RACC: Resource Aware Container Consolidation using a Deep Learning Approach

Saurav Nanda,

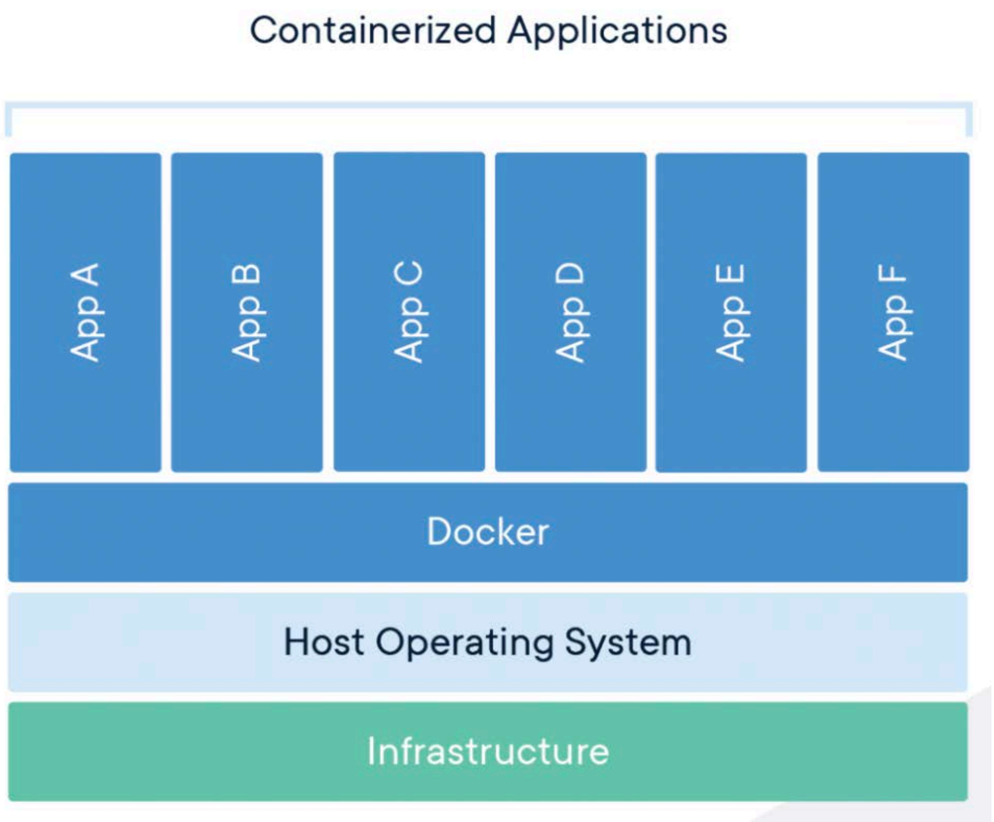
Thomas J. Hacker

Introduction- Container

- Packaged Code + Config + Dependencies
- Lightweight than VM
- Secure – Default isolation

- Example: Docker Image

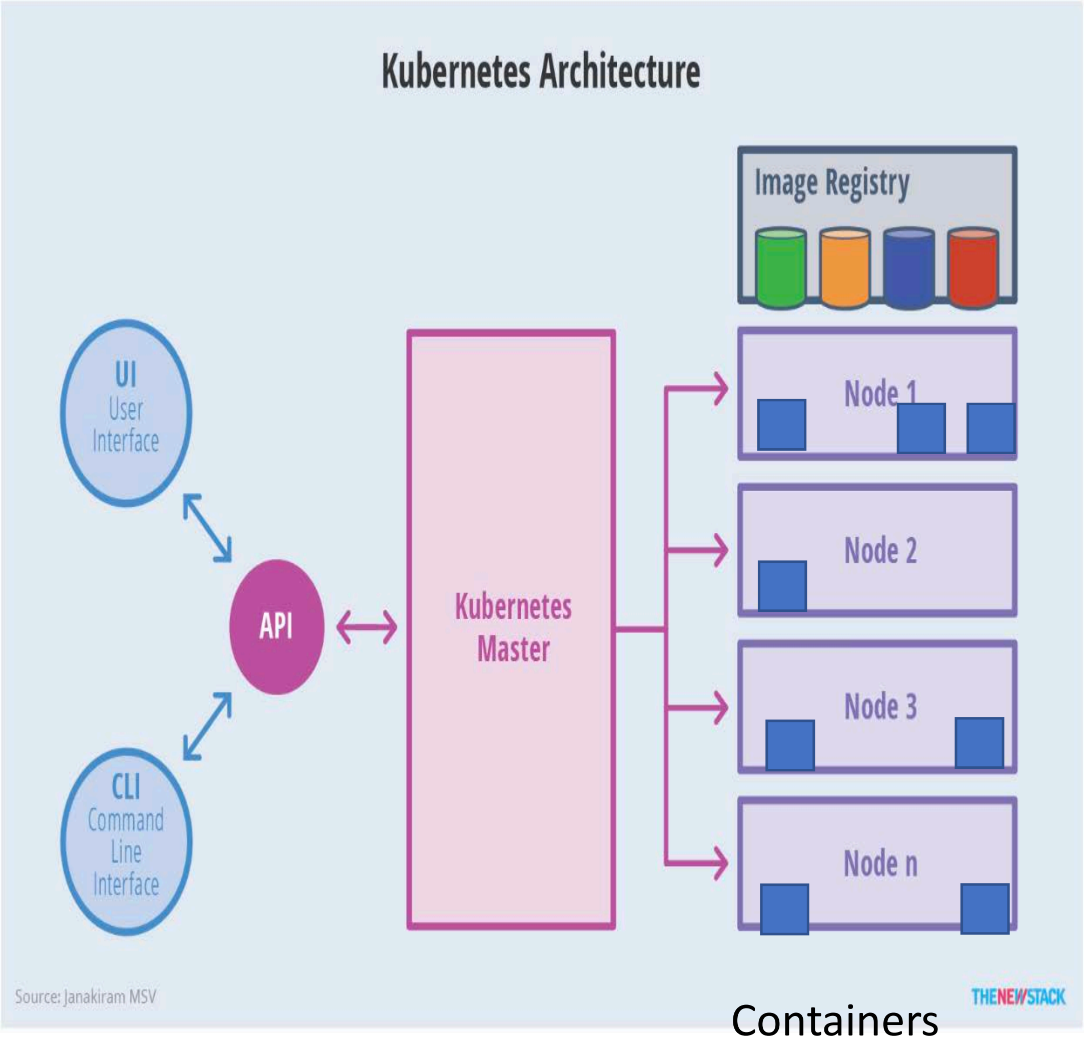
```
FROM debian:stretch-slim
ENV NGINX_VERSION 1.15.11-1~stretch
ENV NJS_VERSION   1.15.11.0.3.0-1~stretch

RUN set -x \
            && apt-get update \
            && apt-get install -y gnupg1 apt-transport-https
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```



Introduction – Resource Optimization

- CaaS (Container as a Service) – pay-as-you-go
- Diverse Resource demands
    - CPU Intensive, Memory Intensive, I/O Intensive, Network Intensive
- Multi-dimensional bin packing – NP Hard
- Heuristics based solutions – First Fit, Best Fit, First Fit decreasing
- Avoid resource fragmentation and over allocation
- Theoretical Model – Takes 30 min for 15 nodes
- Deep Learning based Solution – Fit-for-Packing

Example: Container Scheduler



Source: Janakiram MSV

Containers

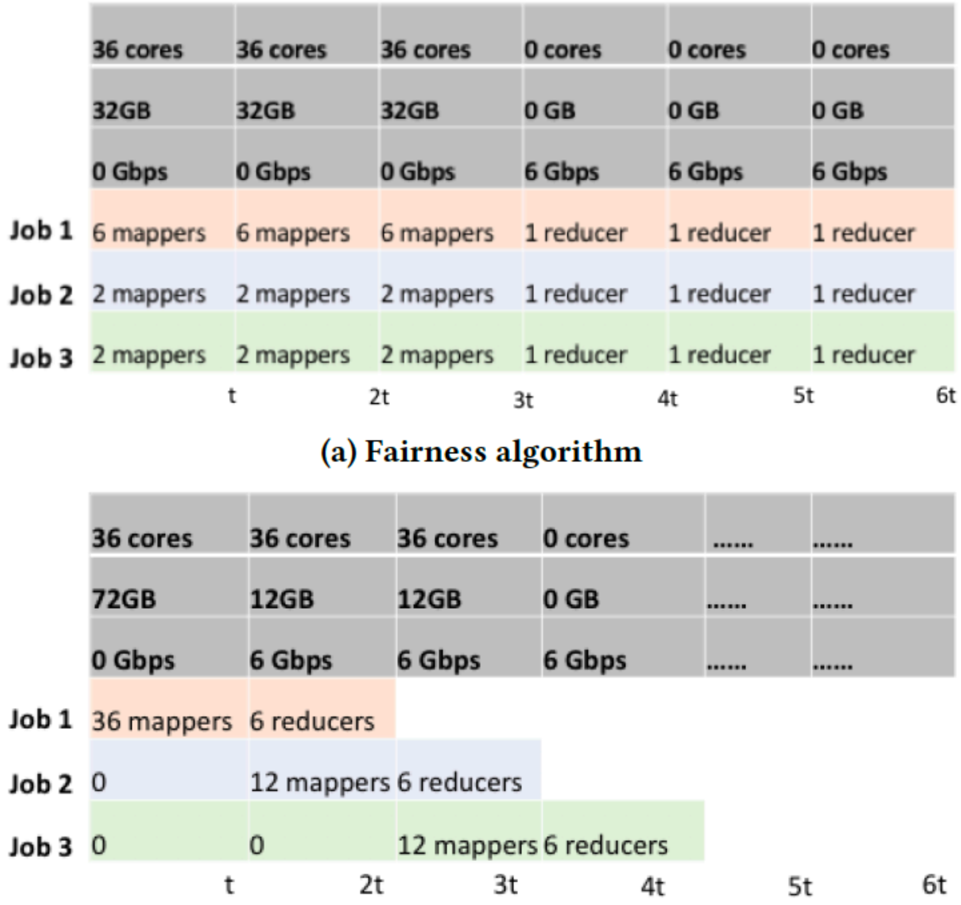
Why pack jobs?

- Machine: CPU cores = 36 , Memory = 7GB, Network Bandwidth = 6Gbps

- Job1 -
  - Mappers – 18, Reducers – 3
  - 1 Mapper: 2 GPU, 4GB Memory
  - 1 Reducer: 2 Gbps network

- Job2 -
  - Mappers – 6, Reducers – 3
  - 1 Mapper: 6 GPU, 2GB Memory
  - 1 Reducer: 2 Gbps network

- Job3 -
  - Mappers – 6, Reducers – 3
  - 1 Mapper: 6 GPU, 2GB Memory
  - 1 Reducer: 2 Gbps network



	36 cores	36 cores	36 cores	0 cores	0 cores	0 cores
	32GB	32GB	32GB	0 GB	0 GB	0 GB
	0 Gbps	0 Gbps	0 Gbps	6 Gbps	6 Gbps	6 Gbps
Job 1	6 mappers	6 mappers	6 mappers	1 reducer	1 reducer	1 reducer
Job 2	2 mappers	2 mappers	2 mappers	1 reducer	1 reducer	1 reducer
Job 3	2 mappers	2 mappers	2 mappers	1 reducer	1 reducer	1 reducer
	t	2t	3t	4t	5t	6t

(a) Fairness algorithm

	36 cores	36 cores	36 cores	0 cores	......	......
	72GB	12GB	12GB	0 GB	......	......
	0 Gbps	6 Gbps	6 Gbps	6 Gbps	......	......
Job 1	36 mappers	6 reducers				
Job 2	0	12 mappers	6 reducers			
Job 3	0	0	12 mappers	6 reducers		
	t	2t	3t	4t	5t	6t

(b) Multi-resource packing algorithm

Figure 1: Comparison of container scheduling based on fairness and multi-resource packing algorithm

Scheduling Framework

- Adaptive learning of resource requirement of job(Jr)
- Monitoring of available resources (Mr)

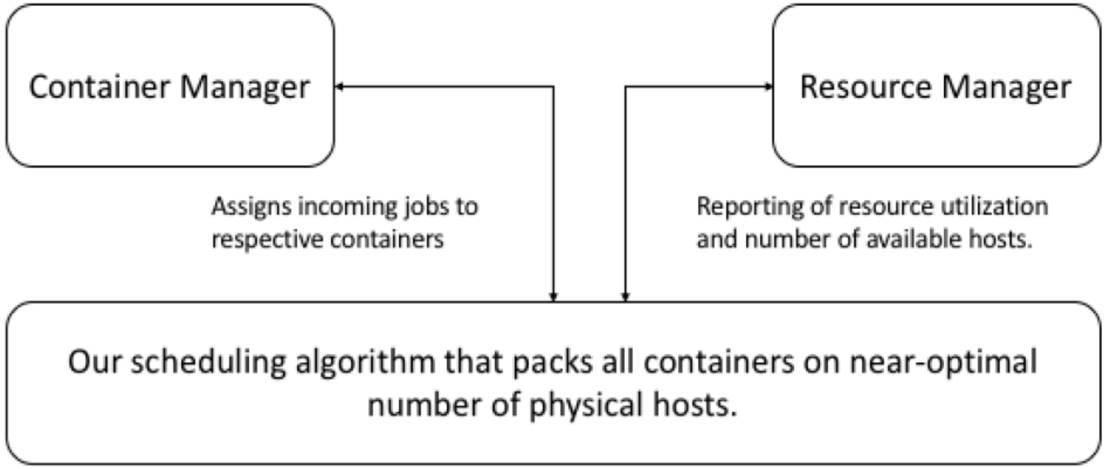

Container Manager
Resource Manager
Assigns incoming jobs to respective containers
Reporting of resource utilization and number of available hosts.
Our scheduling algorithm that packs all containers on near-optimal number of physical hosts.

Figure 2: Overview of System Architecture

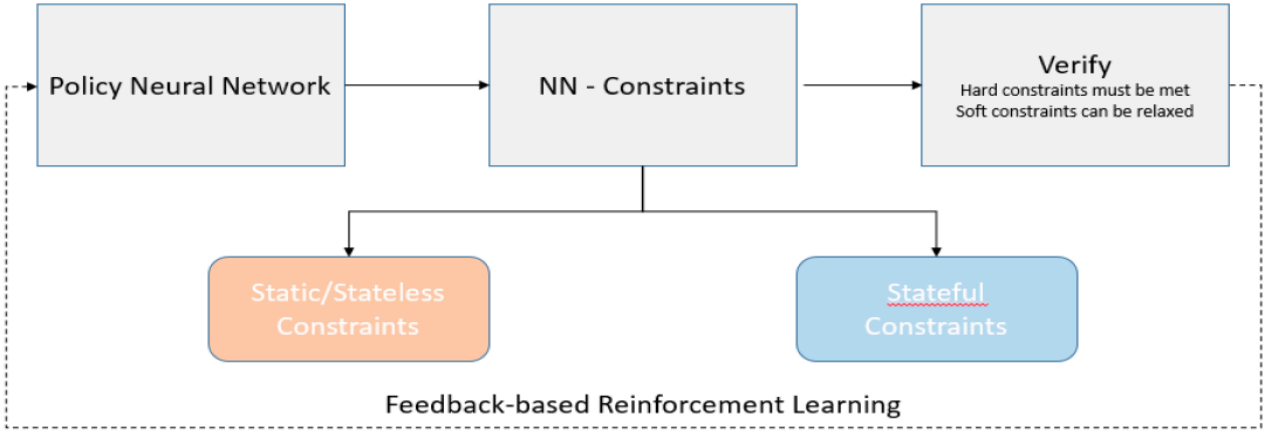

Policy Neural Network
NN - Constraints
Verify
Hard constraints must be met
Soft constraints can be relaxed
Static/Stateless Constraints
Stateful Constraints
Feedback-based Reinforcement Learning

Figure 3: Deep Neural Network based Reinforcement Learning Approach for near-optimal placement of containers.

Constraints: task schedule & resource allocation

- Minimize makespan => Maximize the container consolidation efficiency

$$\sum \alpha_{i,j}^{r,t} \leq C_{i,r} \forall i, t, r$$

- Resource Usage on machine <= capacity

$$0 \leq \alpha_{i,j}^{r,t} \leq D_{j,r} \forall r, i, j, t$$

- Should not exceed maximum requirement

$$\sum_{t=j_{start}}^{j_{end}} \phi_{i,j}^{t} = \left\{ \begin{matrix} j_{duration} \forall\ i \in i_j \\ 0 \ \forall\ other\ machines \end{matrix} \right\}$$

- To avoid preemption – for simplicity

$$j_{duration} = max\left(\frac{A_{j,cpu}}{\sum_t \alpha_{i_j,j}^{cpu,t}}, \frac{A_{j,mem}}{\sum_t \alpha_{i_j,j}^{mem,t}}, \frac{A_{j,dW}}{\sum_t \alpha_{i_j,j}^{dW,t}} \frac{A_{i,j,dR}}{\sum_t \alpha_{i_j,j}^{dR,t}} \right)$$

- Jduration – total job execution time at container j ⍰

$$T_{finish} = max_{containerj \in J} max_{JCT_t} (\phi_{i,j}^{t} > 0)$$

- Job j's finish time

$$P_R = max_{resource_r} \frac{\sum_{i,j \in J} \alpha_{i,j}^{r,t}}{\sum_i C_{i,r}}$$

- Most prominent resource

i – machine,
j - container,
t – discrete time,
α- resource unit,
D – Demand of each container,
Ø – 1 if container j is allocated to machine i at time t
A- allocated
JCT – Job completion time

Results

Job Slowdown = Tcompletion / Texpected

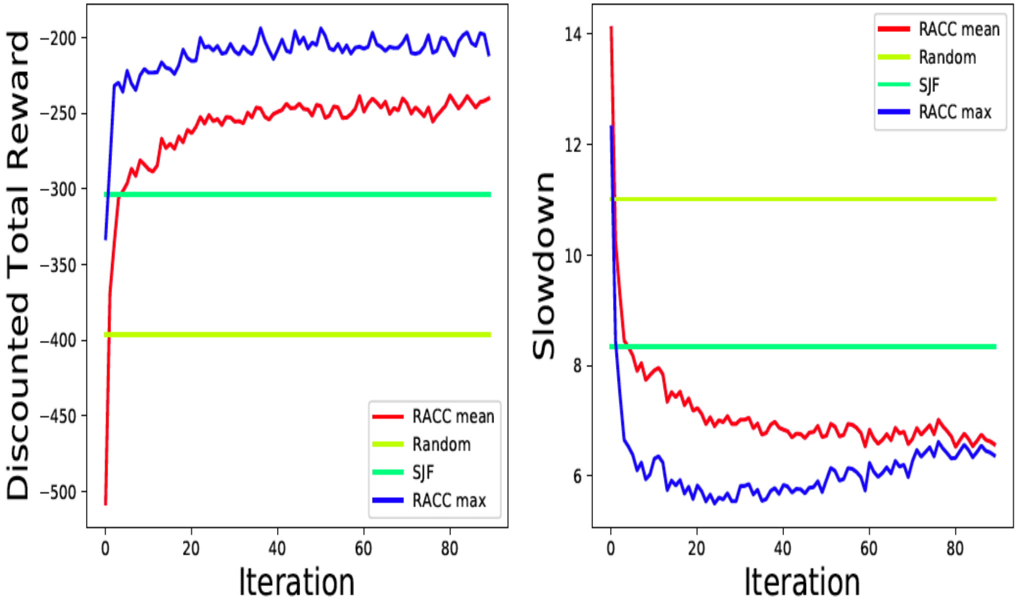
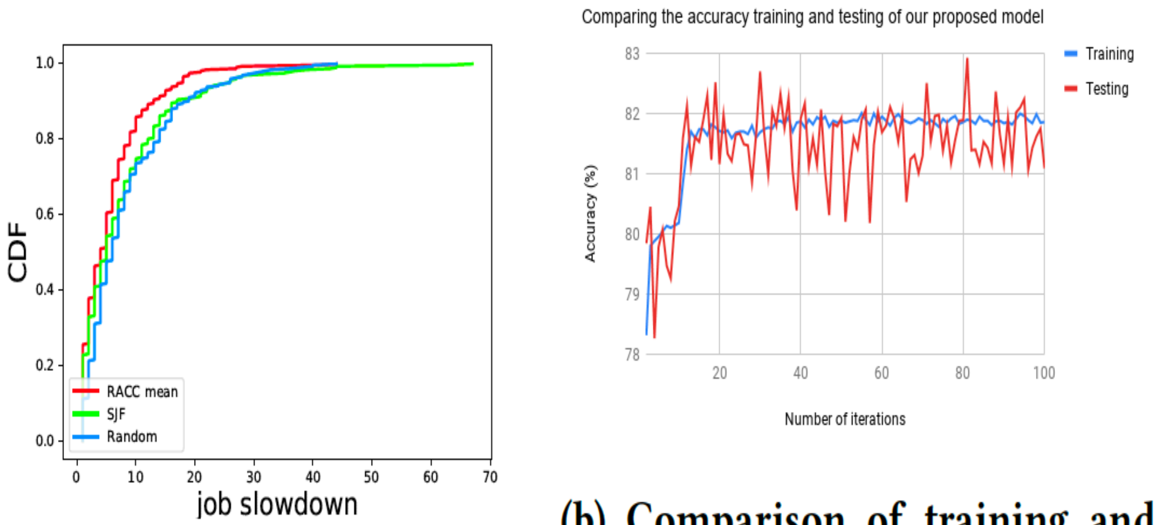


Figure 4: Performance enhancement in discounted total reward and the slowdown due to training process. Random and SJF models are constant as they do not have an incremental learning due to lack of feedback from last iteration.

Results

Training Accuracy – 82.01%, Testing accuracy – 82.93%



(a) CDF comparison

(b) Comparison of training and testing accuracy

Figure 5: Comparison of our proposed RACC model with SJF and Random scheduling approach in terms of the slowdown and the average model accuracy for training/testing.

Thoughts

- CRIU - Checkpoint/Restore In Userspace

  Freeze the running application for live migration.

- Deep or shallow neural network? (25 neurons)
- Comparison with fair scheduling
- Dependency between jobs, the locality issue of machines.

Questions?