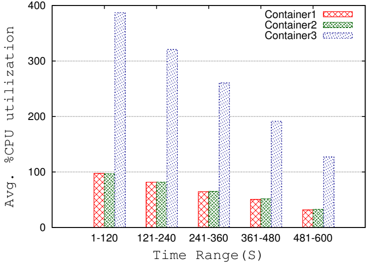
CPU utilization without pinning



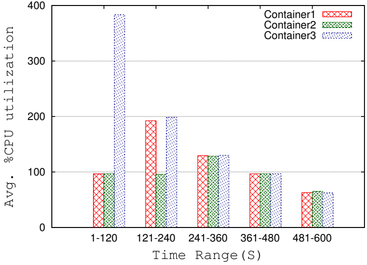
CPU utilization with pinning

Impact of vCPU scaling

- Pinning and scaling $\implies$ ***non-deterministic*** CPU utilization
- **Desired goal:** achieve pinning benefits + maintain CPU share



CPU utilization without pinning



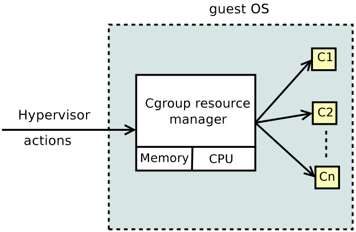
CPU utilization with pinning

Summary of issues in nesting setup

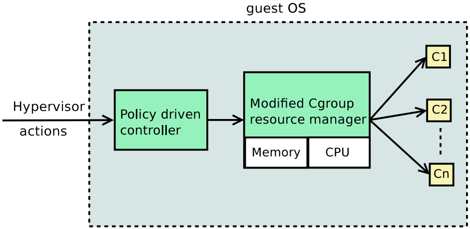
- Ballooning may fail to satisfy container requirements
- vCPU scaling may not respect cpu share with cpu pinning

Our approach

- Native cloud provider can be public or private
- We can't control or change hypervisor in case of public cloud
- We provide solution at guest OS level



Default approach



Our approach

Proposed memory policies

Proportionate memory allocation

- Allocate memory according to credit share of containers

Application-specific differentiated memory reclamation

- Protect memory sensitive container(s) from memory reclamation

Proposed CPU policies

Maximize dedicated vCPU while maintaining allocation ratio

- To get maximum benefits of pinning

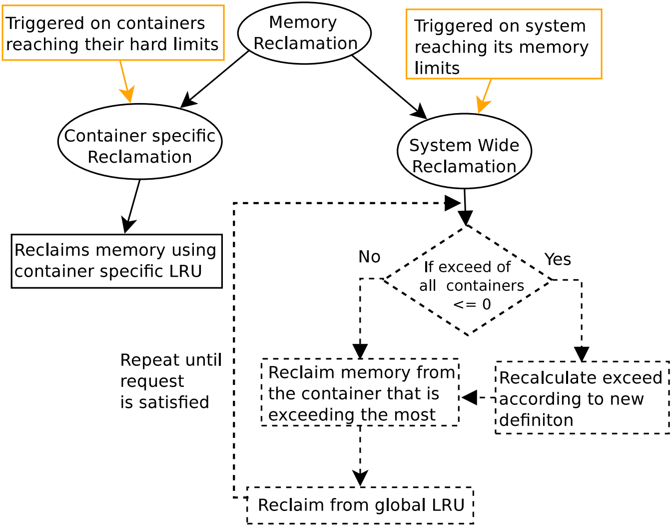
Provide pinned vCPU(s) to a subset of containers

- Based on application nature or user requirement

Contribution

- Modified the memory reclamation logic in *memory cgroup* subsystem
- Provided an additional definition of *exceed*
- Performed several modifications in Linux kernel
  - Added extra parameters in *memory* and *cpu cgroups*
  - Added control to maximize SMR
  - Provided knob to control reclamation chunk size
- Created a *cpuset* calculator in user space

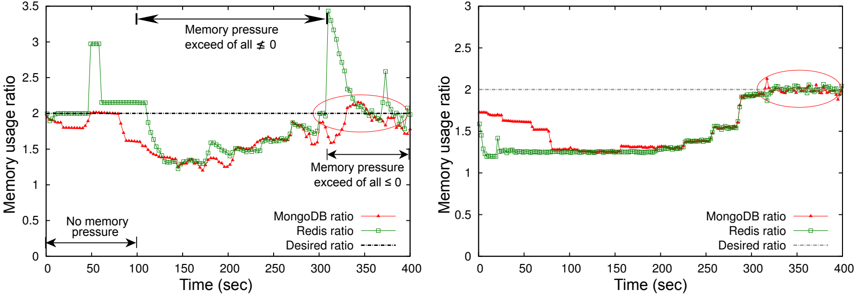
Modified memory reclamation



» ***exceed = memory_usage – proportionate_share***

Effectiveness of memory policies

- Ratio of memory weights: 1:2

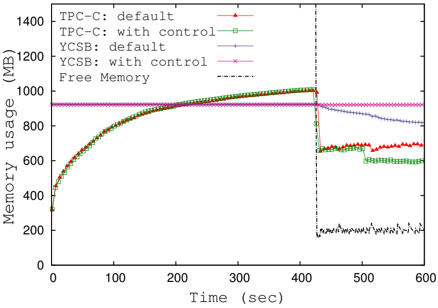


Memory usage ratio (default)

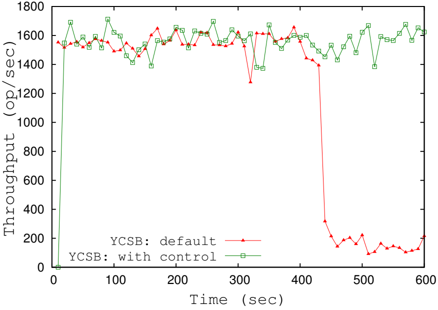
Memory usage ratio (with control)

- Able to maintain memory usage ratio when ***exceed* of all containers become less than or equal zero** (after 300 second)

Application specific reclamation



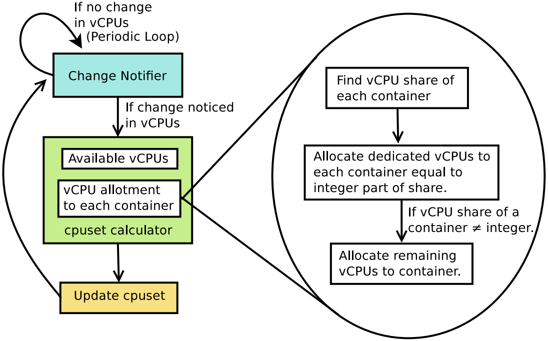
Memory usage

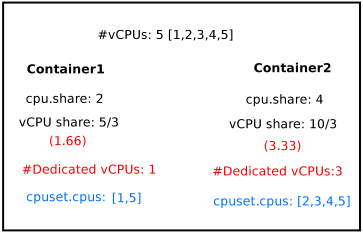


YCSB throughput

- Memory is not reclaimed from YCSB application container (memory sensitive) and it's throughput remains intact

vCPU reallocation design

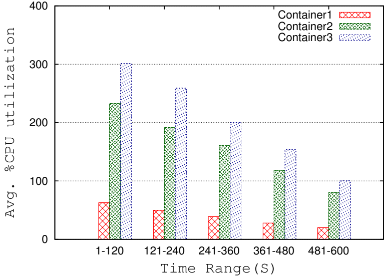




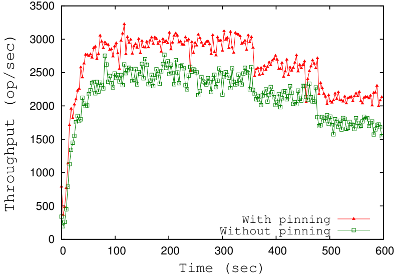
Effectiveness of vCPU reallocation

Experimental setup

VM configuration	7 vCPUs and 8GB Memory
Number of containers inside VM	3
CPU allocation ratio	1:4:5
Benchmark	Sysbench & Twitter
VCPU scaling down frequency	1 vCPU every 120s (vCPU1,2,3,& 4)



CPU utilization by each container



Twitter throughput with scaling down vCPUs

- Able to maintain CPU share along with pinning

Conclusion & future work

Conclusion:

- Quantified the impact of hypervisor actions on containers running inside VM
- Proposed user-defined policies to mitigate the impact of hypervisor actions
- Demonstrated the effectiveness of memory and CPU policies empirically

Future work:

- Design an efficient algorithm for container placement in derivative (nested) setup

Thank you
Questions???

Email id: chandrap@cse.iitb.ac.in

Modifications in Linux kernel

- Added a *weight* parameter in *memory* cgroup and a *pin* parameter in *cpu* cgroup.
- Modified the balance_pgdat() routine (Linux kernel version 4.7).

Listing 1 : Original reclamation logic

```
For every reclamation request:
        SMR();
        GLR();
```

Listing 2 : Modified reclamation logic

```
For every reclamation request:
     NoOfReclaimedPages = SMR();
          if (NoOfReclaimedPages==0):
               GLR();
```

- Created a kernel module to control the reclamation chunk size.