Trimaran-3

An overcommitment-aware scheduler

Demonstration

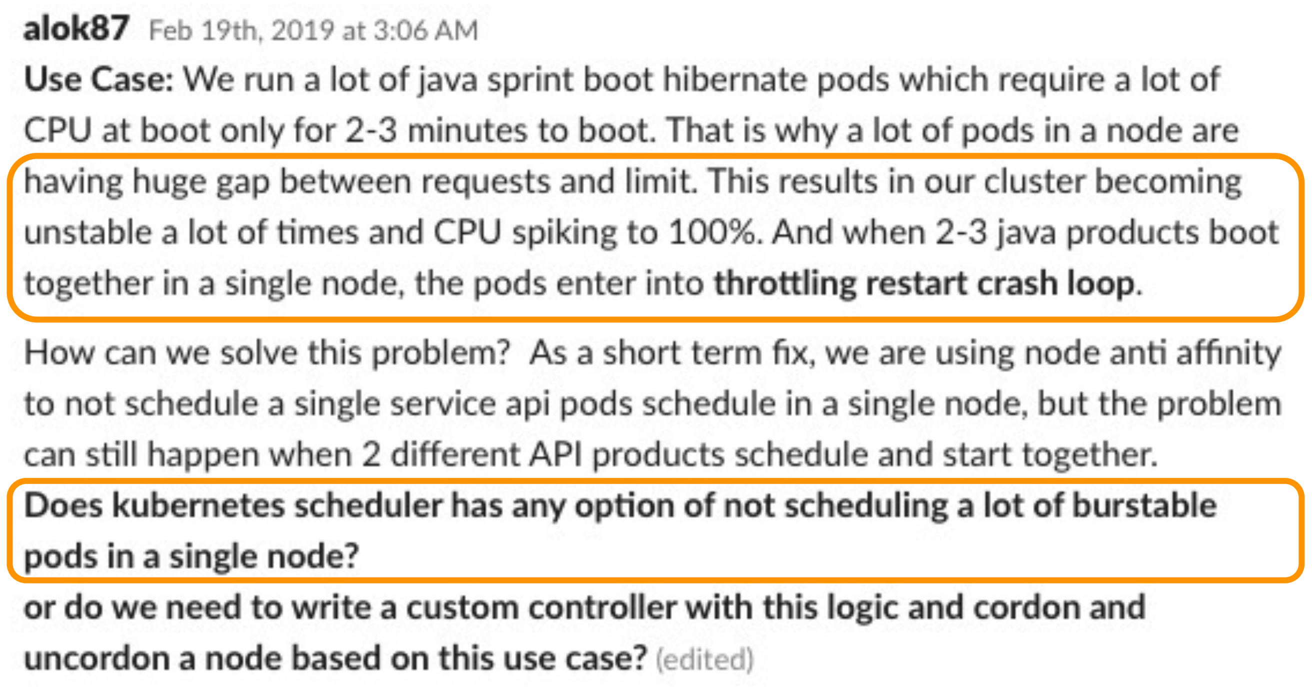
Asser Tantawi

IBM Research

Demo scenarios

- Scenario I
  - Burstable pods spike their resource usage (CPU and/or memory) initially, then usage goes down considerably
  - If pods are scheduled on same node, they run into throttling and/or OOM
  - Example: Spring Boot applications
- Scenario II
  - Burstable, compute-intensive workload (*TBD*)

Scenario I



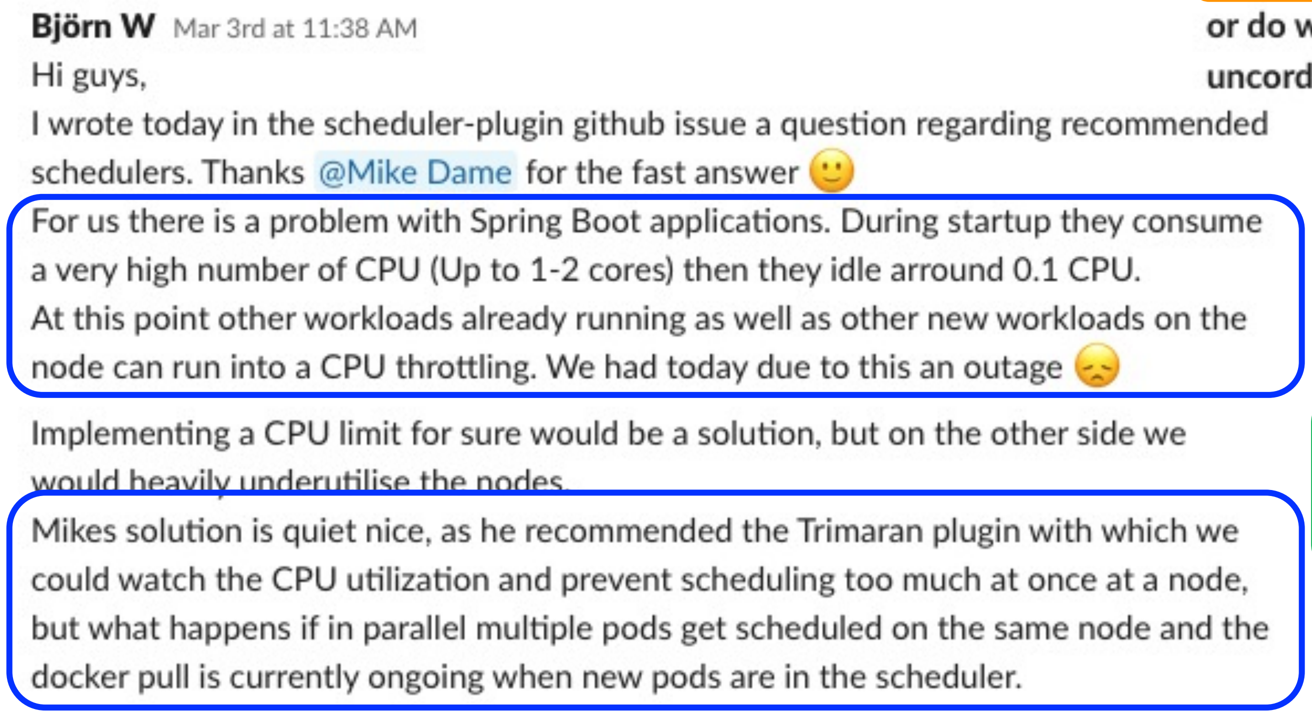
alok87 Feb 19th, 2019 at 3:06 AM

Use Case: We run a lot of java sprint boot hibernate pods which require a lot of CPU at boot only for 2-3 minutes to boot. That is why a lot of pods in a node are having huge gap between requests and limit. This results in our cluster becoming unstable a lot of times and CPU spiking to 100%. And when 2-3 java products boot together in a single node, the pods enter into **throttling restart crash loop**.

How can we solve this problem? As a short term fix, we are using node anti affinity to not schedule a single service api pods schedule in a single node, but the problem can still happen when 2 different API products schedule and start together.

Does kubernetes scheduler has any option of not scheduling a lot of burstable pods in a single node?

or do we need to write a custom controller with this logic and cordon and uncordon a node based on this use case? (edited)



Björn W Mar 3rd at 11:38 AM

Hi guys,

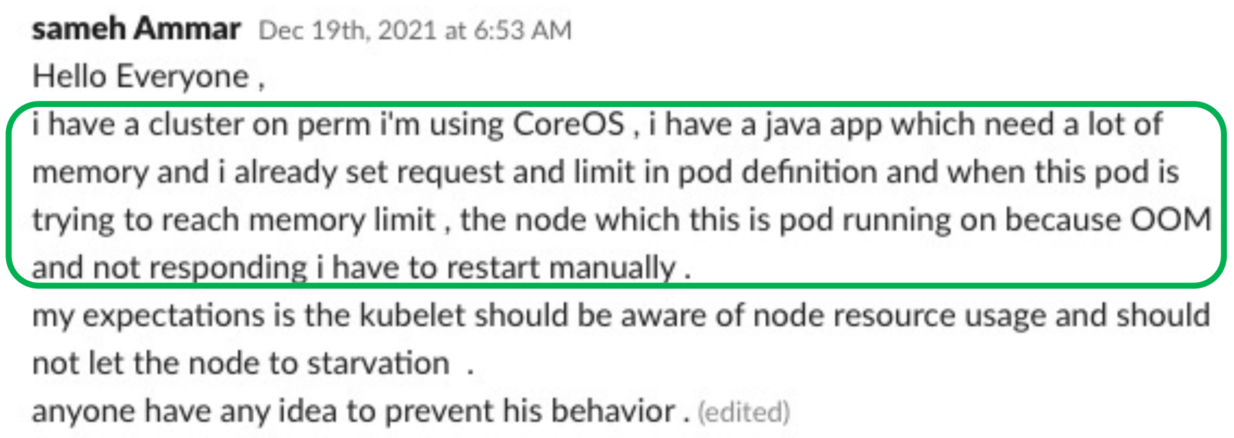
I wrote today in the scheduler-plugin github issue a question regarding recommended schedulers. Thanks @Mike Dame for the fast answer 🙂

For us there is a problem with Spring Boot applications. During startup they consume a very high number of CPU (Up to 1-2 cores) then they idle arround 0.1 CPU.

At this point other workloads already running as well as other new workloads on the node can run into a CPU throttling. We had today due to this an outage 😞

Implementing a CPU limit for sure would be a solution, but on the other side we would heavily underutilise the nodes.

Mikes solution is quiet nice, as he recommended the Trimaran plugin with which we could watch the CPU utilization and prevent scheduling too much at once at a node, but what happens if in parallel multiple pods get scheduled on the same node and the docker pull is currently ongoing when new pods are in the scheduler.



sameh Ammar Dec 19th, 2021 at 6:53 AM

Hello Everyone ,

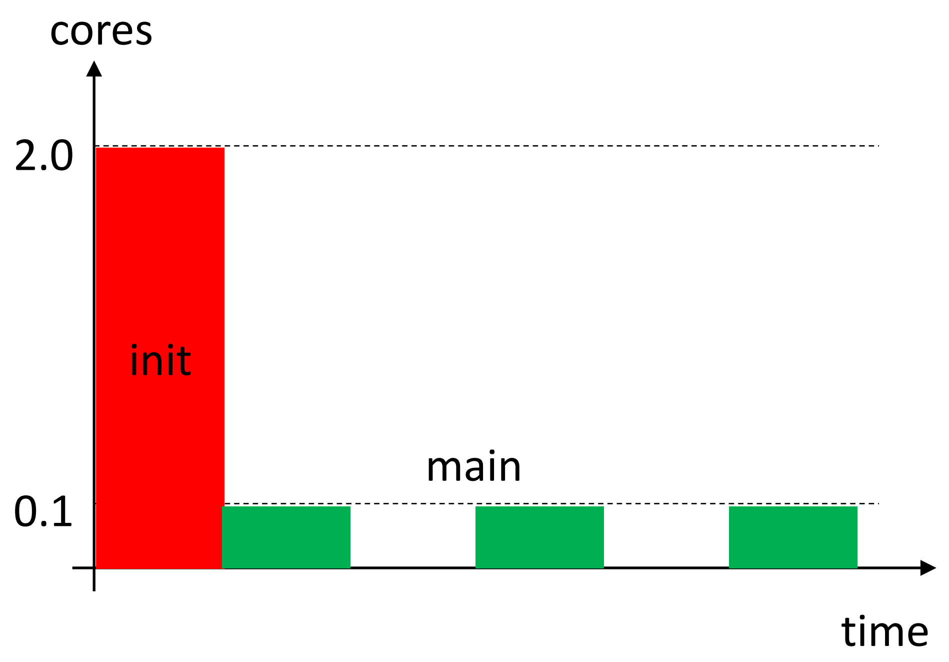
i have a cluster on perm i'm using CoreOS , i have a java app which need a lot of memory and i already set request and limit in pod definition and when this pod is trying to reach memory limit , the node which this is pod running on because OOM and not responding i have to restart manually .

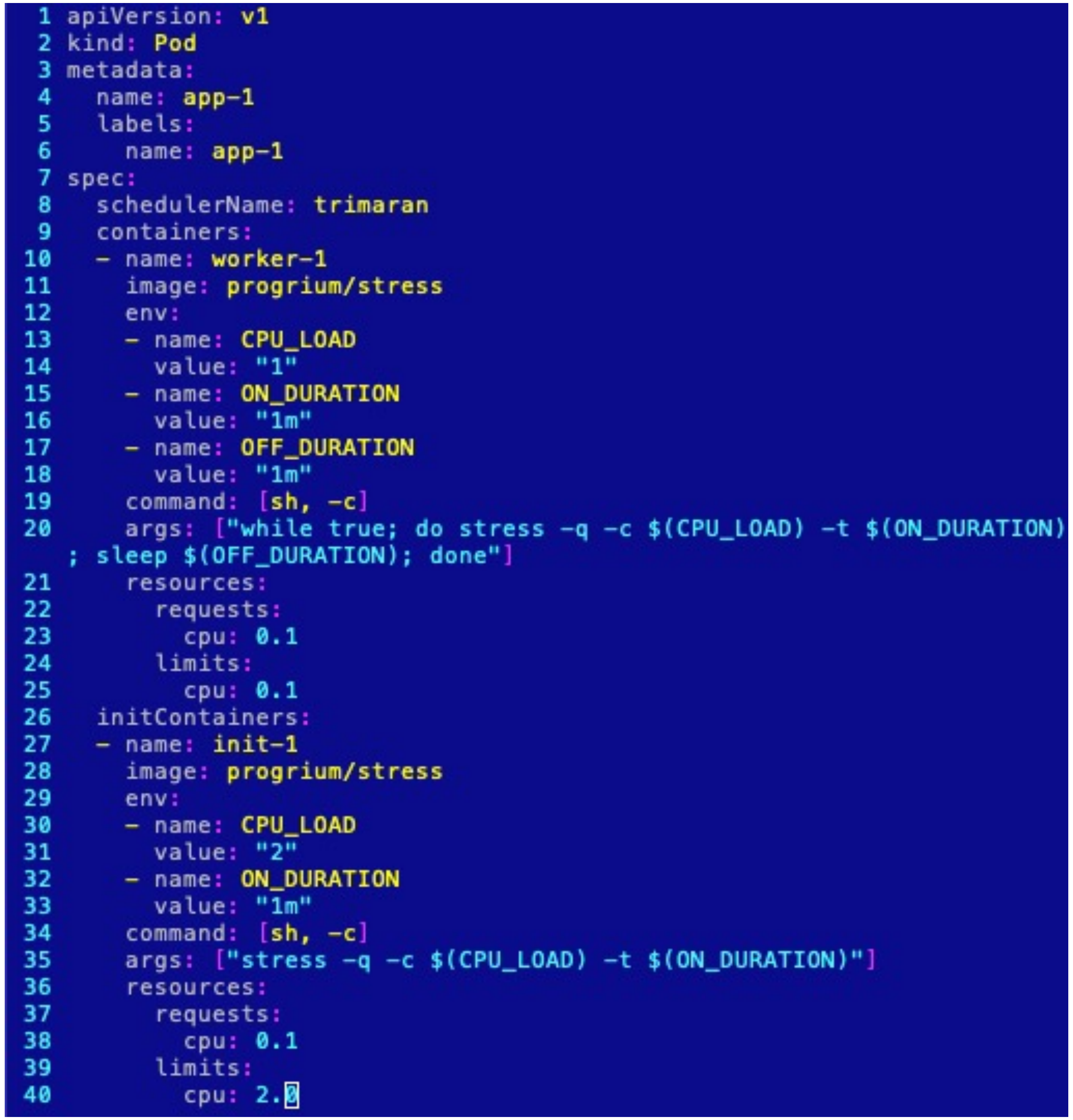
my expectations is the kubelet should be aware of node resource usage and should not let the node to starvation .

anyone have any idea to prevent his behavior . (edited)

Pod spec

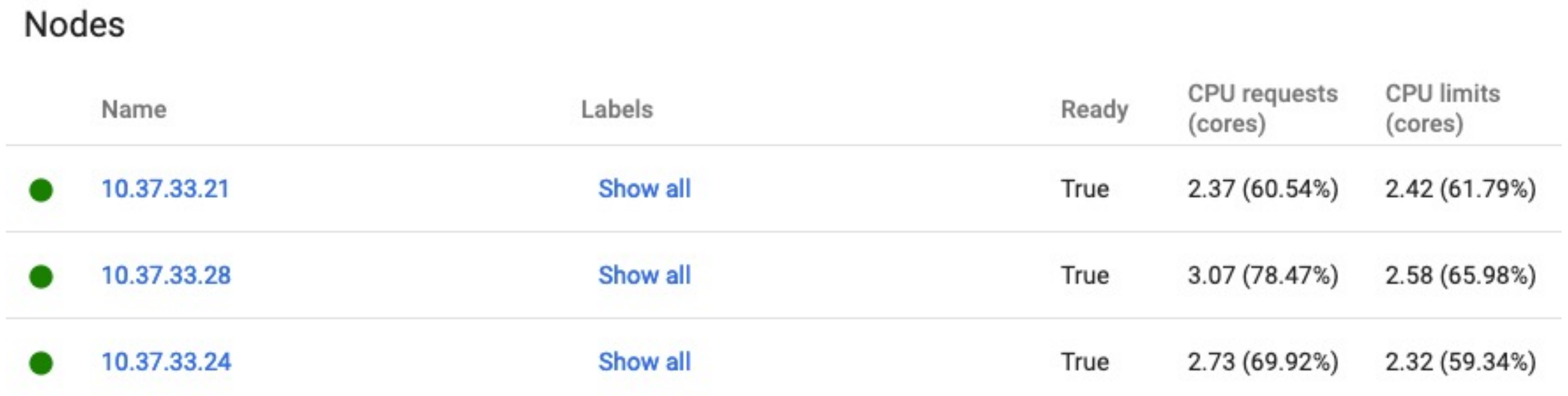
container	request	limit	usage
init	0.1	2.0	2.0
main	0.1	0.1	0.05





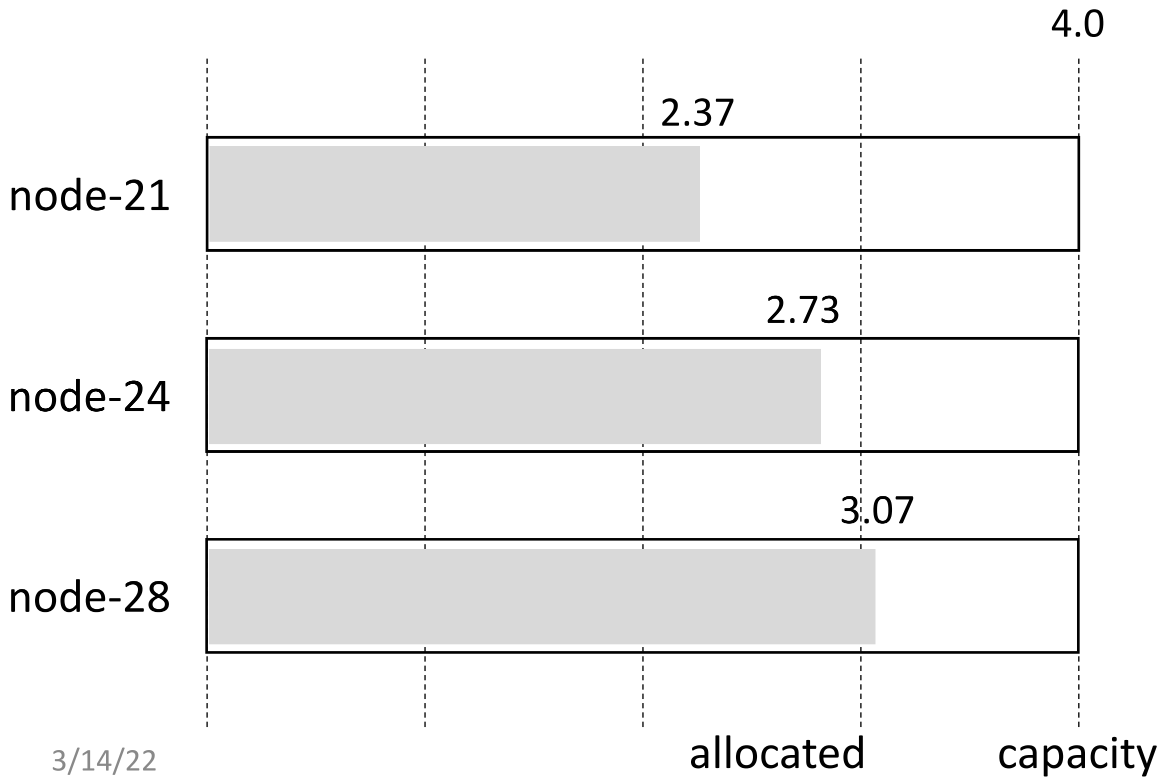
```
apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  schedulerName: trimaran
  containers:
  - name: worker-1
    image: progrium/stress
    env:
    - name: CPU_LOAD
      value: "1"
    - name: ON_DURATION
      value: "1m"
    - name: OFF_DURATION
      value: "1m"
    command: [sh, -c]
    args: ["while true; do stress -q -c $(CPU_LOAD) -t $(ON_DURATION)
; sleep $(OFF_DURATION); done"]
    resources:
      requests:
        cpu: 0.1
      limits:
        cpu: 0.1
  initContainers:
  - name: init-1
    image: progrium/stress
    env:
    - name: CPU_LOAD
      value: "2"
    - name: ON_DURATION
      value: "1m"
    command: [sh, -c]
    args: ["stress -q -c $(CPU_LOAD) -t $(ON_DURATION)"]
    resources:
      requests:
        cpu: 0.1
      limits:
        cpu: 2.0
```

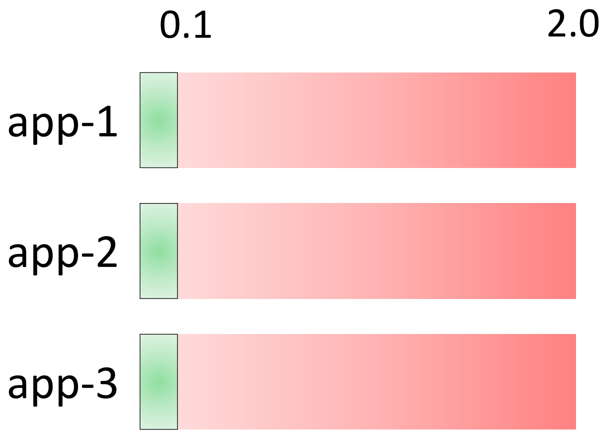
Cluster config



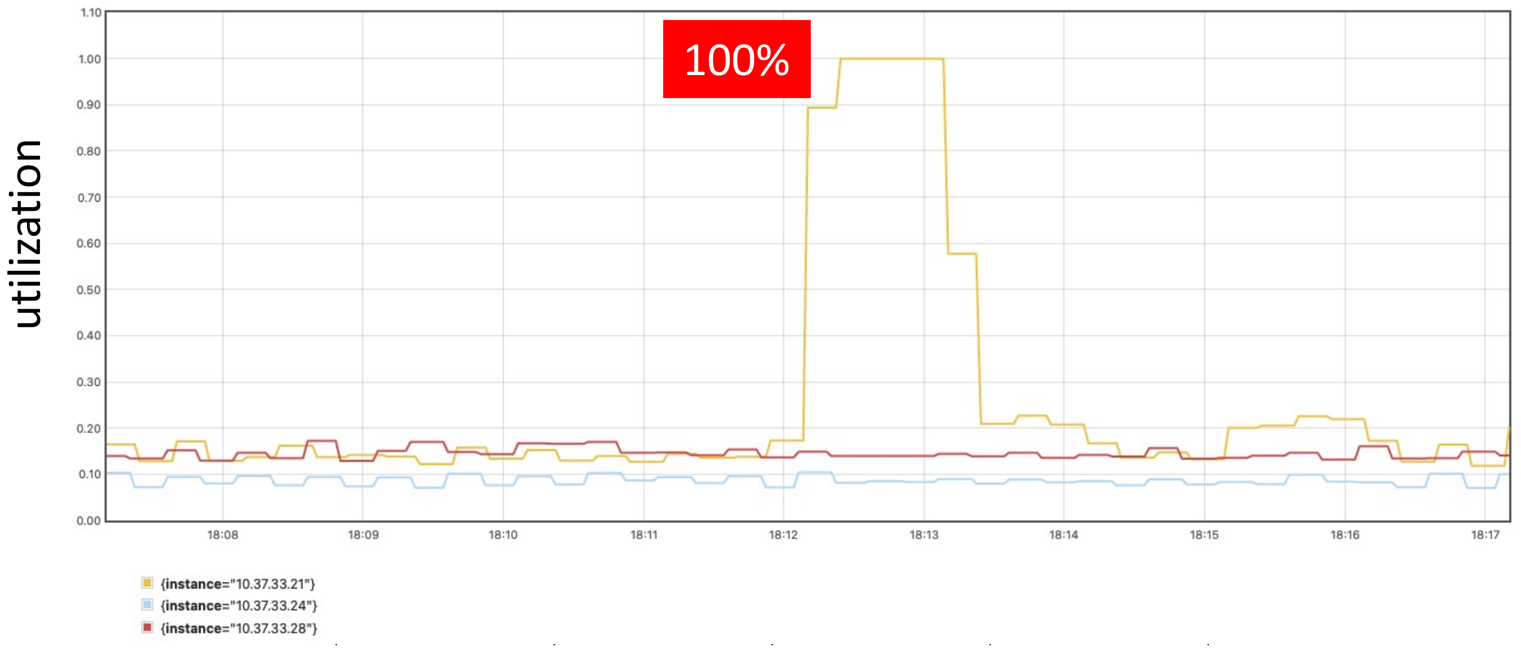
Nodes

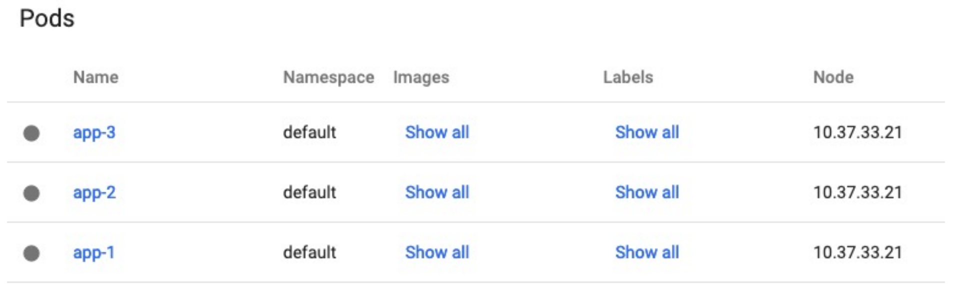
	Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)
●	10.37.33.21	Show all	True	2.37 (60.54%)	2.42 (61.79%)
●	10.37.33.28	Show all	True	3.07 (78.47%)	2.58 (65.98%)
●	10.37.33.24	Show all	True	2.73 (69.92%)	2.32 (59.34%)



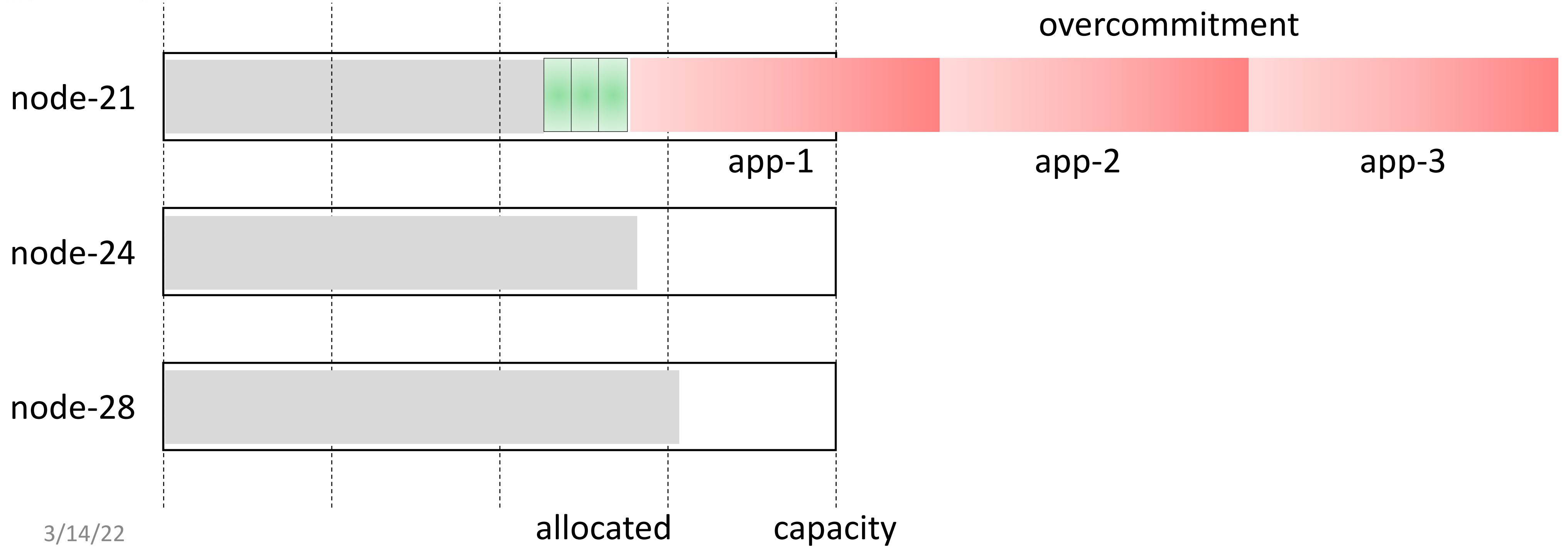


Default

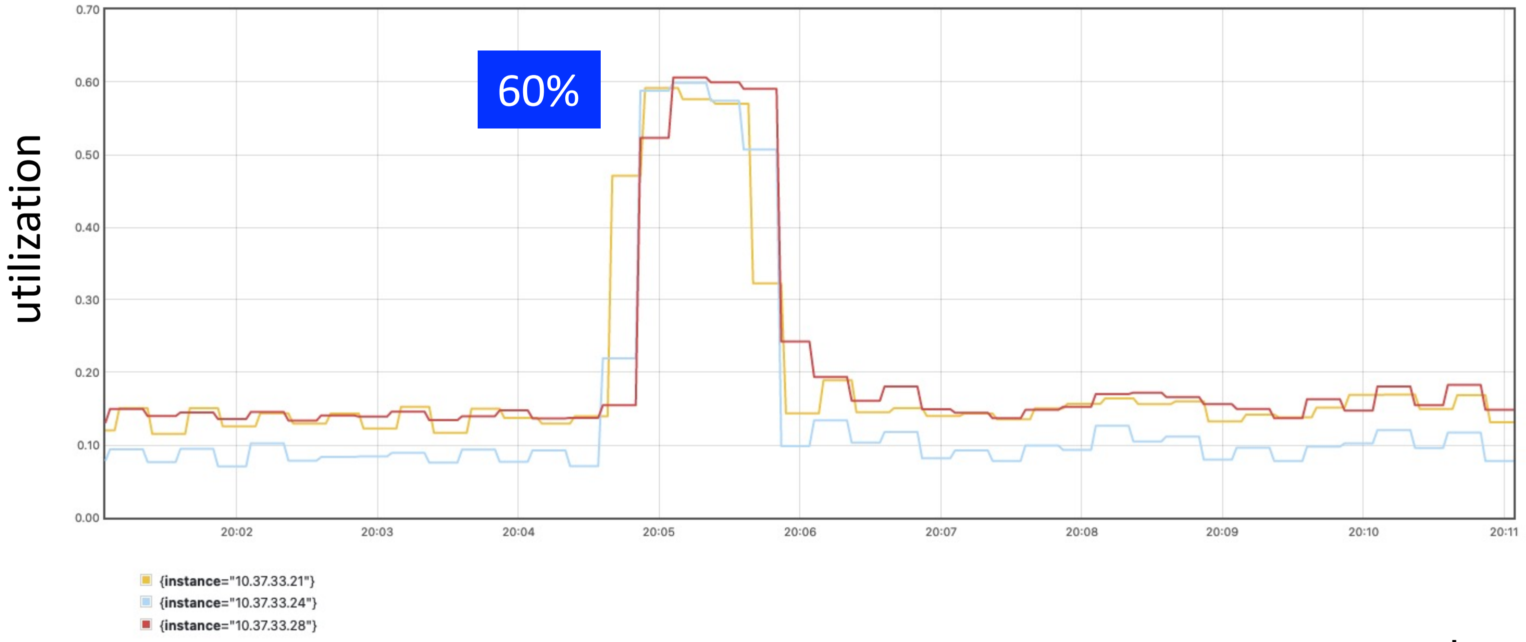


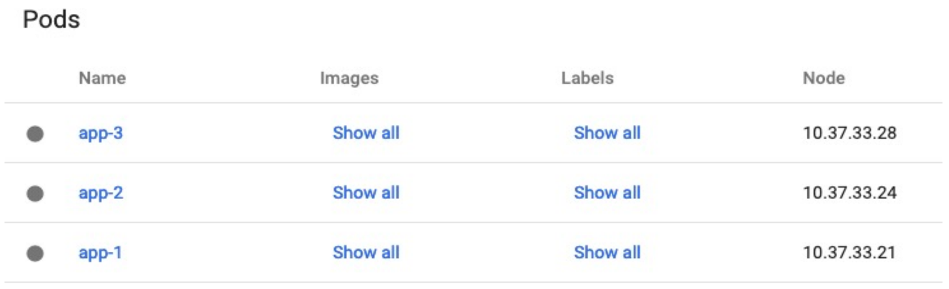

Pods

	Name	Namespace	Images	Labels	Node
●	app-3	default	Show all	Show all	10.37.33.21
●	app-2	default	Show all	Show all	10.37.33.21
●	app-1	default	Show all	Show all	10.37.33.21



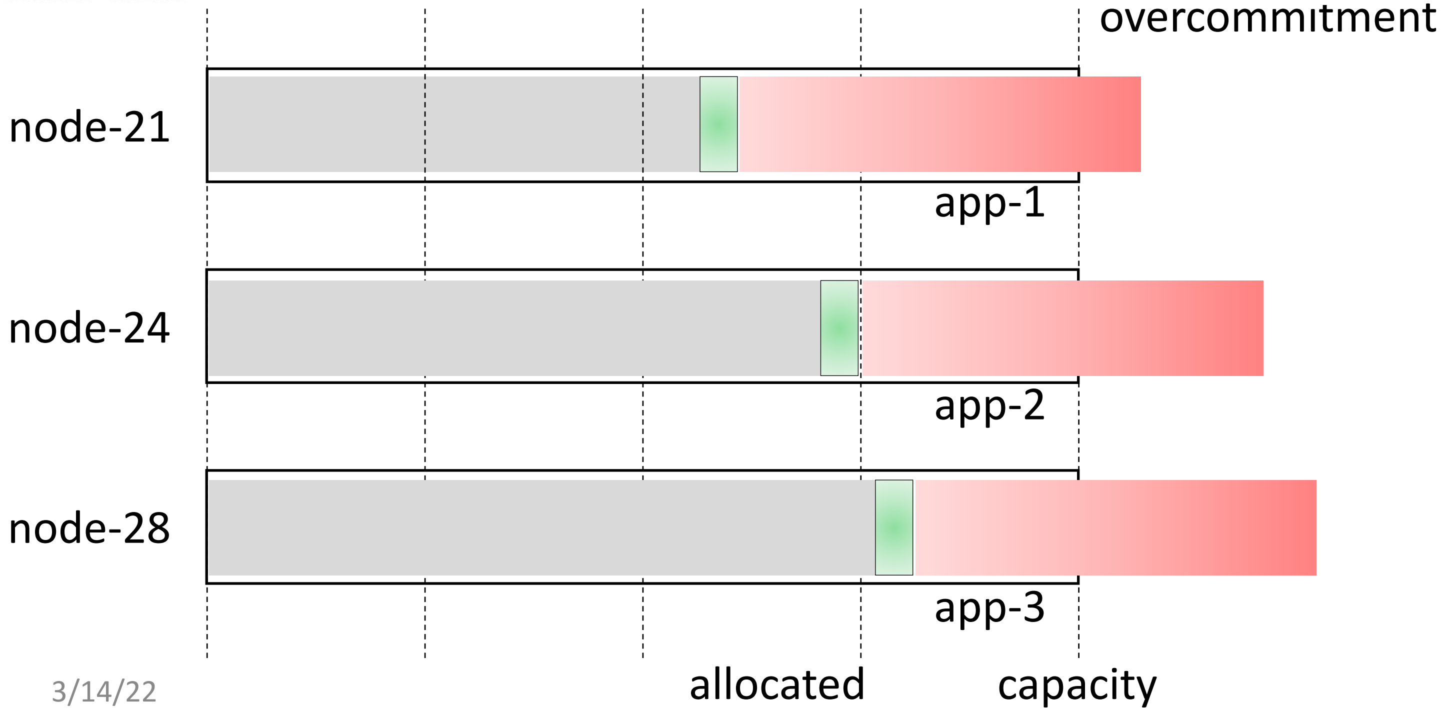
Trimaran-3





Pods

	Name	Images	Labels	Node
●	app-3	Show all	Show all	10.37.33.28
●	app-2	Show all	Show all	10.37.33.24
●	app-1	Show all	Show all	10.37.33.21



parameters	
smoothingWindowSize	5
riskLimitWeight	0.5

Trimaran-3

very light load

place app-1:

	node-21	node-24	node-28
specs			
id	10.37.33.21	10.37.33.24	10.37.33.28
capacity	3,910	3,910	3,910
requests	2,477	2,834	3,168
limits	4,416	4,320	4,580
loadUsage			
usedAvg	374.52	243.93	421.91
usedStd	1.57	1.35	2.19
alpha	10,266	6,129	6,622
beta	96,908	92,109	54,745
mean	0.096	0.062	0.108
var	0.000	0.000	0.000
sigma	0.001	0.001	0.001
overUsage			
allocThreshold	0.608	0.699	0.785
allocProb	1.000	1.000	1.000
overUse	0.000	0.000	0.000
risk			
riskLimit	0.261	0.276	0.475
riskLoad	0.000	0.000	0.000
totalRisk	0.130	0.138	0.237
score			
rank	**0.870**	**0.862**	**0.763**
totalScore	**87**	**86**	**76**

place app-2:

	node-21	node-24	node-28
specs			
id	10.37.33.21	10.37.33.24	10.37.33.28
capacity	3,910	3,910	3,910
requests	2,577	2,834	3,168
limits	6,416	4,320	4,580
loadUsage			
usedAvg	374.52	243.93	421.91
usedStd	1.57	1.35	2.19
alpha	10,266	6,129	6,622
beta	96,908	92,109	54,745
mean	0.096	0.062	0.108
var	0.000	0.000	0.000
sigma	0.001	0.001	0.001
overUsage			
allocThreshold	0.634	0.699	0.785
allocProb	1.000	1.000	1.000
overUse	0.000	0.000	0.000
risk			
riskLimit	0.653	0.276	0.475
riskLoad	0.000	0.000	0.000
totalRisk	0.326	0.138	0.237
score			
rank	**0.674**	**0.862**	**0.763**
totalScore	**67**	**86**	**76**

place app-3:

	node-21	node-24	node-28
specs			
id	10.37.33.21	10.37.33.24	10.37.33.28
capacity	3,910	3,910	3,910
requests	2,577	2,934	3,168
limits	6,416	6,320	4,580
loadUsage			
usedAvg	374.52	243.93	421.91
usedStd	1.57	1.35	2.19
alpha	10,266	6,129	6,622
beta	96,908	92,109	54,745
mean	0.096	0.062	0.108
var	0.000	0.000	0.000
sigma	0.001	0.001	0.001
overUsage			
allocThreshold	0.634	0.699	0.785
allocProb	1.000	1.000	1.000
overUse	0.000	0.000	0.000
risk			
riskLimit	0.653	0.712	0.475
riskLoad	0.000	0.000	0.000
totalRisk	0.326	0.356	0.237
score			
rank	**0.674**	**0.644**	**0.763**
totalScore	**67**	**64**	**76**