# Trimaran-3

## An overcommitment-aware scheduler

*Demonstration*

Asser Tantawi

IBM Research

# Demo scenarios

- Scenario I: Risk of limits
  - 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: Risk of load
  - Burstable, compute-intensive workload (*PARSEC 3.0*)
  - Schedule bursting pod given over-committed nodes
  - Load-awareness shortens running time by a factor of two

# Scenario I

Risk of limits

# 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 ,

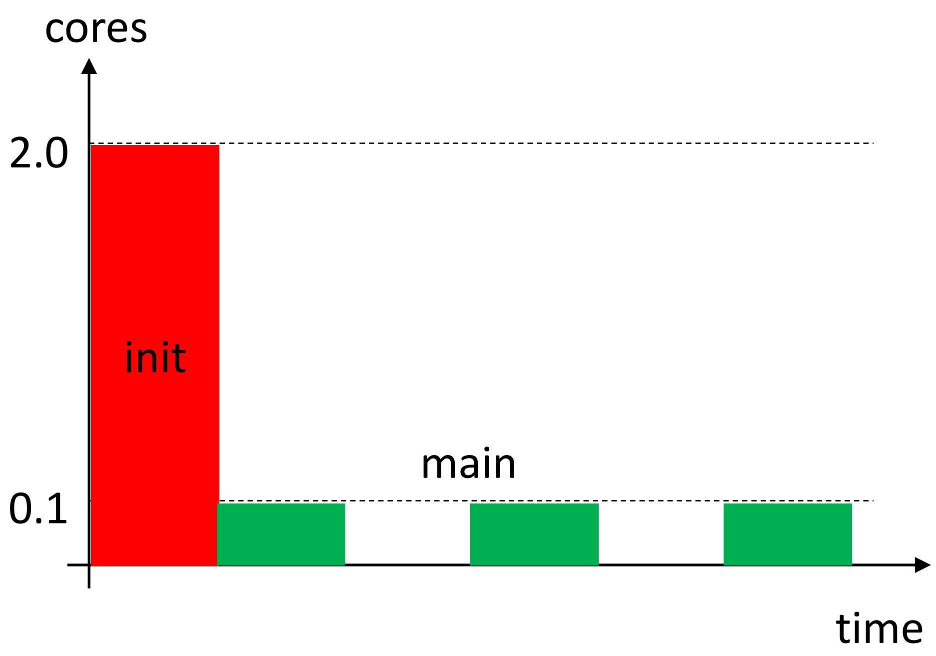
i have a cluster on perm i'm using CoreOS , i have a java app which need a lot of memory and i already set request and limit in pod definition and when this pod is trying to reach memory limit , the node which this is pod running on because OOM and not responding i have to restart manually .

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 |

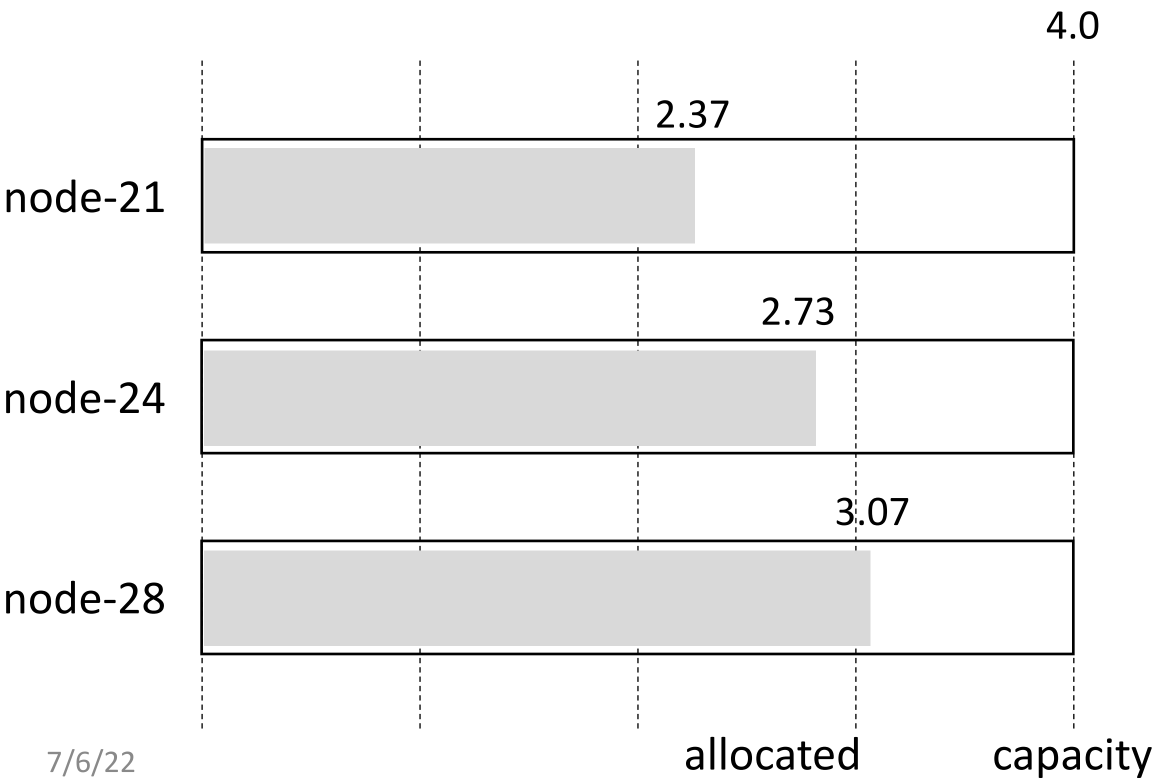


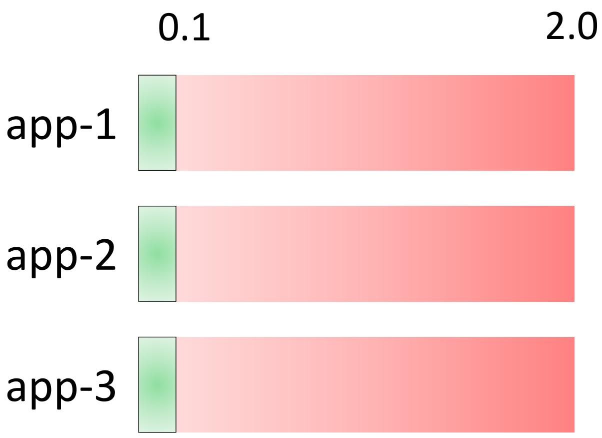

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  schedulerName: trimaran
  containers:
  - name: worker-1
    image: progrium/stress
    env:
    - name: CPU_LOAD
      value: "1"
    - name: ON_DURATION
      value: "1m"
    - name: OFF_DURATION
      value: "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

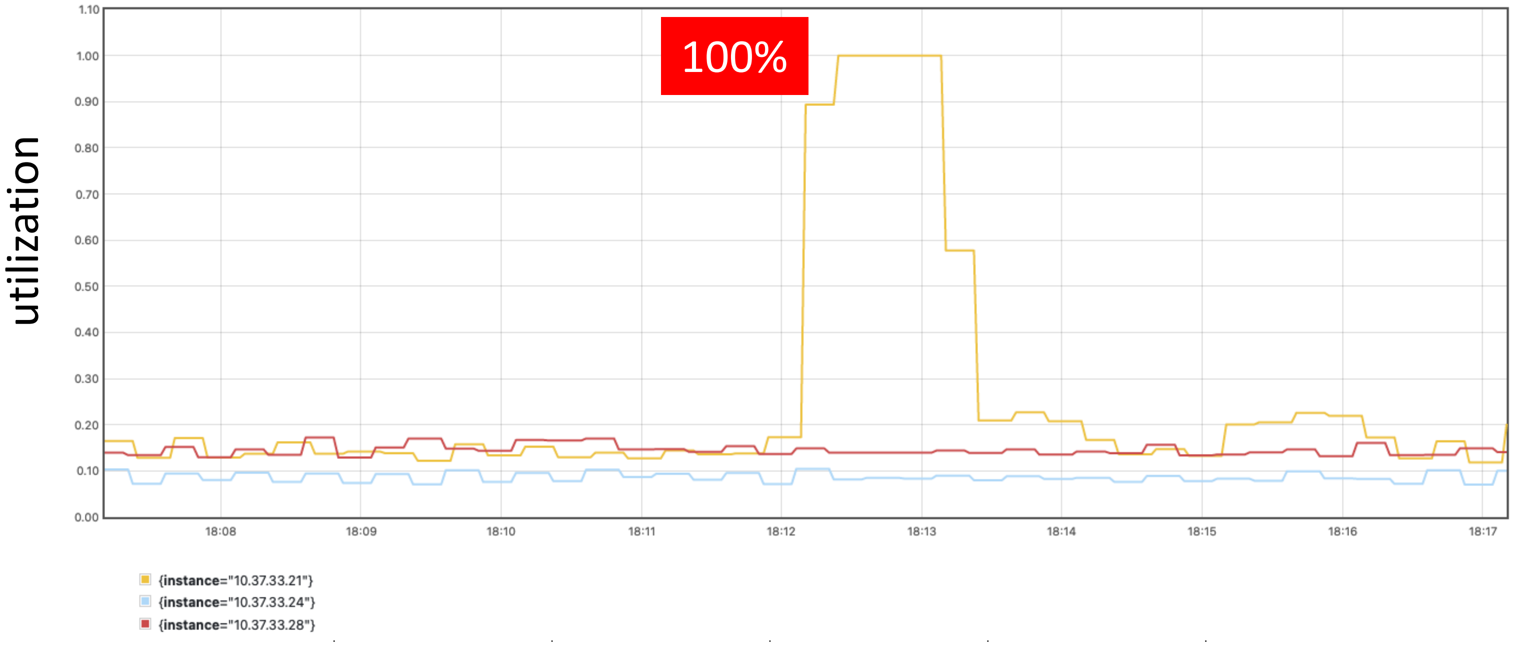
## Nodes

| | 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%) |



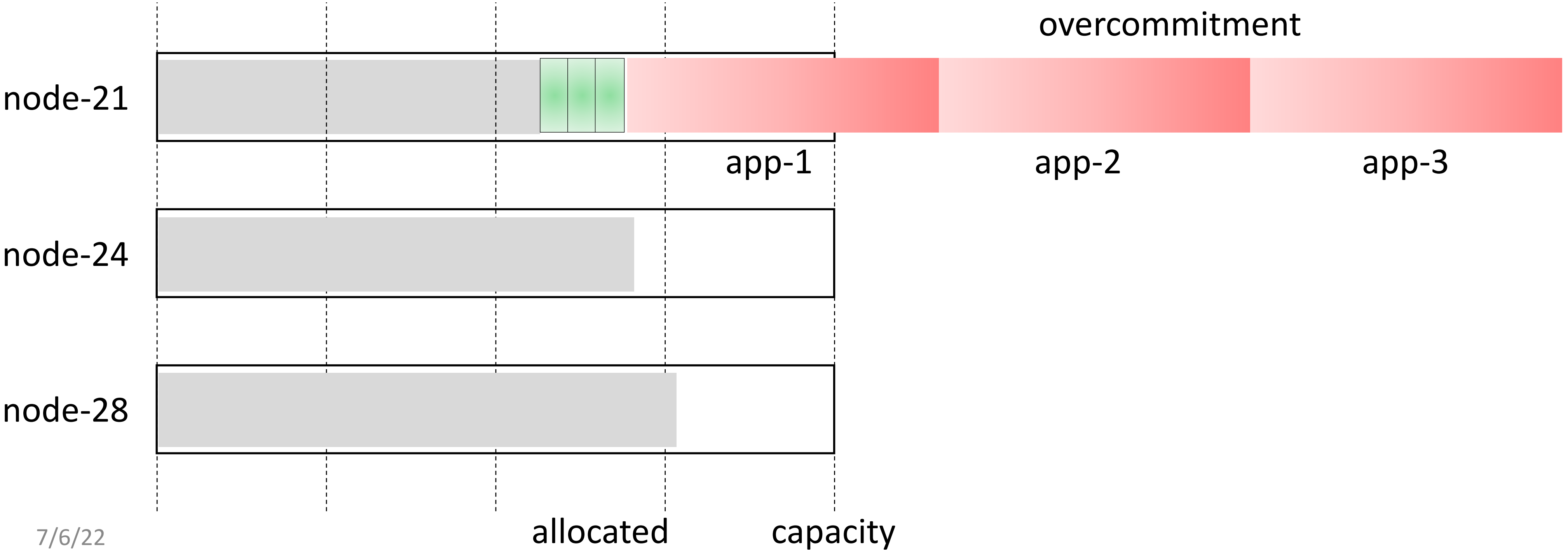



# Default




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 |



| 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** |

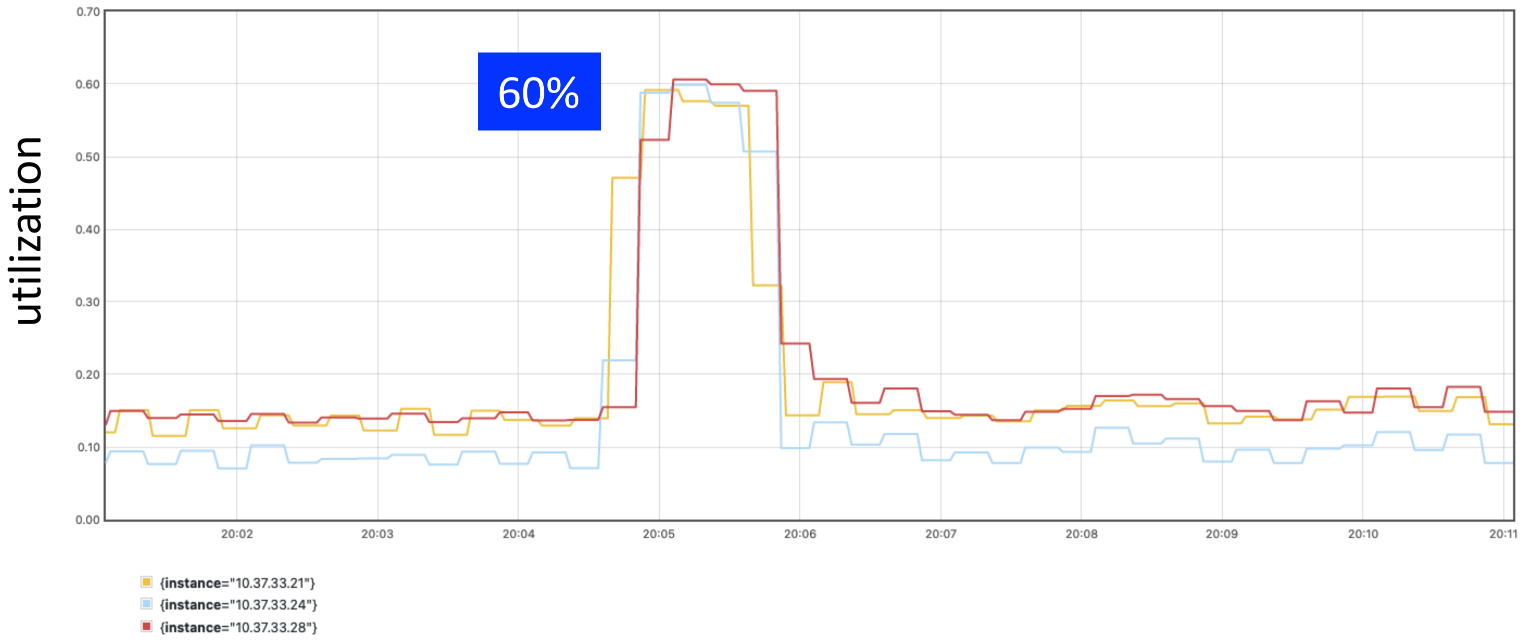
place app-2:

| | 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:

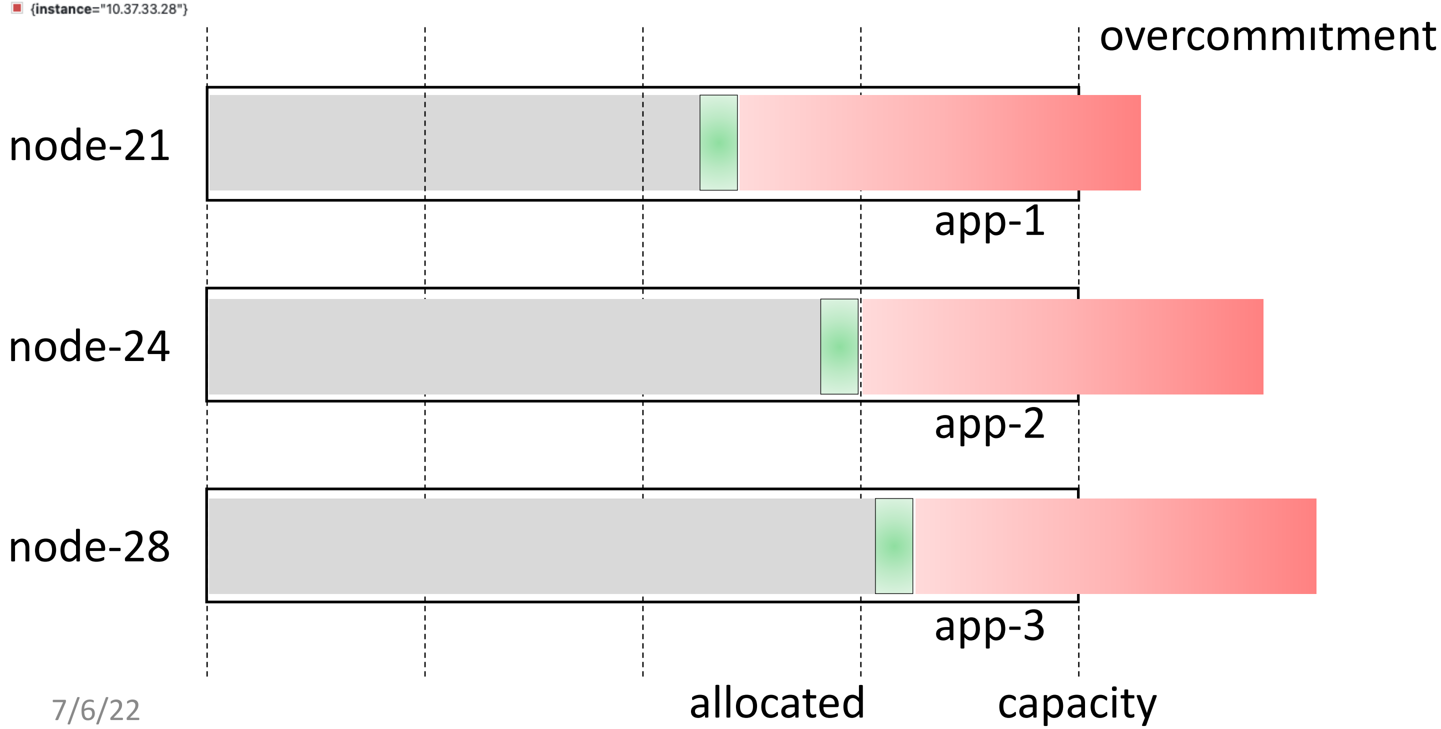
| | node-21 | node-24 | node-28 |
|---|---|---|---|
| specs | | | |
| id | 10.37.33.21 | 10.37.33.24 | 10.37.33.28 |
| capacity | 3,910 | 3,910 | 3,910 |
| requests | 2,577 | 2,934 | 3,168 |
| limits | 6,416 | 6,320 | 4,580 |
| loadUsage | | | |
| usedAvg | 374.52 | 243.93 | 421.91 |
| usedStd | 1.57 | 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** |

# Trimaran-3




Pods

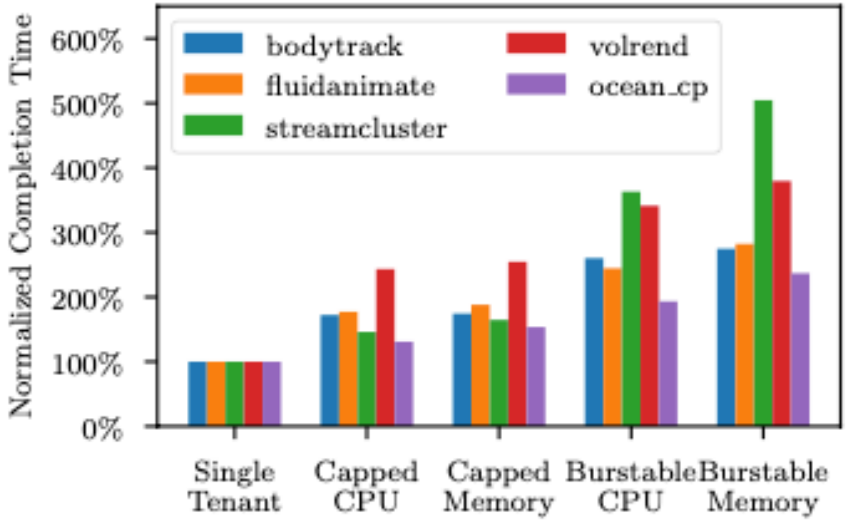
| | Name | Images | Labels | Node |
|---|---|---|---|---|
| ● | app-3 | Show all | Show all | 10.37.33.28 |
| ● | app-2 | Show all | Show all | 10.37.33.24 |
| ● | app-1 | Show all | Show all | 10.37.33.21 |



# Scenario II

Risk of load

# Scenario II




**QoS for best-effort batch jobs in container-based cloud**


Yin-Goo Yim | Hyeon-Jun Jang | Hyun-Wook Jin


**Mind the Gap: Broken Promises of CPU Reservations in Containerized Multi-tenant Clouds**


Li Liu
George Mason University
Fairfax, VA, USA
lliu8@masonlive.gmu.edu

Haoliang Wang
Adobe Research
San Jose, CA, USA
hawang@adobe.com

An Wang
Case Western Reserve University
Cleveland, OH, USA
axw474@case.edu

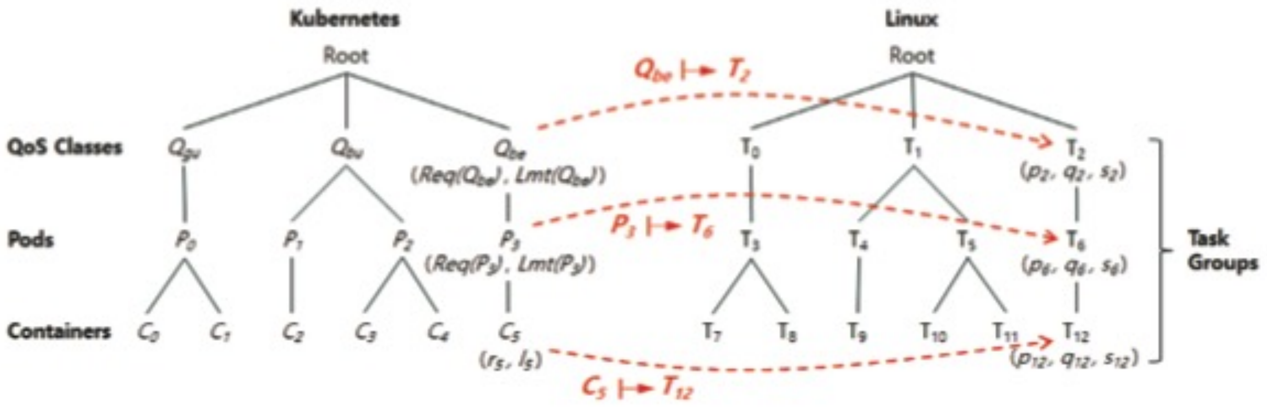




FIGURE 1 Task group hierarchy for containers

# Scenario II

Spirals-Team / benchmark-containers Public

<> Code | Issues 11 | Pull requests | Actions | Projects

master ▾ benchmark-containers / parsec-3.0 /

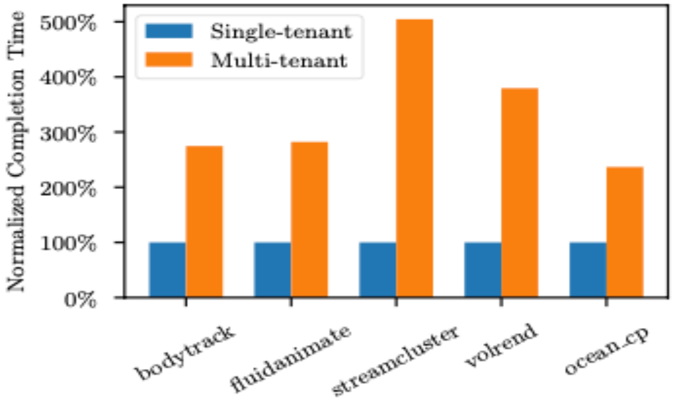
## Workloads

| Program | Application Domain | Parallelization | | Working Set | Data Usage | |
|---|---|---|---|---|---|---|
| | | Model | Granularity | | Sharing | Exchange |
| blackscholes | Financial Analysis | data-parallel | coarse | small | low | low |
| bodytrack | Computer Vision | data-parallel | medium | medium | high | medium |
| canneal | Engineering | unstructured | fine | unbounded | high | high |
| dedup | Enterprise Storage | pipeline | medium | unbounded | high | high |
| facesim | Animation | data-parallel | coarse | large | low | medium |
| ferret | Similarity Search | pipeline | medium | unbounded | high | high |
| fluidanimate | Animation | data-parallel | fine | large | low | medium |
| freqmine | Data Mining | data-parallel | medium | unbounded | high | medium |
| raytrace | Rendering | data-parallel | medium | unbounded | high | low |
| streamcluster | Data Mining | data-parallel | medium | medium | low | medium |
| swaptions | Financial Analysis | data-parallel | coarse | medium | low | low |
| vips | Media Processing | data-parallel | coarse | medium | low | medium |
| x264 | Media Processing | pipeline | coarse | medium | high | high |

## PARSEC
['pär-"sek] A unit of measure

**Goal**: An open-source parallel benchmark suite of emerging applications for evaluating multi-core and multiprocessor systems

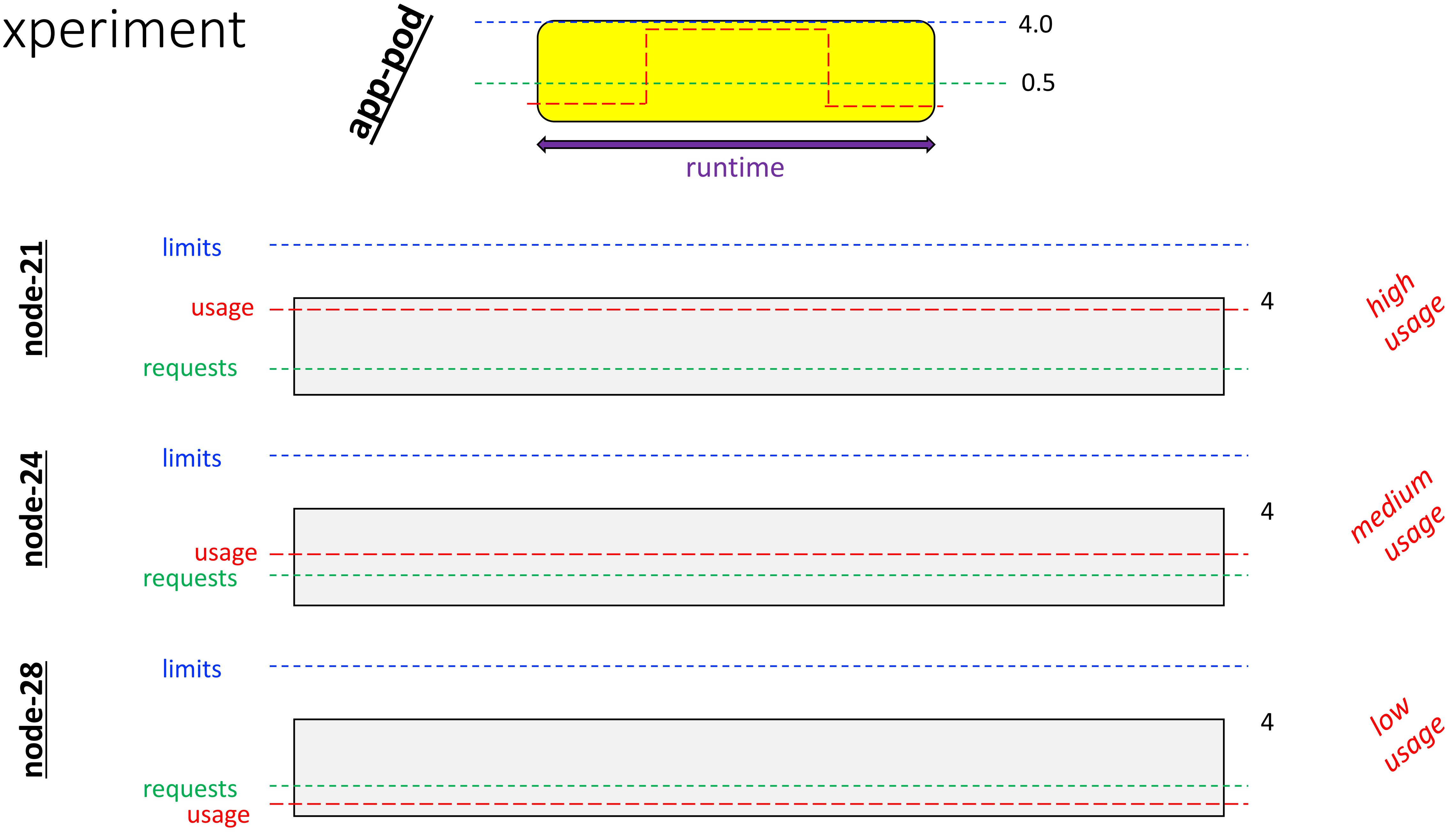
**Application domains**: financial, computer vision, physical modeling, future media, content-based search, deduplication




## What is PARSEC?

- Princeton Application Repository for Shared-Memory Computers
- Benchmark Suite for Chip-Multiprocessors
- Started as a cooperation between Intel and Princeton University, many more have contributed since then
- Freely available at:
  http://parsec.cs.princeton.edu/

# Experiment


app-pod
4.0
0.5
runtime
node-21
limits
usage
requests
4
high usage
node-24
limits
usage
requests
4
medium usage
node-28
limits
requests
usage
4
low usage

Nodes

initially

| | Name | Labels | Ready | CPU requests (cores) | CPU limits (cores) | Memory requests (bytes) | Memory limits (bytes) | Pods |
|---|---|---|---|---|---|---|---|---|
| ● | 10.37.33.21 | Show all | True | 1.72 (44.04%) | 1.87 (47.83%) | 2.00Gi (15.58%) | 2.96Gi (23.10%) | 10 (9.09%) |
| ● | 10.37.33.28 | Show all | True | 2.65 (67.77%) | 3.47 (88.64%) | 3.99Gi (31.16%) | 8.58Gi (66.97%) | 25 (22.73%) |
| ● | 10.37.33.24 | Show all | True | 2.61 (66.70%) | 3.45 (88.13%) | 3.62Gi (28.27%) | 5.00Gi (39.00%) | 22 (20.00%) |

background load

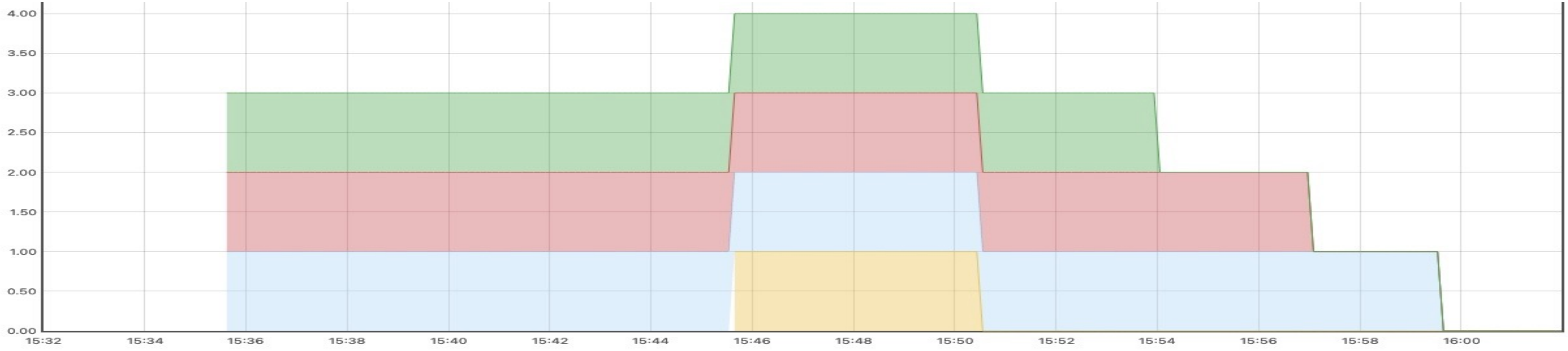
| | Name | Labels | Ready | CPU requests (cores) | CPU limits (cores) | Memory requests (bytes) | Memory limits (bytes) | Pods |
|---|---|---|---|---|---|---|---|---|
| ● | 10.37.33.21 | Show all | True | 2.22 (56.83%) | 5.87 (150.13%) | 6.00Gi (46.78%) | 6.96Gi (54.31%) | 11 (10.00%) |
| ● | 10.37.33.28 | Show all | True | 3.15 (80.56%) | 5.97 (152.58%) | 7.99Gi (62.37%) | 12.58Gi (98.18%) | 26 (23.64%) |
| ● | 10.37.33.24 | Show all | True | 3.11 (79.49%) | 5.95 (152.07%) | 7.62Gi (59.48%) | 9.00Gi (70.20%) | 23 (20.91%) |

# trimaran-3

{instance="10.37.33.21"}
{instance="10.37.33.24"}
{instance="10.37.33.28"}

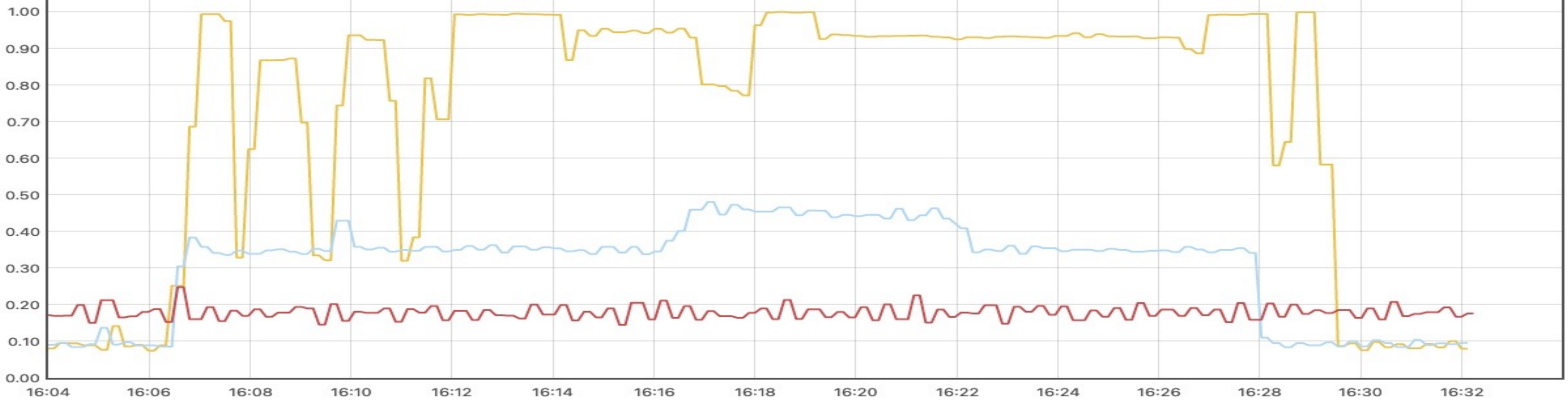
5:05

app-pod -> node-28



kube_pod_container_status_running{**container**="app-1-cnt", **instance**="172.30.9.242:8443", **job**="kube-state-metrics", **namespace**="default", **pod**="app-1", **uid**="b52cfb22-1498-4b4f-bf1d-b25f9498653b"}
kube_pod_container_status_running{**container**="parsec-bg-0-cnt", **instance**="172.30.9.242:8443", **job**="kube-state-metrics", **namespace**="default", **pod**="parsec-bg-0", **uid**="05eaeba2-461c-420d-a2c3-d7fb461996fb"}
kube_pod_container_status_running{**container**="parsec-bg-1-cnt", **instance**="172.30.9.242:8443", **job**="kube-state-metrics", **namespace**="default", **pod**="parsec-bg-1", **uid**="ec100dd4-fc38-429f-a013-05120f5fcada"}
kube_pod_container_status_running{**container**="parsec-bg-4-cnt", **instance**="172.30.9.242:8443", **job**="kube-state-metrics", **namespace**="default", **pod**="parsec-bg-4", **uid**="19ab50bc-226d-450d-96e2-019969ed693d"}

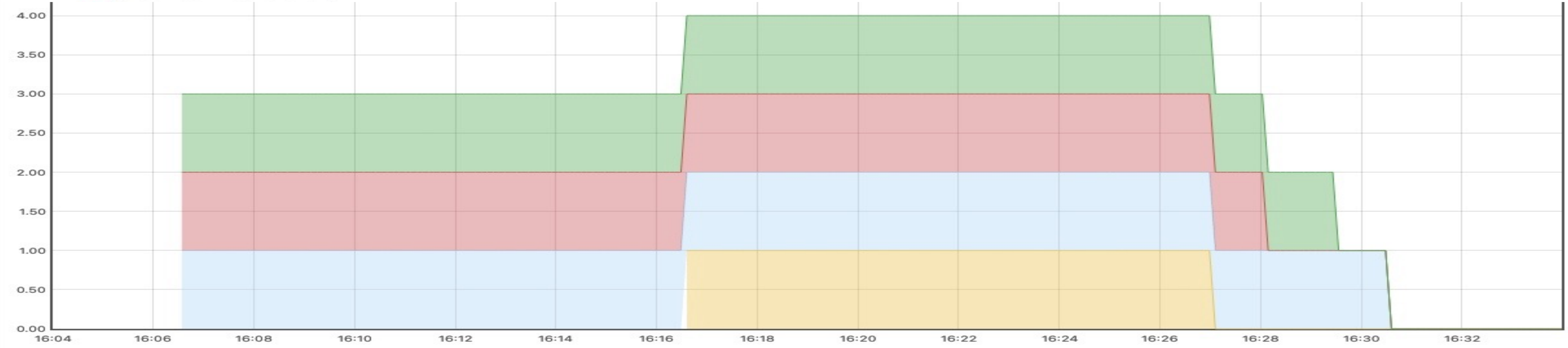
**default**




{instance="10.37.33.21"}
{instance="10.37.33.24"}
{instance="10.37.33.28"}

**10:13**

app-pod -> node-21

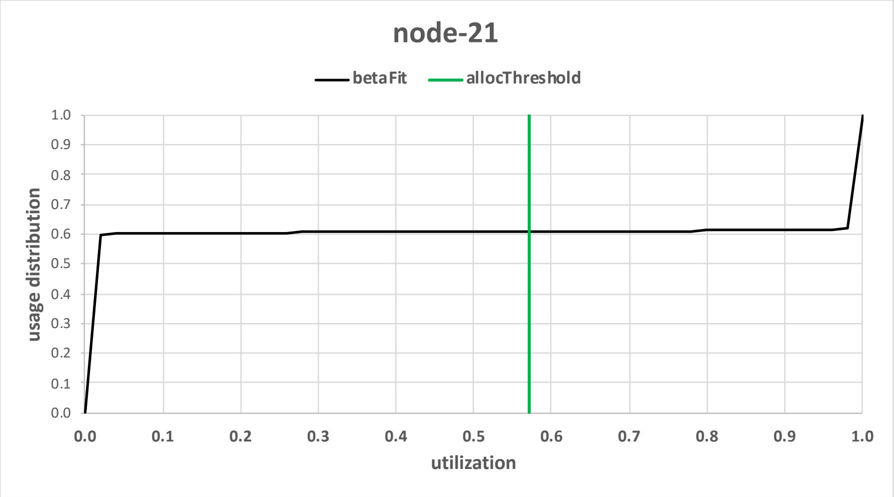


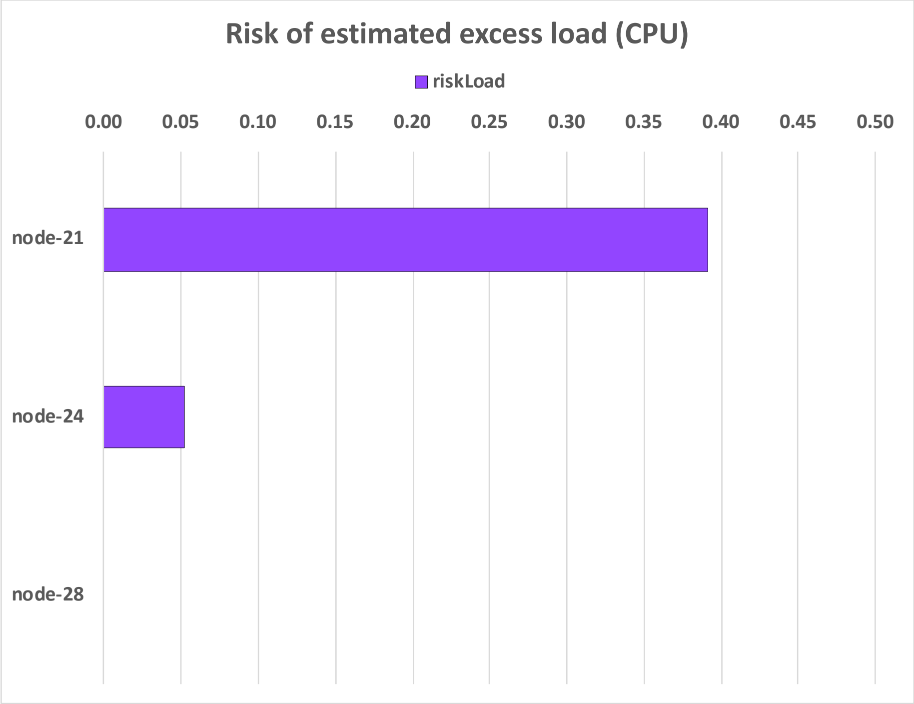

kube_pod_container_status_running{container="app-1-cnt", instance="172.30.9.242:8443", job="kube-state-metrics", namespace="default", pod="app-1", uid="13888bab-28a9-4c6c-85d2-d102ae135be6"}
kube_pod_container_status_running{container="parsec-bg-0-cnt", instance="172.30.9.242:8443", job="kube-state-metrics", namespace="default", pod="parsec-bg-0", uid="7c191f35-c0b2-41ac-aa41-f8f9115703c8"}
kube_pod_container_status_running{container="parsec-bg-1-cnt", instance="172.30.9.242:8443", job="kube-state-metrics", namespace="default", pod="parsec-bg-1", uid="dcb79d65-8500-438e-ac01-9a7ba2710b41"}
kube_pod_container_status_running{container="parsec-bg-4-cnt", instance="172.30.9.242:8443", job="kube-state-metrics", namespace="default", pod="parsec-bg-4", uid="d36fa8c4-f90f-46f1-b480-013bc1b548f2"}

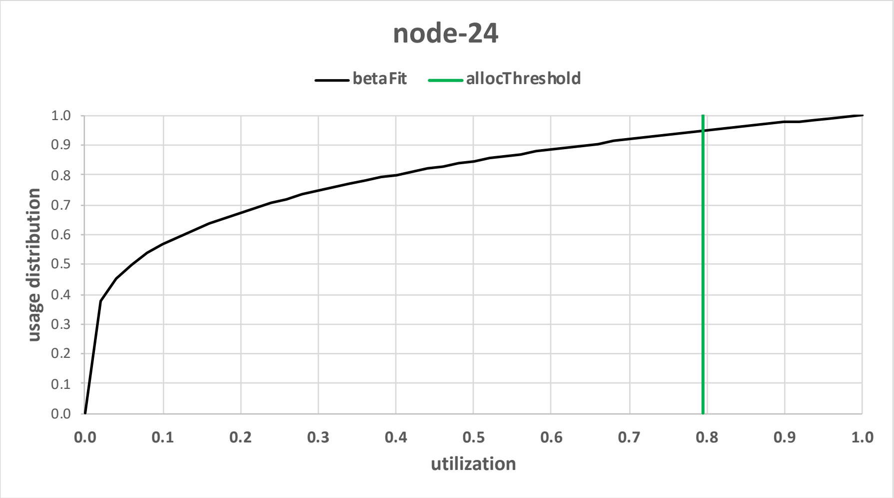
# Trimaran-3

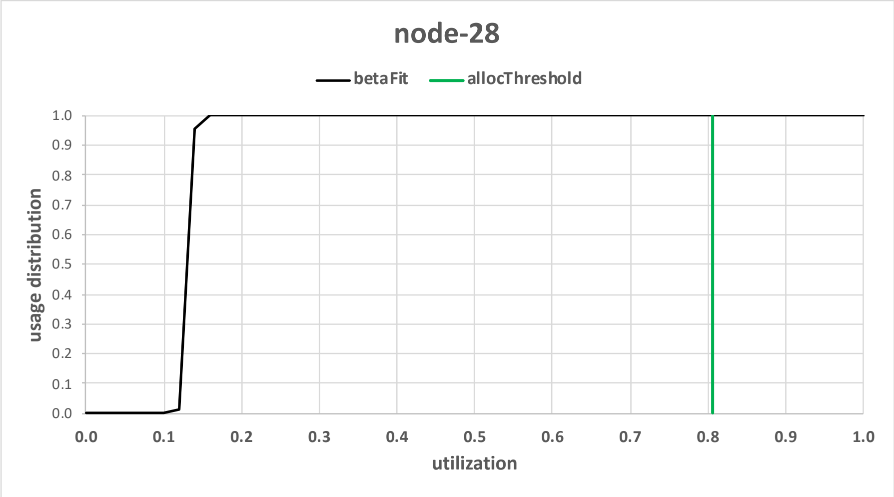
| parameters | |
|---|---|
| smoothingWindowSize | 5 |
| riskLimitWeight | 0.5 |

| | 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** | 2,732 | 3,608 | 3,650 |
| **limits** | 9,870 | 9,946 | 9,966 |
| **loadUsage** | | | |
| **usedAvg** | 1,532.08 | 764.67 | 512.93 |
| **usedStd** | 1,244.43 | 459.17 | 9.07 |
| **alpha** | 0.004 | 0.251 | 555.906 |
| **beta** | 0.006 | 1.031 | 3,681.706 |
| **mean** | 0.392 | 0.196 | 0.131 |
| **var** | 0.236 | 0.069 | 0.000 |
| **sigma** | 0.486 | 0.263 | 0.005 |
| **overUsage** | | | |
| **allocThreshold** | 0.571 | 0.795 | 0.806 |
| **allocProb** | 0.609 | 0.948 | 1.000 |
| **overUse** | 0.391 | 0.052 | 0.000 |
| **risk** | | | |
| **riskLimit** | 0.835 | 0.952 | 0.959 |
| **riskLoad** | 0.391 | 0.052 | 0.000 |
| **totalRisk** | 0.613 | 0.502 | 0.479 |
| **score** | | | |
| **rank** | **0.387** | **0.498** | **0.521** |
| **totalScore** | **39** | **50** | **52** |

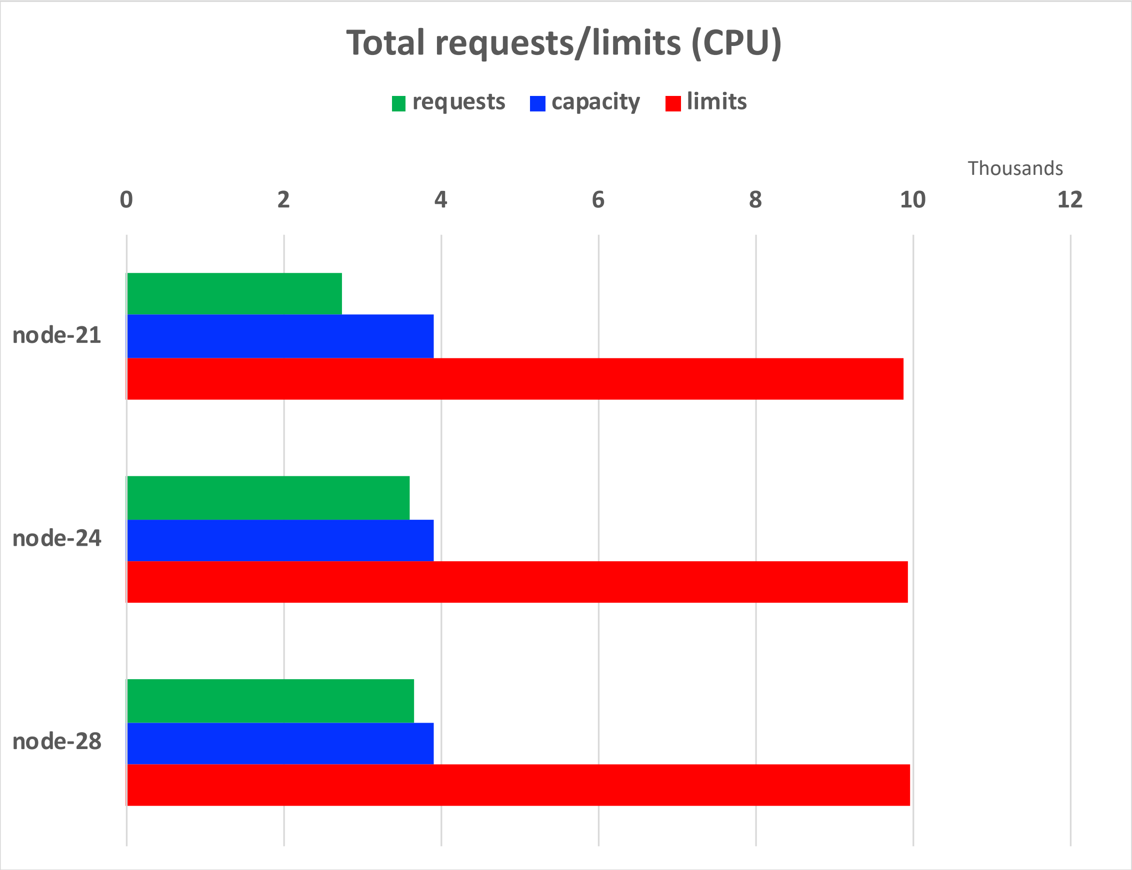

node-21
betaFit
allocThreshold
usage distribution
utilization
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

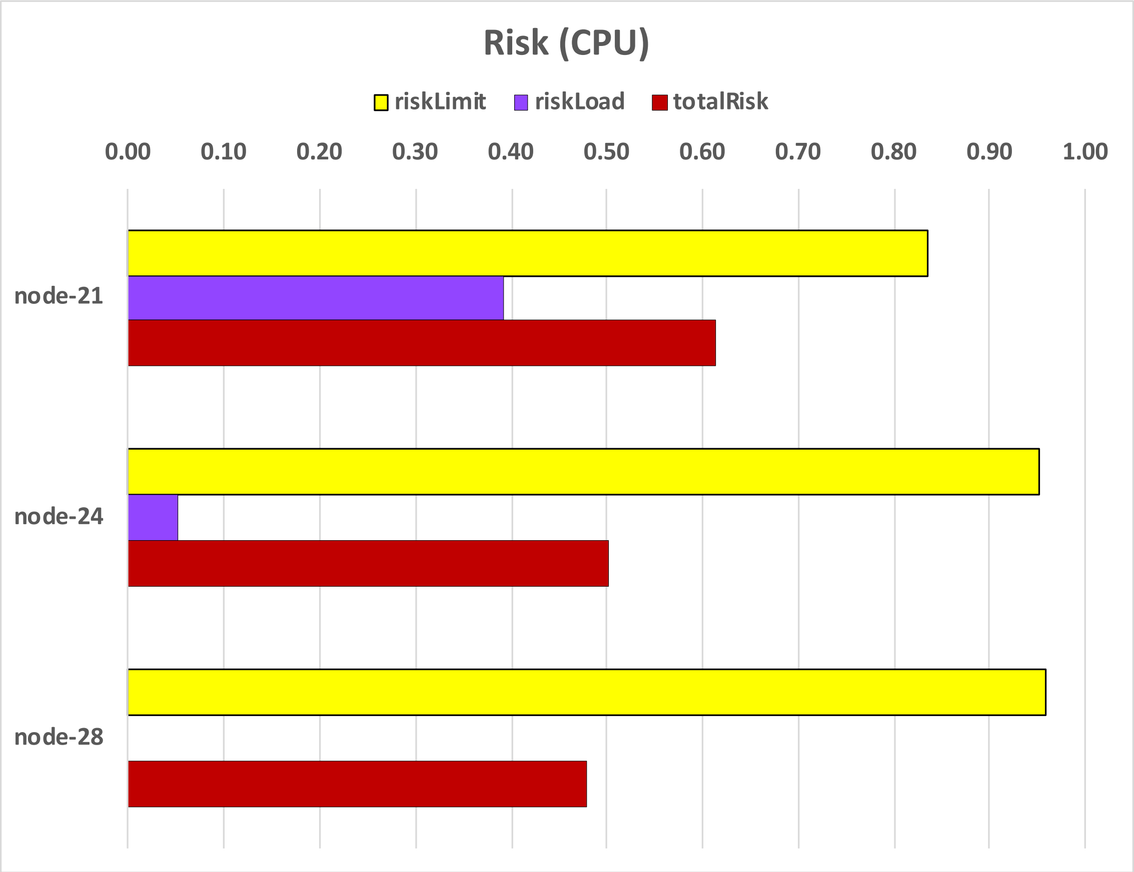

Risk of estimated excess load (CPU)
riskLoad
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
node-21
node-24
node-28

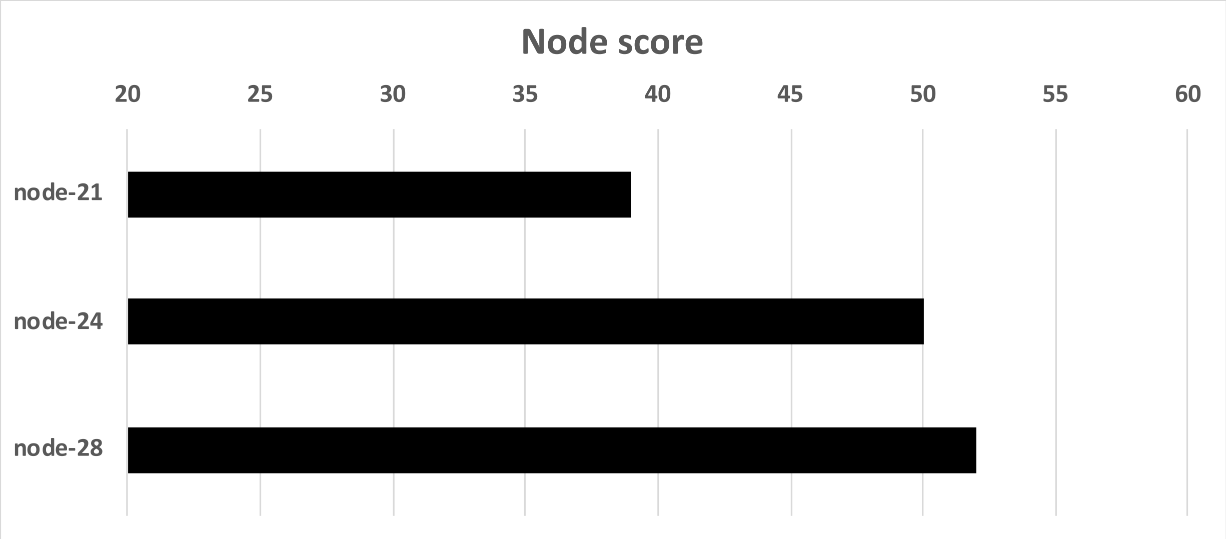

node-24
betaFit
allocThreshold
usage distribution
utilization
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


node-28
betaFit
allocThreshold
usage distribution
utilization
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

# Trimaran-3


Total requests/limits (CPU)
requests
capacity
limits
Thousands
0
2
4
6
8
10
12
node-21
node-24
node-28


Risk (CPU)
riskLimit
riskLoad
totalRisk
0.00
0.10
0.20
0.30
0.40
0.50
0.60
0.70
0.80
0.90
1.00
node-21
node-24
node-28


Node score
20
25
30
35
40
45
50
55
60
node-21
node-24
node-28

# Trimaran-3