Managing Vertical Memory Elasticity in Containers

Carlos Henrique Nicodemus, Cristina Boeres and Vinod Rebello

Instituto de Computação – Universidade Federal Fluminense – Brazil

{boeres,vinod}@ic.uff.br, {maria-cristina.boeres,vinod.rebello}@inria.fr

Paper presented at

13th IEEE/ACM International Conference on Utility and Cloud Computing, 2020

https://doi.org/10.1109/UCC48980.2020.00032

SGC LAB SMART GRID COMPUTING LAB

Instituto de Computação uff

Talk Outline

- Motivation
- Horizontal vs Vertical Container Elasticity
- Related Work
- VEMoC - Vertical Elasticity Management of Containers
- Evaluation Results
- Conclusions and Future Work

Motivation

- Latest research (2018) suggests that the energy consumption of data centres (DCs) accounts for 1% of the energy produced worldwide
  - Although consumption has been rising over the last 2 decades, this percentage is the same as it was in 2010!
  - Due in part to the transitioning to more efficient cloud DCs
- Commercial cloud DCs have strong financial incentives to focus on optimising efficiency
  - While they have achieved significant gains over the last decade, it is getting harder
- Smaller edge DCs have fewer available resources and are thus driven to support multi-tenancy and higher degrees of resource sharing

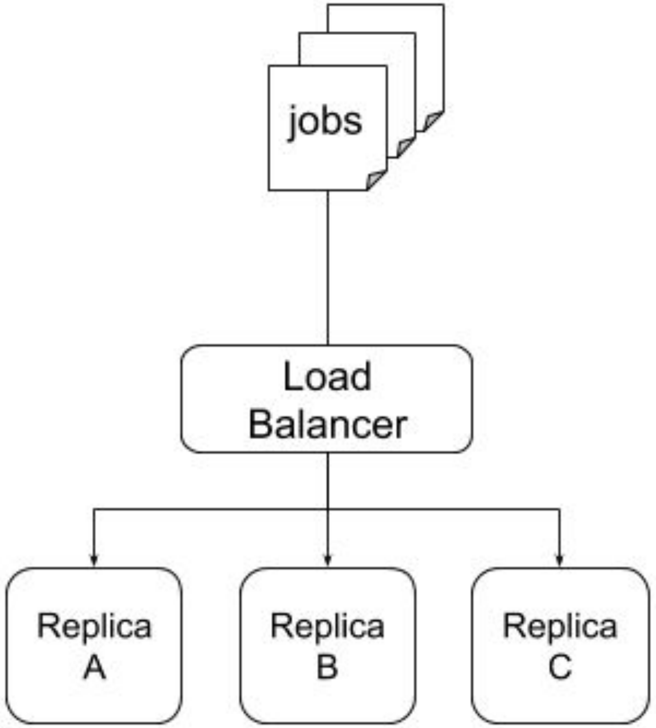
Motivation

- One aspect of efficiency in cloud computing is resource allocation
  - How much of each physical resource should be allocated to each hosted virtual environment?
  - Very difficult to answer without prior knowledge of the application's behavior.
- Both under- and over-provisioning lead to problems
  - Deterioration in performance and/or possible malfunctions in the application.
    - Unnecessary additional costs for the user
  - Idle or underutilized resources on the side of the provider.
- Different applications have different demands that generally vary over time
  - Elastic environments expand and contract allocations to meet demand.

Instituto de Computação

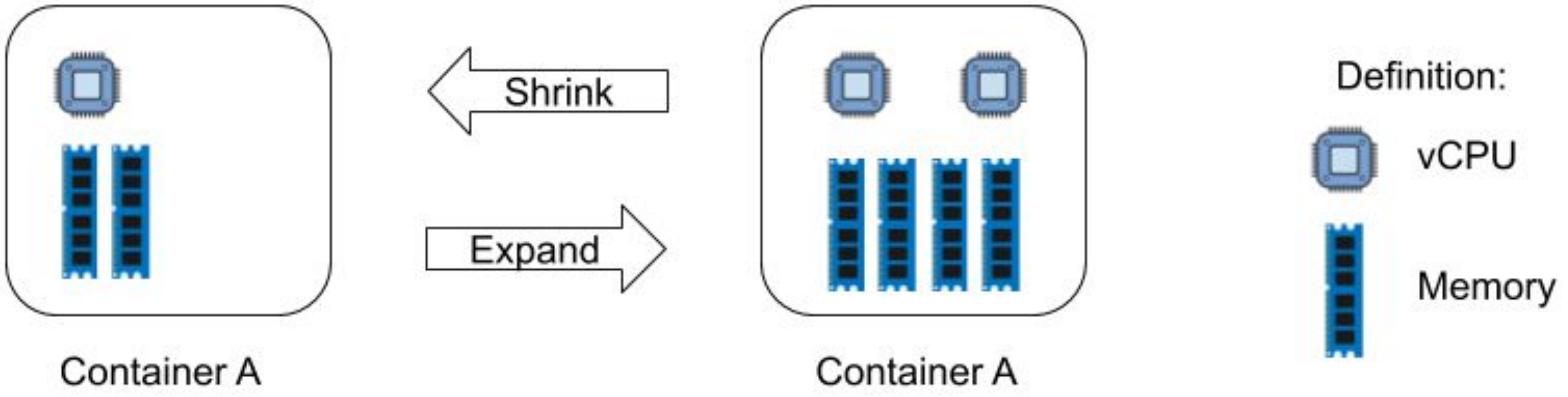
Horizontal Resource Elasticity

- Horizontal elasticity is achieved by replicating containers
  - Appropriate for **online** applications compatible with replication
  - Used extensively by enterprise cloud orchestrators
  - Focuses more on maintaining a given service QoS than maximizing resource utilization efficiency



Vertical Resource Elasticity

- Expand or shrink the resources allocated to a single container
  - Involves changing the resource allocation limits
  - Changes are made at runtime, without having to stop and restart the container
- Known to be useful for **non-distributed** applications



Challenges/Goals

- Improving resource utilization and maximising throughput are two of many goals service providers strive for to reduce operating costs
- While containers consume resources elastically, frameworks are still required to:
  - ***Allocate*** resources according to availability, and;
  - ***Limit*** resource allocations to avoid interference.
- This work aims to manage vertical memory elasticity in containers
  - "Task scheduling" and "Resource allocation" in unison
  - To help providers increase server utilization without incurring significant degradations in performance of individual co-allocated containers

Instituto de Computação uff

Related Work

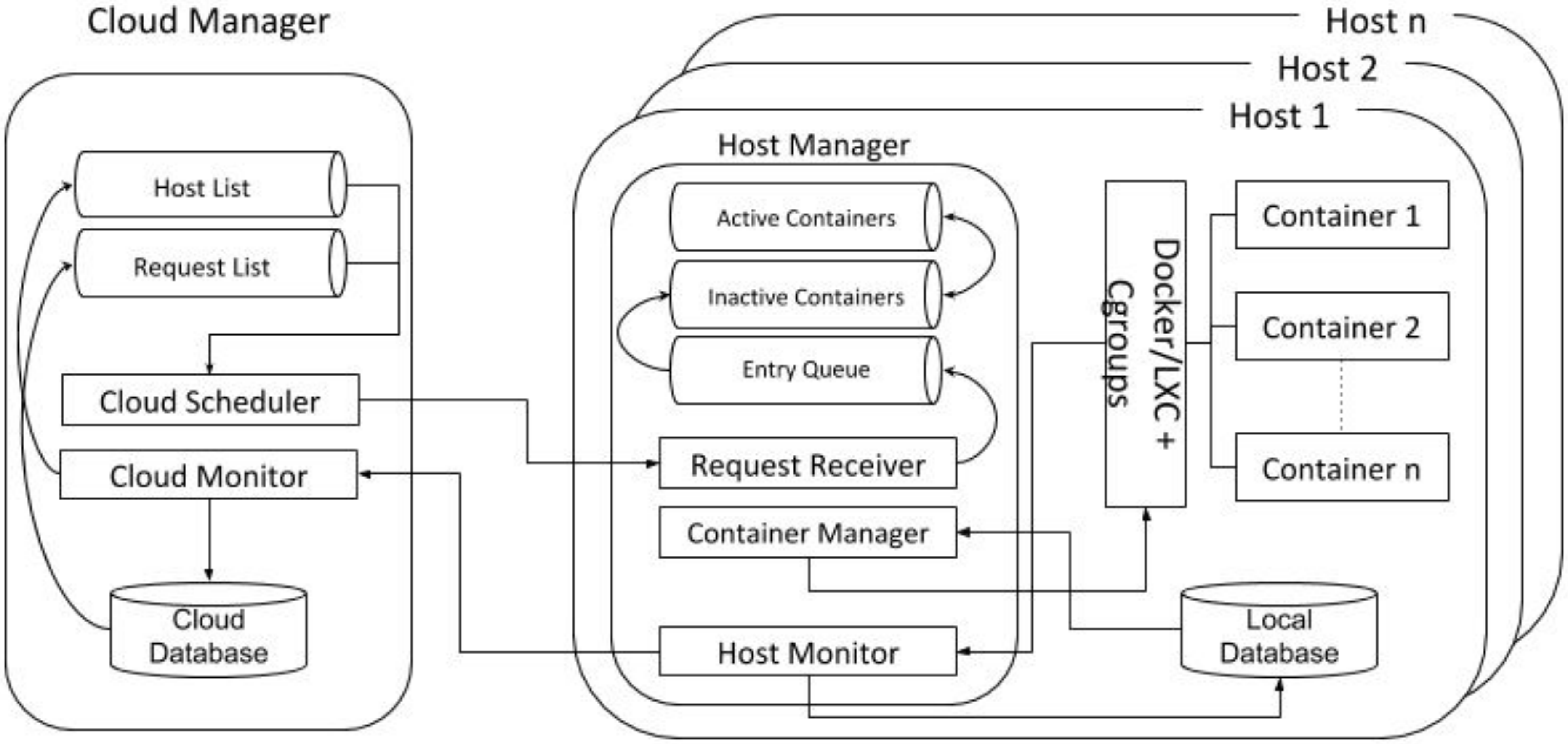
- Enterprise Orchestrators: Docker swarm, Kubernetes, Openshift
  - Focus mostly on horizontal resource elasticity
  - Need user interaction and environment configuration
- In the literature:
  - Vertical Elasticity of Memory based on upper/lower threshold limits;
  - With fixed elasticity adjustment ratios;
  - Long scheduling cycles (> 20 seconds) can mean approaches are more susceptible to making decisions too late.

Vertical Elasticity Management of Containers - VEMoC

- An architecture to manage the life cycles of co-located containers
- This paper focuses on the distribution of host memory
  - Manipulating *Container Memory Limit* (CML) at runtime
- The predicted memory requirement of container is based on:
  - Fine-grained monitoring of container and host metrics
  - Optimised use of rates of changes to determine consumption trends
- If host memory becomes scarce or insufficient to meet demand
  - Containers may "collaborate" by donating some (or be suspended and donate all) of their memory allocation to others in need

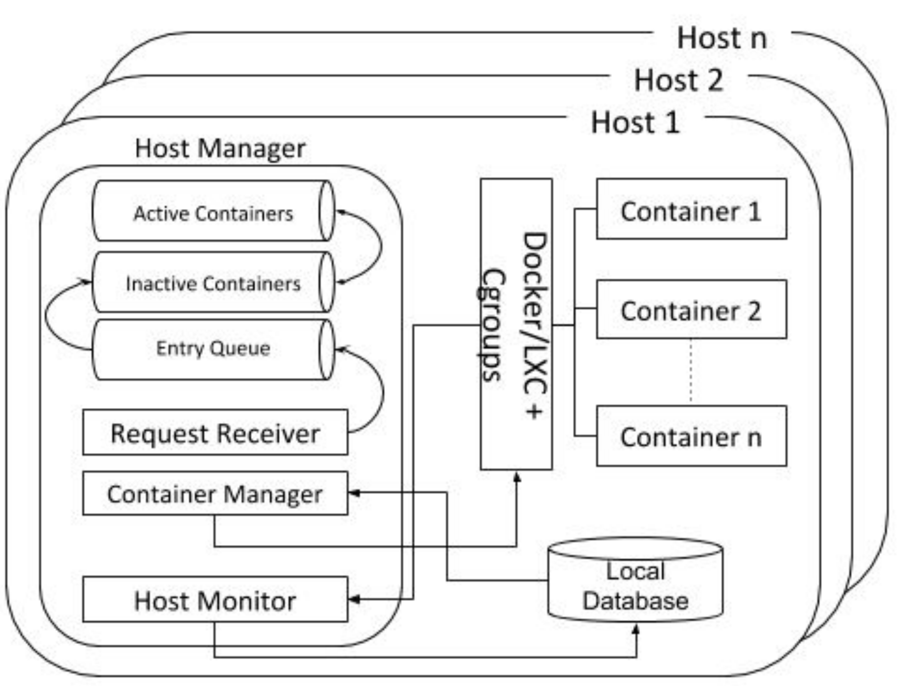
Instituto de Computação

VEMoC Architecture



Instituto de Computação uff

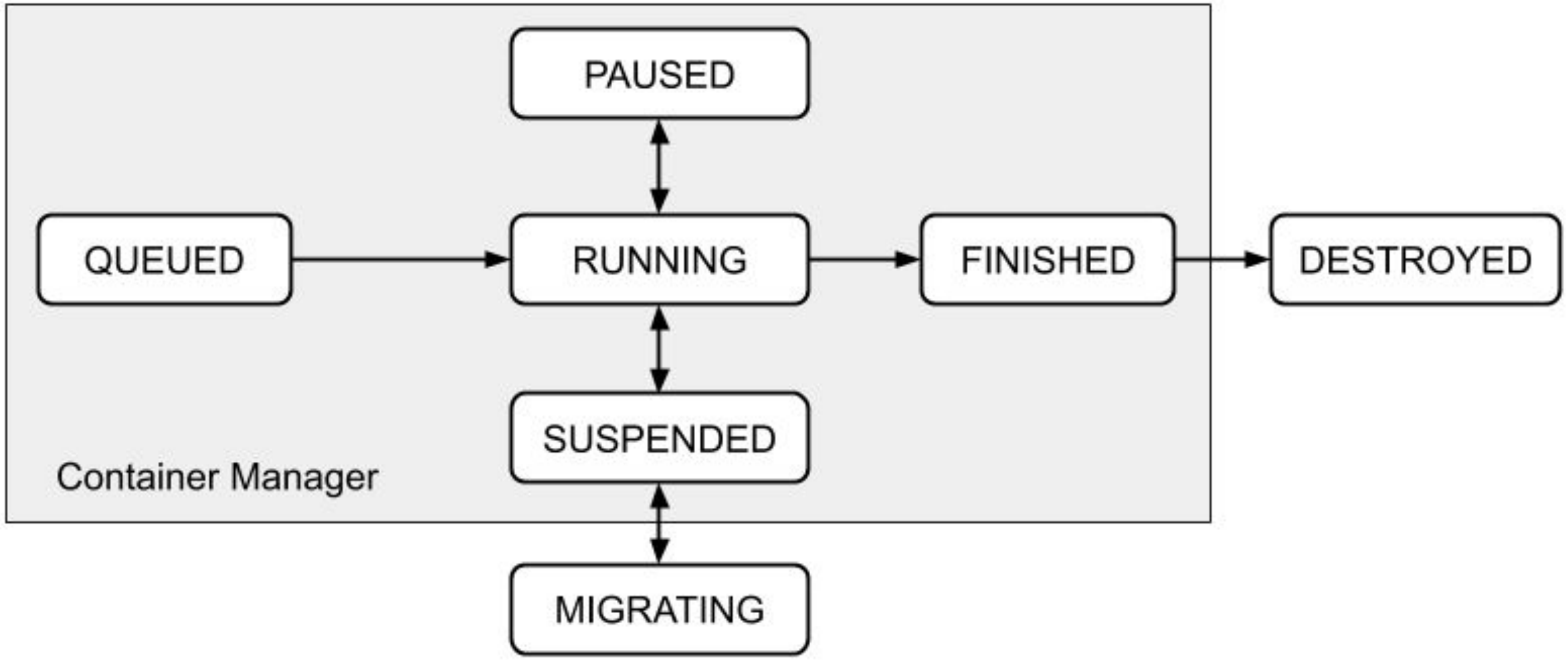
Host Manager



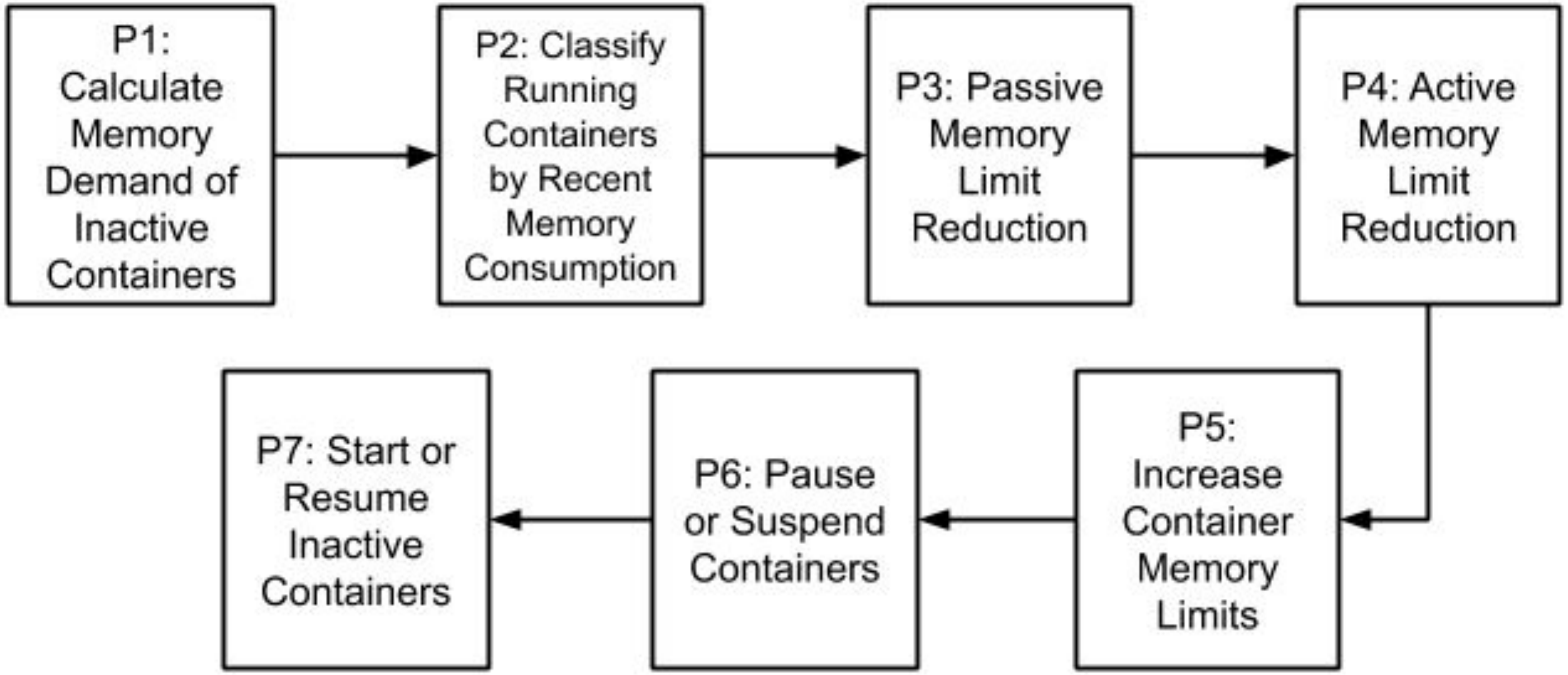
- Host Monitor
  - Obtains monitoring data;
- Request Receiver
  - Receives new jobs from Cloud Manager;
  - “Creates” and queues the container request
- Container Manager
  - Manages the distribution of the host’s memory amongst containers
  - Manages the life cycle of containers allocated to this host

Instituto de Computação uff

Container Life-Cycle



VEMoC Algorithm



Instituto de Computação

Phase 1

- Receive and queue container requests from Cloud Manager
- Calculates the amount of memory required to start all currently inactive containers
  - This includes containers in the state QUEUED or SUSPENDED
- Calculate the amount of non allocated memory is available on the host

Instituto de Computação uff

Phase 2

- Classifies active containers by extrapolating their memory consumption from the previous scheduling interval
  - Containers are classified as RISING, FALLING or STABLE based on their major page faults, page-in, page-out rates and memory and swap usage
  - Predicts the amount of memory expected to be consumed by a container until the CML can be updated again during the following scheduling interval

Instituto de Computação

Phases 3 and 4

- In general, CML is defined to include a reserve to cover any unforeseen spike in memory consumption during the next scheduling interval
- Phase 3 remove any over-estimation of the CML of STABLE and FALLING containers
- Phase 4 repossess memory from candidate STABLE containers
  - Reducing the CML below a container’s memory consumption to force it to swap-out some of its inactive memory
  - The suitability of containers is determined soon after they become STABLE

Instituto de Computação uff

Phase 5

- Phases 1 and 2 determine the total memory demand for current scheduling interval
- If the host does not have enough available memory to meet the demand, Phases 3 and 4 tries to extract additional memory
- Phase 5 thus distributes this memory to those containers that need theirs CMLs to be increased in accordance to the following priority:
  1. Paused Containers;
  2. Containers that brought in pages from swap during the last interval;
  3. Other containers whose consumption is expected to exceed their current CML.

Phase 6

- If Phase 5 cannot meet the needs of the active containers, Phase 6 considers preempting containers
- Of the remaining unsatisfied containers, some may need to be:
  - Paused to prevent excessive performance degradation due to swap utilisation, or be;
  - Suspended in order to free up enough memory for other containers in need.

Phase 7

- Considers initiating inactive containers if there was no need for Phase 6
- First, available memory permitting, VEMoC attempts to
  - Resume suspended containers, then
  - Start queued containers awaiting execution
- Priority in each group, is given to the container with longest runtime or wait time

Instituto de Computação uff

Experiments

- Our tests were executed on a host with:
  - 2x 6 core/12 threads Intel Xeon X5650 @ 2.67GHz;
  - 24 GiB of DDR3 RAM memory;
  - 8 GiB of swap memory;
  - 2 TB of SATA disk;
- CentOS Linux 7.7, kernel 4.20.11 and LXC 3.2.1
- Executing two synthetic jobs:
  - J1 – iterates over the elements a given vector of size s
  - J2 – similar to J1, but exploits data locality by dividing and processing the vector in blocks of size s/n, block by block.

Instituto de Computação

Experiments

- VEMoC performance is compared with three commonly adopted forms of defining CMLs (loosely based on Kubernetes QoS terminology):
  - **Guaranteed** - The CML is set *a priori* to the maximum amount of memory required, and the job can only be submitted when that amount is available - effectively, the required amount of memory is pre-reserved;
  - **Fair Share** - Prior to execution, the available memory is divided equally among the jobs to be executed;
  - **Best Effort** - Runs jobs as they arrive if the minimum memory limit is available. During execution, containers can use whatever free memory is available, up to their maximum memory limit.

Instituto de Computação

Experiments

- The comparisons are based on five metrics:
  - **Total Scenario Execution Time** (**TSET**) is the wallclock time from first job submission to end of the last job’s execution, in seconds;
  - **Average Job Turnaround Time** (**AJTT**) in seconds;
  - **Average Memory Utilization** (**MemUtil**) is the average of the average percentage memory utilisation of each job;
  - **Total Memory-Time Product** (**TMTP**) is a cost metric (in millions of page-seconds) for clients, which considers the duration the memory was reserved for in that scenario;
  - The **Average Host Memory Utilization** (**AHMU**) percentage represents the effective use of host memory during the scenario's execution, i.e., over the period TSET.

Instituto de Computação uff

Results – Scenario 1

- Executes two J1 jobs of 4GiB, with an interval of 50 seconds between them, on a host with 6GiB of available memory
- There is not enough memory, so who gets what?

	TSET (s)	AJTT (s)	MemUtil (%)	TMTP	AHMU (%)
Guaranteed	843.2 (4.9%)	606.3 (2.6%)	87.1 (10.1%)	886.98 (10.9%)	58.1 (5.7%)
Fair Share	1245.2 (54.9%)	1193.2 (101.8%)	96.4 (0.5%)	1876.71 (134.6%)	92.2 (49.7%)
Best Effort	1284.0 (59.7%)	1236.9 (109.2%)	95.2 (1.8%)	2019.6 (152.5%)	92.8 (50.6%)
VEMoC	804.0	591.2	96.9	799.8	61.6

Instituto de Computação

Results – Scenario 2

- Executes a J2 job of 4GiB, followed by a J1 job also of 4GiB, 100 seconds later, on a host with 6 GiB of available memory
- Still not enough memory, can we get some from thin air?

	TSET (s)	AJTT (s)	MemUtil (%)	TMTP	AHMU (%)
Guaranteed	841.1 (58.8%)	582.8 (36.7%)	73.1 (22.3%)	884.8 (50.6%)	48.8 (28.0%)
Fair Share	793.2 (49.8%)	561.8 (31.8%)	83.1 (11.7%)	883.6 (50.4%)	60.5 (10.8%)
Best Effort	637.9 (20.4%)	483.1 (13.2%)	71.1 (24.4%)	1002.7 (70.7%)	65.3 (3.7%)
VEMoC	529.7	426.2	94.1	587.5	67.8

Instituto de Computação uff

Results – Scenario 3

- Execute a J2 jobs with 4GiB, followed by five J1 jobs with 4GiB, at 10 seconds interval, using the 24 GiB of host memory
- What happens under extreme stress?

	TSET (s)	AJTT (s)	MemUtil (%)	TMTP	AHMU (%)
Guaranteed	867.8 (59.6%)	506.9 (3.1%)	82.6 (11.8%)	2788.4 (5.4%)	42.8 (42.2%)
Fair Share	779.6 (43.4%)	663.6 (35.0%)	84.9 (9.3%)	4093.0 (54.7%)	73.7 (0.4%)
Best Effort	715.1 (31.5%)	543.4 (10.5%)	72.6 (22.4%)	4407.8 (66.7%)	63.0 (14.9%)
VEMoC	543.6	491.7	93.6	2644.9	74.0

Instituto de Computação uff

Conclusions

- VEMoC obtains better utilization:
  - by using page level predicted memory consumption rates;
  - using fine grain vertical elasticity of memory;
  - combining techniques of memory stealing and container preemption.
- VEMoC also demonstrated higher efficiencies for the service provider, and better performances and lower costs for the client.
- As future work, intend to investigate:
  - Alternative libraries that implement container suspension;
  - the impact of scheduling policies on memory utilization efficiencies, and;
  - With VEMoCs elastic management of CPU, integrate CPU throttling with the management of vertical memory elasticity.

Instituto de Computação uff

Acknowledgements

CAPES

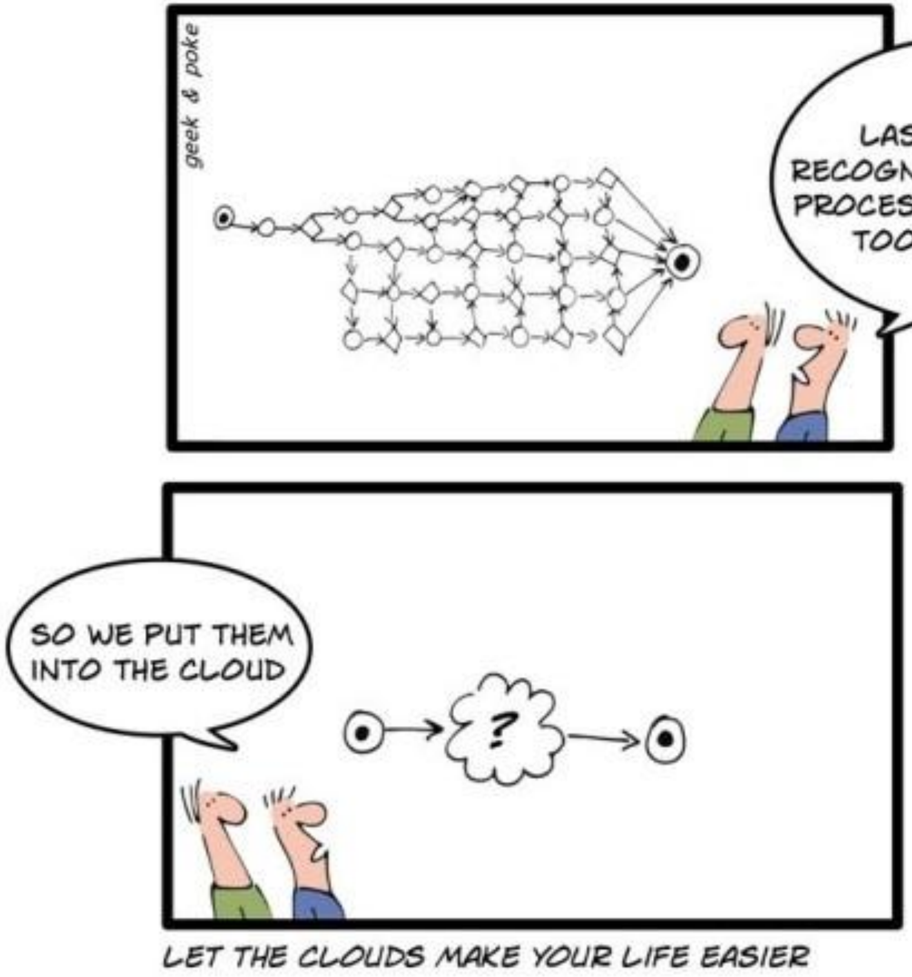
FAPERJ
Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro

PROPPI uff
Pró-Reitoria de Pesquisa, Pós-graduação e Inovação

PRINT
Programa de Internacionalização

Instituto de Computação uff

Any Questions?



Instituto de Computação uff