Scheduling in Warehouse-scale Data Centers

Francisco Brasileiro

fubica@computacao.ufcg.edu.br

COMPUTAÇÃO UFCG

Universidade Federal de Campina Grande

Agenda

- Day one: state-of-practice
  - What is a warehouse-scale data center
  - Typical workloads
  - The scheduling problem
  - Google's Borg priority-based scheduler
- Day two: QoS-driven scheduling
  - Typical Service Level Agreements (SLA)
  - Limitations of priority-based schedulers
  - QoS-driven scheduling
  - Availability-driven scheduler

Typical Service Level Agreements (SLA)

AWS (https://aws.amazon.com/compute/sla/)

- The promise
  - "AWS will use commercially reasonable efforts to make the Included Services each available for each AWS region with a Monthly Uptime Percentage of at least 99.99%, in each case during any monthly billing cycle (the 'Service Commitment')"

Typical Service Level Agreements (SLA)

AWS (https://aws.amazon.com/compute/sla/)

- The penalties
  - "In the event any of the Included Services do not meet the Service Commitment, you will be eligible to receive a Service Credit calculated as a percentage of the total charges paid by you … for the individual Included Service in the affected AWS region for the monthly billing cycle in which the Unavailability occurred …"

Typical Service Level Agreements (SLA)

AWS (https://aws.amazon.com/compute/sla/)

- Penalty calculation

Monthly Uptime Percentage	Service Credit Percentage
Less than 99.99% but equal to or greater than 99.0%	10%
Less than 99.0% but equal to or greater than 95.0%	30%
Less than 95.0%	100%

Typical Service Level Agreements (SLA)

Azure (https://azure.microsoft.com/en-ca/support/legal/sla/virtual-machines/v1_8/)

- The promise

  "For all Virtual Machines that have two or more instances deployed across two or more Availability Zones in the same Azure region, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.99% of the time.

  For all Virtual Machines that have two or more instances deployed in the same Availability Set, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.95% of the time.

  For any Single Instance Virtual Machine using premium storage for all Operating System Disks and Data Disks, we guarantee you will have Virtual Machine Connectivity of at least 99.9%."

Typical Service Level Agreements (SLA)

Azure (https://azure.microsoft.com/en-ca/support/legal/sla/virtual-machines/v1_8/)

- The penalties

  "If we do not achieve and maintain the Service Levels for each Service as described in this SLA, then you may be eligible for a credit towards a portion of your monthly service fees.

  …

  Service Credits are your sole and exclusive remedy for any performance or availability issues for any Service under the Agreement and this SLA."

Typical Service Level Agreements (SLA)

Azure (https://azure.microsoft.com/en-ca/support/legal/sla/virtual-machines/v1_8/)

- Penalty calculation

MONTHLY UPTIME PERCENTAGE	SERVICE CREDIT
< 99.99%	10%
< 99%	25%
< 95%	100%

Limitations of priority-based schedulers

fairness among equals

Service classes

Prod	Batch	Free
Priority: 2	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 50%

Pending queue

Prior.: 0	Priority: 0	Priority: 0
SLO: 50%	SLO: 50%	SLO: 50%
QoS: 0%	QoS: 0%	QoS: 0%

Server 1

Priority: 2	Priority: 1	Priority: 1	Priority: 1
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 90%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 100%

Server 2

Priority: 2	Priority: 1	Priority: 1	Priority: 1
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 100%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 100%

Limitations of priority-based schedulers

fairness among equals

Service classes

Prod	Batch	Free
Priority: 2	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 50%

Pending queue

Prior.: 0
SLO: 50%
QoS: 0%

Server 1

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 0%

Server 2

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 0%

Limitations of priority-based schedulers

fairness among equals

Service classes

Prod	Batch	Free
Priority: 2	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 50%

Pending queue

Prior.: 0
SLO: 50%
QoS: 0%

Server 1

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 70%

Server 2

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 70%

Limitations of priority-based schedulers

fairness among equals

Service classes

Prod	Batch	Free
Priority: 2	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 50%

Pending queue

Prior.: 0
SLO: 50%
QoS: 0%

Server 1

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 99%

Server 2

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 99%

Limitations of priority-based schedulers

inefficient resource usage

Service classes

Prod	Batch	Free
Priority: 2	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 50%

Pending queue

Prior.: 0	Priority: 0	Priority: 0
SLO: 50%	SLO: 50%	SLO: 50%
QoS: 49%	QoS: 49%	QoS: 49%

Server 1

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 49%

Server 2

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 100%	QoS: 49%

Limitations of priority-based schedulers

inefficient resource usage

Service classes

Prod	Batch	Free
Priority: 2	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 50%

Pending queue

Prior.: 1	Priority: 1	Priority: 1
SLO: 90%	SLO: 90%	SLO: 90%
QoS: 100%	QoS: 100%	QoS: 100%

Server 1

Priority: 2	Priority: 0	Priority: 0	Priority: 0
SLO: 100%	SLO: 50%	SLO: 50%	SLO: 50%
QoS: 100%	QoS: 0%	QoS: 0%	QoS: 50%

Server 2

Priority: 2	Priority: 1	Priority: 0	Priority: 0
SLO: 100%	SLO: 90%	SLO: 50%	SLO: 50%
QoS: 100%	QoS: 100%	QoS: 0%	QoS: 50%

Limitations of priority-based schedulers

inefficient resource usage

Service classes

Prod	Batch	Free
Priority: 2	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 50%

Pending queue

Prior.: 0	Priority: 0	Priority: 0
SLO: 50%	SLO: 50%	SLO: 50%
QoS: 52%	QoS: 51%	QoS: 51%

Server 1

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 94%	QoS: 94%	QoS: 51%

Server 2

Priority: 2	Priority: 1	Priority: 1	Priority: 0
SLO: 100%	SLO: 90%	SLO: 90%	SLO: 50%
QoS: 100%	QoS: 94%	QoS: 93%	QoS: 51%

QoS-driven scheduling
acknowledgements

- This is work done in cooperation with Ericsson Research
  - Funding
    - Centro de Inovações da Ericsson Telecomunicações S.A.
    - EMBRAPII-CEEI-UFCG
  - Team
    - Giovanni Farias, Raquel Lopes, Marcus Carvalho, Fabio Morais, João Mafra, Vinícius da Silva, Daniel Turull
- It is part of the PhD work by Giovanni Farias, developed under my supervision at UFCG

QoS-driven scheduling
rationale

- Priority-based schedulers use the priority of a service class as a proxy to the level of QoS that needs to be offered
- QoS-driven schedulers use metrics related to the actual Service Level Objectives (SLO) used to define the SLA
  - We will focus on availability, since this is by far the most common measure currently used

QoS-driven scheduling functioning

- Scheduling decisions are made considering the promised QoS (SLO) and the current QoS being delivered to each task
- Tasks with QoS above the promised one, can be preempted to make room for tasks whose QoS is below the SLO
  - The service class of a task does not need to be taken into account

Availability-driven scheduler
defining a metric to drive scheduling

- The current availability of a task t:

$$A_t = a_t / (a_t + p_t)$$

where:

a_t is the amount of time that t has had resources allocated to it since its admission, and

p_t is the amount of time that t has been in pending queue since its admission

Availability-driven scheduler

defining a metric to drive scheduling

- Which of these two tasks should be allocated, if there is space for just one of them?

Task 1	running: 20 pending: 30 SLO: 50%
$A_1 = 20 / (20 + 30) = 40\%$	

Task 2	running: 90 pending: 10 SLO: 50%
$A_2 = 90 / (90 + 10) = 90\%$	

Availability-driven scheduler
defining a metric to drive scheduling

- What about now?

Task 3	running: 6 pending: 4 SLO: 50%
$A_3 = 6 / (6 + 4) = 60\%$	

Task 4	running: 40 pending: 40 SLO: 50%
$A_4 = 60 / (60 + 40) = 60\%$	

  - Any?
  - The one admitted first?
  - The one admitted last?
  - Something else?

Availability-driven scheduler
defining a metric to drive scheduling

- What about now?

Task 3	running: 6 pending: 4 SLO: 50%
$A_3 = 6 / (6 + 4) = 60\%$	

Task 4	running: 40 pending: 40 SLO: 50%
$A_4 = 60 / (60 + 40) = 60\%$	

  - Select the task that would violate sooner its SLO, were it to be kept in the pending queue
    - This is the time to violate (TTV) metric
      - $TTV_3 = 2$, $TTV_4 = 20$

Availability-driven scheduler
defining a metric to drive scheduling

- TTV is calculated for tasks with current availability equal or greater than the SLO

$$TTV_t = a_t / SLO - (a_t + p_t)$$

Note that, since $a_t / (a_t + p_t) \geq SLO$, TTV_t is never negative

- For tasks with the availability smaller than the SLO, we compute the task recoverability, using the same formula above
  - Task recoverability is always negative
  - The smaller the value the smaller the recoverability

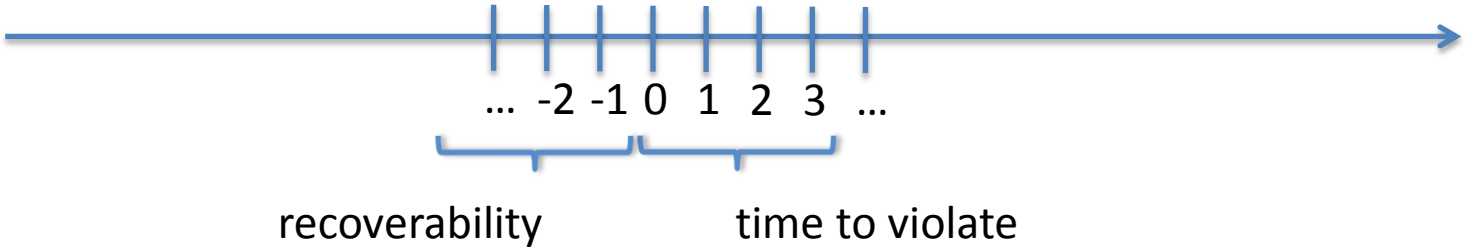
Availability-driven scheduler
defining a metric to drive scheduling

- In summary:

$$metric = a_t / SLO - (a_t + p_t)$$

when $a_t / (a_t + p_t) \geq SLO$ the metric is called time to violate (TTV)

when $a_t / (a_t + p_t) < SLO$ the metric is called recoverability



Availability-driven scheduler
main algorithm

On admission of task t
    insert t into the pending queue
    trigger scheduling
On completion of task t at server s
    free resources allocated to t in s
    trigger scheduling
On removal of server s (failure or maintenance)
    insert all tasks allocated in s into the pending queue
    trigger scheduling
On addition of server s
    trigger scheduling
On watchdog fired
    trigger scheduling
On scheduling triggered
    sort pending queue based on the tasks' TTV
    for all tasks t in the pending queue
        perform feasibility check for t
        if there is at least one feasible host
            perform ranking of feasible servers
            allocate t to the first ranked server
    reset watchdog

Availability-driven scheduler

main algorithm

On admission of task t
    insert t into the pending queue
    trigger scheduling

```
On completion of task t at server s
        free resources allocated to t in s
        trigger scheduling
On removal of server s (failure or maintenance)
        insert all tasks allocated in s into the pending queue
        trigger scheduling
On addition of server s
        trigger scheduling
On watchdog fired
        trigger scheduling
On scheduling triggered
        sort pending queue based on the tasks' TTV
        for all tasks t in the pending queue
                perform feasibility check for t
                if there is at least one feasible host
                        perform ranking of feasible servers
                        allocate t to the first ranked server
        reset watchdog
```

Availability-driven scheduler
main algorithm

On admission of task t
    insert t into the pending queue
    trigger scheduling

On completion of task t at server s
    free resources allocated to t in s
    trigger scheduling

On removal of server s (failure or maintenance)
    insert all tasks allocated in s into the pending queue
    trigger scheduling
On addition of server s
    trigger scheduling
On watchdog fired
    trigger scheduling
On scheduling triggered
    sort pending queue based on the tasks' TTV
    for all tasks t in the pending queue
        perform feasibility check for t
        if there is at least one feasible host
            perform ranking of feasible servers
            allocate t to the first ranked server
    reset watchdog

Availability-driven scheduler

main algorithm

On admission of task t
    insert t into the pending queue
    trigger scheduling
On completion of task t at server s
    free resources allocated to t in s
    trigger scheduling

On removal of server s (failure or maintenance)
    insert all tasks allocated in s into the pending queue
    trigger scheduling

On addition of server s
    trigger scheduling
On watchdog fired
    trigger scheduling
On scheduling triggered
    sort pending queue based on the tasks' TTV
    for all tasks t in the pending queue
        perform feasibility check for t
        if there is at least one feasible host
            perform ranking of feasible servers
            allocate t to the first ranked server
    reset watchdog

Availability-driven scheduler

main algorithm

On admission of task t
    insert t into the pending queue
    trigger scheduling
On completion of task t at server s
    free resources allocated to t in s
    trigger scheduling
On removal of server s (failure or maintenance)
    insert all tasks allocated in s into the pending queue
    trigger scheduling

On addition of server s
    trigger scheduling

On watchdog fired
    trigger scheduling
On scheduling triggered
    sort pending queue based on the tasks' TTV
    for all tasks t in the pending queue
        perform feasibility check for t
        if there is at least one feasible host
            perform ranking of feasible servers
            allocate t to the first ranked server
    reset watchdog

Availability-driven scheduler

main algorithm

On admission of task t
    insert t into the pending queue
    trigger scheduling
On completion of task t at server s
    free resources allocated to t in s
    trigger scheduling
On removal of server s (failure or maintenance)
    insert all tasks allocated in s into the pending queue
    trigger scheduling
On addition of server s
    trigger scheduling

On watchdog fired
    trigger scheduling

On scheduling triggered
    sort pending queue based on the tasks' TTV
    for all tasks t in the pending queue
        perform feasibility check for t
        if there is at least one feasible host
            perform ranking of feasible servers
            allocate t to the first ranked server
    reset watchdog

Availability-driven scheduler

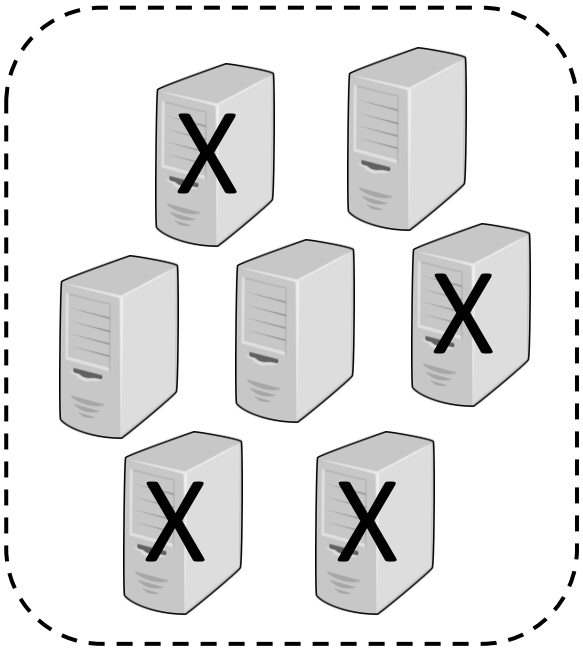
main algorithm

```
On admission of task t
        insert t into the pending queue
        trigger scheduling
On completion of task t at server s
        free resources allocated to t in s
        trigger scheduling
On removal of server s (failure or maintenance)
        insert all tasks allocated in s into the pending queue
        trigger scheduling
On addition of server s
        trigger scheduling
On watchdog fired
        trigger scheduling
```

```
On scheduling triggered
    sort pending queue based on the tasks' TTV/recoverability
    for all tasks t in the pending queue
        perform feasibility check for t
        if there is at least one feasible host
            perform ranking of feasible servers
            allocate t to the first ranked server
    reset watchdog
```

Availability-driven scheduler
feasibility checking

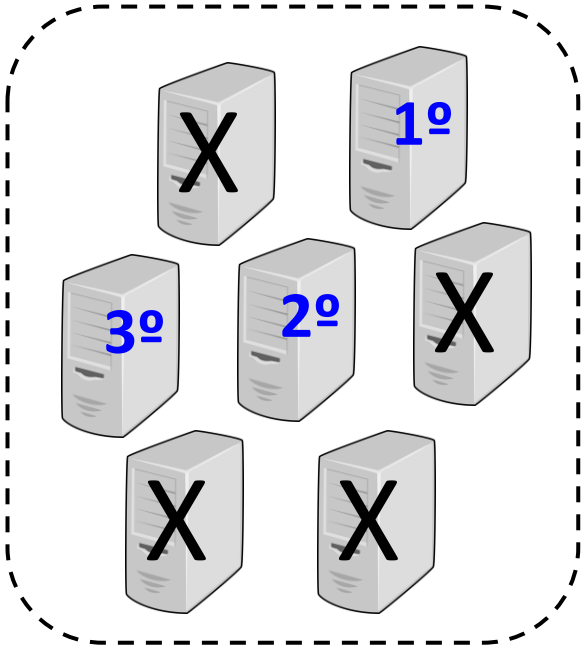
- Randomly selects a server and checks whether it satisfies the requirements for the task to be allocated
  - A task with a TTV larger than the one being allocated can be preempted
  - If the QoS of the task being allocated is already below the SLO, then other tasks with recoverability larger may also be preempted
- (Like Borg) Stops after a number of feasible servers have been found
  - This may require checking all hosts
  - During contention, there might be no feasible server



Availability-driven scheduler

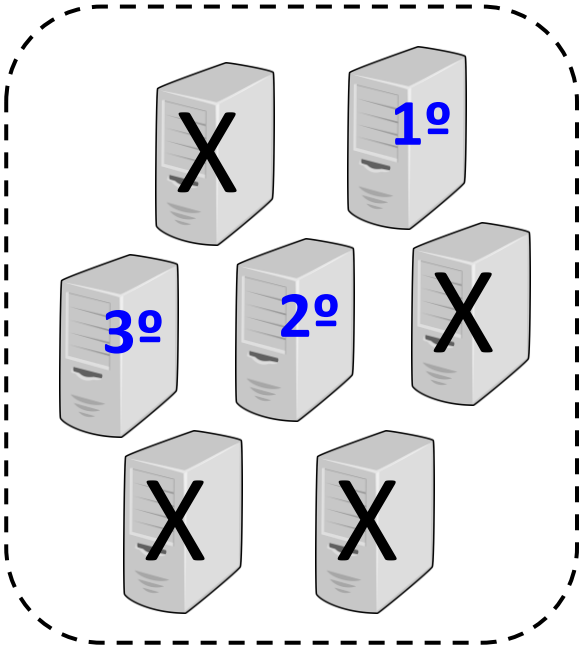
ranking

- Let F be the set of feasible servers computed in the previous step
- If there are servers in F that can accommodate t without preempting other tasks, then rank them according to some criterea
  - Ex. Borg's approach to minimize resource stranding
- Chose the server with the largest score



Availability-driven scheduler
ranking

- Otherwise, compute for each server s the sequence:
  - $(ar_1, ar_2, ..., ar_n, at)$, where:
    - ar_i is the aggregate recoverability for all tasks of service class i that need to be preempted at s to accommodate t, and
    - at is the aggregate TTV for all tasks that need to be preempted at s to accommodate t
  - Service classes are ordered in relative importance
- Rank servers based on their sequence score, from the largest to the smallest
- Chose the server with the largest score



Availability-driven scheduler evaluation

- Goal
  - Compare the performance of priority-based and QoS-driven schedulers
    - Borg's scheduler
    - Availability-driven scheduler
- Strategy
  - Built simulation models for both schedulers
  - Developed a proof-of-concept system based on Kubernetes to validate the simulators with controlled synthetic workloads
  - Performed simulation experiments fed with data from Google's trace

Availability-driven scheduler evaluation

- Materials and methods
  - Simulators developed in Erlang on top of the Sim-diasca simulation framework
  - Kubernetes scheduler developed in Go
  - Extracted 10 subsets of Google's trace to define different workloads
    - Grouped users based on the requests they sent to the system using k-means
    - Randomly selected 10% of the users in each of the identified groups
    - Considered all requests submitted to the system by these users

Availability-driven scheduler evaluation

- Materials and methods
  - Considered three classes of services
    - Prod, with an SLO of 100%
    - Batch, with an SLO of 90%
    - Free, with an SLO of 50%
  - Three sizes of infrastructure
    - N
    - 0.9N
    - 0.8N

Availability-driven scheduler evaluation

- Materials and methods
  - Generating infrastructures with different sizes for each workload
    - Let N be the aggregate capacity needed to serve a workload without preemptions, were the infrastructure be comprised by a single server
      - We compute this by simulating the priority-based scheduler with an infrastructure that has a single server with infinite capacity
    - For each workload, we randomly select servers from Google's trace, until the aggregate capacity reaches N
    - Then, we remove servers from the previously generated infrastructure until the aggregate capacity reaches 0.9N
    - We follow the same procedure to generate the infrastructure of size 0.8N

Availability-driven scheduler evaluation

- Materials and methods

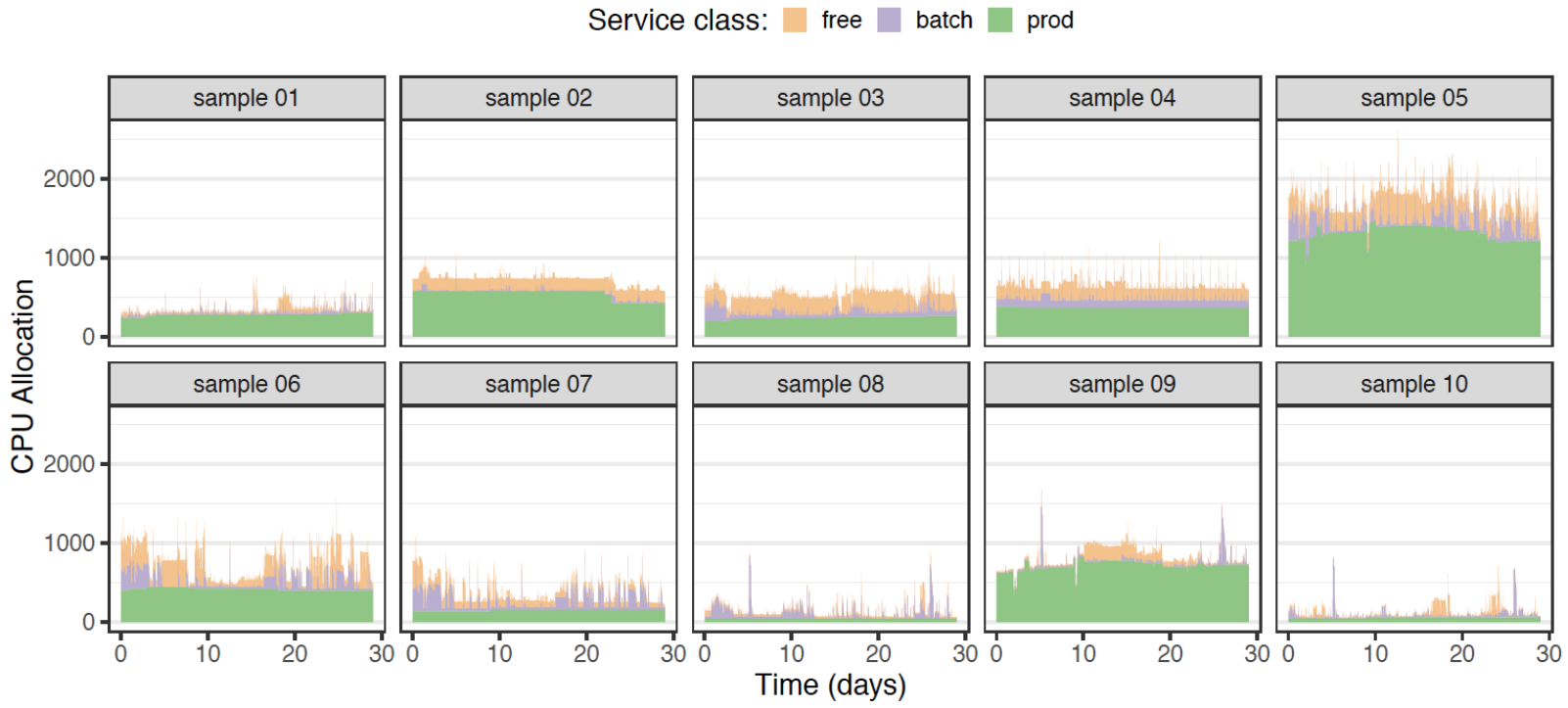


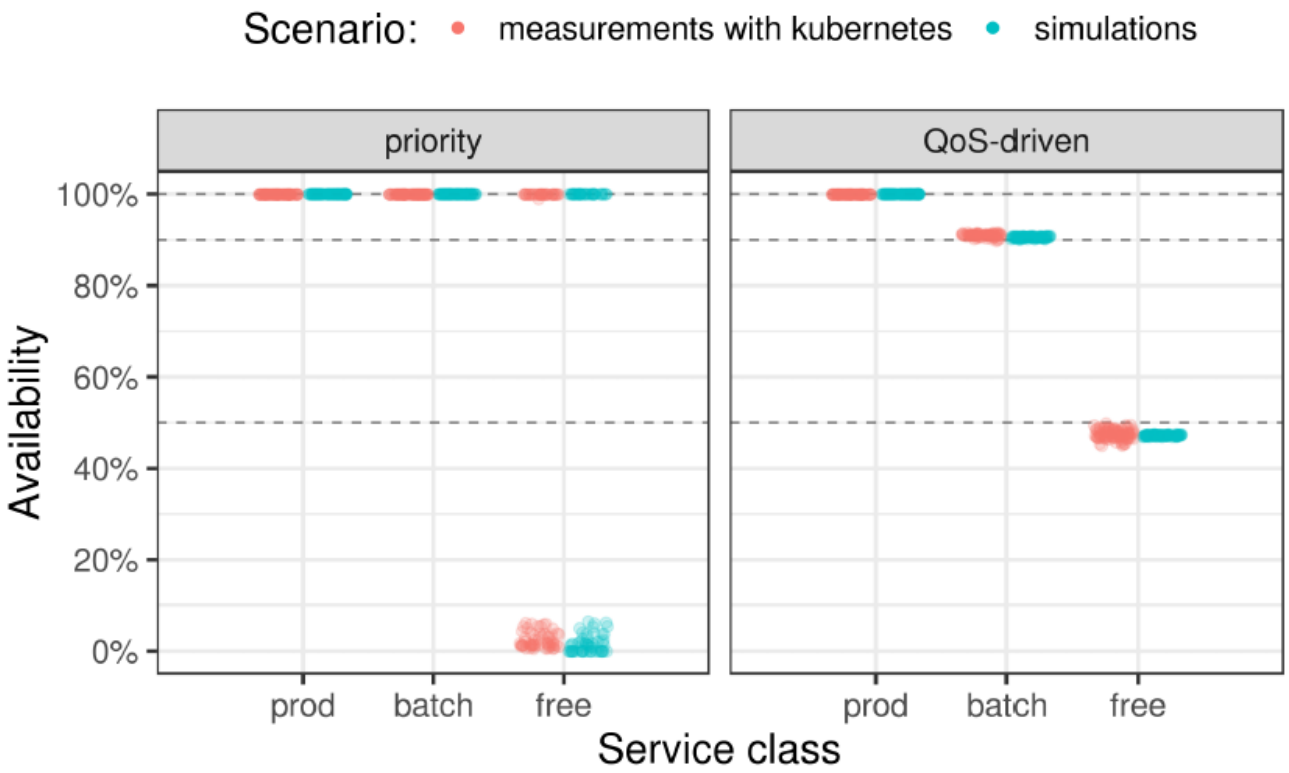
Fig. 1. CPU usage over 1-minute time intervals in the ten workload samples generated.

Availability-driven scheduler evaluation

- Materials and methods
  - Metrics per service class
    - Average QoS (availability)
    - Average QoS deficit
      - SLO minus the QoS delivered, when QoS delivered is smaller than the SLO
    - SLO fulfilment
      - Percentage of requests whose QoS was at or above the SLO
    - Gini coefficient
      - How fair resources are shared
      - Varies from 0 to 1
        - » The lowest, the better

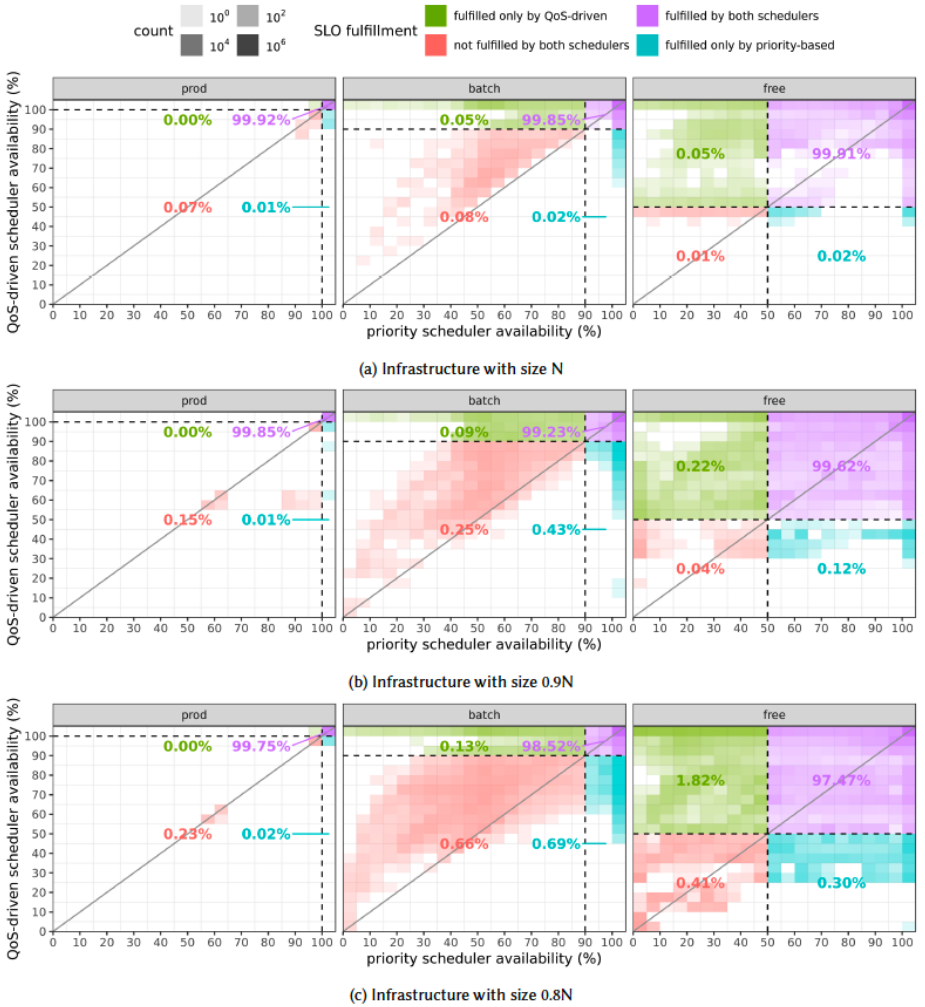
Availability-driven scheduler evaluation

- Simulator validation



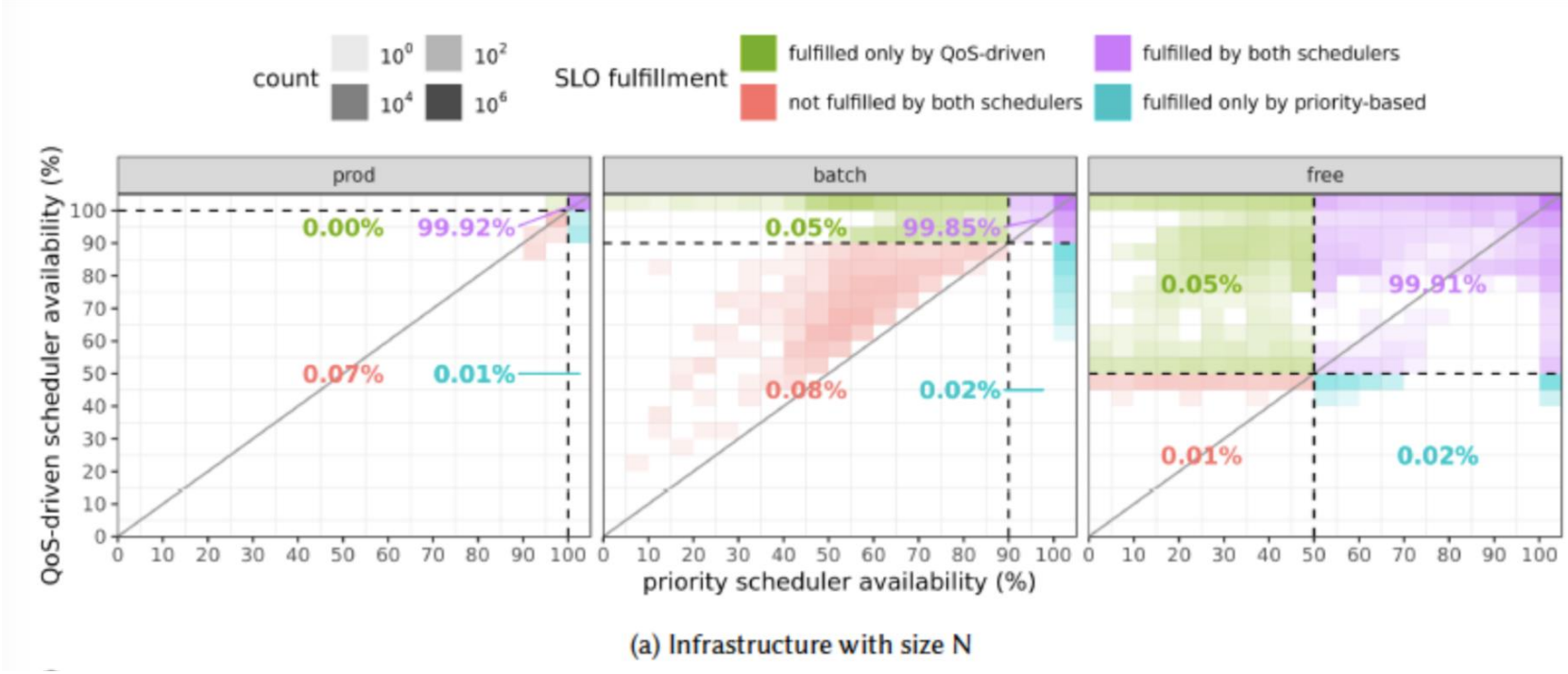
Availability-driven scheduler evaluation

- Simulation results
  - QoS



Availability-driven scheduler evaluation

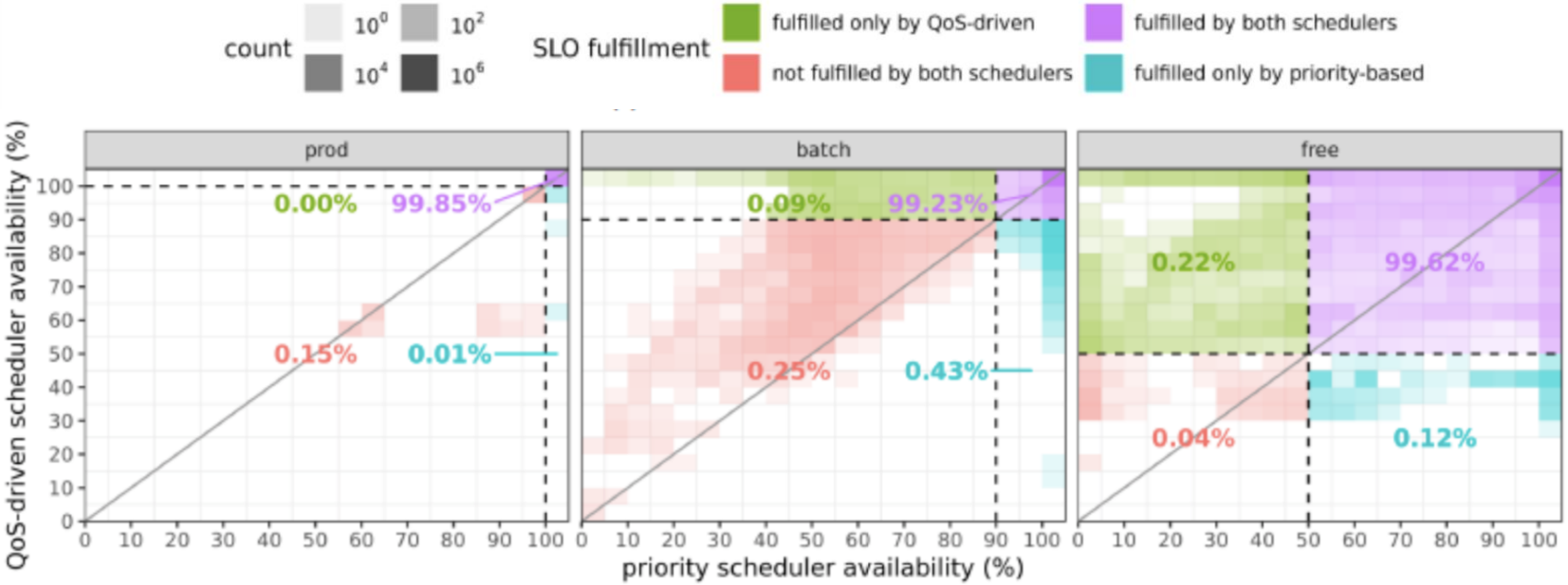
- Simulation results
  - QoS



(a) Infrastructure with size N

Availability-driven scheduler evaluation

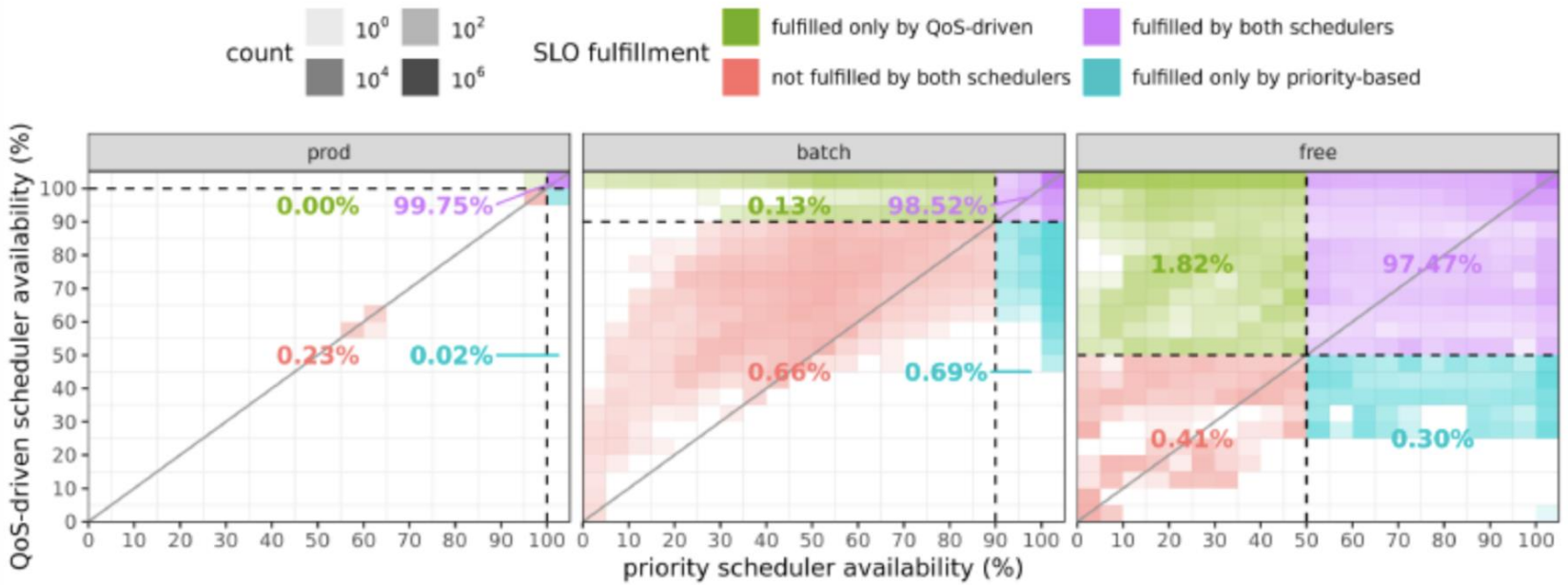
- Simulation results
  - QoS



(b) Infrastructure with size 0.9N

Availability-driven scheduler evaluation

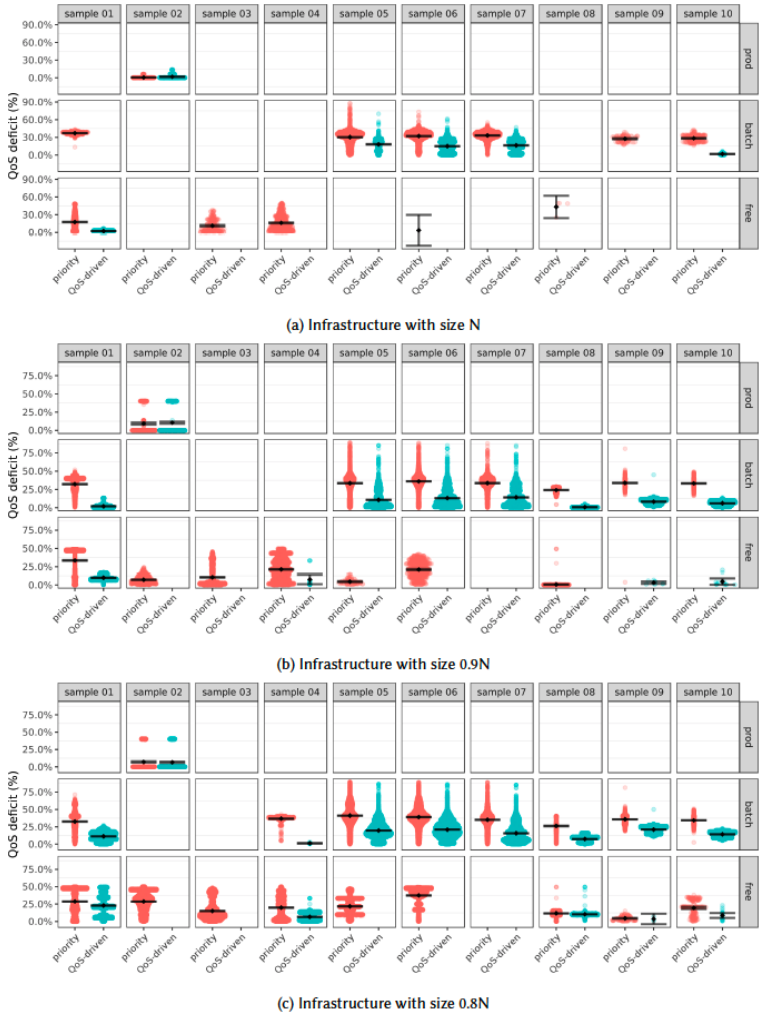
- Simulation results
  - QoS



(c) Infrastructure with size 0.8N

Availability-driven scheduler evaluation

- Simulation results
  - QoS deficit



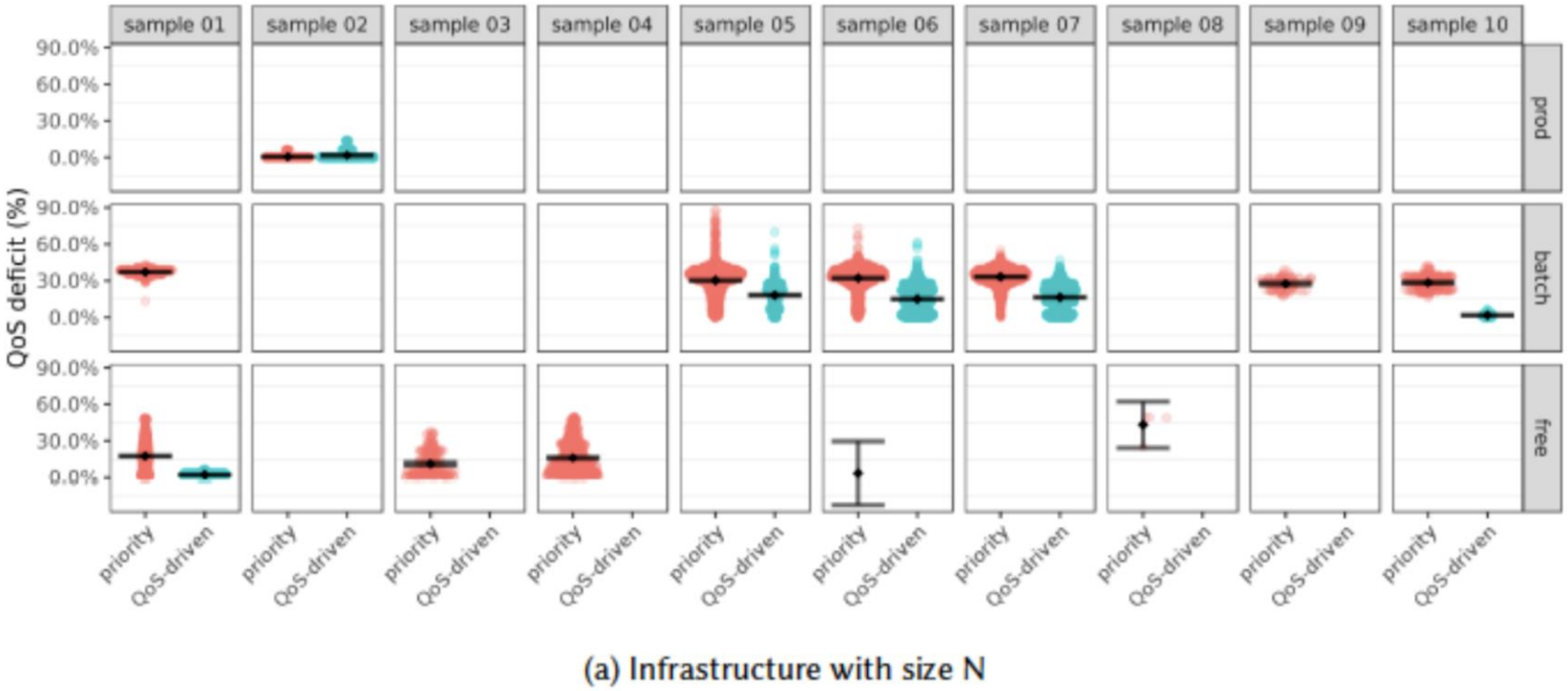
(a) Infrastructure with size N

(b) Infrastructure with size 0.9N

(c) Infrastructure with size 0.8N

Availability-driven scheduler evaluation

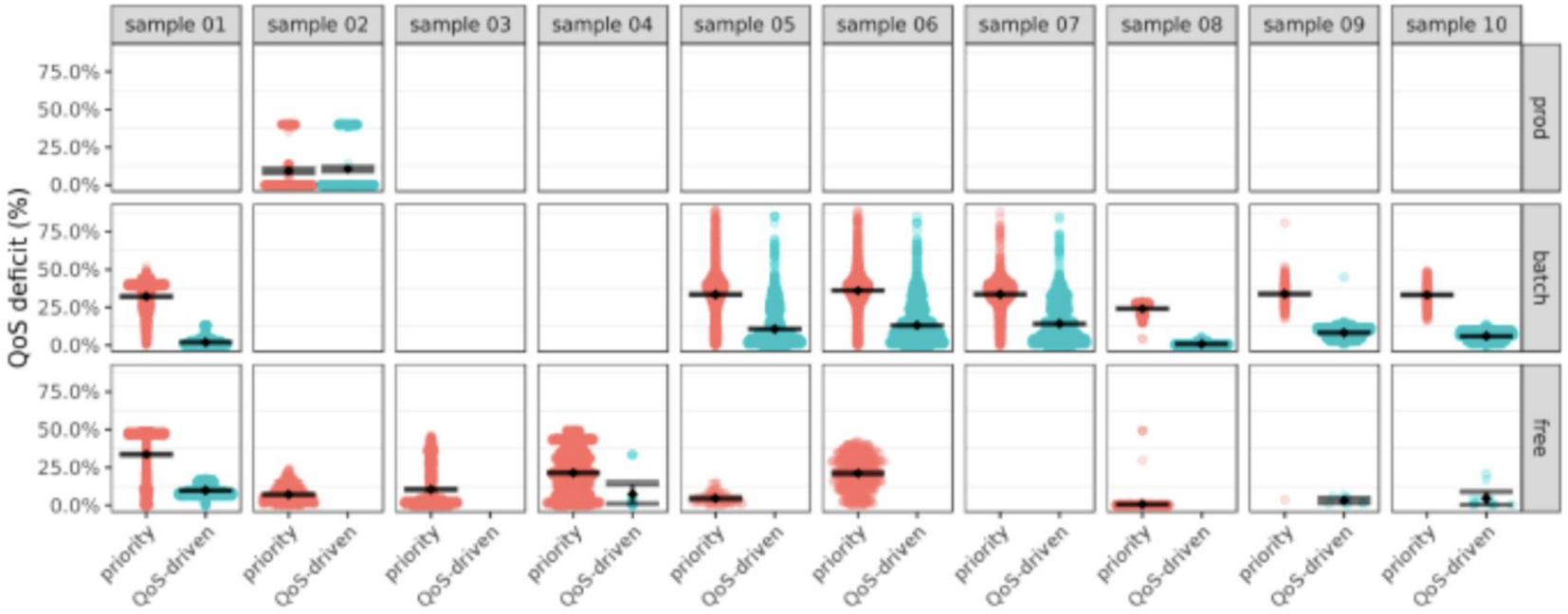
- Simulation results
  - QoS deficit



(a) Infrastructure with size N

Availability-driven scheduler evaluation

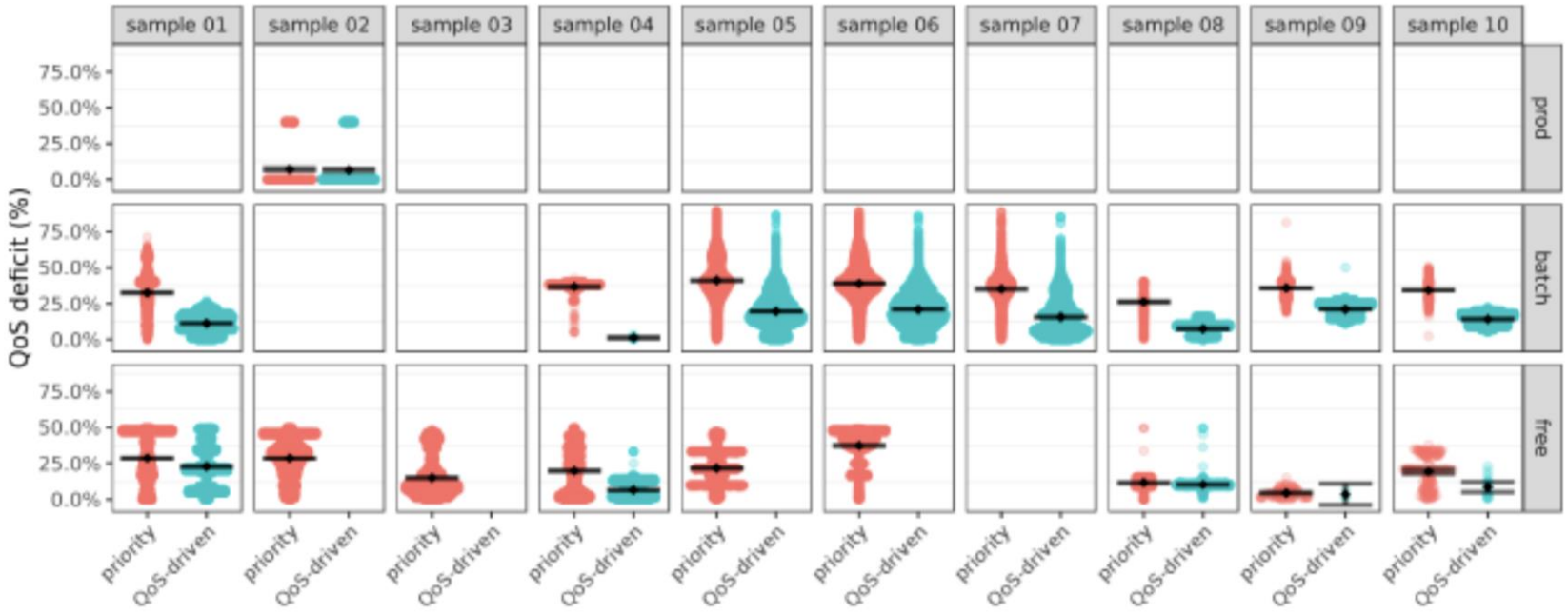
- Simulation results
  - QoS deficit



(b) Infrastructure with size 0.9N

Availability-driven scheduler evaluation

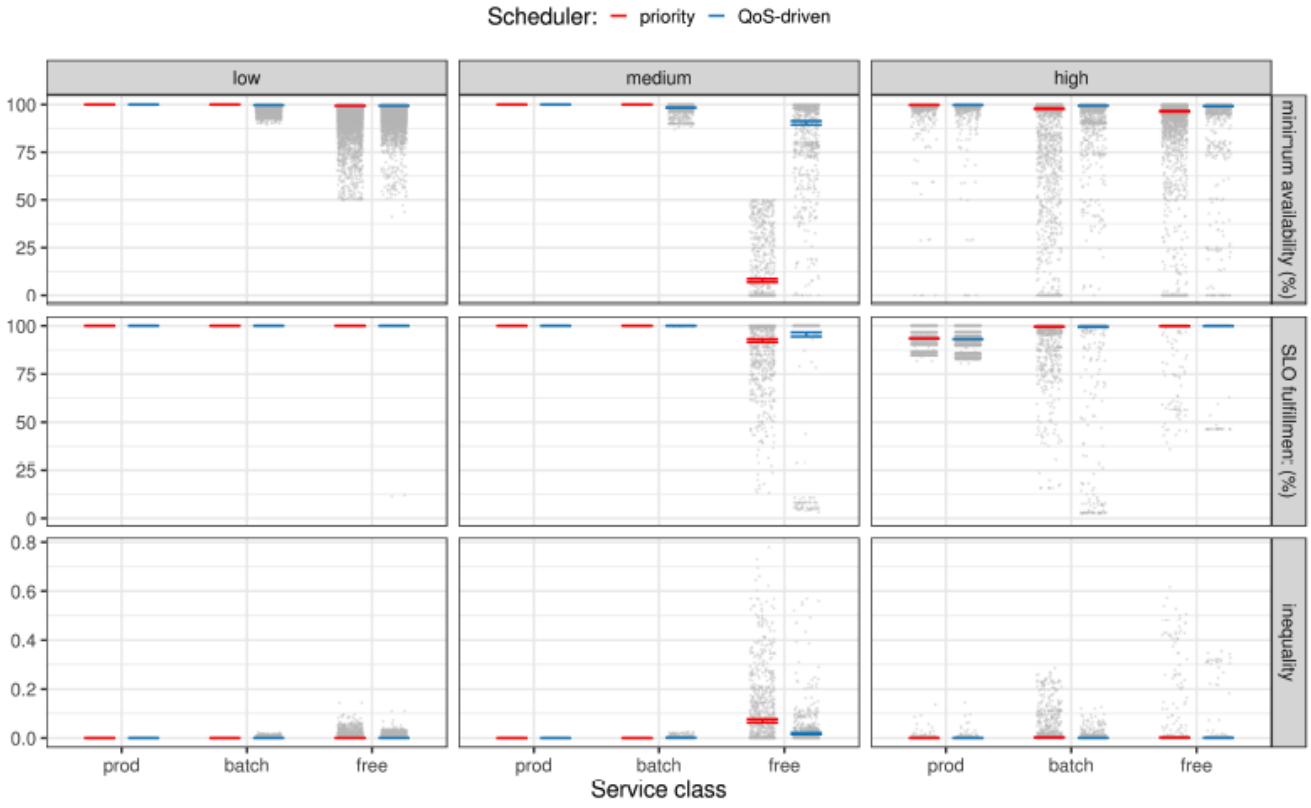
- Simulation results
  - QoS deficit



(c) Infrastructure with size 0.8N

Availability-driven scheduler evaluation

- Simulation results
  - Fairness



Take away messages

- Current scheduling practices are unfit to provide better SLA to customers
- QoS-driven scheduling provides an alternative way that might be a better fit to this purpose
- However, there are several open issues that still need further understanding
  - How to consider QoS dimensions other than availability?
  - How to consider multiple QoS dimensions at the same time?
  - How to do all this in a scalable way?