Deterministic memory management for container based services

M.Tech Stage-1 Presentation

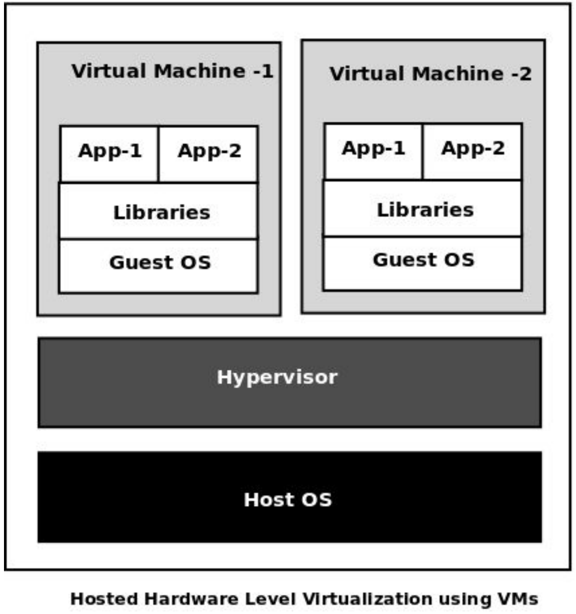
Prashanth, 153050095
Guide: Prof. Puru

Difference between Virtual Machines and Containers

Container is a virtual environment that contains **a set of processes grouped along with its dependent resources** into a single logical OS entity

Share OS-kernel among containers

OS-level virtualization



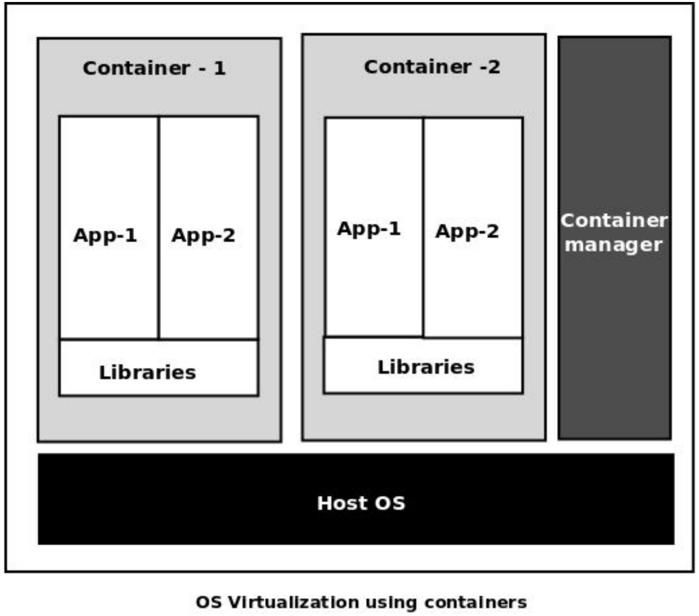


Fig-1: Difference in system stacks used between VMs and Containers

Comparison between native container and derivative cloud environment

Nested hosting

Derivative cloud environment is proposed in Spotcheck [7]

Repackages and resells resources purchased **from native IaaS platforms**

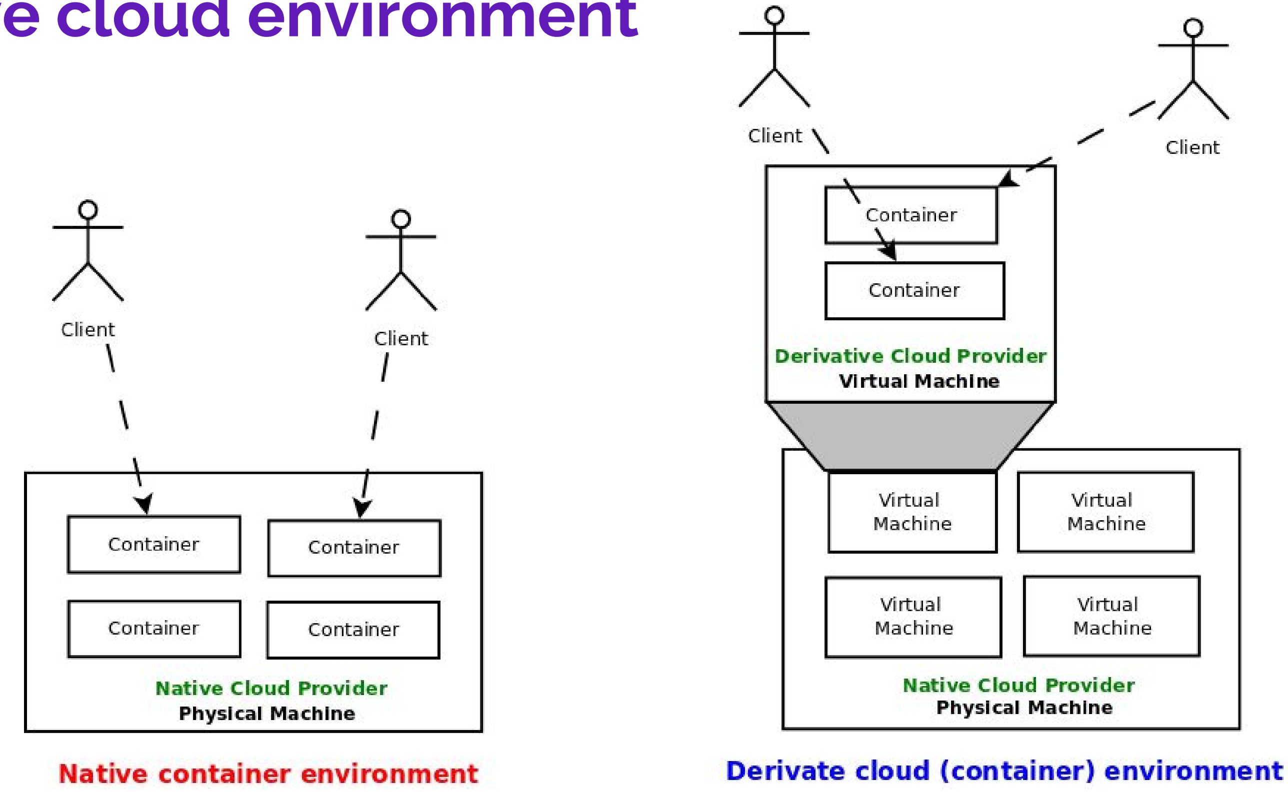


Fig-2: Comparison between native container cloud and derivative cloud provider

Motivation

Consider **2 mongoDB clients with memory intensive workloads** on a cloud provider who uses containers to provision clients,

Table-1: Client requirements and their provisioning using existing container knobs

Clients	Cost Paid for service	Avg. Memory usage	Memory Provisioned using existing knobs	Desired App throughput
Mongo-Low	1x	1x	1x	**1x**
Mongo-High	2x	2x	2x	**1x**

Desired App throughput = Memory Provisioned / Avg. Memory usage

Containers under no memory pressure versus under pressure

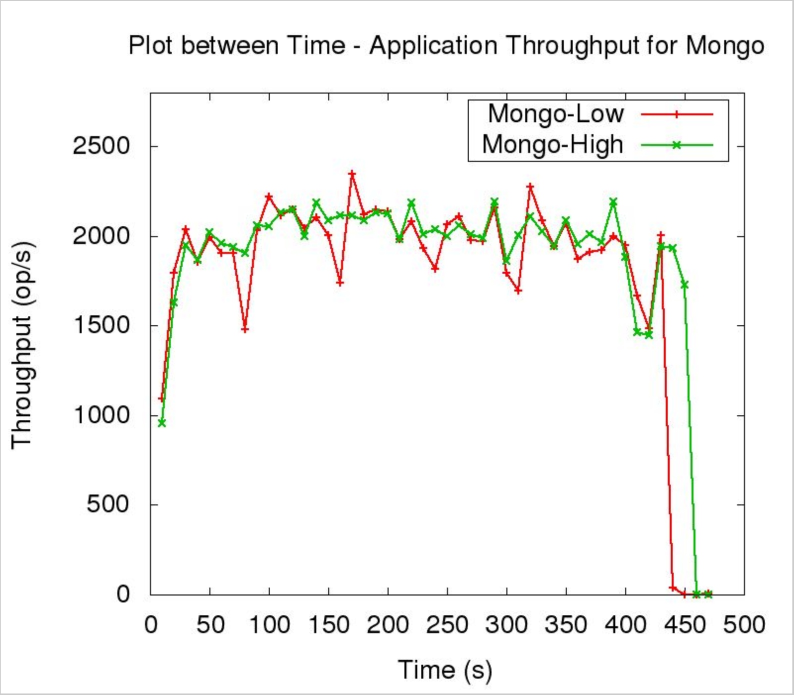


Fig 3: Containers under no memory

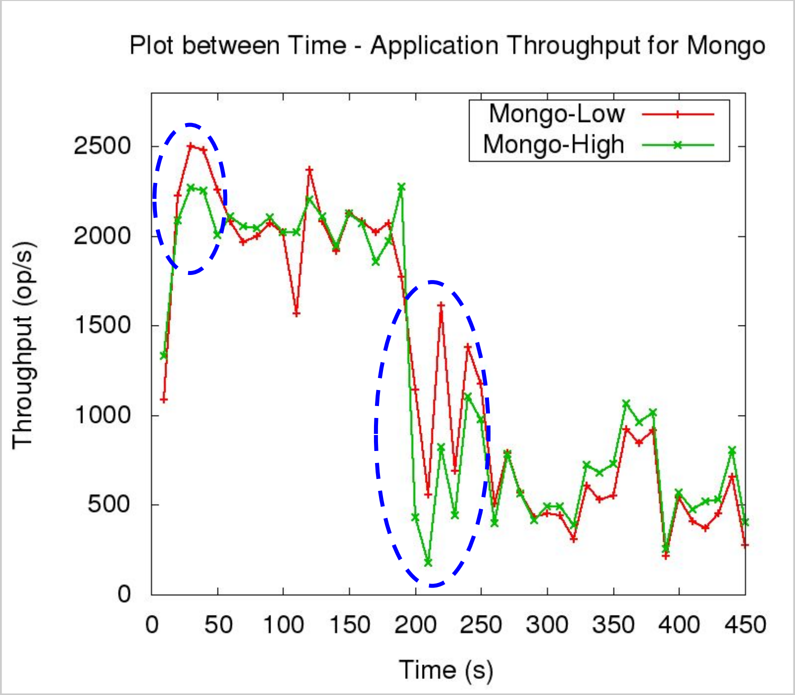


Fig 4: Containers under memory pressure

Not achieving desired throughput ratios of 1:1
Penalizing higher priority (allocation) container

Desired characteristics

Table-2: Average throughput in each case (op/s)

Client	No pressure	Observed with pressure	Desired with pressure
Mongo low	1825	1268	1255(-)
Mongo high	1972	1242	1255(+)

Desired characteristics while provisioning

- **Differentiated**
  Notion of priority
- **Deterministic**
  1:1 throughputs every time in this case
- **Adaptive**
  External factors shouldn't disrupt
- **Elastic**
  Scale as and when required
- **Satisfy SLAs**
  Satisfy all SLAs promised

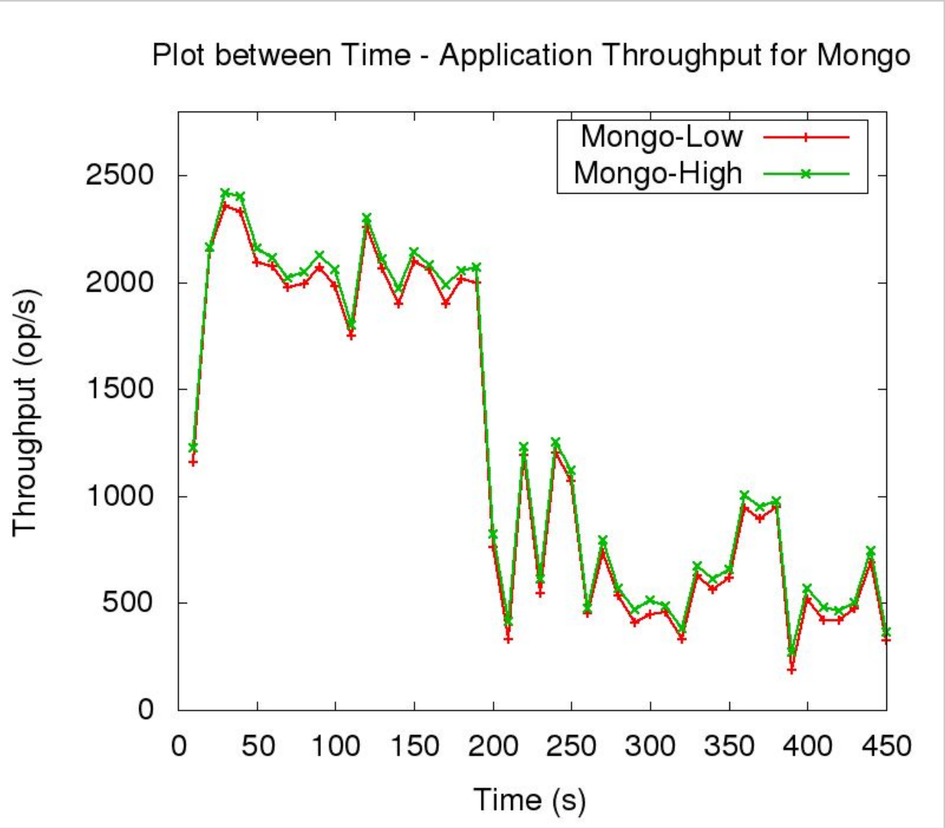


Fig 5: Desired throughputs based on allocation

Problem Description

- **Understand memory management policies** used by Linux **for containers** by going through existing literature and perform empirical analysis when necessary
- Extrapolate its **implications on applications running** in a,
    - Native container environment
    - Derivative cloud environment
- **Identify issues** with existing policies in both environments
- Make a new of **requirements for a new policy**
- **Design and implement** a policy that satisfies the requirements

Related Works

- **Memory management**

  Ballooning [1], Page deduplication [2] , **Elastic memory management in IaaS using VM** (Vertical Scaling) [3], Overdriver [4], Difference Engine [5], Ex-tmem [6]

- **Resource provisioning**

  Aneka [10], **Elastic Application Container** [9], **VM placement for deterministic N/w provisioning**[21]

- **Container characterization**

  Comparative analysis between VMs and Containers [12] [13] [14] [15] [16]

- **Derivative cloud**

  Spotcheck [7]

Deterministic memory management for container based services

Memory cgroup

- Provide **memory accounting and control**
- Tracks,
  - Anonymous/Page-cache pages
  - Kernel memory (when enabled)
- Provide control on
  - Soft limit (SL)

    Min. memory limit
  - Hard limit (HL)

    Max. memory limit
  - Swappiness
  - OOM

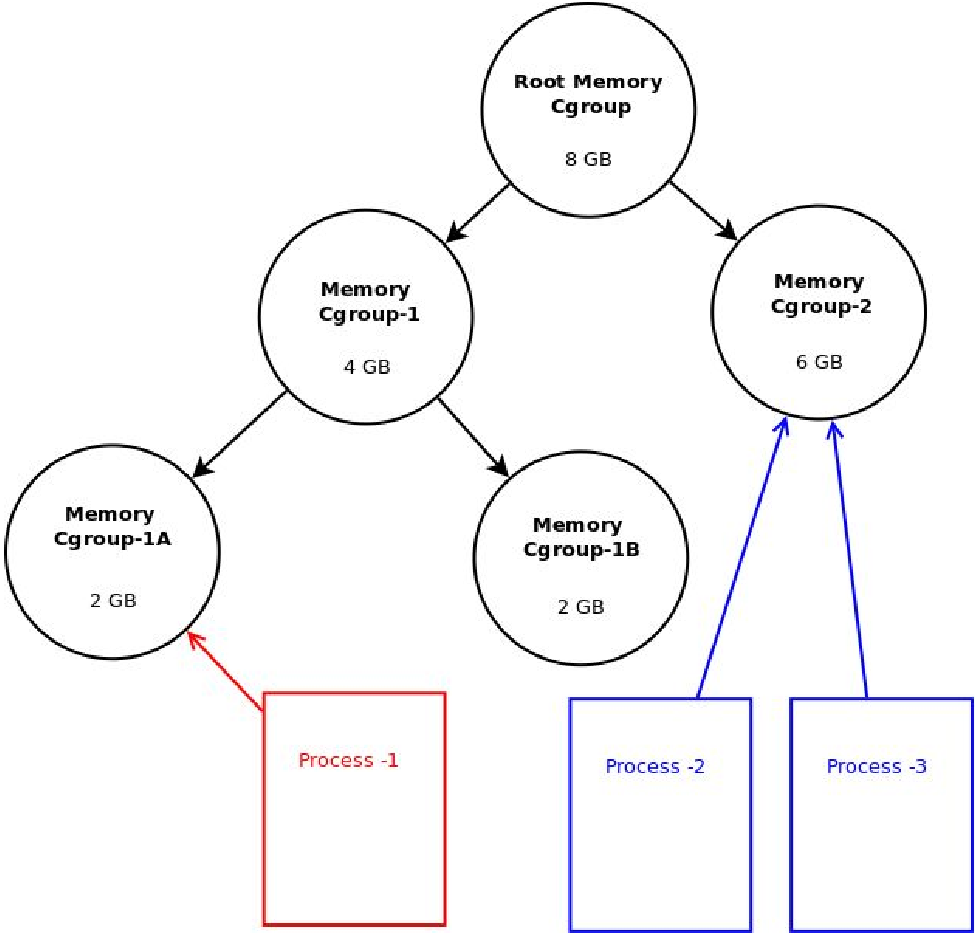


Fig-6: Attaching processes to memory cgroups (containers)

Memory Pressure Generators

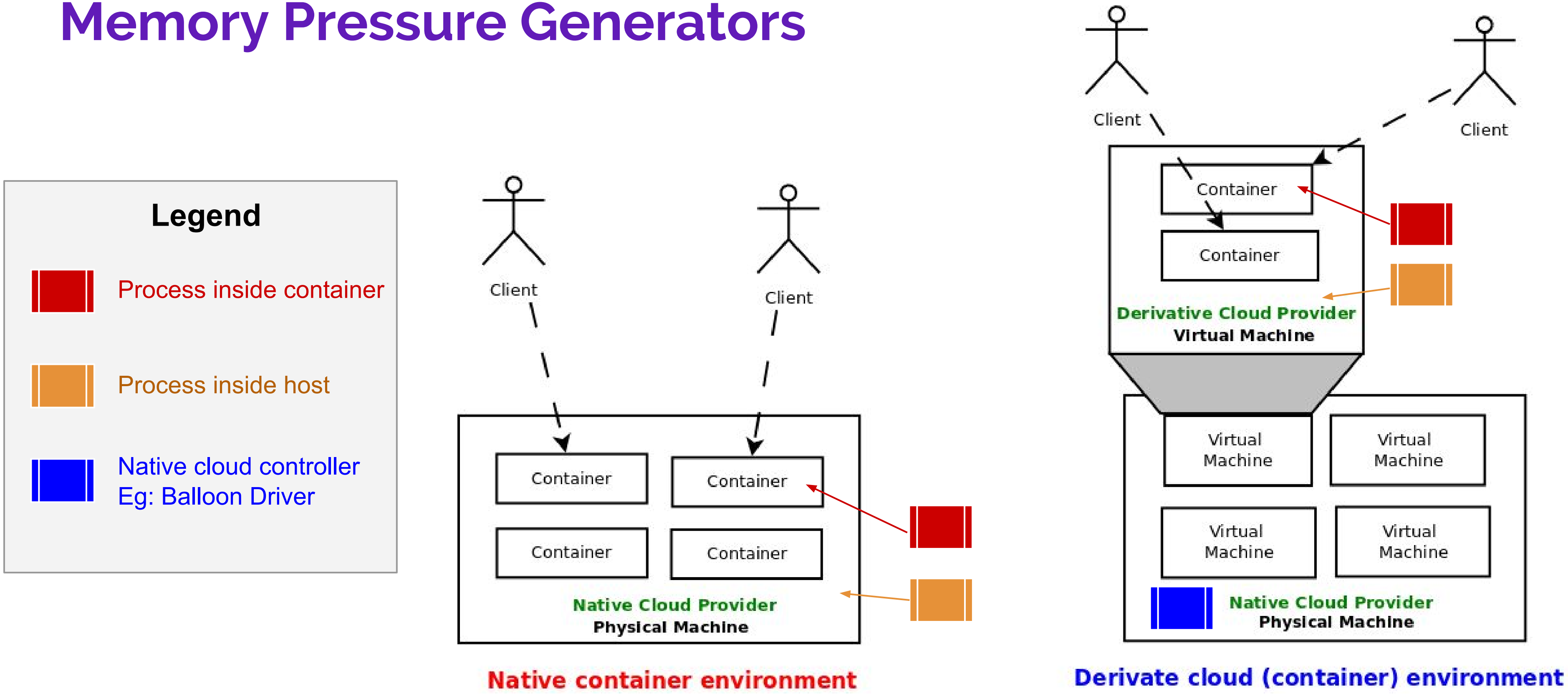


Fig-7: Memory pressure generators that will trigger reclamation when provider system's free memory is low

Container specific LRUs

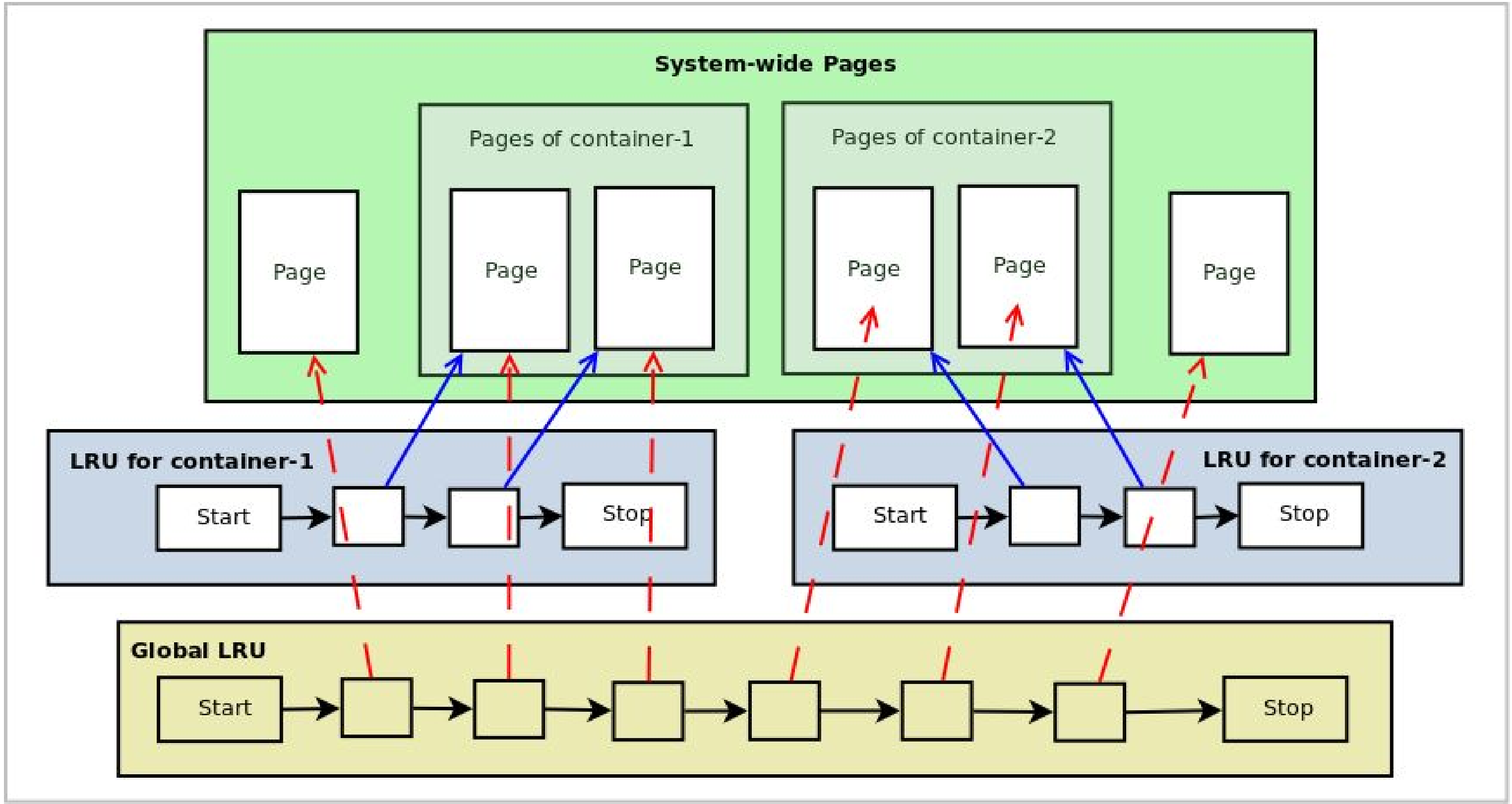


Fig-8: Addition of container specific LRUs

Memory Reclamation in Linux

We focus on system wide reclamation

Exceed = Usage - SL

Two types

1. Soft Memory Reclamation **(SMR)**
2. Global LRU Reclamation **(GLR)**

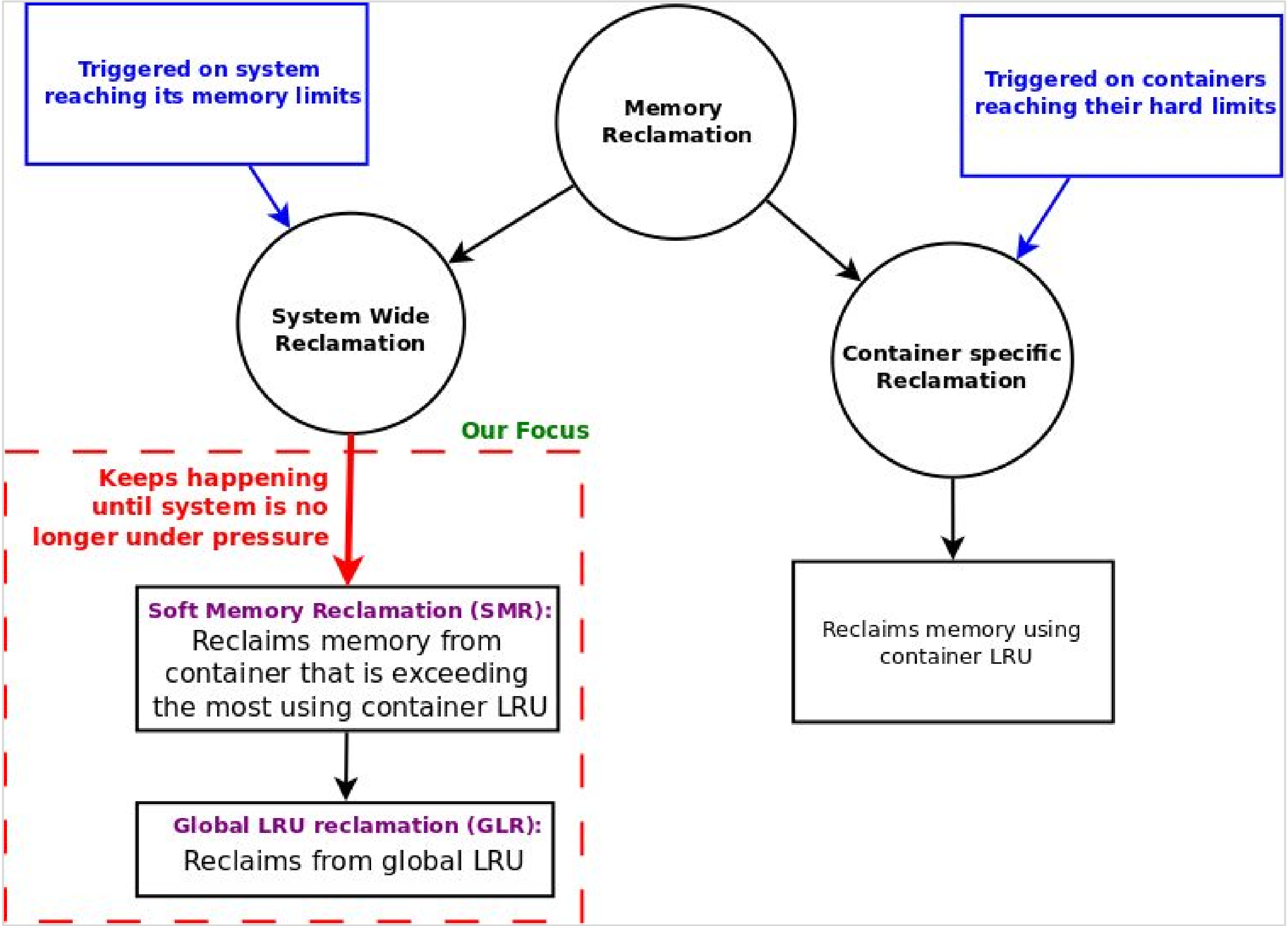


Fig-9: Memory reclamation in a Linux system

Need for empirical study

- Presented **background** is based on Kernel implementation with **no concrete proof**
- Many **unanswered questions**
- Need for empirical analysis to,
    1. To verify proposed hypotheses
    2. Understand parts for which hypotheses couldn’t be drawn

Hypotheses to be verified

➔ SMR purely based on exceed at occurs above soft limits.

➔ Reclamation below soft limits / No soft limits falling back to native GLR hold good ?

Questions

➔ In what order and how much memory reclaimed on a reclamation request from all containers ?

➔ What are the parameters that affect reclamation ?

➔ When containers are exceeding by the same values and different values, how is the reclamation occurring ?

➔ Do existing knobs work as desired ?

➔ What effect does all this have on application performance ?

Experimental parameters and metrics

Parameters

1. No. of containers
2. **Soft limits (SL)**
3. Hard limits (HL)
4. **Usage**
5. Workload
6. **External memory pressure**
7. Memory size of host machine

Metrics

Per container

1. Memory assigned
2. Memory newly allocated
3. Memory reclaimed using **SMR**
4. Total memory reclaimed
5. Application metrics

On the whole,

1. Memory reclaimed using **GLR**

Experimental setup - Native container

Table-3: Base configuration for native container setup

	Container-1 (M1)	Container-2 (M2)
HL (MB)	1000	1000
SL (MB)	150	150
Usage (MB)	500	500
Exceed (MB)	**350**	**350**
Ext Pressure (MB)	200 - 400 - 600 - 800 - 1000 Changed every 50(s)	
Size of VM (GB)	2	

Reference: Stress[17]

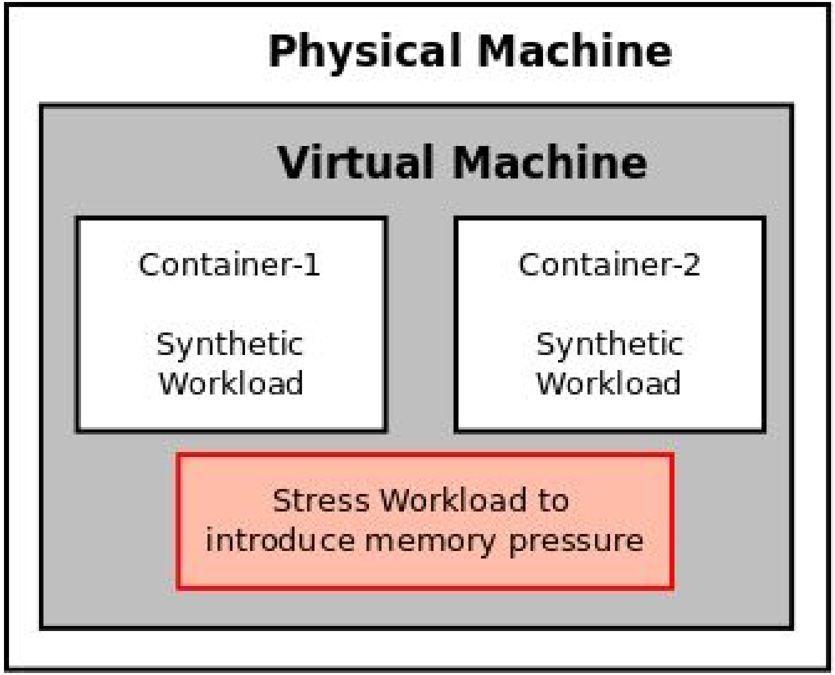
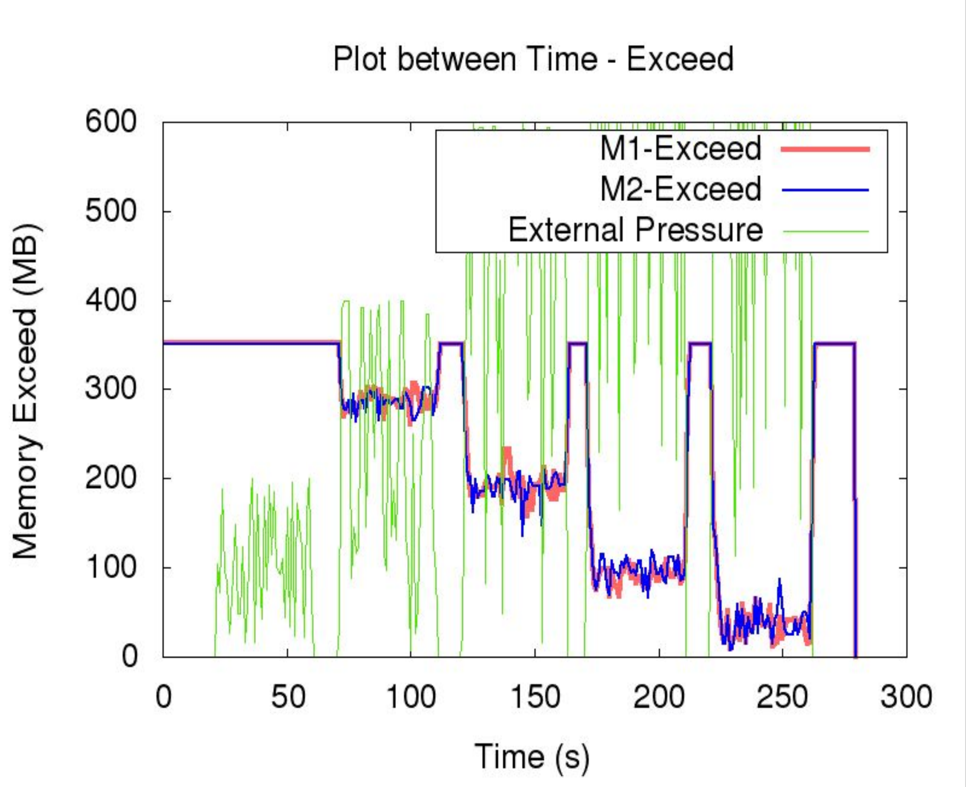
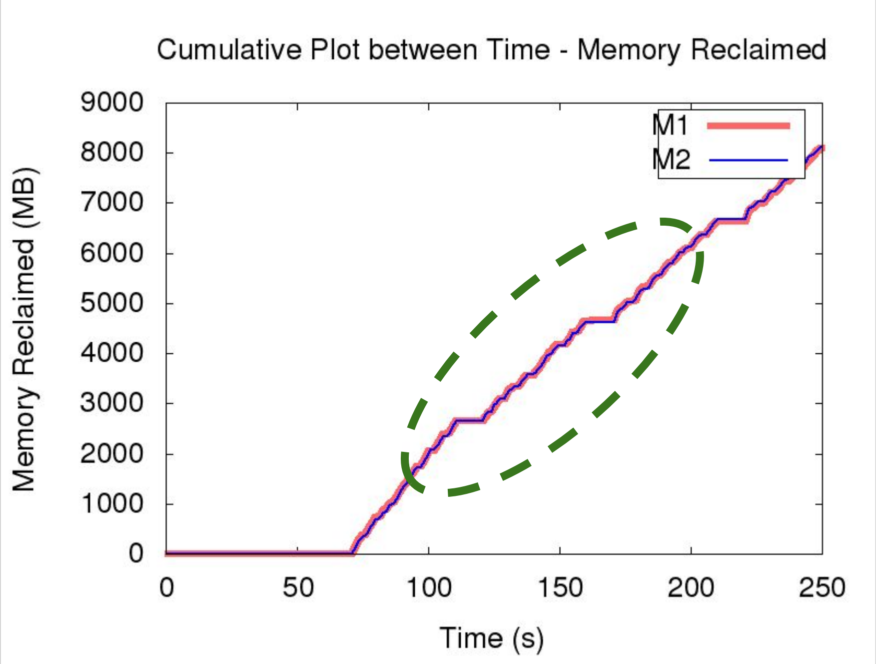


Fig-10: Native container testbed

Reclamation above soft limits





SMR is based on exceed only
Targets containers with equal exceeds iteratively

*Fig-11: Plots for analysis of reclamation when both containers are **exceeding by same value (Above SL)***
*M1 (**Usage: 500**, SL: 150, Exceed: 350), M2 (**Usage: 650**, SL: 300, Exceed: 350)*

Reclamation above soft limits

Most reclamation using SMR

Small amounts using GLR (Removes any inactive pages in global LRU)

Pressure increases, GLR increases

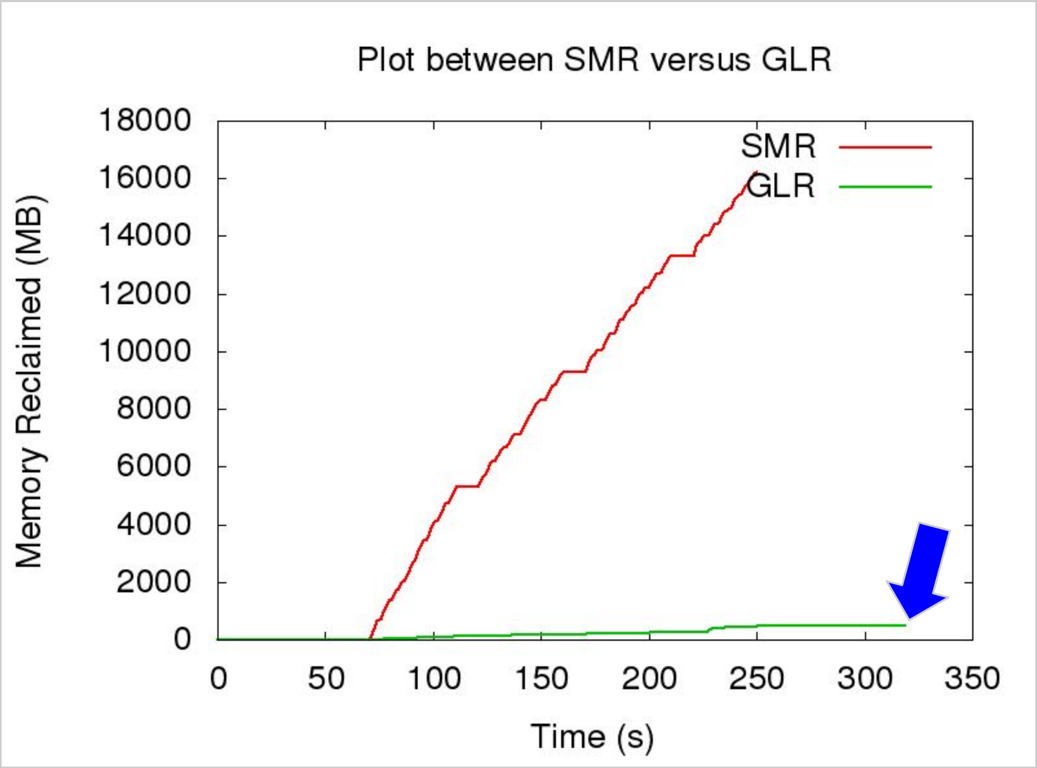
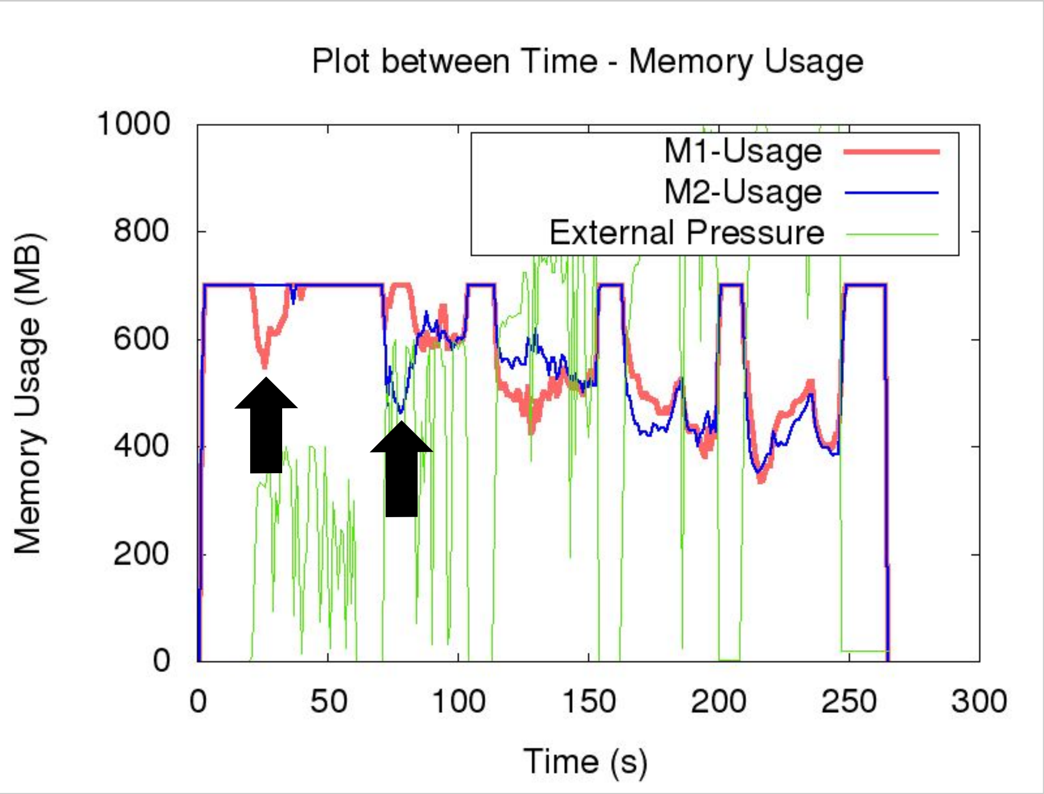
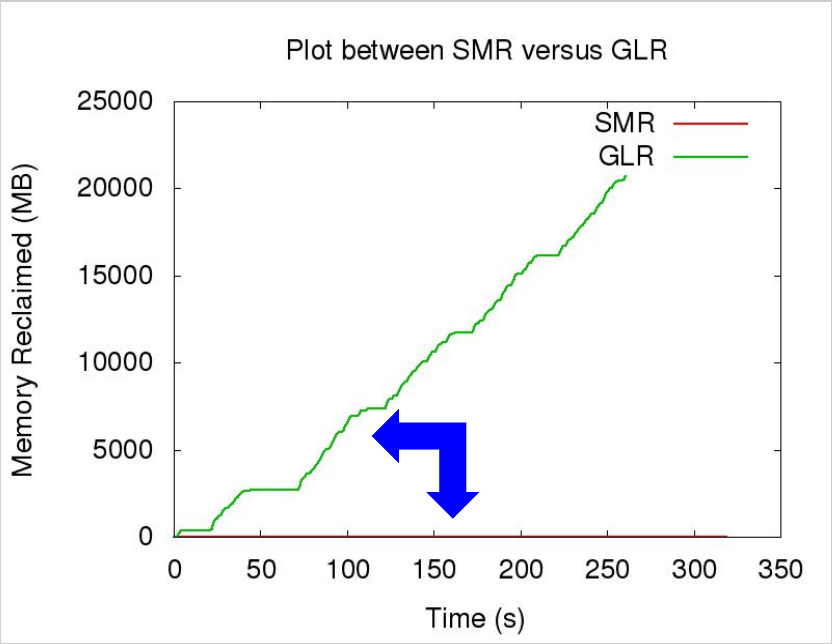


Fig-12: Cumulative plot for SMR versus GLR when both containers are ***exceeding by same value (Above SL)***

Reclamation above below limits



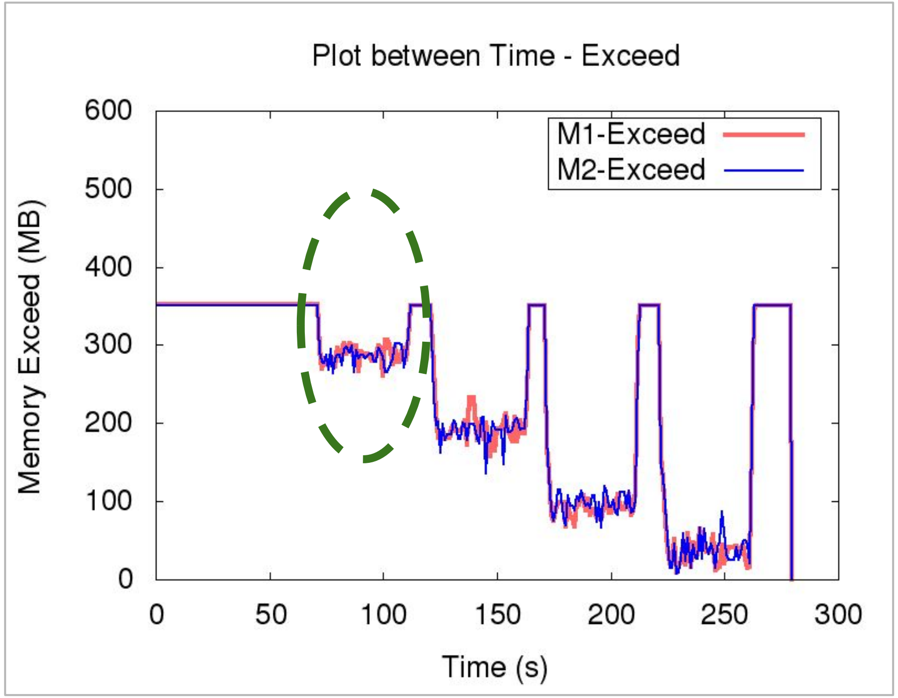


Non deterministic reclamations although usage is same
All reclamation using GLR

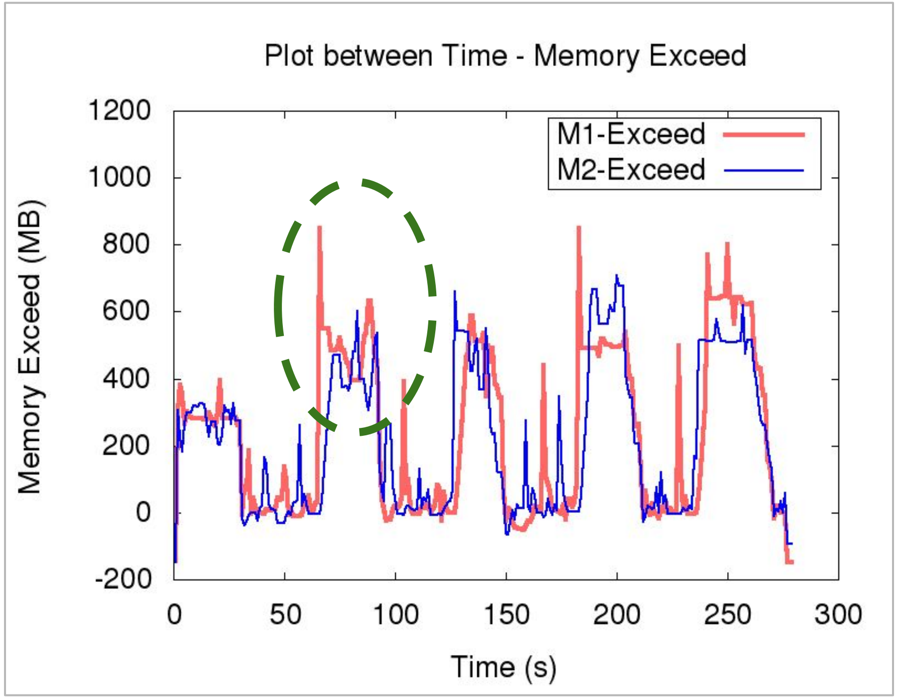
Fig-13: Plots for analysis of reclamation when both containers are ***having same usage (No SL/Below SL)***
*M1 (**Usage: 700,** SL:1000), M2 (**Usage: 700,** SL:1000)*

Effect of workload on reclamation

Anonymous memory consumer



Page cache memory consumer



Abnormalities were observed in reclamation chunks, hence further analysis was done on reclamation chuck / request

Fig-15:Plots for analysis of reclamation when ***workloads characteristic in the containers change***

Empirical analysis for reclamation chunk

Table: Reclamation chunk per reclamation request to a container (MB/req) with base config

Workload characteristic	Min	Avg	Max
Anonymous memory consumer	0	11.57	22.57
Page cache page consumer	0	20.42	202.23

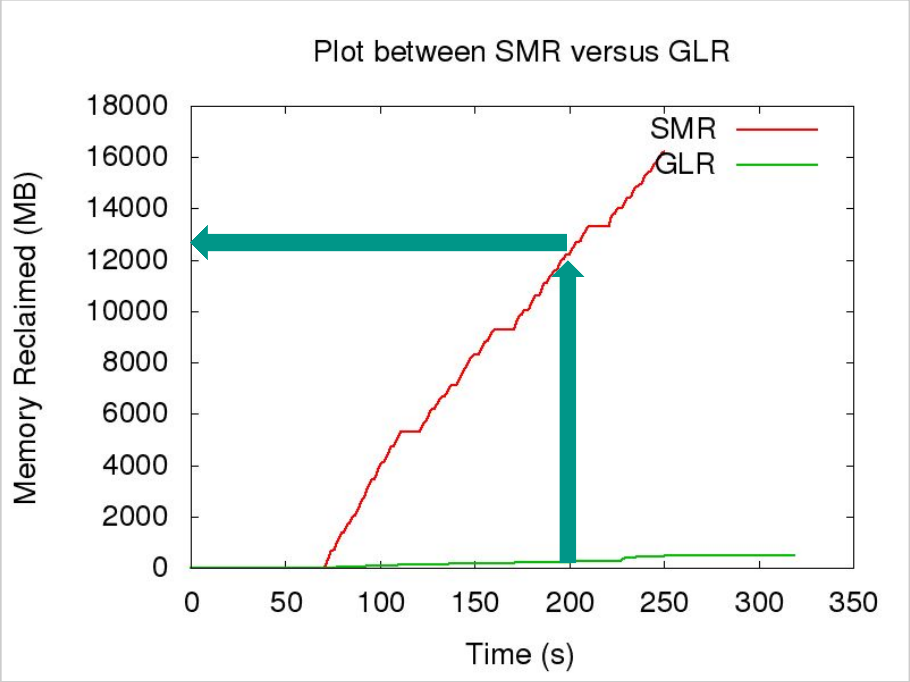
Observations

- Further analysis with higher workloads it was found that
  - Maximum of 6400 pages (25MB) can be reclaimed from anonymous memory / per request
  - All the page cache pages of the container can be reclaimed if necessary
- Workloads with page cache pages targeted more in single request

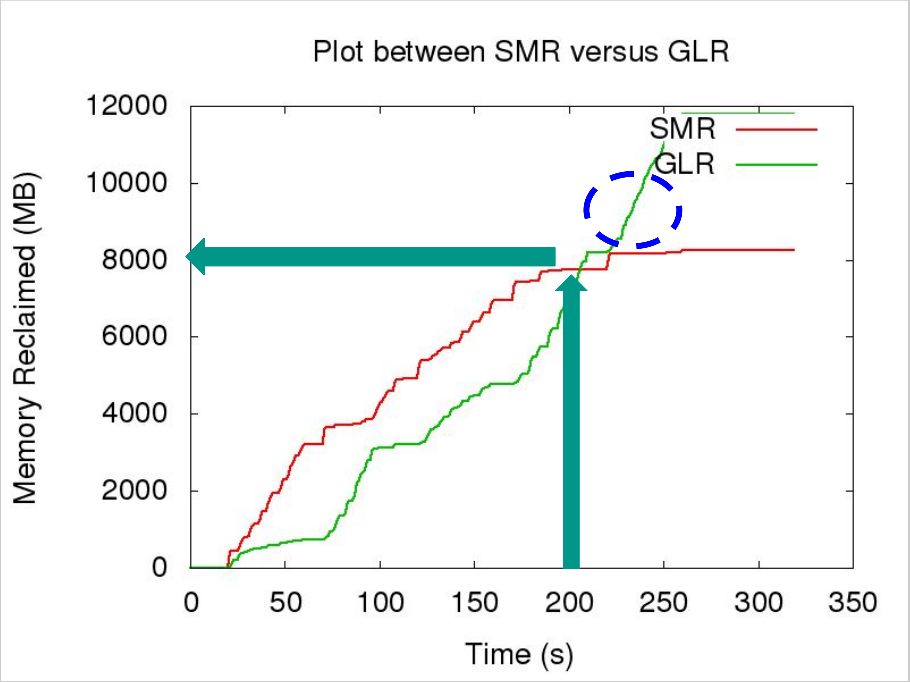
Reclamation chunk = Anonymous memory pages (<25MB) + Page cache pages

Reclamation moving above SL to below SL

Lower soft limits (150MB)



Higher soft limits (300MB)



Lower soft limits gives more opportunity for SMR
Once no more exceeding containers, only GLR

Fig-16: Plots for analysis of reclamation when ***soft limits are increased***

Violation of soft limit

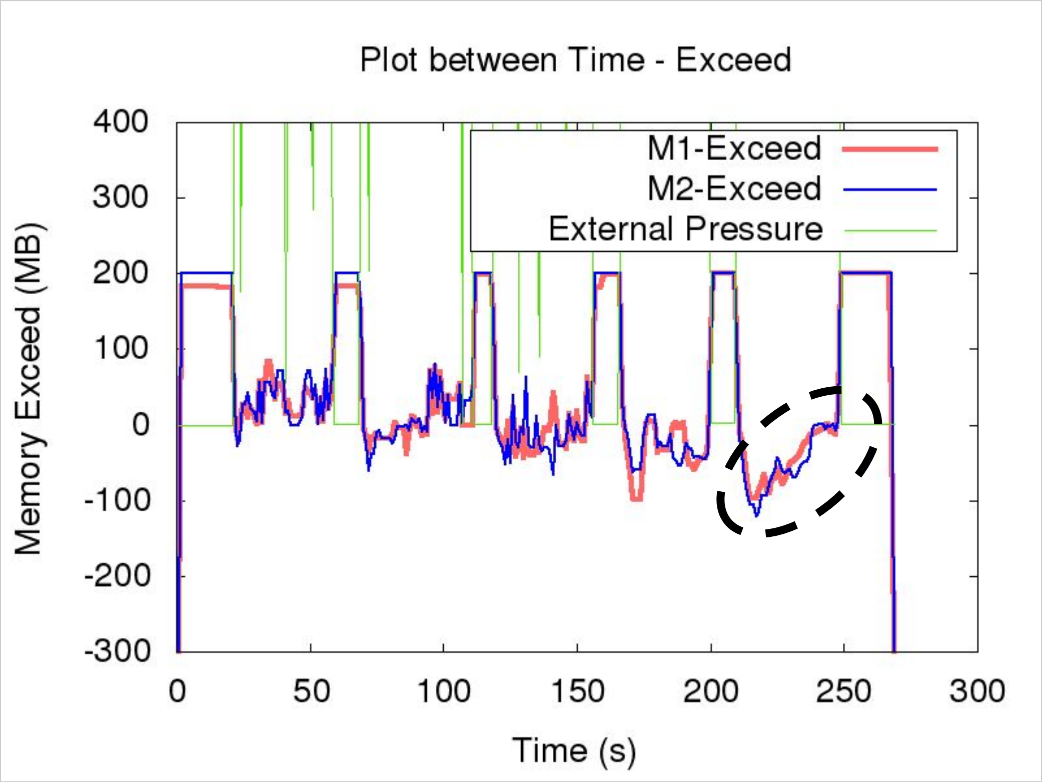


Fig-17: Exceed plot for in previous where, soft limits are increased

SL is violated

Key inferences

Above SL

- Most reclamation occurs using SMR that is based on exceed
- Page cache pages are reclaimed at larger chunks
- Exceeds are equal, equally and iteratively targeted

Below SL

- Usage < SL reclamation falls back to GLR
- GLR is haphazard and there is no control over it
- Soft limit is not an absolute guarantee, best effort approach

Experimental setup - Derivative cloud

Physical Machine:
Native Cloud Provider

Virtual Machine-1 :
Derivative Cloud
Provider

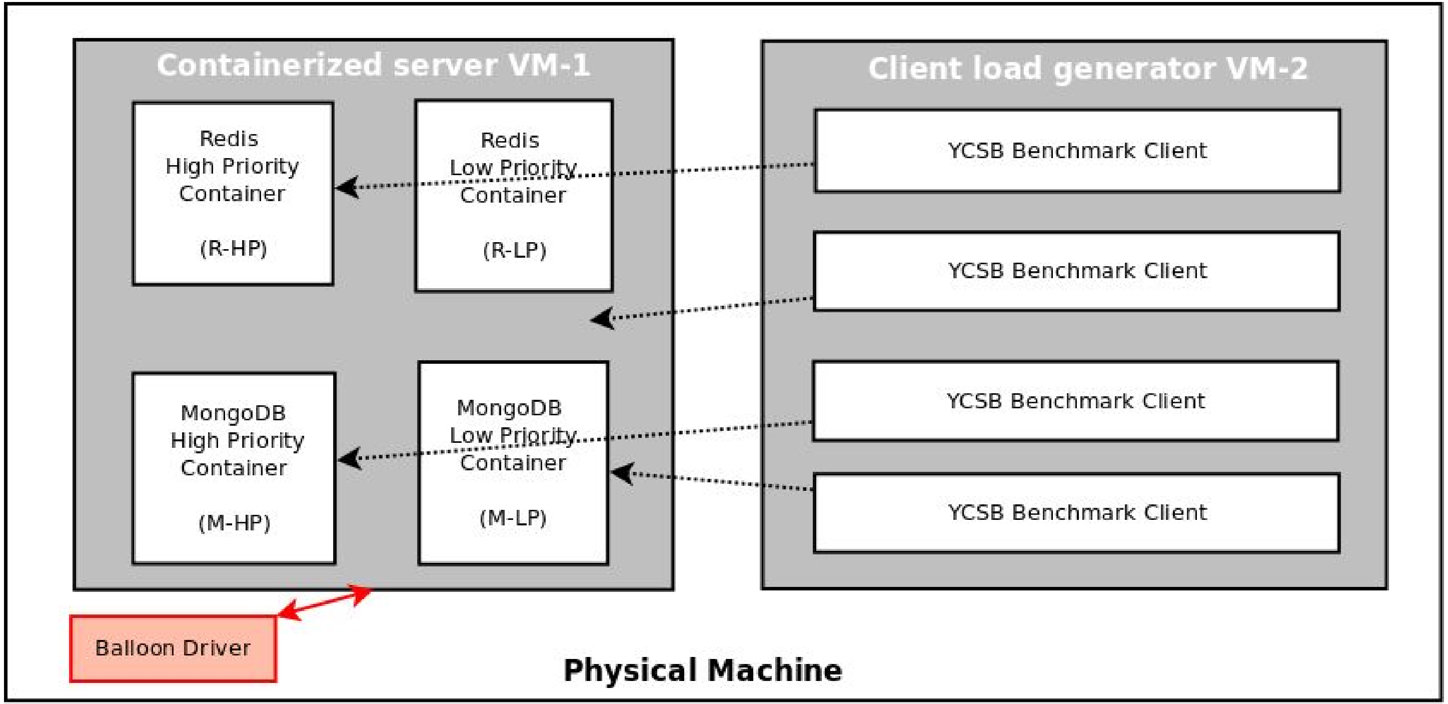


Fig-19: Derivative cloud testbed

References: MongoDB[18], Redis [19], YCSB [20]

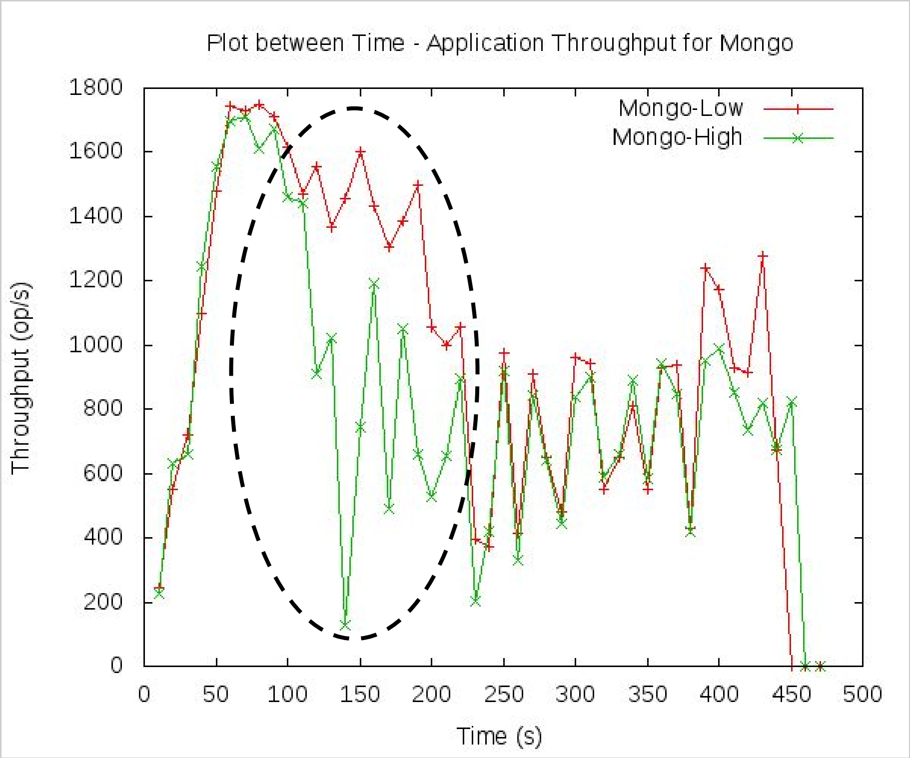
Experimental config - Derivative Cloud

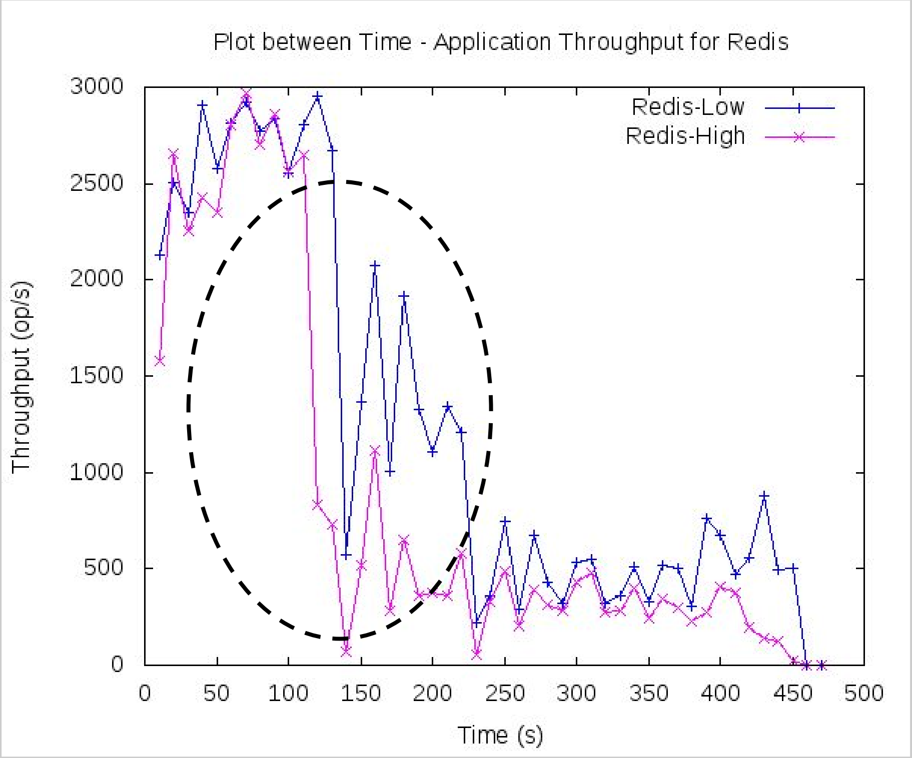
- Containers were loaded with data offline
- During the experiment one client (with 2 threads) connected to one server each
- No pressure from balloon driver in the initial 100s, after which balloon driver increases pressure every 30s

Table: Base configuration for derivative cloud setup

Container	HL (GB)	SL (GB)	Workload size (records)	Avg. Usage (GB)
Redis-Low	2	0.5	500K	1.3
Mongo-Low	2	0.5	500K	1.3
Redis-High	4	1	1000K	2.6
Mongo-High	4	1	1000K	2.6

Impact on application when reclamation above soft limits





Reached desired throughputs when no pressure (100s)
Higher priority containers are affected negatively

Fig-20: Plots for analysis on application throughputs when all containers are ***above SL***

Reclamation in derivative cloud above soft limits

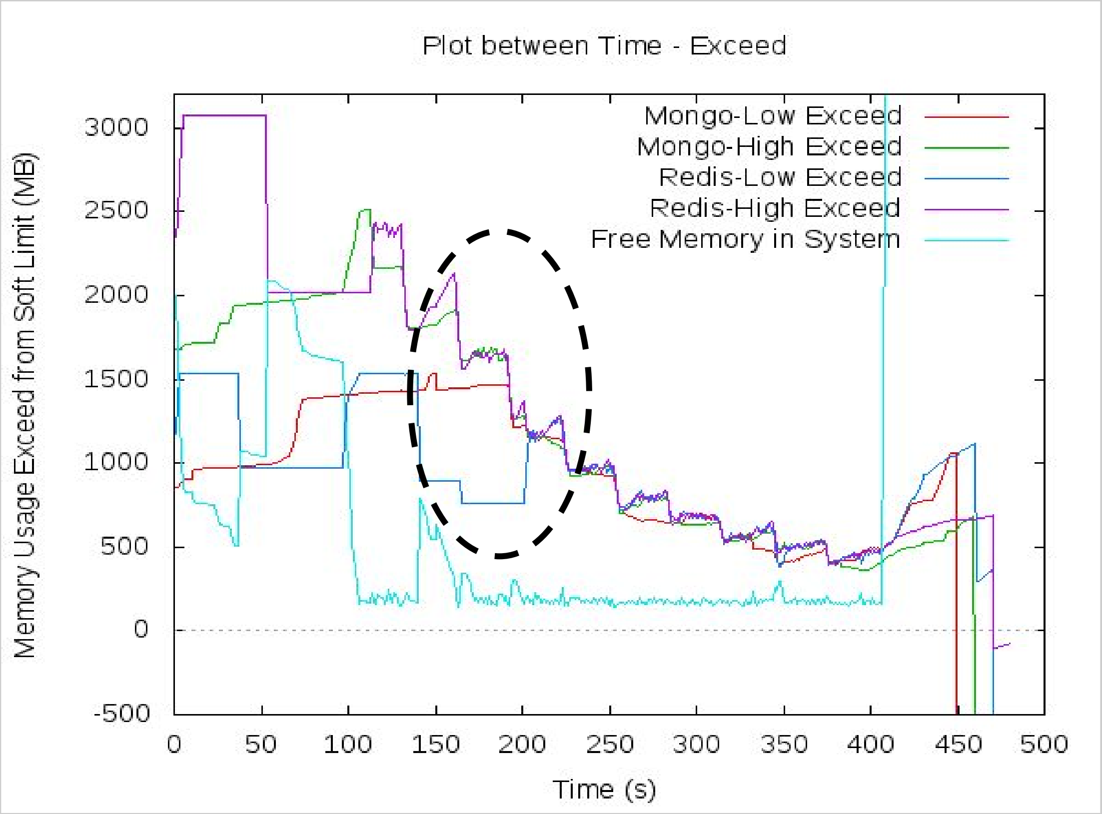
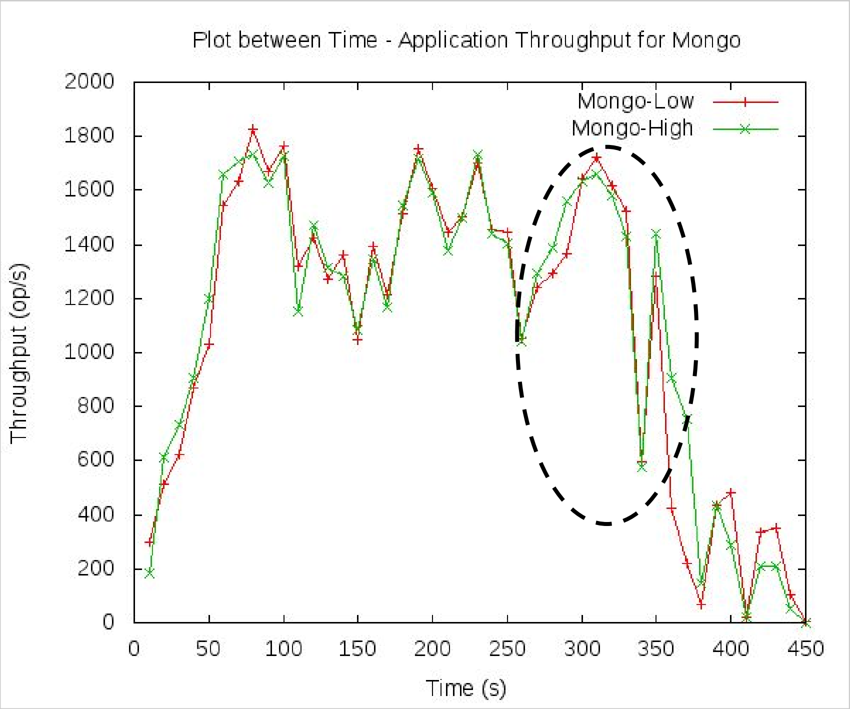
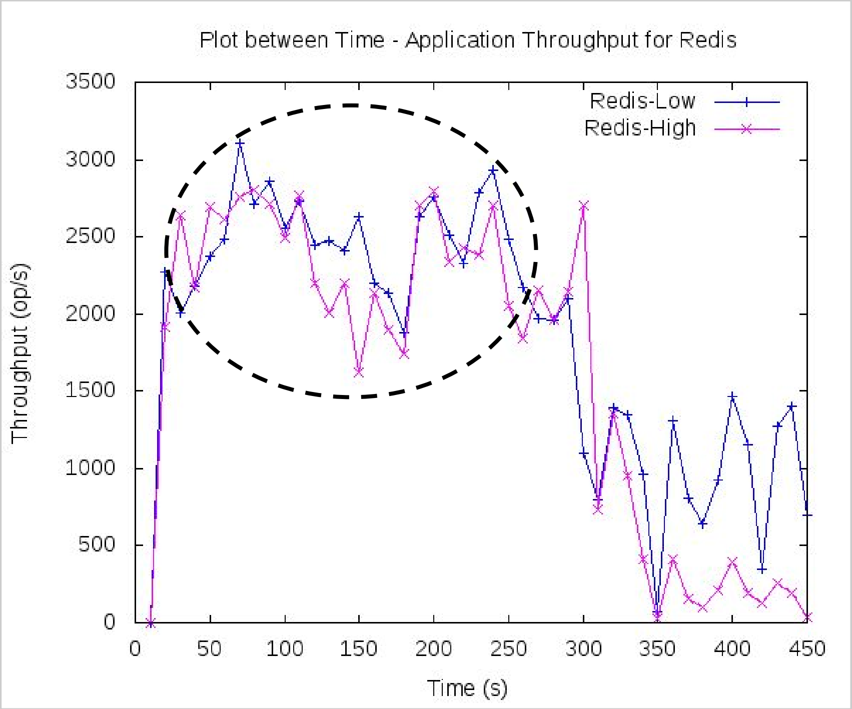


Fig-21: Plot for analysis on reclamation when all containers are ***above SL***

Containers with different exceeds equalize and then reclaim alternatively

Impact on application when reclamation below soft limits





Throughputs vary in a haphazard fashion
Avg. throughputs seem to better, however that's since all containers are actively using memory

Fig-21: Plots for analysis on application throughputs when all containers are ***below SL / No SL***

Reclamation in derivative cloud below soft limits

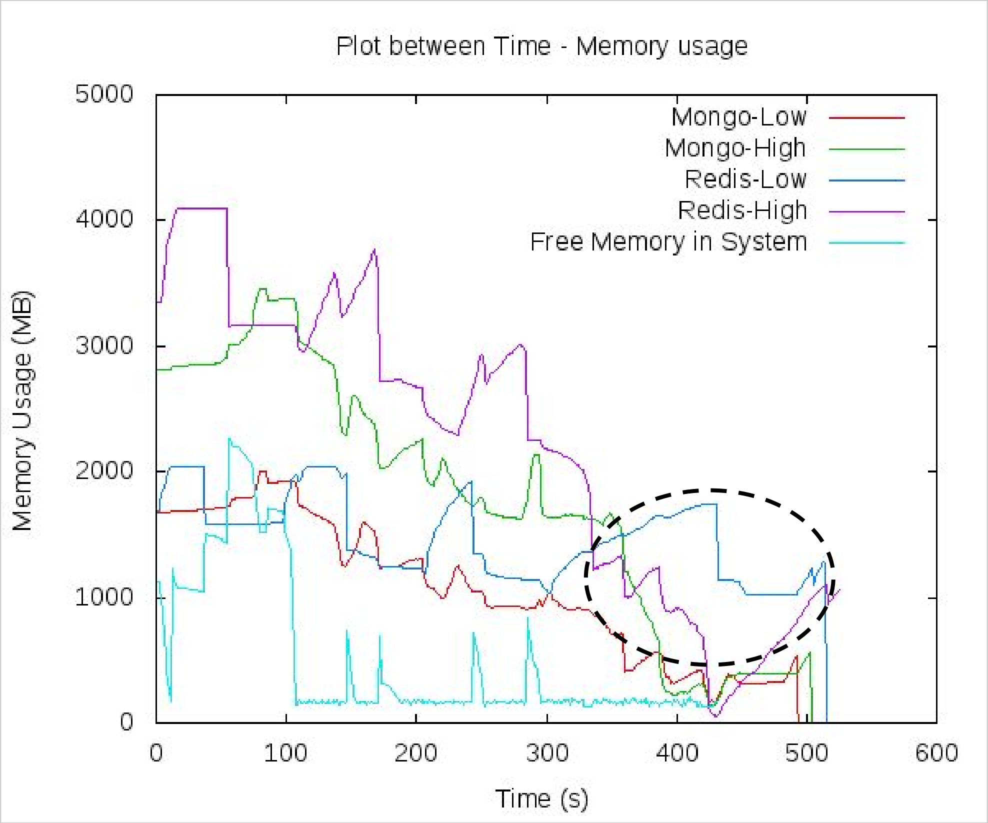


Fig-22: Plot for analysis on reclamation when all containers are ***Below SL / No SL***

No deterministic reclamation

Non-determinism is highly undesirable to cloud providers who promise QOS guarantees

Key inferences

- Reclamation **above SL may impact negatively** while we try to provision containers based on QOS guarantees
- Reclamation **below SL causes non-determinism** which is **not a desired property for a cloud provider**
- **Soft Limit is not a definite guarantee**, it is mere best effort approach

Requirements of a new policy

1. **Differentiated memory reclamation:**
   Provide a differentiated policy where each container can be assigned a priority / is assigned a priority based on its configuration

2. **Deterministic provisioning:**
   The memory allocated to a given container at every instance must be predictable

3. **Elastic Provisioning:**
   Memory must be resizable as and when required

4. **Adaptive provisioning:**
   The policy must be able adapt itself when there is a change in memory in such a way that it doesn't negatively affect containers with higher priority

5. **Strict enforcement of limits:**
   Existing notion of soft limits must be strengthened

Possible Solution Design

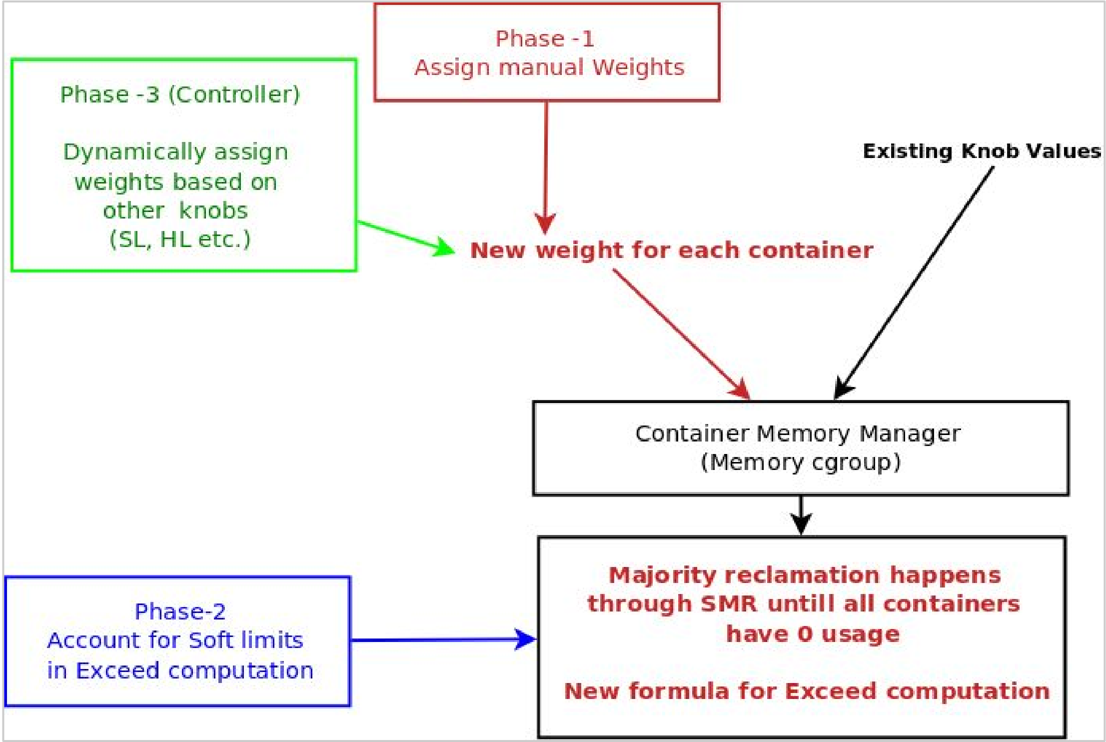


Fig-22: High level overview of proposed design

- Memory Usage for a container (MU)
- Total memory Usage for all containers in system **(TU) = Σ (MU)**
- Relative weight for each container (W)
- Total weight in system **(TW) = Σ (W)**
- **Expected Allocation for a container (EA)**
  = TU x W / TW
- **New exceed** for each container **(EX)**
  = U - EA

Demonstration of exceed value computation

- **Expected Allocation for a container (EA) = TU x W / TW**
- **New exceed** for each container **(EX) = U - EA**
- Consider 3 containers with below given configurations
- U, EA, EX are in MB in our case)

Table: Exceed computation using new formula for 3 containers with same usage and different weights

	Container 1	Container 2	Container 3	
Weight (W)	1	2	3	**TW = 6**
Usage (U)	2000	2000	2000	**TU = 6000**
Expected Allocation (EA)	1000	2000	3000	
New Exceed (EX)	**1000**	**0**	**- 1000**	

Variation of usage over time

Table: Variation of usage over time as pressure increases 500MB each interval ***(-Memory reclaimed in interval)***

Time (t)	Total Usage (TU)	Container 1	Container 2	Container 3
1	6000	2000	2000	2000
2	5500	**(-500)** 1500	2000	2000
3	5000	**(-500)** 1000	2000	2000
4	4500	**(-166)** 833	**(-333)** 1667	2000
5	4000	**(-166)** 666	**(-333)** 1338	2000
6	3500	**(-166)** 500	**(-172)** 1167	**(-166)** 1833
7	3000	500	**(-166)** 1000	**(-366)** 1500
8	2500	**(-83)** 417	**(-166)** 833	**(-250)** 1250

Conclusions & Future Work

Conclusions

1. Initial attempt to understand memory management between containers
2. Proposed a new design to fix existing issues

Future work

1. Evaluate the correctness of the proposed design
2. Implement the new memory management policy in the 3 phases proposed
3. Analyze memory hierarchy in cgroups
4. Proper accounting for shared memory pages
5. Explore other resource controllers
6. **End goal** is to provide an **adaptive resource provisioning** framework **for containers**

References

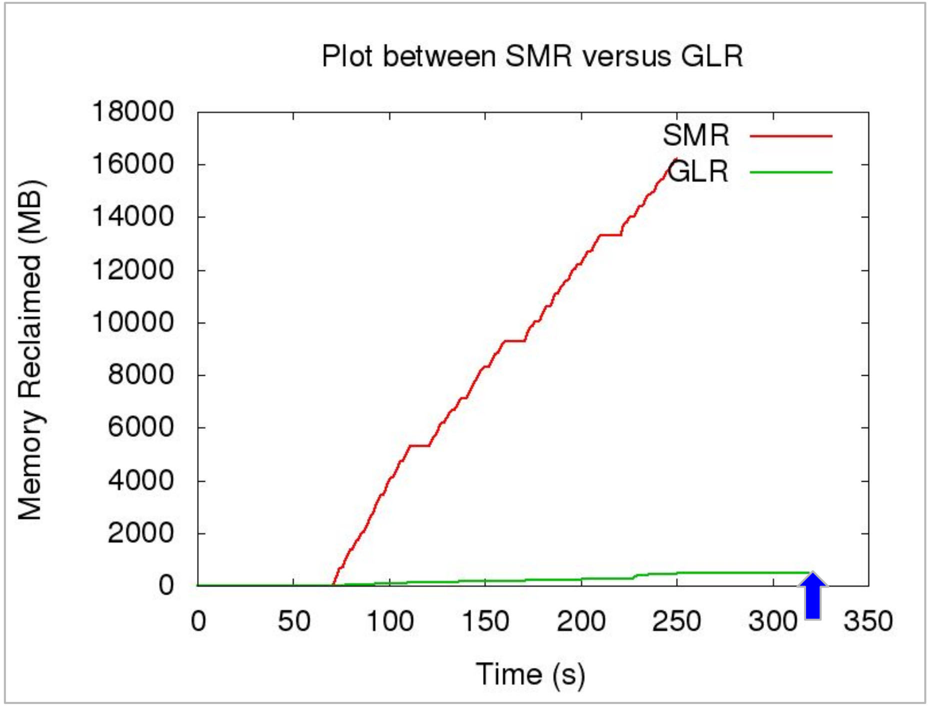
1. C. A. Waldspurger, “Memory resource management in vmware esx server,” ACM SIGOPS Operating Systems Review, vol. 36, no. SI, pp. 181–194, 2002
2. Wang, Yufeng, Chiu C. Tan, and Ningfang Mi. "Using elasticity to improve inline data deduplication storage systems." 2014 IEEE 7th International Conference on Cloud Computing. IEEE, 2014.
3. G. Molt ́o, M. Caballer, E. Romero, and C. de Alfonso, “Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements,” Procedia Computer Science, vol. 18, pp. 159–168, 2013
4. D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, “Overdriver: Handling memory overload in an oversubscribed cloud,” in ACM SIGPLAN Notices, vol. 46, pp. 205–216, ACM, 2011
5. Gupta, Diwaker, et al. "Difference engine: Harnessing memory redundancy in virtual machines." *Communications of the ACM* 53.10 (2010): 85-93.
6. V. Venkatesan, W. Qingsong, and Y. Tay, “Ex-tmem: Extending transcendent memory with non-volatile memory for virtual machines,” in High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on, pp. 966–973, IEEE, 2014
7. P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, “Spotcheck: Designing a derivative iaas cloud on the spot market,” in Proceedings of the Tenth European Conference on Computer Systems, p. 16, ACM, 2015
8. Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, “Cloudscale: elastic resource scaling for multi-tenant cloud systems,” in Proceedings of the 2nd ACM Symposium on Cloud Computing, p. 5, ACM, 2011
9. S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, “Elastic application container: A lightweight approach for cloud resource provisioning,” in 2012 IEEE 26th International Conference on Advanced Information Networking and Applications, pp. 15–22, IEEE, 2012

References

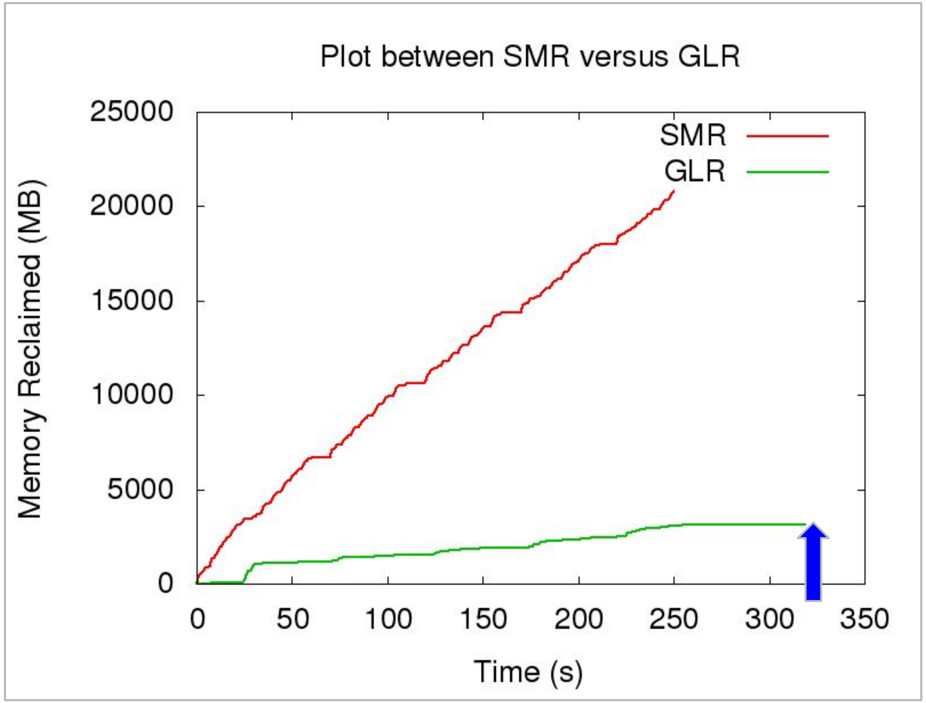
10. R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, “The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds,” Future Generation Computer Systems, vol. 28, no. 6, pp. 861–870, 2012
11. T. D¨ornemann, E. Juhnke, and B. Freisleben, “On-demand resource provisioning for bpel workflows using amazon’s elastic compute cloud,” in Cluster Computing and the Grid, 2009. CCGRID’09. 9th IEEE/ACM International Symposium on, pp. 140–147, IEEE, 2009.
12. K. Agarwal, B. Jain, and D. E. Porter, “Containing the hype,” in Proceedings of the 6th Asia-Pacific Workshop on Systems, p. 8, ACM, 2015
13. W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On, pp. 171–172, IEEE, 2015
14. R. Morabito, J. Kjallman, and M. Komu, “Hypervisors vs. lightweight virtualization: a performance comparison,” in Cloud Engineering (IC2E), 2015 IEEE International Conference on, pp. 386–393, IEEE, 2015
15. D. Beserra, E. D. Moreno, P. Takako Endo, J. Barreto, D. Sadok, and S. Fernandes, “Performance analysis of lxc for hpc environments,” in Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference on, pp. 358–363, IEEE, 2015.
16. M. S. Rathore, M. Hidell, and P. Sjodin, “Kvm vs. lxc: comparing performance and isolation of hardware-assisted virtual routers,” American Journal of Networks and Communications, vol. 2, no. 4, pp. 88–96, 2013.
17. “Stress workload generator.” http://people.seas.harvard.edu/~apw/stress/.
18. “Mongodb.” https://docs.mongodb.com/v3.2/.
19. “Redis in-memory key-value store.” http://redis.io/.
20. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with ycsb,” in Proceedings of the 1st ACM symposium on Cloud computing, pp. 143–154, ACM, 2010.
21. Jin, Hao, et al. "Efficient VM placement with multiple deterministic and stochastic resources in data centers." Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE, 2012.

Backup Slides

Lesser memory pressure



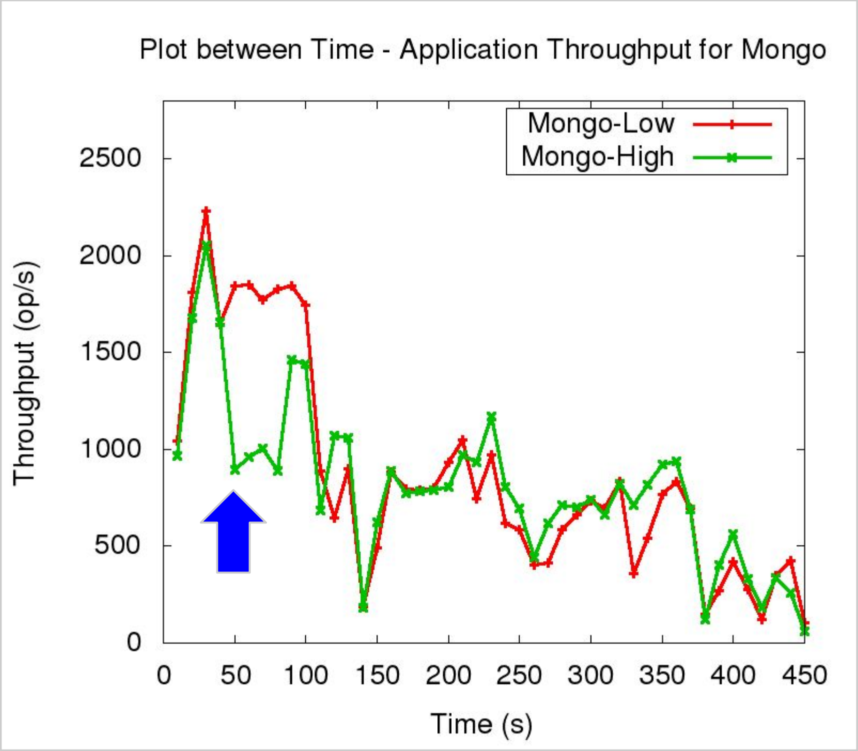
Higher memory pressure



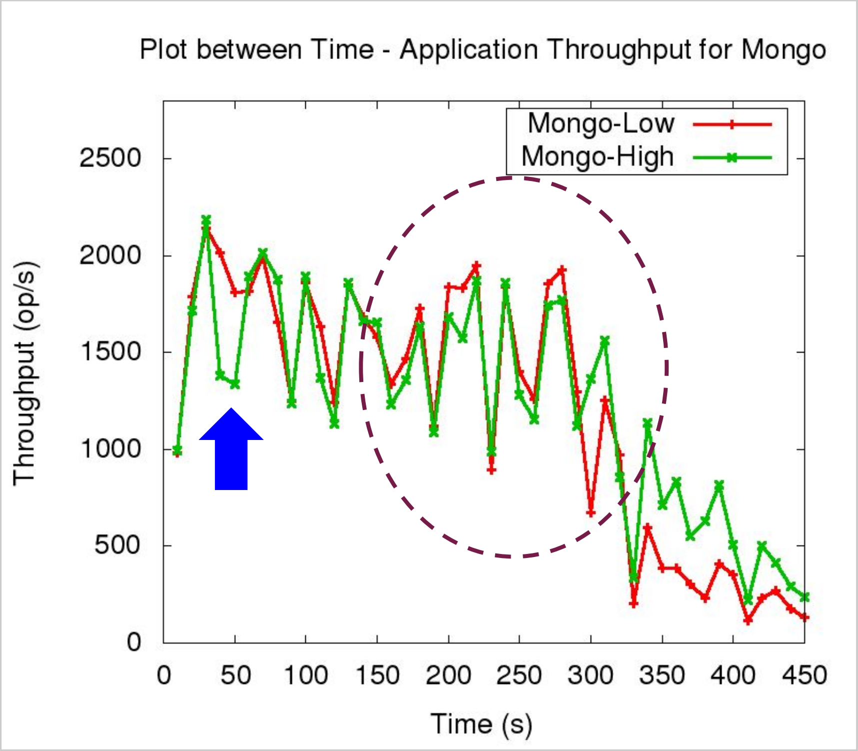
Lesser SMR, More GLR

Fig-14: Plots for analysis of reclamation when ***system memory pressure increases***

Above SL



Below SL / No SL



Low

SL: 0.5 GB Avg. Usage: 1.3 GB

High

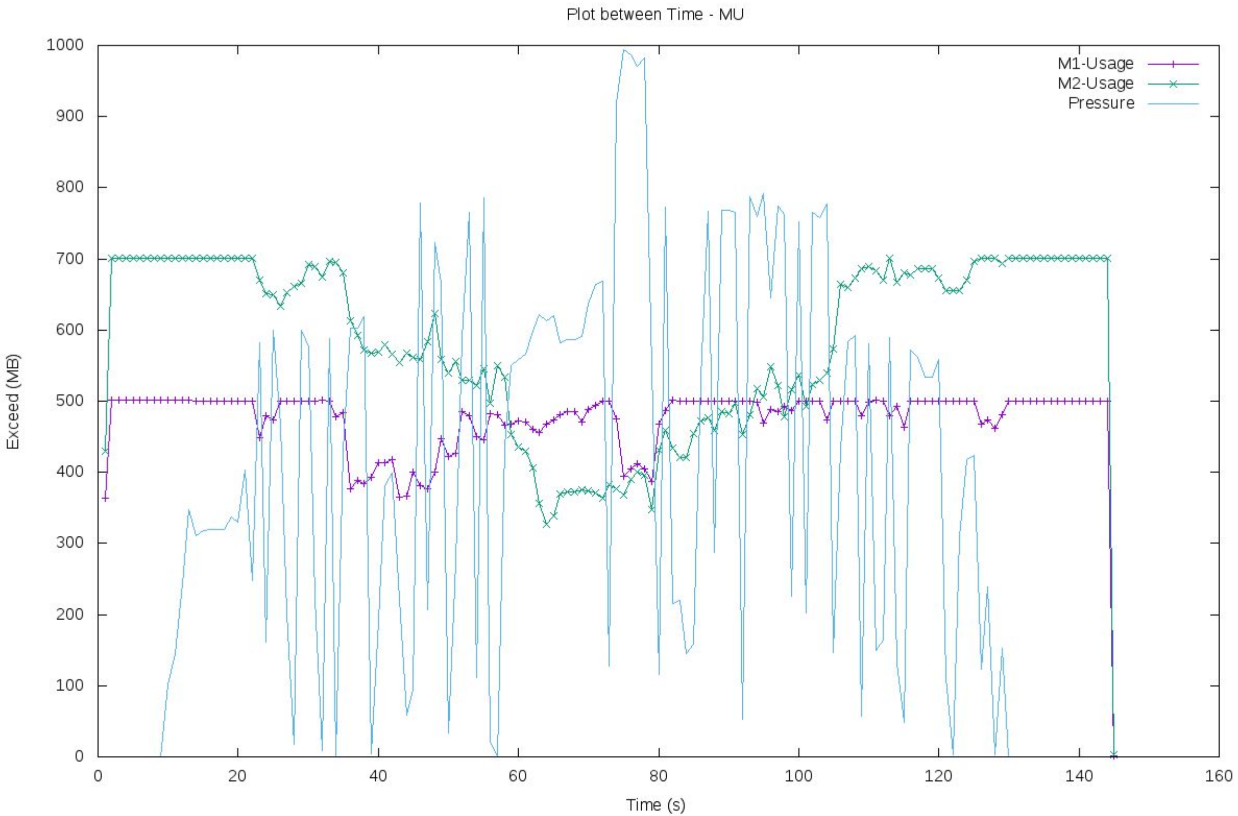
SL: 1 GB Avg. Usage: 2.6 GB

Negatively impacts containers with higher allocations

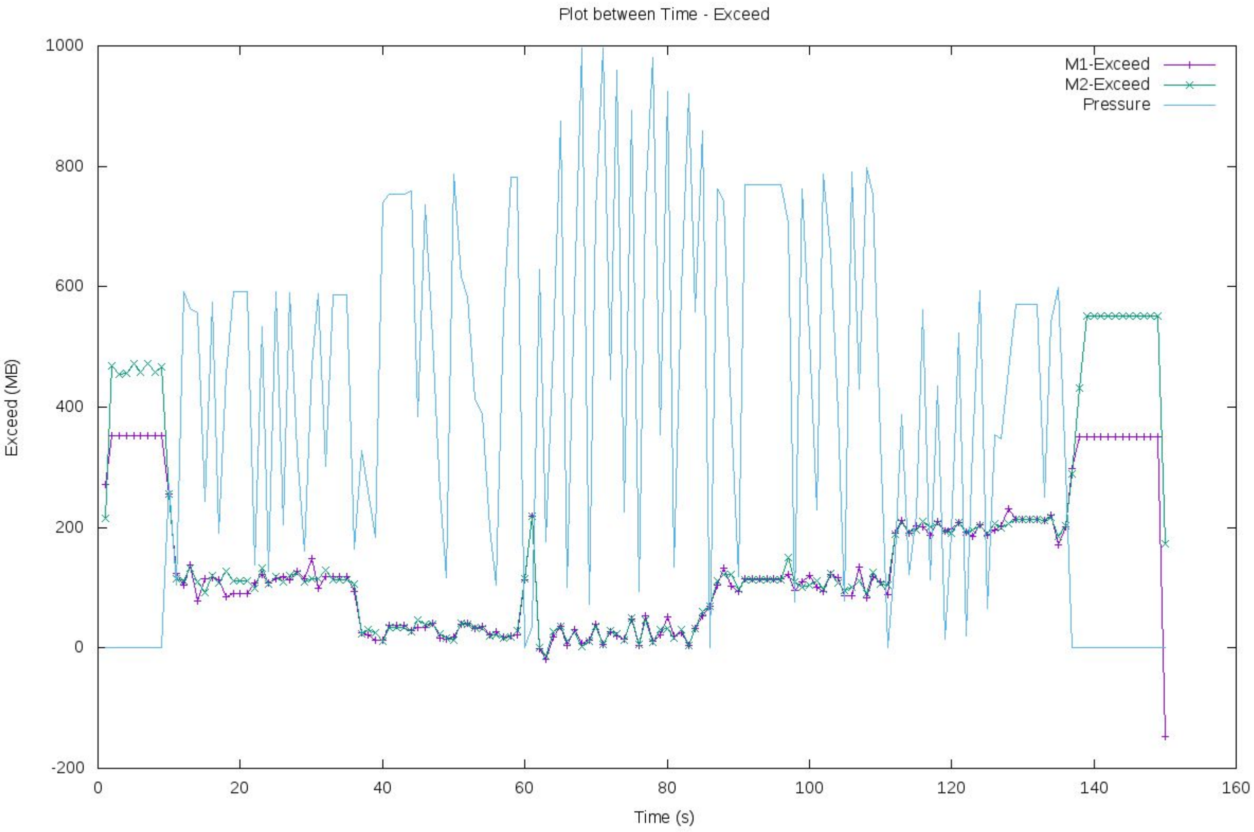
Non deterministic performance

*Fig-18: Plots to understand **impact on applications running inside** prioritized containers*

Reassignment without SL



Reassignment with SL



Reassignment without SL