RACC: Resource-Aware Container Consolidation using a Deep Learning Approach

Conference Paper · May 2018

DOI: 10.1145/3217871.3217876

2 authors, including:

Saurav Nanda
Purdue University
7 PUBLICATIONS **25** CITATIONS

SEE PROFILE

RACC: Resource-Aware Container Consolidation using a Deep Learning Approach

Saurav Nanda, Thomas J. Hacker
Department of Computer and Information Technology
Purdue University
West Lafayette, Indiana
{nandas,tjhacker}@purdue.edu

ABSTRACT

Resource optimization has always been a big challenge in modern data centers. The process of performing workload consolidation on a minimal number of physical machines is becoming more complex when these data center began supporting containers in addition to virtual machines (VMs). With the increasing usage of containers with VMs in data centers, it becomes critical to address this problem from container's point of view - that is to optimally allocate containers in the fewest number of physical hosts. Depending on the type of application workload or tasks, infrastructure providers may provision separate containers to handle each task. These tasks may have different resource demands, such as: some of these task are CPU intensive, some memory intensive, some I/O intensive and some may be network intensive. Also, it is not necessary for all physical machines in the data center are even so they may have different kinds of machines with different resource capacity. Hence, the challenge is to consolidate all the active containers with different resource requirements on the minimum number of physical machines that are not even. We formulate a multi-resource bin packing problem and propose a *Deep Learning* technique called *Fit-for-Packing* to place near-optimal number of containers on a physical machine. Experimental results show that our model achieves an average training accuracy of 82.01% and an average testing accuracy of 82.93%.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Learning paradigms**; **Reinforcement learning**;

KEYWORDS

Container Consolidation, Deep Learning, Quality of Service (QoS)

1 INTRODUCTION

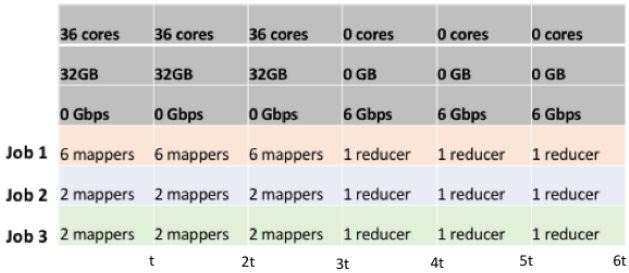
The usage of cloud computing infrastructure in small/large scale organizations is rapidly increasing primarily due to their *pay-as-you-go* model in which you pay only for resources that you actually use and you can also choose on-demand scaling of your resources. Apart from standard cloud services, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), Piraghaj et al. [7] described about a new kind of infrastructure service called *Containers as a Service*. Docker [6] is a good example of CaaS that leverage the same kernel with the physical machine, lightweight compared to VMs and provides portability to the application developers. Resource optimization using workload consolidation is critical research problem to maintain application performance, optimal power consumption and load balancing in a cloud infrastructure. Within the addition of a CaaS layer in the cloud infrastructure, there is a need to address the container consolidation problem to ensure the optimal usage of resources. The problem of optimally allocating VMs/containers in the fewest number of hosts is a classic *bin packing* problem that is NP-Hard. Our problem is slightly different from classic bin packing problem as we are dealing with different size of balls (containers with different resource requirements) and different size of bins (physical machines with different resource capacity) There are a good amount of different theoretical models that tried to solve this optimization problem. Beloglazov et al. [1] described that these theoretical models are not practical enough to be used in real-world cloud infrastructure providers as the optimization solver may take up to 30 minutes for just 15 nodes. Hence, we use a deep reinforcement learning approach to transform this multi-resource bin packing problem into a deep learning problem. We validate our proposed *Deep Reinforcement Learning* approach by implementing our model on top of *DeepRM*, which was developed by [5] for resource scheduling using deep learning, and achieve an average training accuracy of 82.01% and an average testing accuracy of 82.93%. To the best of our knowledge, our proposed methodology is the first one that leverages *Deep Learning* approach to solve container consolidation problem while keeping track of resource demands of each container.

2 MOTIVATION

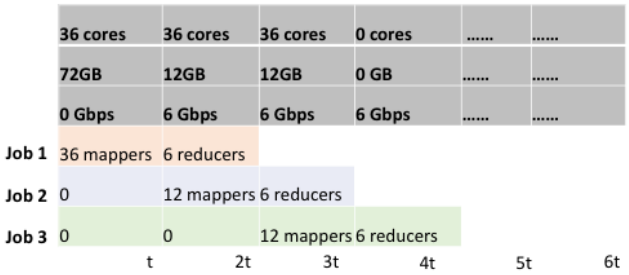
In this section, we explain our motivation based on an example that is inspired by Grandl et al. [3], we are considering a cluster with a total resource capacity of of 36 CPU cores, 72GB of memory, and 6Gbps of network bandwidth. Let us consider a scenario where the cluster is used to execute 3 map reduce jobs job-1, job-2 and job-3. The job-1 has 18 mapper tasks, job-2 / job-3 has 6 mapper tasks, and each of them has 3 reducer tasks. Each mapper task for job-1 requires 2 CPU and 4GB memory, and for job-2 and job-3 they require 6 CPU and 2GB memory. On the other hand, each reducer task requires 2Gbps of network and a negligible amount of CPU and memory. We consider that the infrastructure is designed in such a

MLCS'18, June 12, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5865-1/18/06…$15.00
https://doi.org/10.1145/3217871.3217876



(a) Fairness algorithm



(b) Multi-resource packing algorithm

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

way that we instantly create a new container for each mapper and reducer task as per the requirement. Hence, scheduling/consolidating these jobs in an optimal way is same as consolidating/scheduling these containers that is running each mapper or reducer task.

Any typical fairness algorithm will schedule these jobs in such a way that every job gets an equal amount of resource, so the 6 mappers of job-1, the 2 mappers of job-2 and the 2 mappers of job-3. All the mapper tasks will be completed by time 3t. However, this kind of fairness algorithm is inducing a 40GB of unused memory in the mapper phase. During the execution reducer tasks, the fair scheduler will run one reducer from each job so each of them get equal amount of network resource and all reducer tasks are completed at time 6t as shown in Figure 1a. Now, consider a scheduler that packs these jobs/containers based on their resource requirement rather than the fairness. Based on resource-aware scheduling: 1) all 36 mappers of job-1 are scheduled first so that all the memory available in the cluster gets utilized; 2) all 12 mapper tasks of job-2 are scheduled along with 6 reducer tasks job-1; and 3) finally all jobs are completed by time 4t as shown in Figure 1b. Hence, such multi-resource packing algorithm can reduces the total job completion of the cluster by 33% and we get more physical machines free by consolidating these containers, which are running these mappers and reducers tasks.

3 PROBLEM FORMULATION

This section presents the variables and list of assumptions related to multi-resource packing to consolidate the containers running different jobs/applications into minimum number of physical machines. The derivation steps we present below for our multi-resource bin-packing problem follows the derivation developed and described by Grandl et al. [3] to solve grid scheduling problem using their *Tetris* scheduler. We consider a grid computing center responsible for executing multiple jobs, such as batch jobs, map-reduce jobs or any other type of high performance computing jobs, as described by Higgins et al. [4]. Every job may be a executed on a separate container or they may share it with some other job(s) and tries to schedule these jobs to minimize the makespan which is equivalent to maximize the efficiency of consolidating these containers. The *makespan* is defined as the the amount of time taken to complete the processing of all the jobs. Current data centers tend to schedule these jobs based on a fair share algorithm so that each job gets equal amount of computing resources. However, fair schedulers are generally not efficient with respect to dynamically changing resource demands as the do not consider the individual resource demands of each job. The jobs may have different resource requirements, such as some of them may be CPU intensive, some are memory intensive, some are i/o intensive and some are network intensive jobs. Similarly, different containers running these different jobs also have different resource requirements ($D_{i,r}$) and current VM/container scheduling algorithms do not consider the individual resource demands of individual jobs that leads to resource fragmentation and over allocation of resources. Thus, the current scheduling algorithms that are based on fairness can schedule all containers with similar resource requirements (example: containers with all I/O intensive jobs) on same physical machine (with a total capacity of $C_{i,r}$) to maintain the fairness but that may lead to wastage of all CPU resources.

Our problem is bit different from classic bin packing problem as the resource demands of all containers are highly dynamic and it depends upon the current physical machine location. We focus on two major objectives: 1) adaptive learning of resource requirements of each job (J_r); and 2) monitoring of available resources (M_r). Prior scheduling algorithms, such as *Smallest Remaining Time First* (SRTF) does not have an optimal packing process and *Fair Schedulers* have a higher job completion time. Hence, we need a technique that is packing efficient called *Fit-for-packing*. We assume that network resources are uniform throughout the grid and we

Table 1: Variable Definition

Symbol	Definitions
$C_{i,r}$	capacity (c) of each resource (r) on machine (i).
$D_{j,r}$	demand of each container (j) for resource (r).
$\phi_{i,j}^{t}$	A boolean value. 1 if a container j is allocated to machine i at time t (time is discrete).
$\alpha_{i,j}^{r,t}$	α units of resource type r allocated to container j on machine i at time t
i_j	container j assigned on machine i
$d_{j,cpu}, A_{j,cpu}$	CPU demanded (d) and CPU allocated (A) to a container (j).
$d_{j,mem}$, $A_{j,mem}$	Amount of memory demanded (d) and memory allocated (A) by a container (j).
$d_{j,dR}$, $d_{j,dW}$, $A_{j,dR}$, $A_{j,dW}$	Amoung of disk read and write bandwidth demanded (d), and the amount of (A) of bytes to be read and write from container (j).
$C_{i,cpu}, C_{i,mem}$, $C_{i,dR}, C_{i,dW}$	Maximum number of CPU, memory size, and bandwidth for disk read/write by a machine (i).

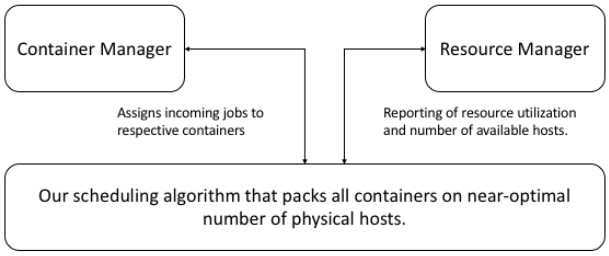


Figure 2: Overview of System Architecture

are not considering any network latency in our model. Also, we do consider the number of machines may change because of failure or new addition. Based on the above problem description, we have following constraints:

- Total resource usage on a machine i should never exceed its capacity.

$$\sum \alpha_{i,j}^{r,t} \leq C_{i,r} \forall i, t, r \tag{1}$$

where $\alpha_{i,j}^{r,t}$ represents α units of resource type r allocated to container j on machine i at time t.

- Tasks should not exceed their maximum resource requirement, and no resources should be allocated to an inactive task.

$$0 \leq \alpha_{i,j}^{r,t} \leq D_{j,r} \forall r, i, j, t \tag{2}$$

- To avoid the overhead of preemption, a scheduling technique may choose to provide uninterrupted access of resource until the container scheduling and job execution on these containers is completed.

$$\sum_{t=j_{start}}^{j_{end}} \phi_{i,j}^{t} = \begin{Bmatrix} j_{duration} \ \forall \ i \in i_j \\ 0 \ \forall \ other \ machines \end{Bmatrix} \tag{3}$$

where j_{start} represents the starting time of job execution at container j, j_{end} represents the end time of job execution at container j, and $j_{duration}$ represents the total job execution time at container j.

- The time taken by a container to finish the jobs depends on both container placement and resource allocation.

$$j_{duration} = max\left(\frac{A_{j,cpu}}{\sum_t \alpha_{i_j,j}^{cpu,t}}, \frac{A_{j,mem}}{\sum_t \alpha_{i_j,j}^{mem,t}}, \frac{A_{j,dW}}{\sum_t \alpha_{i_j,j}^{dW,t}} \frac{A_{i,j,dR}}{\sum_t \alpha_{i_j,j}^{dR,t}}\right) \tag{4}$$

where $A_{j,cpu}$, $A_{j,mem}$ represents the amount of CPU and memory allocated (A) to a container (j) and $A_{j,dR}$, $A_{j,dW}$ represents the amount of (A) of bytes to be read and write from container (j).

Hence, we formulate our objective function based on minimizing the *makespan* that is equivalent of maximizing packing efficiency and to minimize the job completion time (JCT) of job J, we can formulate the following equation for job J's finish time (T_{finish}):

$$T_{finish} = max_{container j \in J} max_{JCT_t}(\phi_{i,j}^{t} > 0) \tag{5}$$

and to maintain the fairness throughout the process, we have following constraint at time t for the share of most prominent resource (P_R) of job J:

$$P_R = max_{resource_r} \frac{\sum_{i,j \in J} \alpha_{i,j}^{r,t}}{\sum_i C_{i,r}} \tag{6}$$

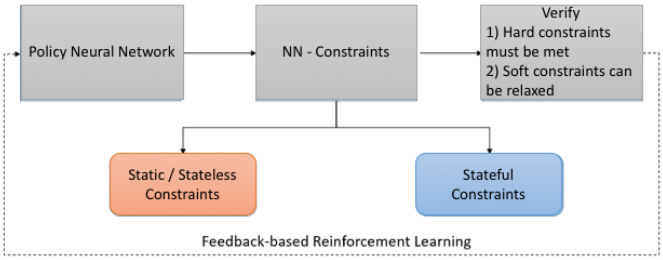


Figure 3: Deep Neural Network based Reinforcement Learning Approach for near-optimal placement of containers.

4 SYSTEM ARCHITECTURE - DEEP LEARNING APPROACH

In this section, we describe our *Deep Learning* based approach to solve this multi-resource scheduling problem. Figure 2 shows the overall architecture of our system that uses a container manager to assign incoming jobs on respective containers and sends a set of containers to our scheduling algorithm as an input. Our scheduling algorithm packs these container based on the resource requirement of these jobs based on the training provided to our deep neural network. Finally, we have a resource manager module that is responsible for reporting the resource utilization of each physical host and the total number of available hosts. Within our scheduling module, we follow a step-by-step process as shown in Figure 3: 1) firstly, we classify all incoming jobs and their corresponding containers into four different categories based on their resource requirements, such as CPU intensive, memory intensive, I/O intensive and network intensive; 2) using the historical data of container placement we train our neural network (NN) for all the policies discussed section 7.3 that includes both hard and soft constraints; 3) we assign probabilistic weights to pack these containers based on their resource requirements; and 4) after testing the output from constraint-based NN, we verify if all the hard constraints are met and soft constraint violations are within a threshold value.

5 EXPERIMENTAL SETUP

In this section, we describe our experiments that demonstrates the results of our Deep Learning approach to solve this problem resource-aware container consolidation. To validate our proof-of-concept, we conducted some initial experiments using DeepRM that is developed by Mao et al. [5] for resource scheduling using deep learning technique. DeepRM was primarily developed for grid scheduling where all incoming jobs are scheduled as per their resource requirement which is very similar to our scenario except that we are scheduling containers rather than the actual jobs running on top of these containers. We use standard python libraries deep learning and data analysis, such as Theano [1], Lasagne=0.1, numpy, scipy, and matplotlib. We leverage a neural network (NN)

[1] https://pypi.python.org/pypi/Theano

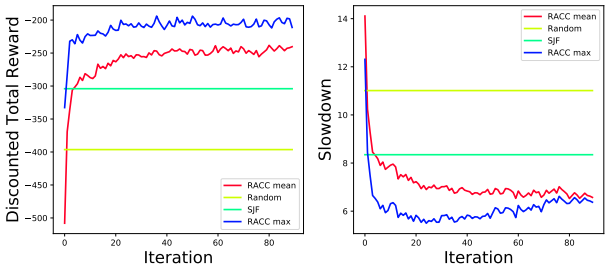
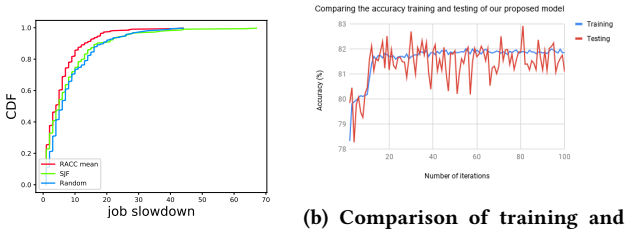


Figure 4: Performance enhancement in discounted total reward and the slowdown due to training process. Random and SJF models are constant as they do not have an incremental learning due to lack of feedback from last iteration.



(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.

with hidden layers that are completely connected with each other and has a total number of 25 neurons. We conducted our evaluation on a computer with an Intel(R) Core(TM) $i7-5930K$ processor with 3.50 GHz speed, 12 CPU cores and 32 GB memory, Ubuntu 16.04 as the primary operating system. The computer has a graphics processing unit (GPU) with NVIDIA Tesla K40c.

5.1 Results and Discussion

We evaluated our system by conducting a set of experiments to present the preliminary results. We use a Bernoulli distribution generate a synthetic data of incoming containers that are executing jobs with different resource requirements. We choose a fixed value of 10 time steps for the duration since last container requests were received. We consider these containers to be executing batch jobs and we consider two different size of jobs: 1) smaller jobs with expected completion time between $1t$ and $3t$, where t represents a unit time; and 2) bigger jobs with expected completion time between $10t$ and $15t$. We currently consider three types resource intensive jobs (out of which only one is dominant): CPU, memory and disk I/O that are assigned randomly to each container. We evaluate our model based on two metrics: 1) *job slowdown* that is calculated as $JS = T_c/T_e$, where T_c represents the job completion time and T_e is expected job duration; and 2) training and testing accuracy of the model. After running 100 iterations, 100 is a significant number (> 30) as per Central Limit Theorem [2], we analyzed the performance enhancement in discounted total reward (measure of model convergence defined by the Theano python package) and the slowdown due to the training process. The second part of Figure 4 shows the mean slowdown of the policy NN (refer Figure 3) in each iteration. We compare our RACC model with other baselines approaches, such as the shortest job first (SJF) algorithm and the random placement of containers. We observe incremental enhancement of our RACC with an increasing number of iterations. At the beginning of iteration, SJF and Random algorithms perform better than our RACC algorithm. However, after $\sim$ 10 iterations slowdown performance of RACC is much better than that of SJF and Random algorithms. To demonstrate the model convergence, first part of Figure 4 shows the maximum and mean value of discounted total reward that increases with increasing number of iterations as the policy NN keeps improving based on the feedback received from previous iteration. Also, Figure 5a shows the cumulative distribution function (CDF) of average slowdown and we observe that for first 40 iterations RACC has a higher probability compared to that of SJF and Random algorithms but, after 40 iterations the difference between all these algorithms becomes small. Finally, Figure 5b shows the training accuracy of 82.01% and testing accuracy of 82.93% of our RACC model. To summarize, RACC is capable of minimizing the job completion time of the jobs running of different containers by 1) performing a near-optimal packing; and 2) conducting resource-aware container placement to minimize the wastage of system resources.

6 CONCLUSION

In this paper, we used a *Deep Reinforcement Learning* approach to consolidate all the active containers with different resource requirements on minimum number of physical machines. We proposed and implemented a multi-resource bin packing algorithm (RACC) that leverages a *Deep Learning* technique called *Fit-for-Packing* to place near-optimal number of containers on a physical machine. Based on preliminary results, we observe that RACC achieves better job slowdown compared to other baseline algorithms, and RACC also shows a significant training accuracy of 82.01% and an average testing accuracy of 82.93%. We are working further to evaluate our proposed methodology in a real environment by using CRIU [2] software tool that can be leveraged to perform container migration.

REFERENCES

[1] Anton Beloglazov and Rajkumar Buyya. 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience* 24, 13 (2012), 1397–1420.
[2] Jay L Devore. 2011. *Probability and Statistics for Engineering and the Sciences*. Cengage learning.
[3] Robert Grandl, Ananthanarayanan, Kandula, Rao, and Akella. 2015. Multi-resource packing for cluster schedulers. *ACM SIGCOMM* 44, 4 (2015), 455–466.
[4] Joshua Higgins, Violeta Holmes, and Colin Venters. 2015. Orchestrating docker containers in the HPC environment. In *International Conference on High Performance Computing*. Springer, 506–513.
[5] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 50–56.
[6] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.
[7] Piraghaj, Dastjerdi, Calheiros, and Rajkumar Buyya. 2015. A framework and algorithm for energy efficient container consolidation in cloud data centers. In *DSDIS, 2015 IEEE International Conference on*. IEEE, 368–375.

[2] https://criu.org/Main_Page