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

```bash
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
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
Scheduling Framework

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

**Figure 2: Overview of System Architecture**

**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

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

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

\[ \sum_{t=j_{\text{start}}}^{j_{\text{end}}} \phi_{i,j}^t = \begin{cases} j_{\text{duration}} \lor i \in i_j & \text{if } \forall i_j \in i \text{ and } \exists \phi_{i,j}^t > 0 \\ 0 \lor \text{other machines} & \text{otherwise} \end{cases} \]

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

\[ T_{\text{finish}} = \max_{\text{container } j} \max_{t} JCT_t (\phi_{i,j}^t > 0) \]

\[ P_R = \max_{\text{resource } r} \frac{\sum_{i,j \in J} \alpha_{i,j}^{r,t}}{\sum_i \alpha_{i,r}} \]

- Resource Usage on machine <= capacity
- Should not exceed maximum requirement
- To avoid preemption – for simplicity
- Jduration – total job execution time at container j
- Job j’s finish time
- 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?