50% Utilization and Beyond

Apache Mesos Oversubscription in Practice

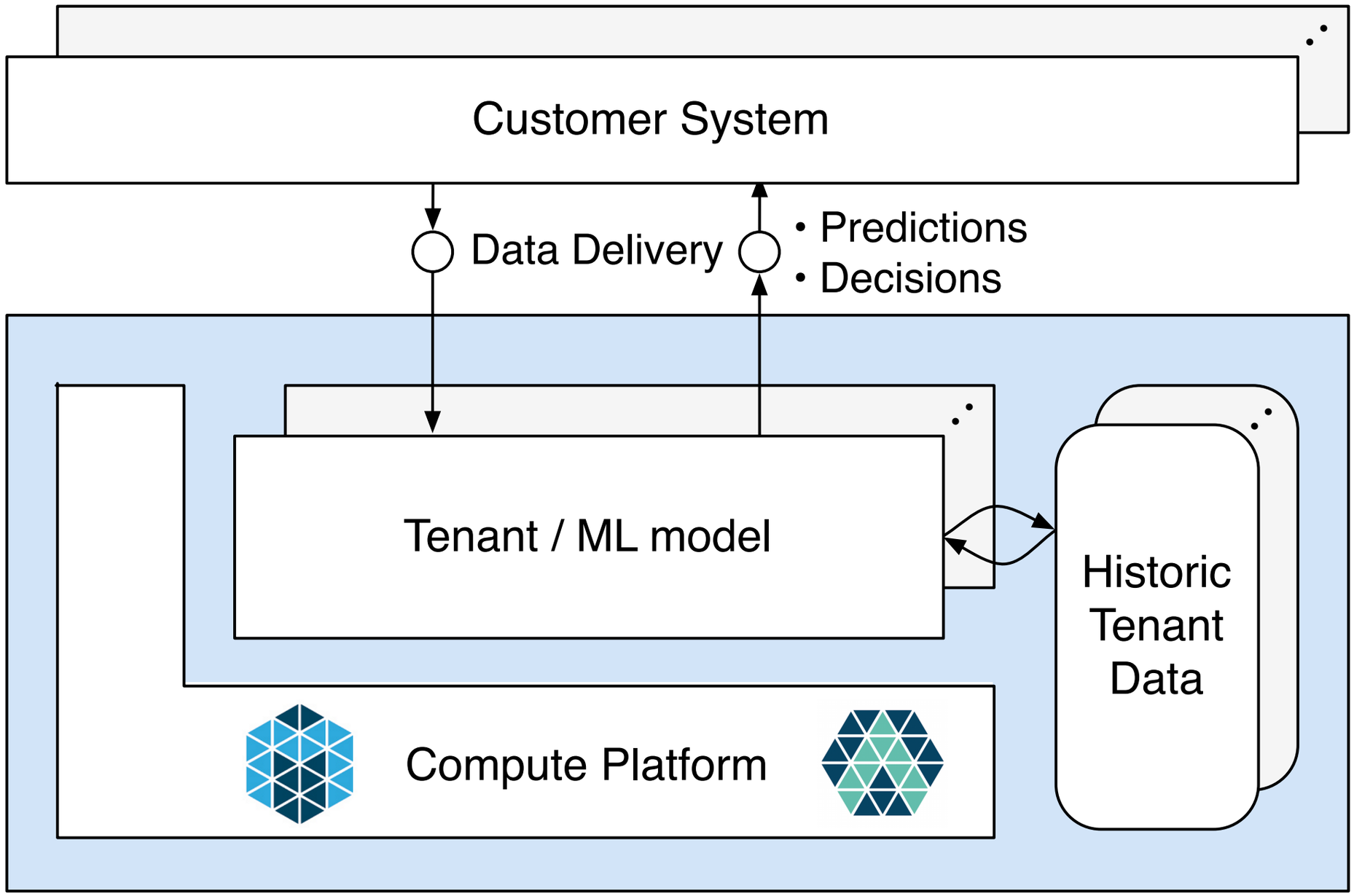
ContainerSched
London 2017

Stephan Erb
@ErbStephan

BlueYonder
Best decisions, delivered daily

blue yonder

Photo by liz west https://flic.kr/p/7qYh21


Customer System
Data Delivery
• Predictions
• Decisions
Tenant / ML model
Historic Tenant Data
Compute Platform

Apache Aurora

https://aurora.apache.org

Mesos framework for the deployment and scaling of stateless and fault tolerant services in a datacenter

Apache Mesos

https://mesos.apache.org

Cluster manager providing fault-tolerant, fine-grained multitenancy via containers

Apache Aurora

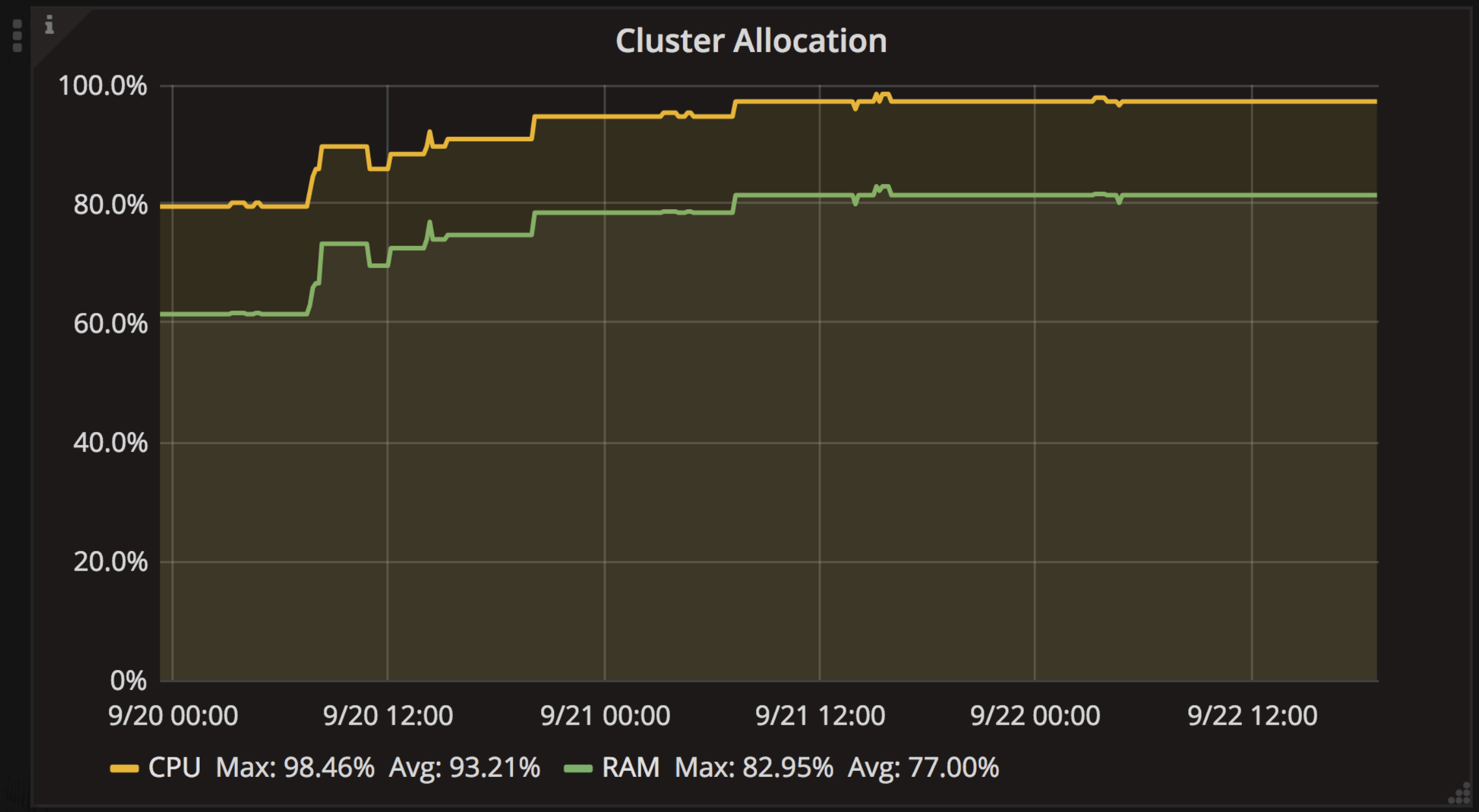
https://aurora.apache.org

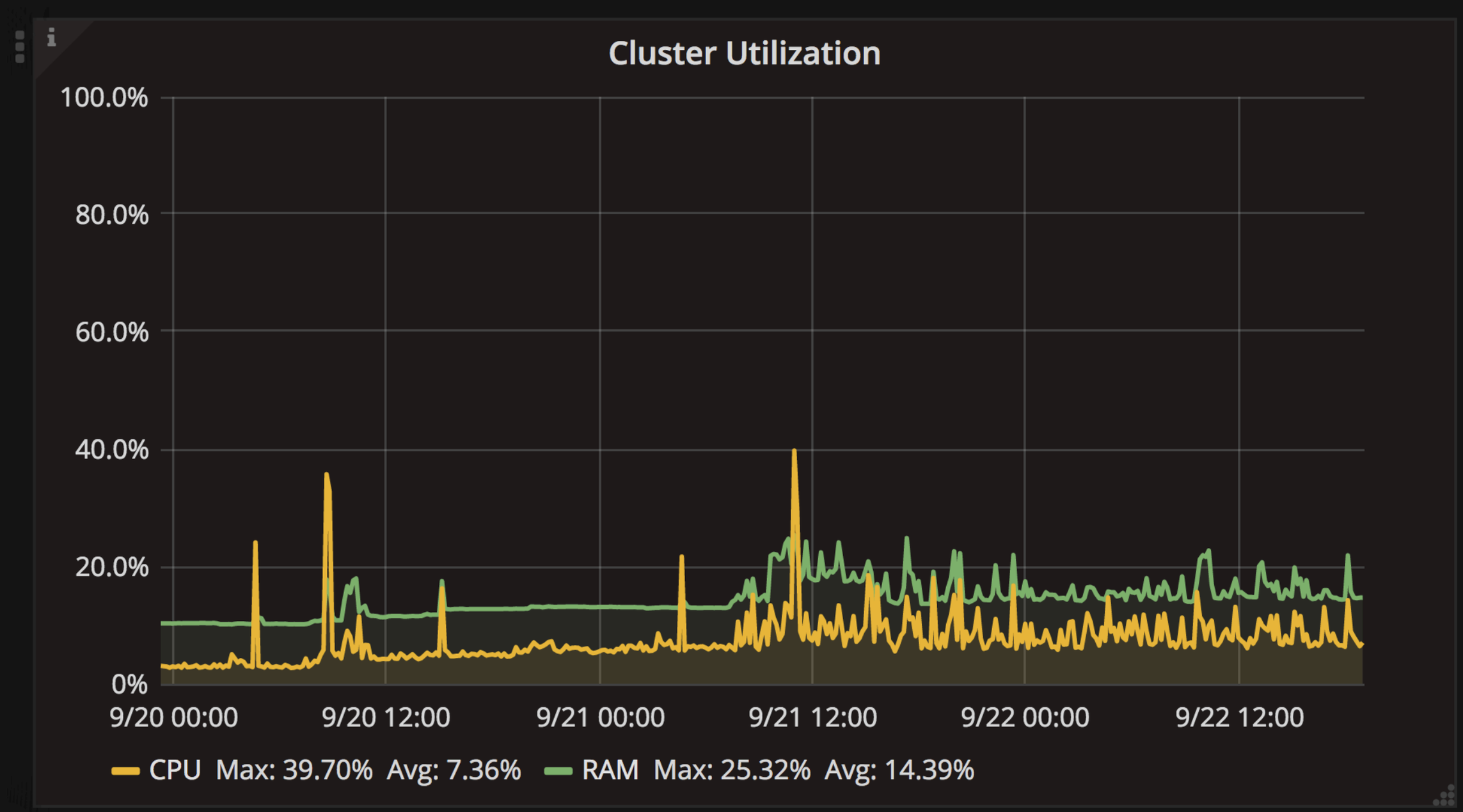
"distributed supervisord"

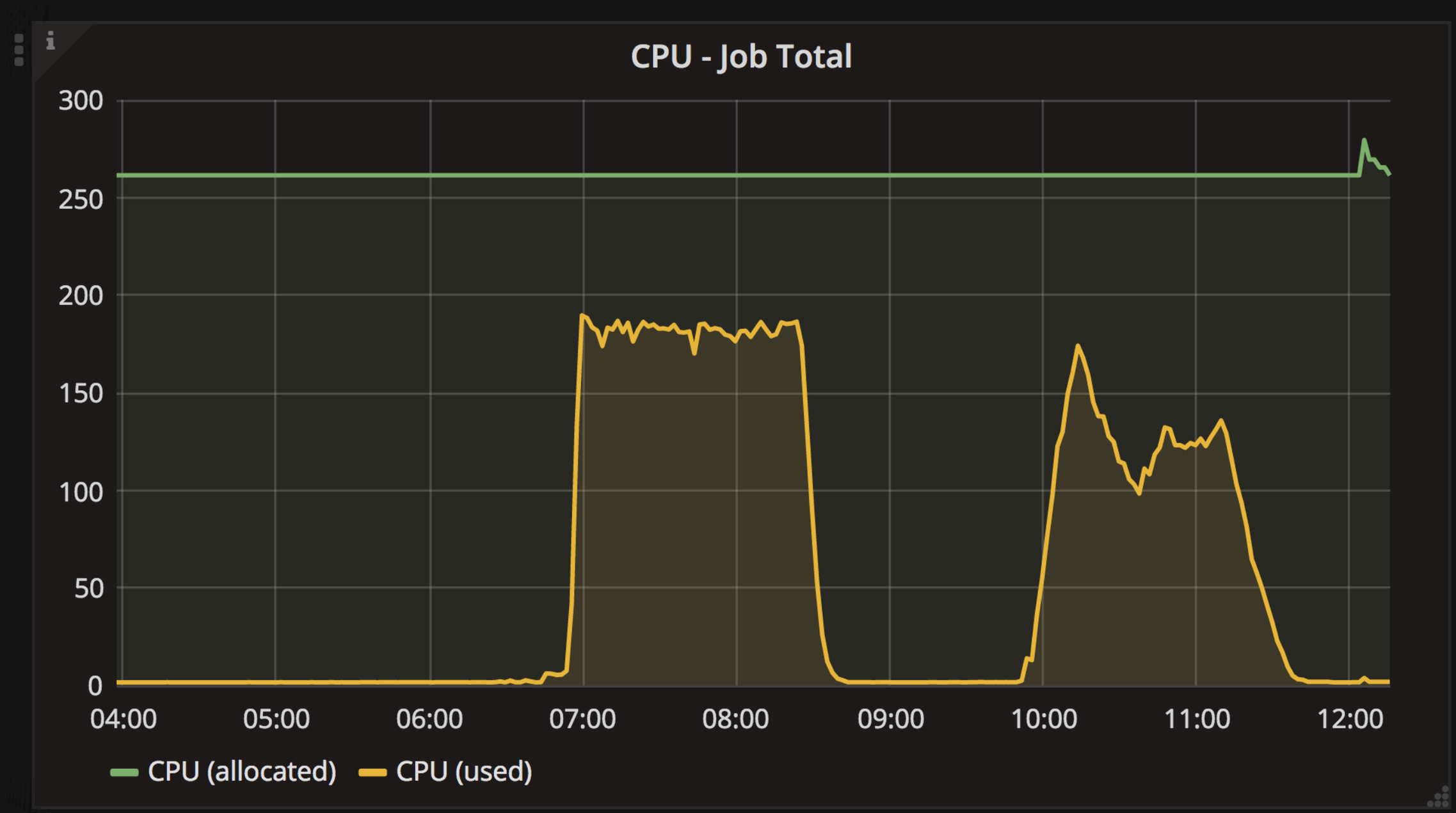
Apache Mesos

https://mesos.apache.org

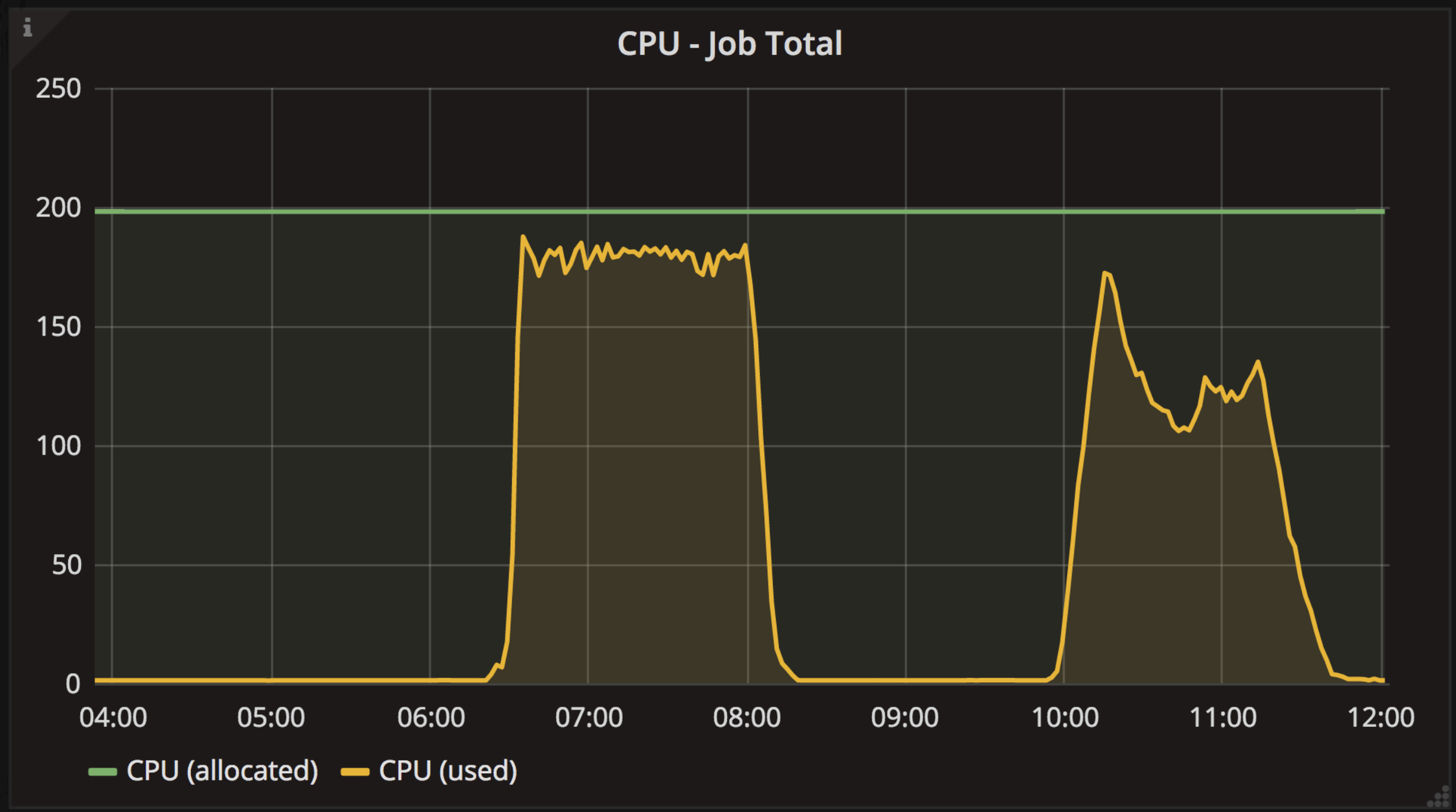
"plumbing"


Cluster Allocation
100.0%
80.0%
60.0%
40.0%
20.0%
0%
9/20 00:00
9/20 12:00
9/21 00:00
9/21 12:00
9/22 00:00
9/22 12:00
CPU Max: 98.46% Avg: 93.21%
RAM Max: 82.95% Avg: 77.00%


Cluster Utilization
100.0%
80.0%
60.0%
40.0%
20.0%
0%
9/20 00:00
9/20 12:00
9/21 00:00
9/21 12:00
9/22 00:00
9/22 12:00
CPU Max: 39.70% Avg: 7.36%
RAM Max: 25.32% Avg: 14.39%

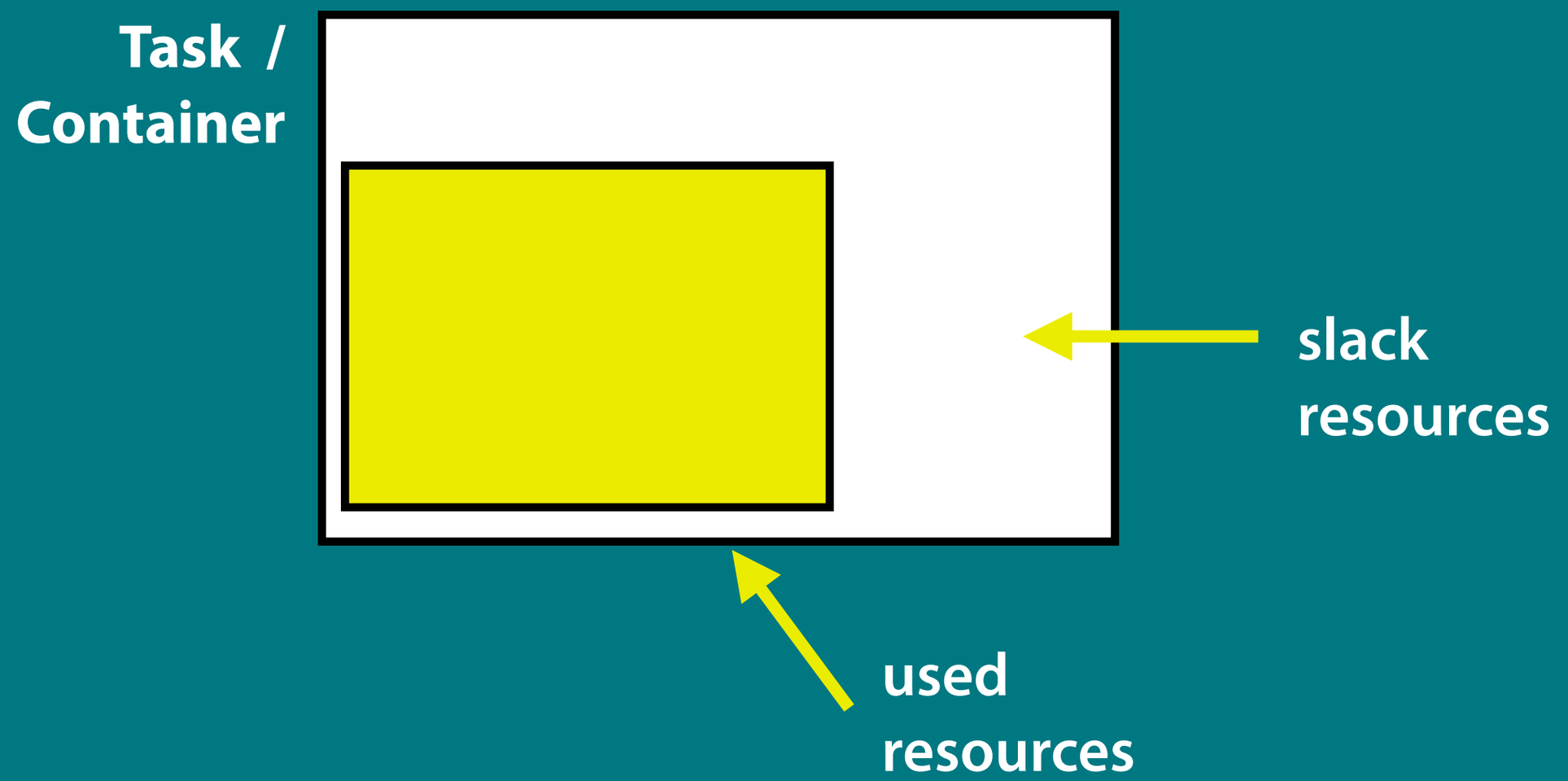

CPU - Job Total
300
250
200
150
100
50
0
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
CPU (allocated)
CPU (used)

Monitoring & Observability


CPU - Job Total
250
200
150
100
50
0
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
CPU (allocated)
CPU (used)

Oversubscription

Use slack resources of production tasks to run low priority best effort tasks



Large-scale cluster management at Google with Borg

Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

1. Introduction

The cluster management system we internally call Borg admits, schedules, starts, restarts, and monitors the full range of applications that Google runs. This paper explains how.

Borg provides three main benefits: it (1) hides the details of resource management and failure handling so its users can focus on application development instead; (2) operates with very high reliability and availability, and supports applications that do the same; and (3) lets us run workloads across tens of thousands of machines effectively. Borg is not the first system to address these issues, but it's one of the few operating at this scale, with this degree of resiliency and completeness. This paper is organized around these topics, concluding with a set of qualitative observations we have made from operating Borg in production for more than a decade.

[†] Work done while author was at Google.

[‡] Currently at University of Southern California.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

EuroSys'15, April 21–24, 2015, Bordeaux, France.
Copyright is held by the owner/author(s).
ACM 978-1-4503-3238-5/15/04.
http://dx.doi.org/10.1145/2741948.2741964

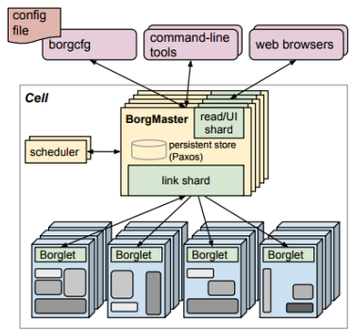


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

2. The user perspective

Borg's users are Google developers and system administrators (site reliability engineers or SREs) that run Google's applications and services. Users submit their work to Borg in the form of *jobs*, each of which consists of one or more *tasks* that all run the same program (binary). Each job runs in one Borg *cell*, a set of machines that are managed as a unit. The remainder of this section describes the main features exposed in the user view of Borg.

2.1 The workload

Borg cells run a heterogenous workload with two main parts. The first is long-running services that should "never" go down, and handle short-lived latency-sensitive requests (a few μs to a few hundred ms). Such services are used for end-user-facing products such as Gmail, Google Docs, and web search, and for internal infrastructure services (e.g., BigTable). The second is batch jobs that take from a few seconds to a few days to complete; these are much less sensitive to short-term performance fluctuations. The workload mix varies across cells, which run different mixes of applications depending on their major tenants (e.g., some cells are quite batch-intensive), and also varies over time: batch jobs

Large-scale cluster management at Google with Borg

- large clusters / cells
- multiple users
- heterogenous workloads
  - prod service
  - non-prod service
  - batch jobs

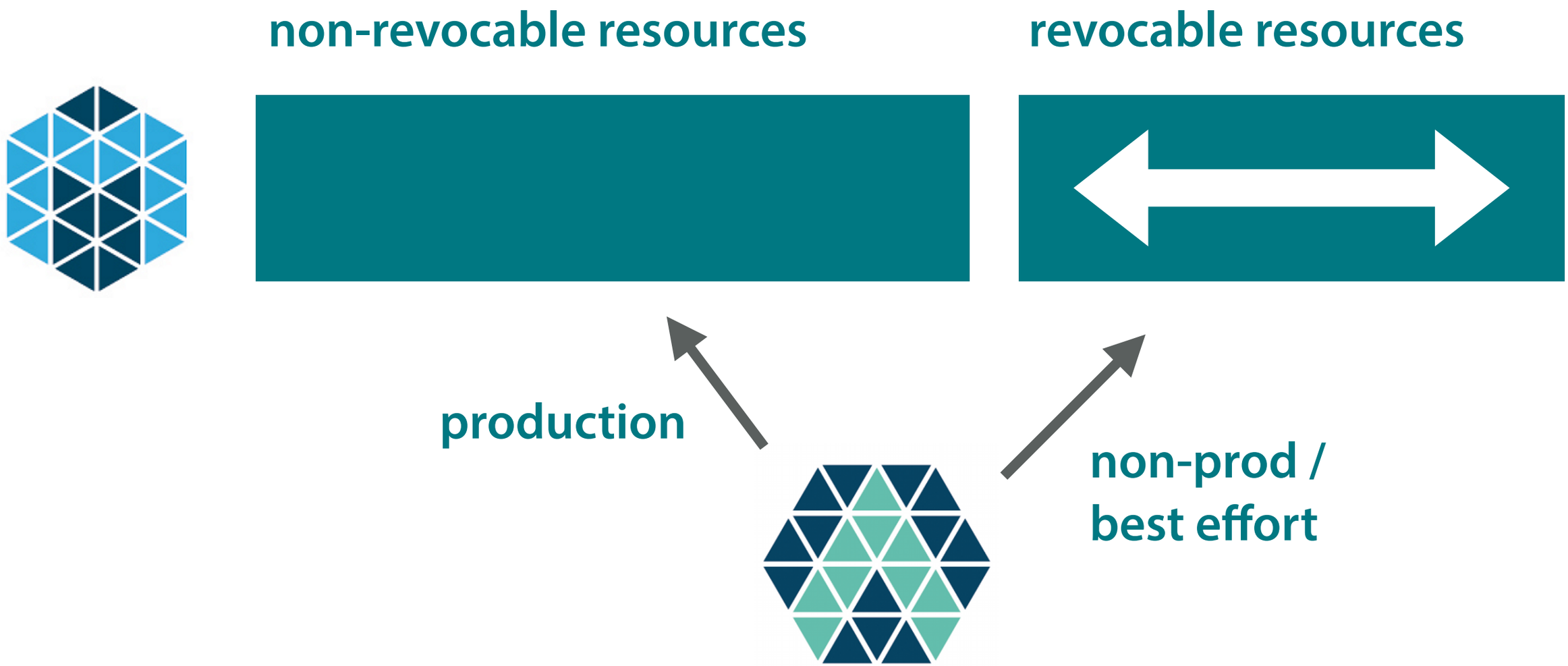
http://dl.acm.org/ft_gateway.cfm?id=2741964&type=pdf

Mesos Oversubscription

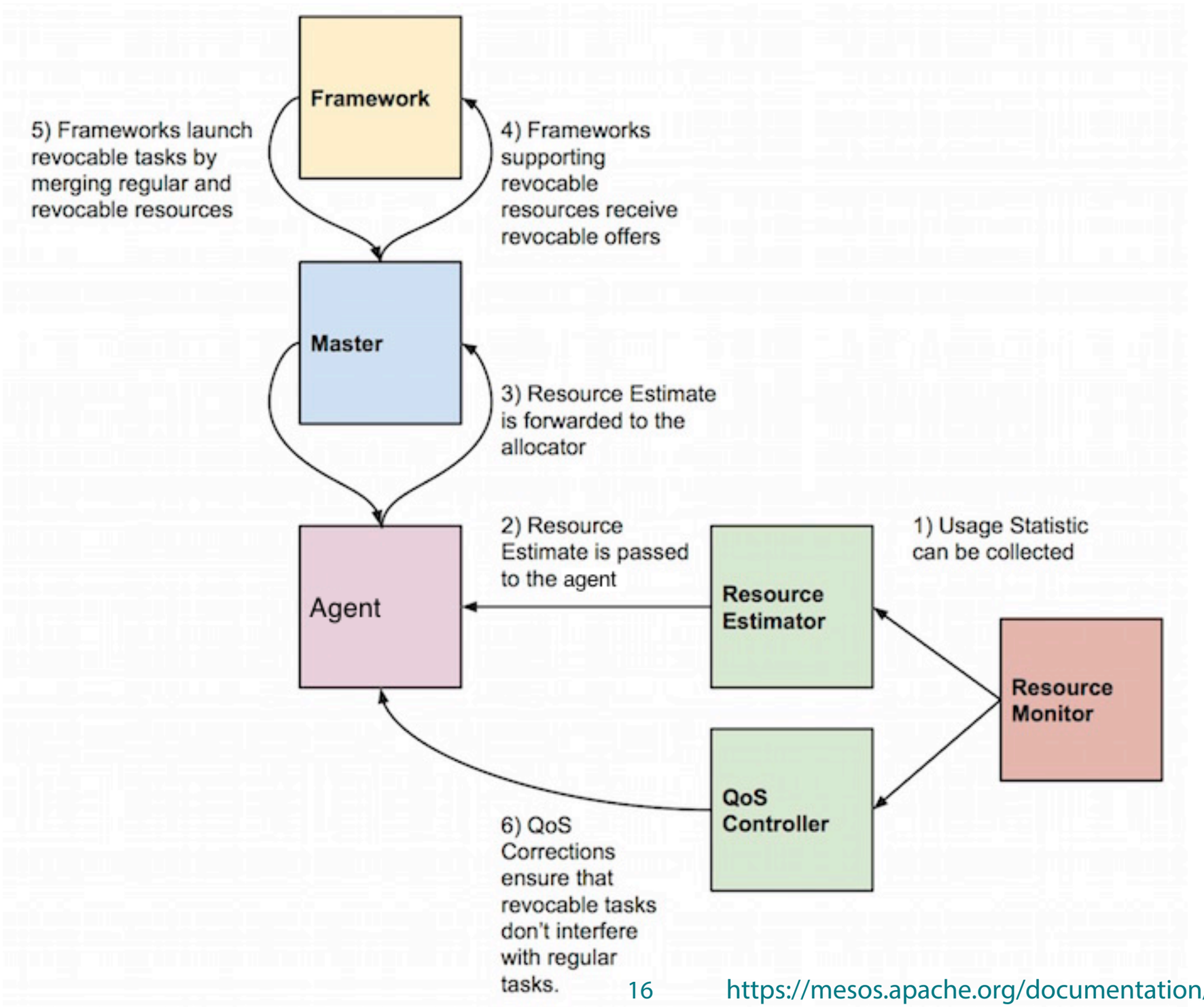
non-revocable resources

revocable resources

Mesos Oversubscription



Mesos Oversubscription



https://mesos.apache.org/documentation/latest/oversubscription/

Photo by Jan Tik https://flic.kr/p/29Jjq

Our way out:

Do the simplest thing that could possibly work…

Components

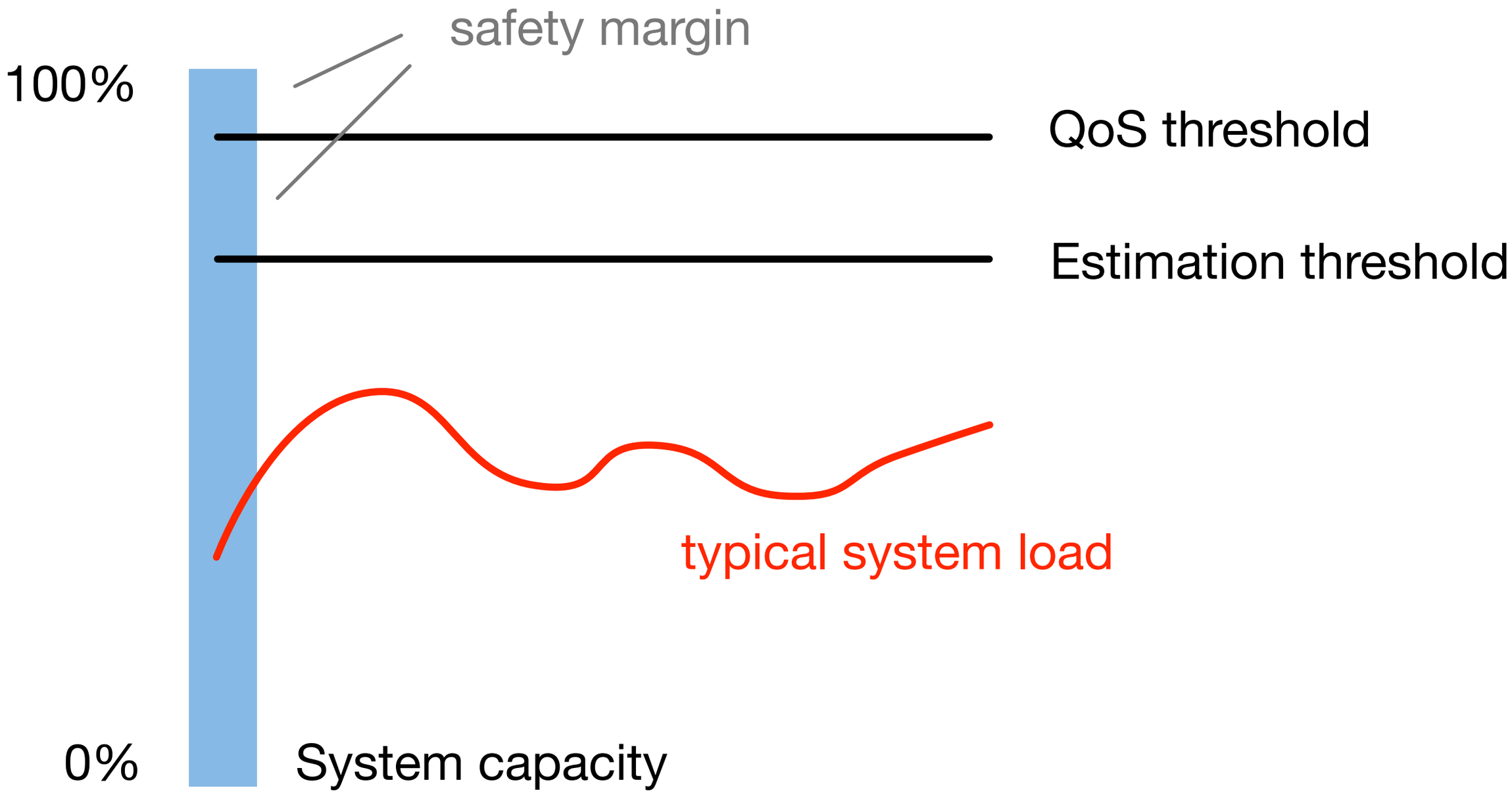
Threshold Resource Estimator:

- **advertises a fixed** amount of oversubscribable revocable **resources** per agent
- no further scheduling once utilization threshold reached

Threshold QoS Controller:

- **kill** revocable tasks if resource estimations that turned out to be over-optimistic
- decision based on second utilisation threshold

System Threshold Approach



Signals

Linux system load

- counts running, waiting, and tasks in uninterruptible sleep, i.e. all threads not completely idle
- 1m, 5m, 15m moving average indicates if load is increasing or decreasing

Memory Usage

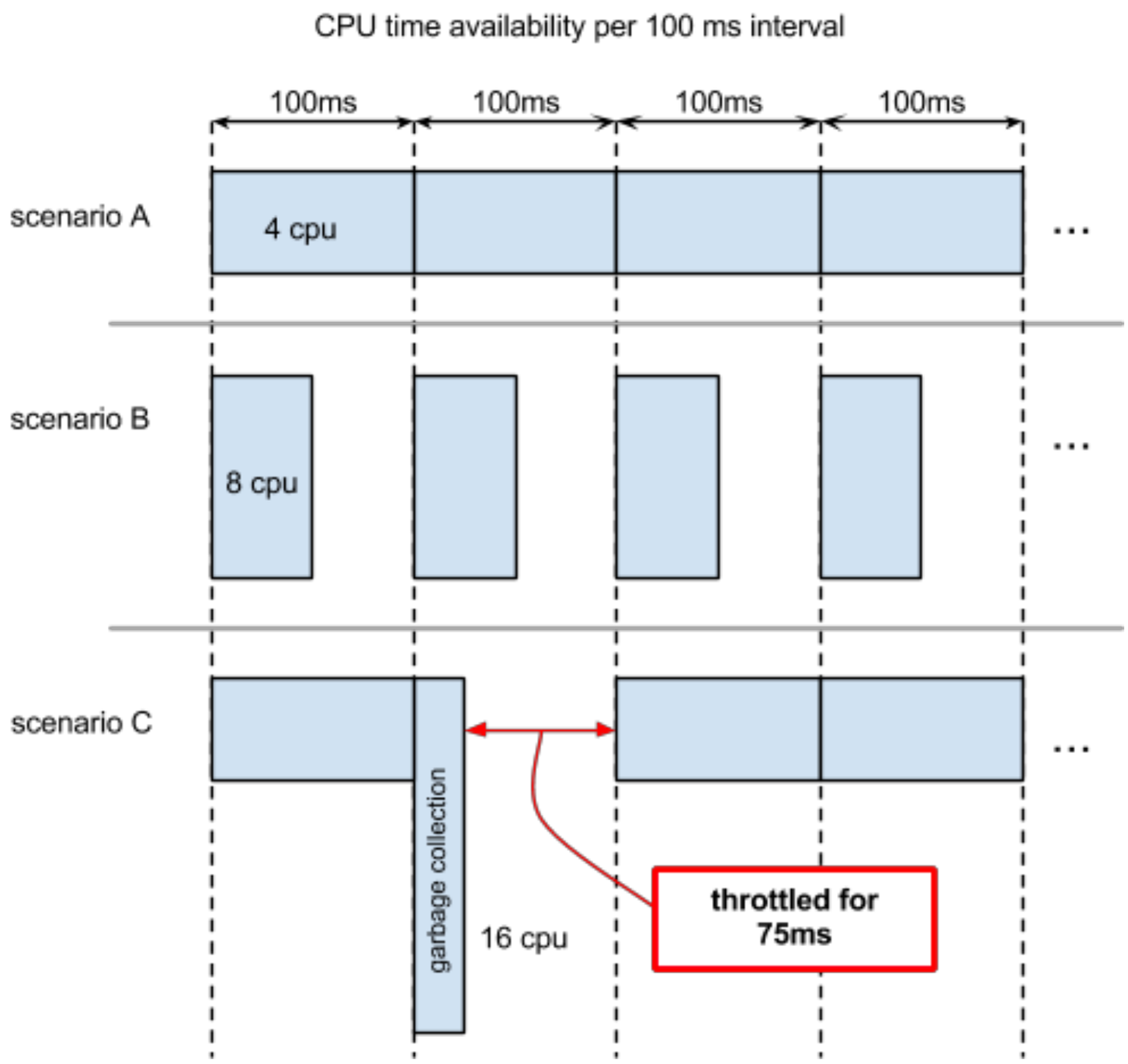
- current memory available for applications (excluding caches, buffers, …)

CPU Shares (cgroups)

		revocable	revocable
4 cores	8 cores	2 cores	8 cores
4 * 1024 shares	8 * 1024 shares	2 * 10 shares	8 * 10 shares

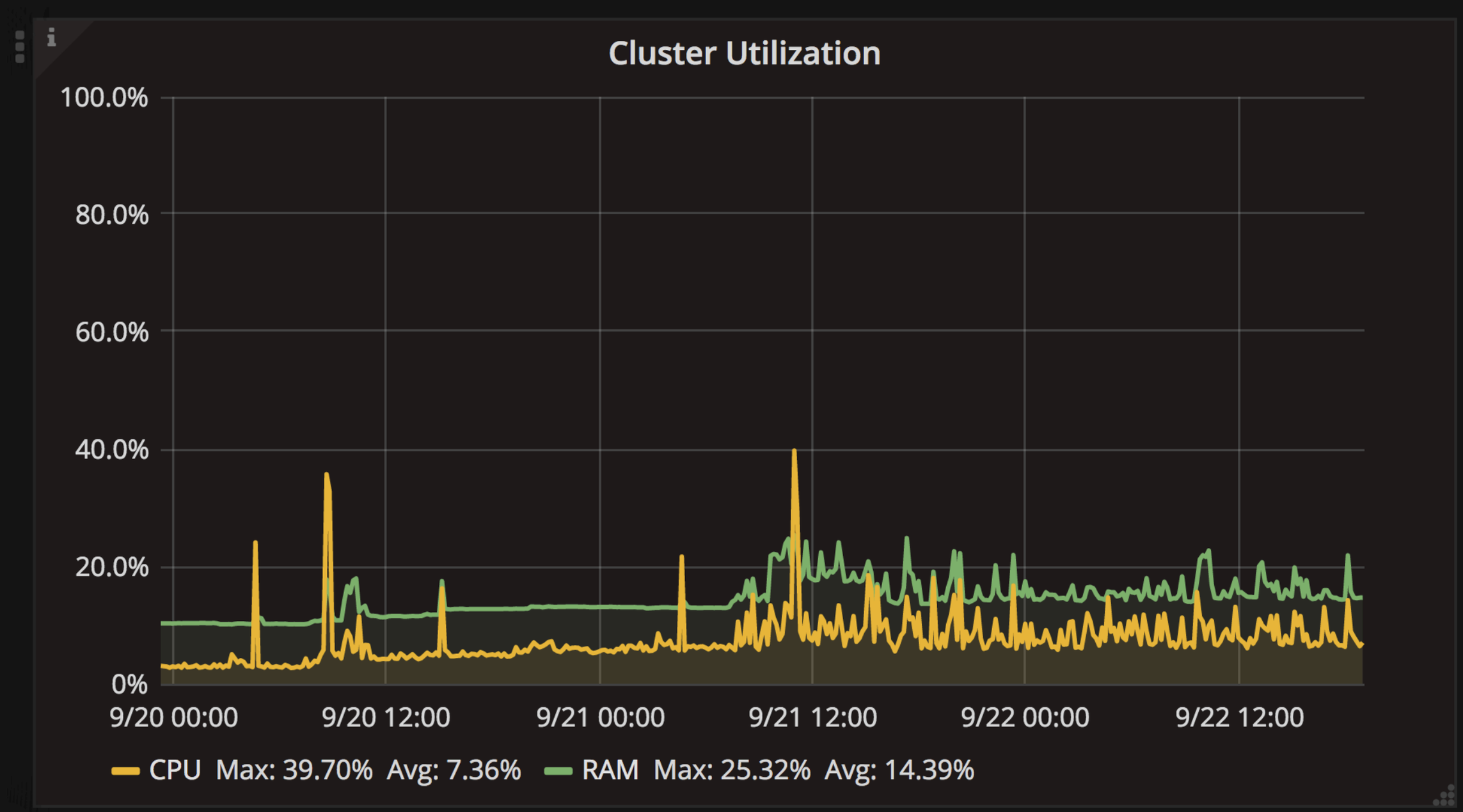
- runtime is weighted inverse proportion to the number of CPU shares: **weighted runtime = runtime / shares**
- groups with smallest weighted virtual runtime are eligible for execution

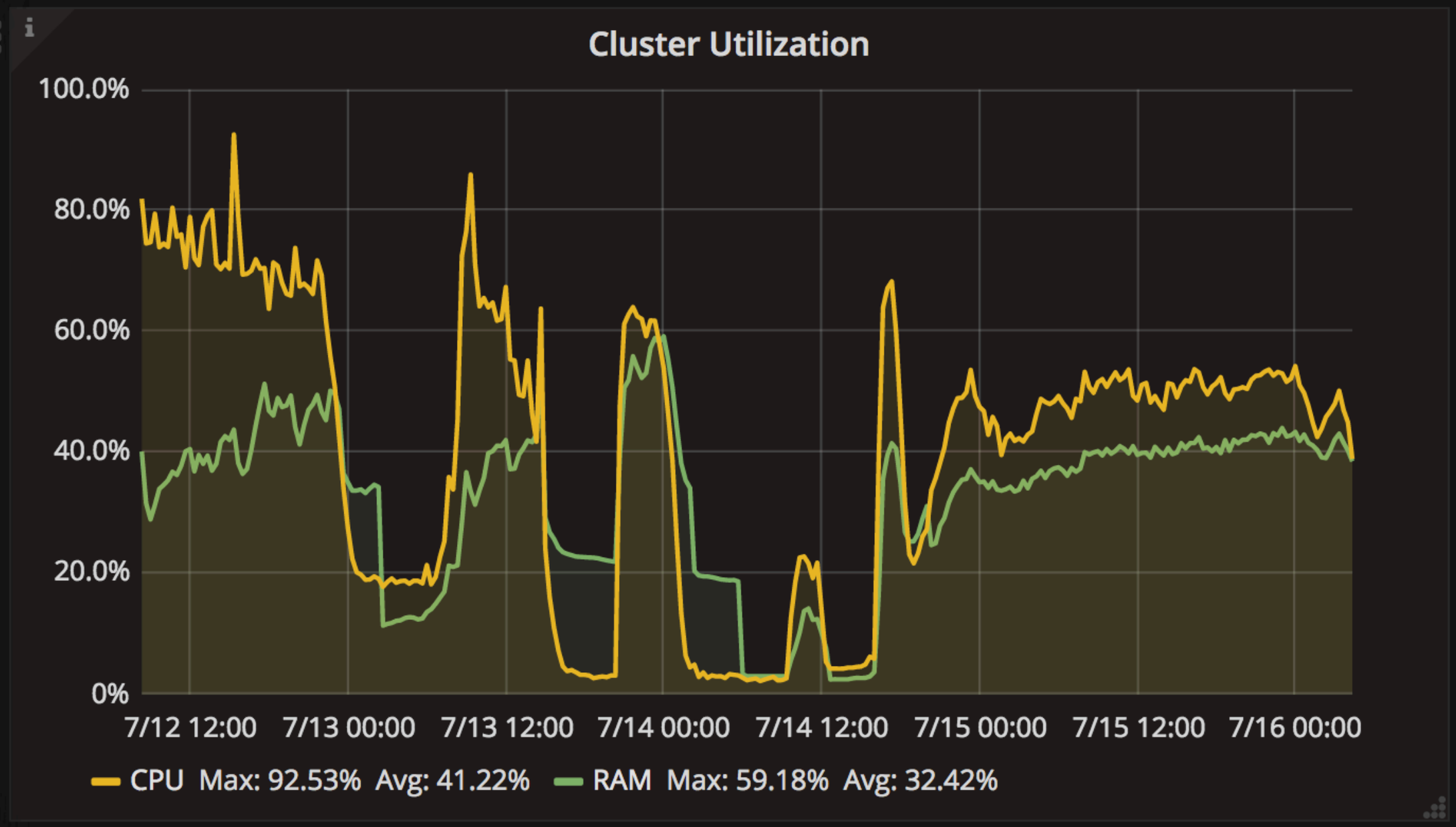
CFS bandwidth limiting

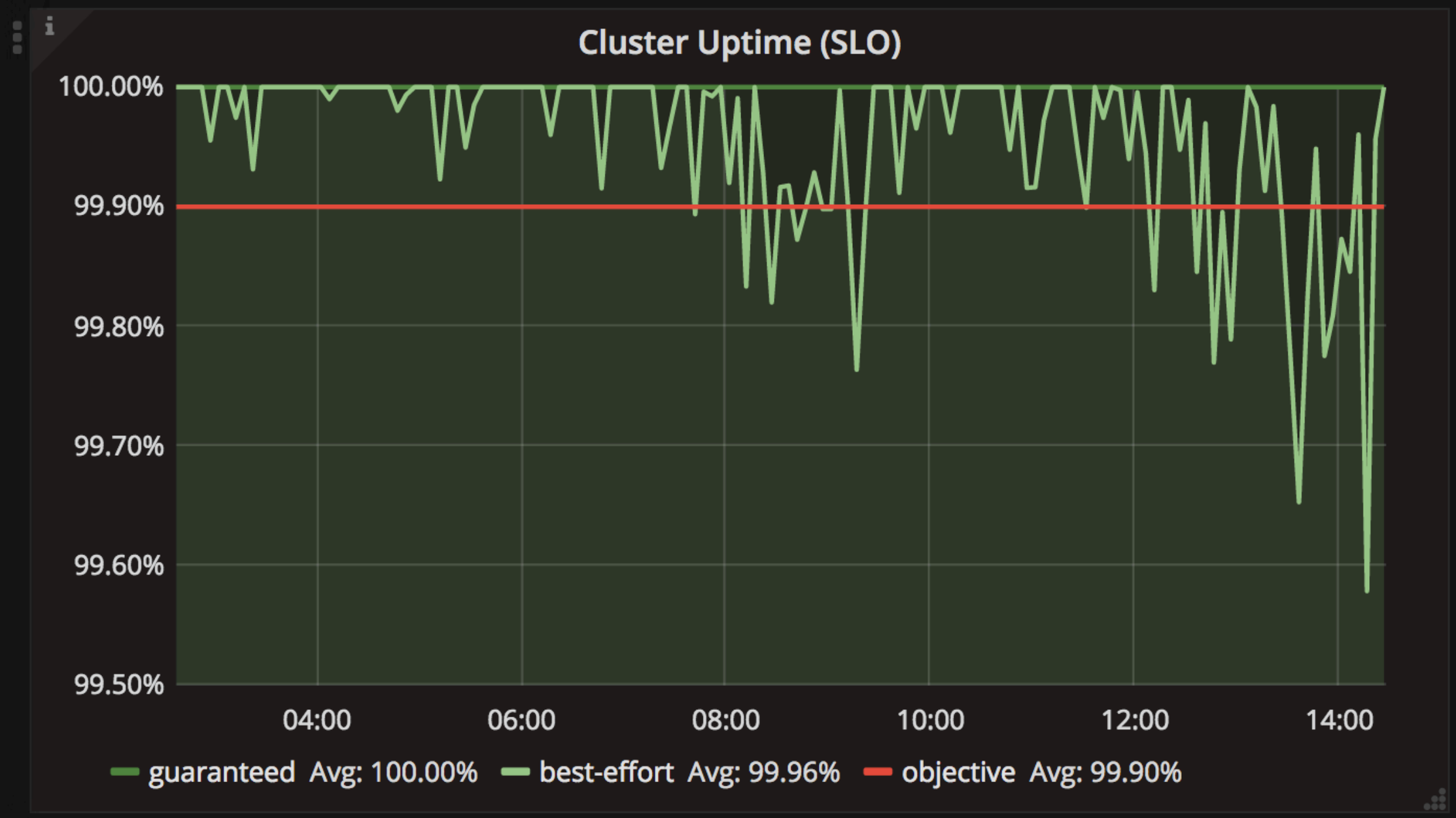


https://aurora.apache.org/documentation/latest/features/resource-isolation/

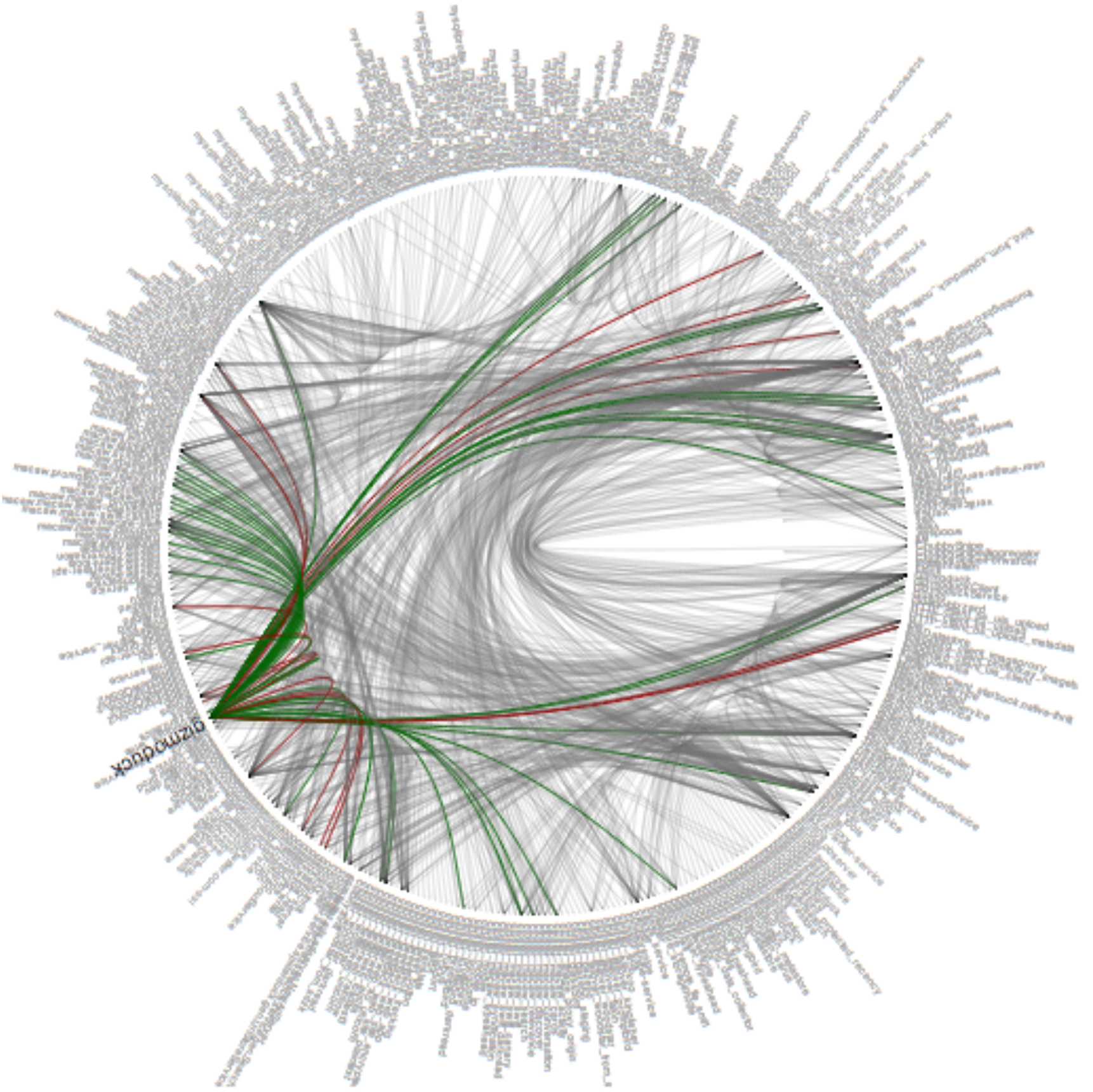
Photo by Dusty Culver https://flic.kr/p/8LyZz1


Cluster Utilization
100.0%
80.0%
60.0%
40.0%
20.0%
0%
9/20 00:00
9/20 12:00
9/21 00:00
9/21 12:00
9/22 00:00
9/22 12:00
CPU Max: 39.70% Avg: 7.36%
RAM Max: 25.32% Avg: 14.39%


Cluster Utilization
100.0%
80.0%
60.0%
40.0%
20.0%
0%
7/12 12:00
7/13 00:00
7/13 12:00
7/14 00:00
7/14 12:00
7/15 00:00
7/15 12:00
7/16 00:00
CPU Max: 92.53% Avg: 41.22%
RAM Max: 59.18% Avg: 32.42%


Cluster Uptime (SLO)
100.00%
99.90%
99.80%
99.70%
99.60%
99.50%
04:00
06:00
08:00
10:00
12:00
14:00
guaranteed Avg: 100.00%
best-effort Avg: 99.96%
objective Avg: 99.90%

```
ALERT ServiceLevelObjectiveBreached
  IF (
        sum(
          rate(http_request_latency_seconds_bucket{le="1"}[5m])
        ) by (job)
      /
        sum(
          rate(http_request_latency_seconds_count[5m])
        ) by (job)
    ) < 0.95
  FOR 5m
  LABELS { severity = "page" }
  ANNOTATIONS {
    summary = "Service {{$labels.job}} is slow.",
  }
```


gizmoduck

Ooopses

- running out of **disk** and **ports** (non-revocable resources)
  - subsequently overlapping Mesos and ephemeral port range
- running into **ZooKeeper** connection limits
  - subsequently running out of file descriptors
- Aurora Thermos **UI slowdowns** due to high number of containers per host

Try it!

https://github.com/
blue-yonder/
mesos-threshold-oversubscription

Additional References

(on top of the other links within the talk)

- **Linux Load Averages: Solving the Mystery**
  - http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html
- **CFS per-entity load tracking**
  - https://lwn.net/Articles/504013/
- **CPI 2: CPU performance isolation for shared compute clusters**
  - https://pdfs.semanticscholar.org/5848/da5058fed3b97bfd801ca19e5265f489abfe.pdf
- **The Tail at Scale**
  - https://research.google.com/pubs/pub40801.html
- **Reconciling high server utilization and sub-millisecond quality-of-service**
  - https://pdfs.semanticscholar.org/6d44/790b6d952eff28f302998e8121f90786e3ff.pdf
- **Idle Resources Oversubscription: Improving Cluster Utilization**
  - https://www.youtube.com/watch?v=qrulRFp3UEY&index=50&list=PLbzoR-pLrL6qAEnkhkh5tGl6oX_xXD3X4
- **Don't Let Linux Control Groups Run Uncontrolled**
  - https://engineering.linkedin.com/blog/2016/08/don_t-let-linux-control-groups-uncontrolled

Thank you!

ContainerSched
London 2017

Stephan Erb
@ErbStephan

BlueYonder
Best decisions, delivered daily

Blue Yonder
Best decisions, delivered daily

Blue Yonder GmbH
Ohiostraße 8
76149 Karlsruhe
Germany
+49 721 383117 0

Blue Yonder Software Limited
19 Eastbourne Terrace
London, W2 6LG
United Kingdom
+44 20 3626 0360

Blue Yonder Analytics, Inc.
5048 Tennyson Parkway
Suite 250
Plano, Texas 75024
USA

BlueYonder
Best decisions, delivered daily