BurScale: Using Burstable Instances for Cost-Effective Autoscaling in the Public Cloud

Ata Fatahi, Timothy Zhu, Bhuvan Urgaonkar

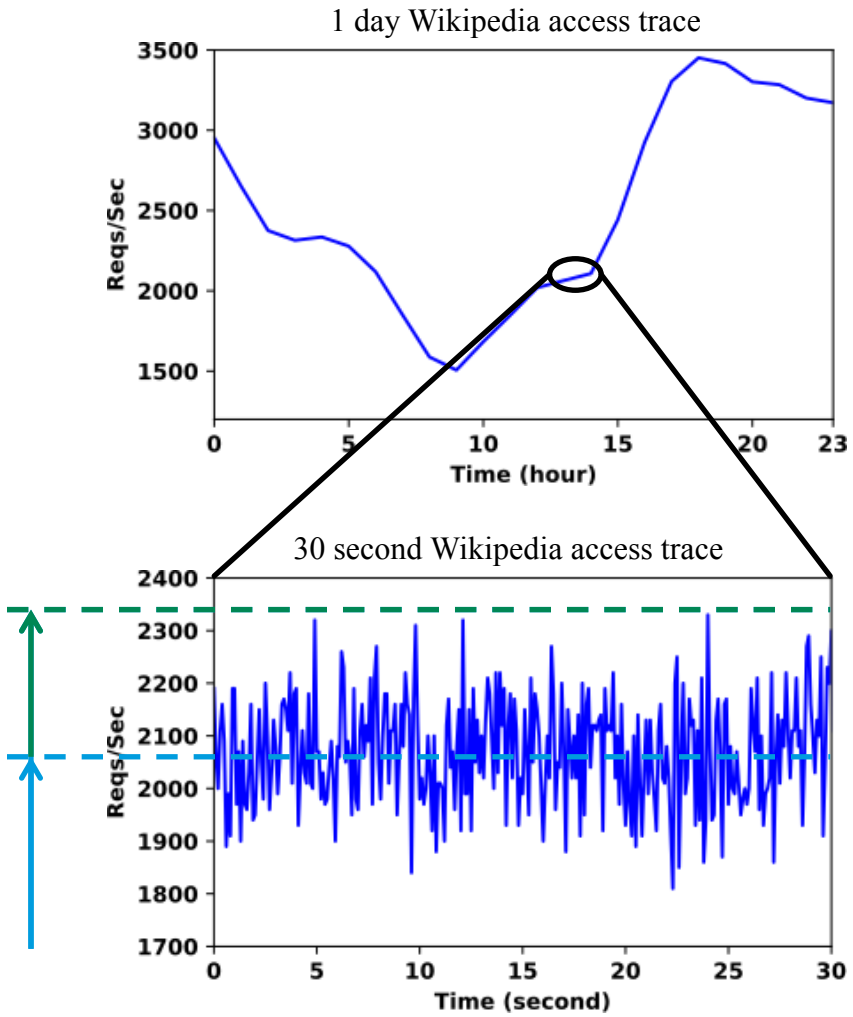
PennState

Problem and Motivation

- Context:
  Autoscaling in the cloud

- Problem:
  Uses expensive regular instances

- Solution:
  Use cheaper burstable instances



Load Variability

Short-term Burstiness

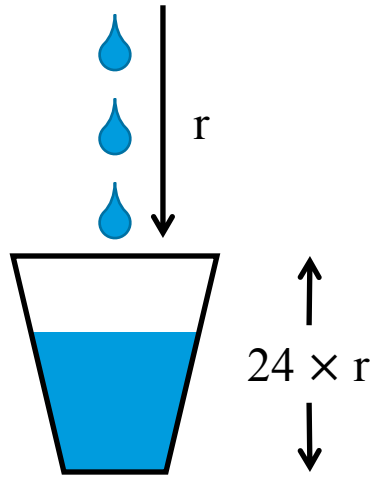
SALE
UP TO
95%
OFF

Goal: Cost-effective autoscaling using burstable instances

PennState

Burstable Instances

- CPU capacity rate-limited by a token (credit) bucket mechanism
  - Credits accrue at baseline rate up to max bucket size (24x baseline rate)
  - 1 credit = 100% CPU utilization for 1 min
    = 50% CPU utilization for 2 min
- Example: AWS t3.small accrues 24 credits/hour
  = 0.4 credits/min
  = 40% baseline CPU utilization

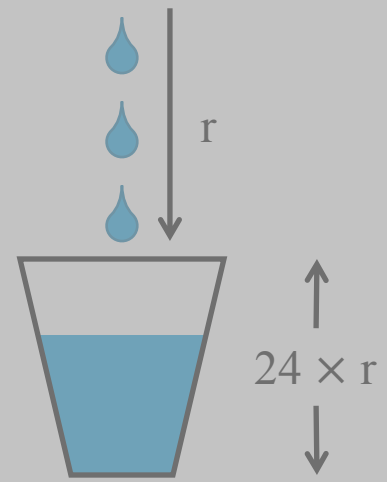


Burstable instance = "Fractional" instance with burst capability

PennState

Burstable Instances

- CPU capacity rate-limited by a token (credit) bucket mechanism
  - Credits accrue at baseline rate up to max bucket size (24x baseline rate)
  - 1 credit = 100% CPU utilization for 1 min
    = 50% CPU utilization for 2 min
- Example: AWS t3.small accrues 24 credits/hour
  = 0.4 credits/min
  = 40% baseline CPU utilization



Pros

- **Cheaper (up to 95%)**
- **Ability to burst**

Cons

- **Performance is rate limited**
- **More expensive than regular for performance**

Burstable instance = "Fractional" instance with burst capability

PennState

How to Effectively Use Burstable Instances?

1. How many burstable/regular instances to provision?

2. How to avoid running out of credits?

3. How to handle flash crowds?

Resource Provisioning

- Scaling policy
  Determines # of instances (k)

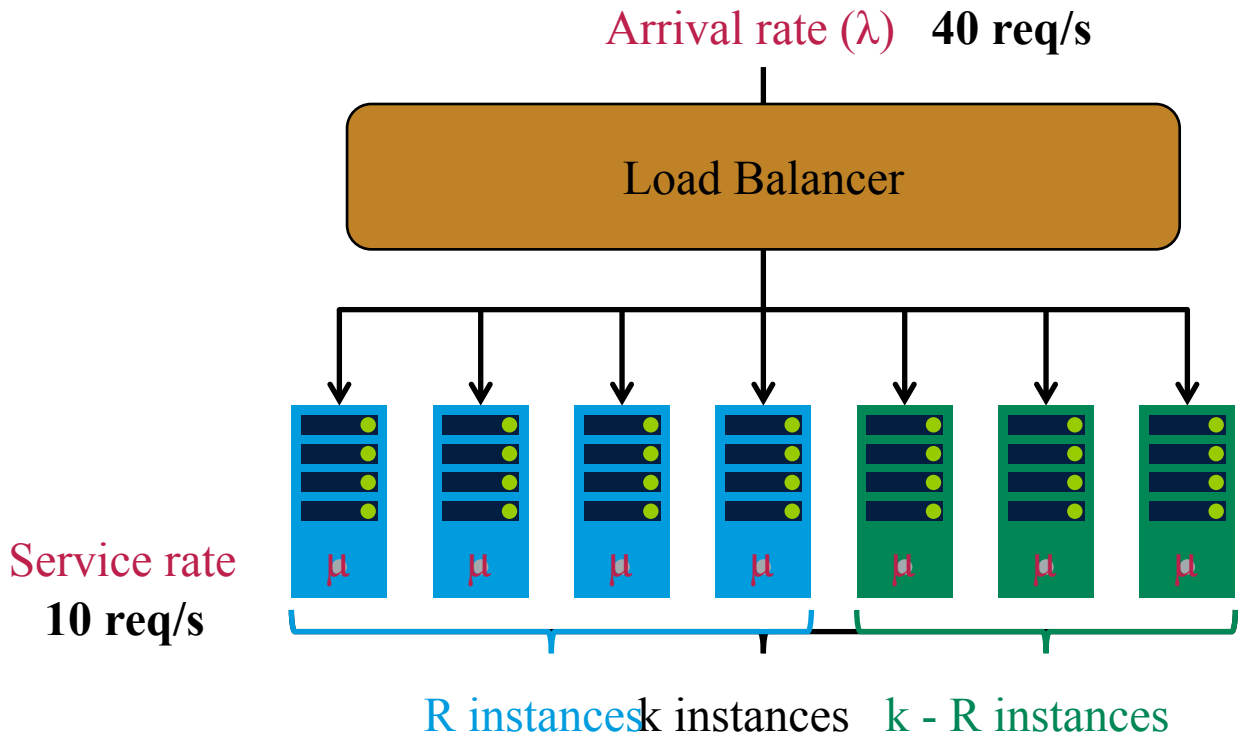
- What is the minimum # instances?

  $R = \lambda / \mu$

- $k > R$ for latency SLOs

- Square Root Staffing Rule Scaling Policy:

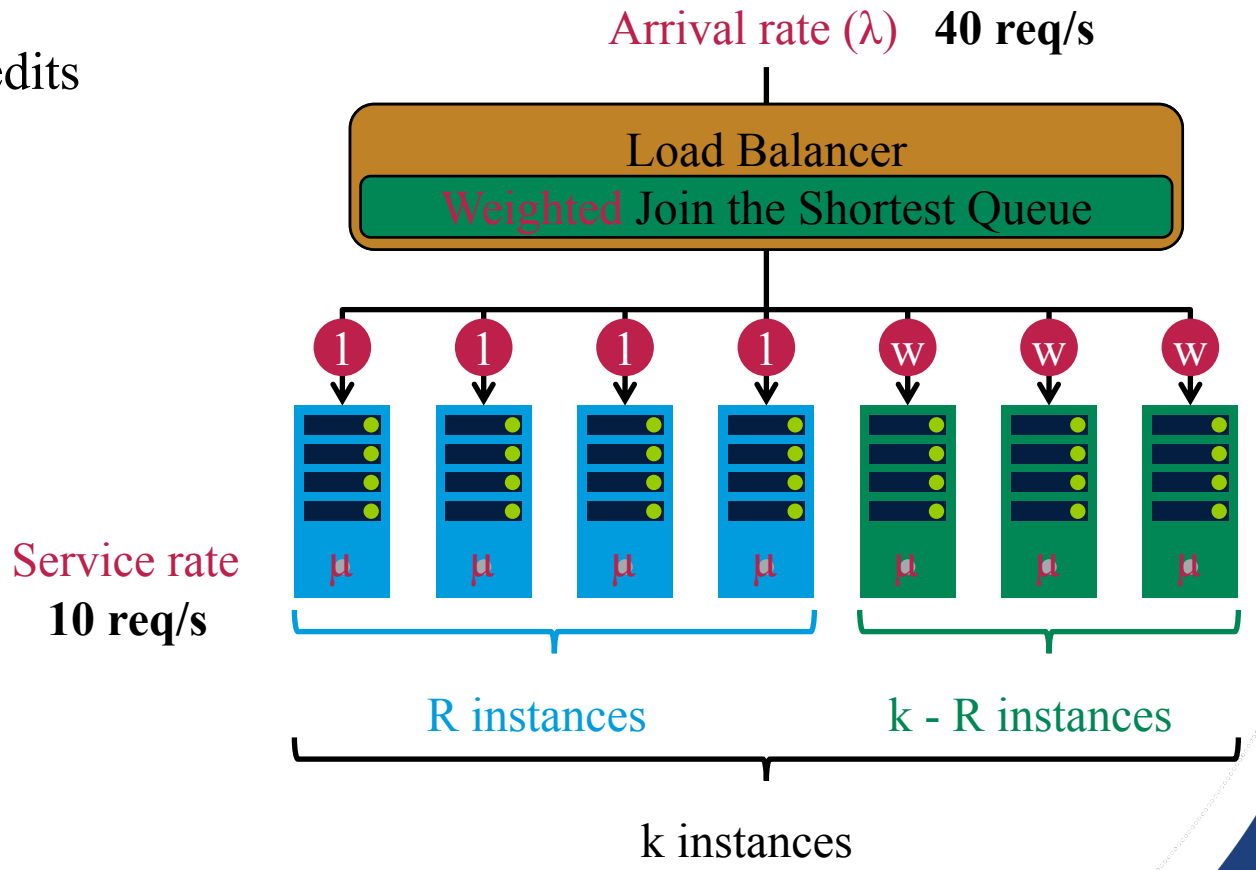
  $k = R + c\sqrt{R}$



Idea: Use burstable instances for standby variable capacity

How to Avoid Running Out of Credits?

- Problem:
  Burstable instances overused → run out of credits
- Solution:
  Unbalance the load
- How to set weight w?

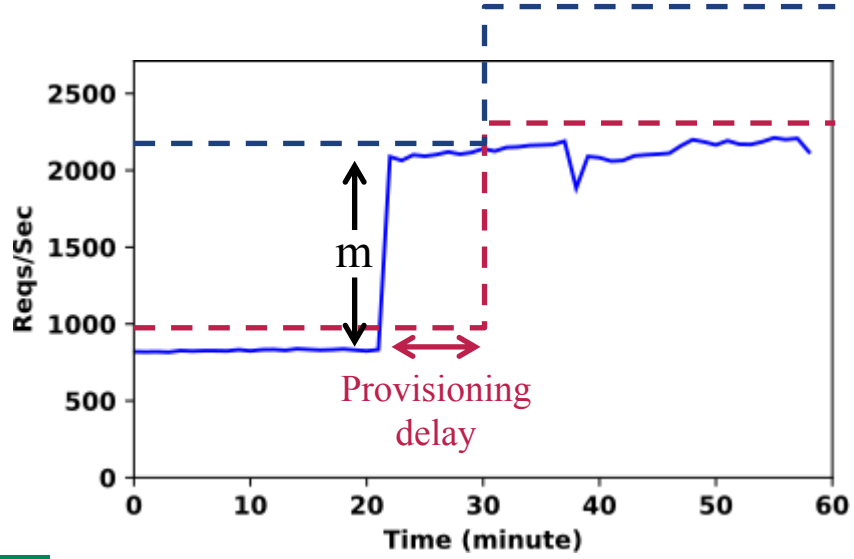


Solution: Monitor credits & Dynamically adjust weight to earn credits

PennState

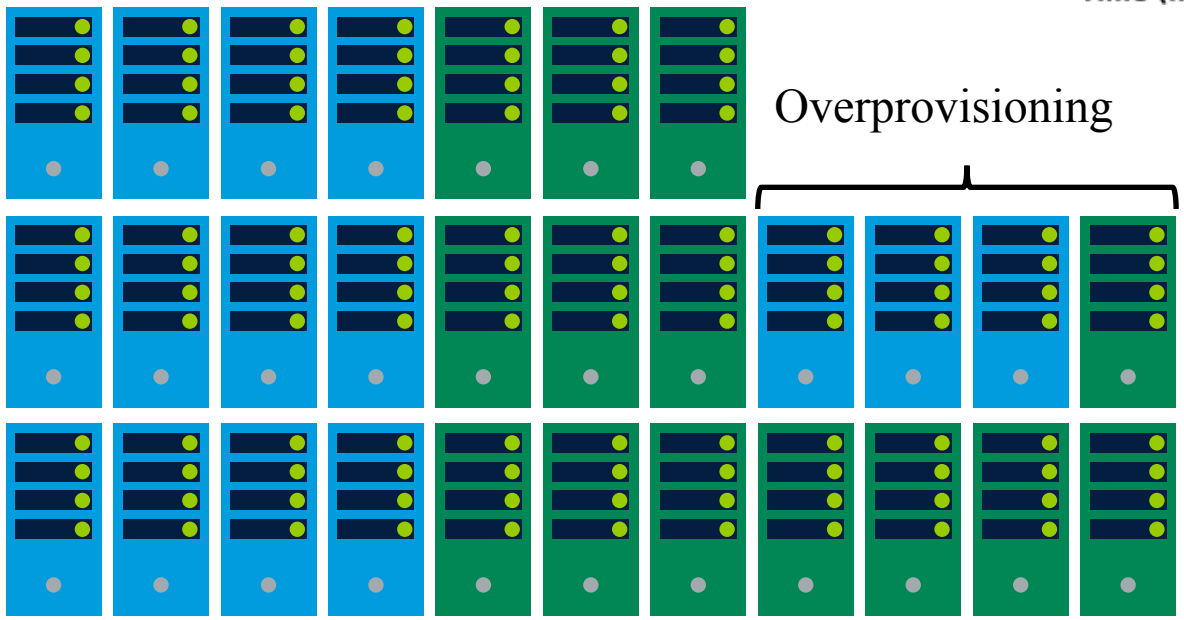
Flash Crowds

- Flash crowds are unpredictable sudden load increases



- Challenge:
  Delay in acquiring and warming up new resources

- Solution:
  Overprovision capacity (e.g., Netflix Project Nimble)



Normal Provisioning (λ, R)

Flash Crowd Provisioning ($m\lambda$, mR)

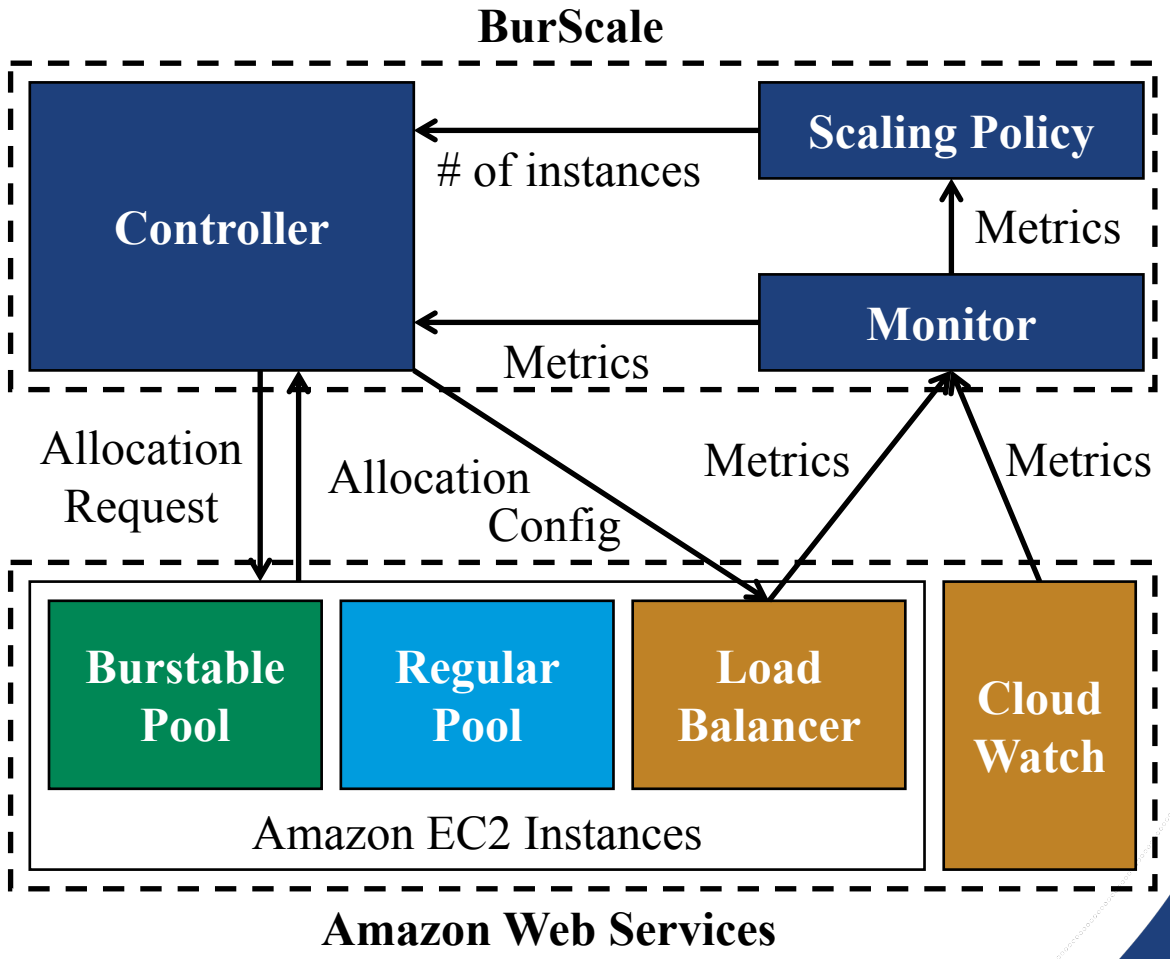
BurScale ($m\lambda$, R)

Idea: Use burstable instances for standby capacity

PennState

BurScale Design and Implementation

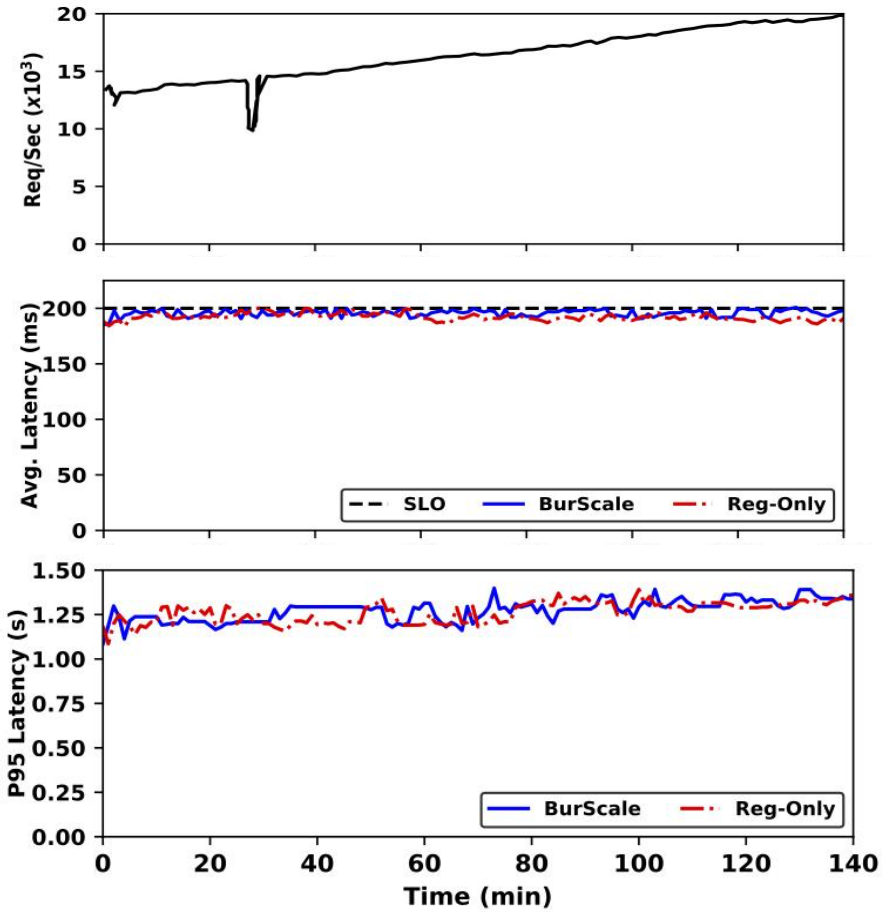
- Monitor:
  - Collects system stats
- Scaling Policy:
  - Determines cluster size
- Controller:
  - Determines # burstable/regular instances
  - Allocates/deallocates instances
  - Detects flash crowds
  - Adjusts load balancer weights

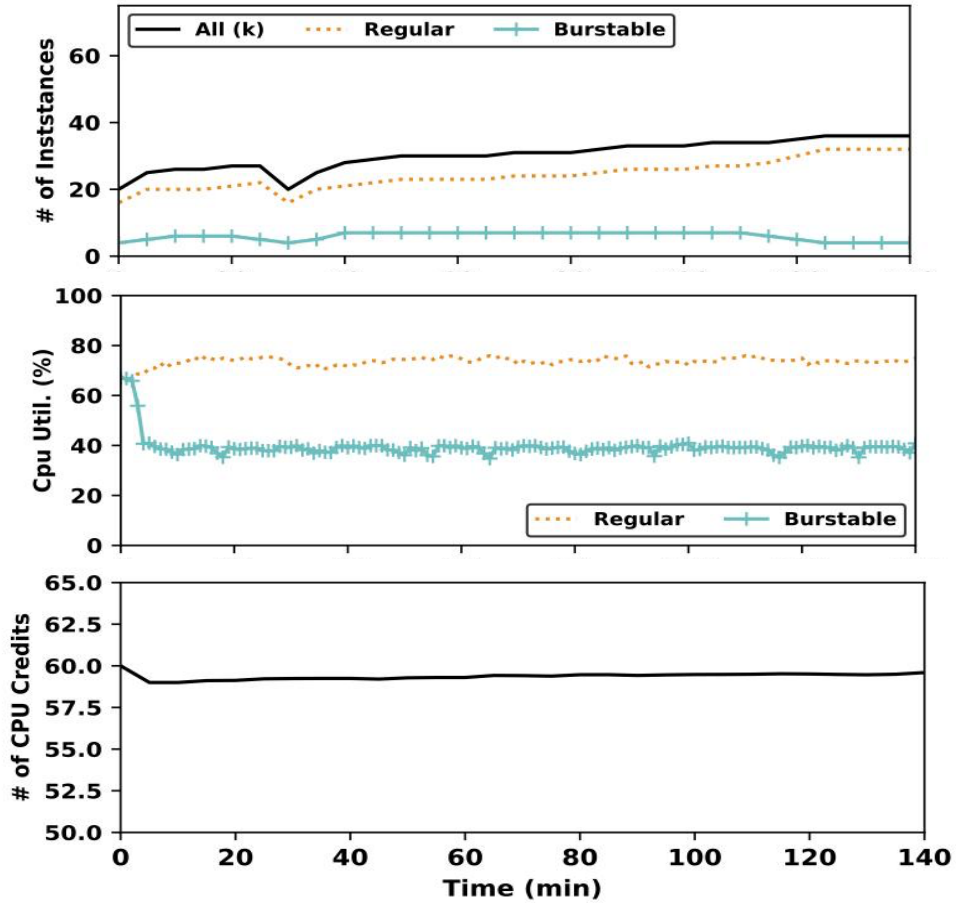


Evaluation

- Workload: WikiMedia application using Wikipedia access traces
- Regular instances: m5.large, 2 vCPUs, $0.096 / hr
- Burstable instances: t3.small, 2 vCPUs, $0.0208 / hr
- Moderate cluster size ranging from 20 to 70 instances
- Comparisons
  - Reg-Only: Cluster of only regular instances
  - BurScale: Combines burstable and regular instances

Handling Transient Queueing

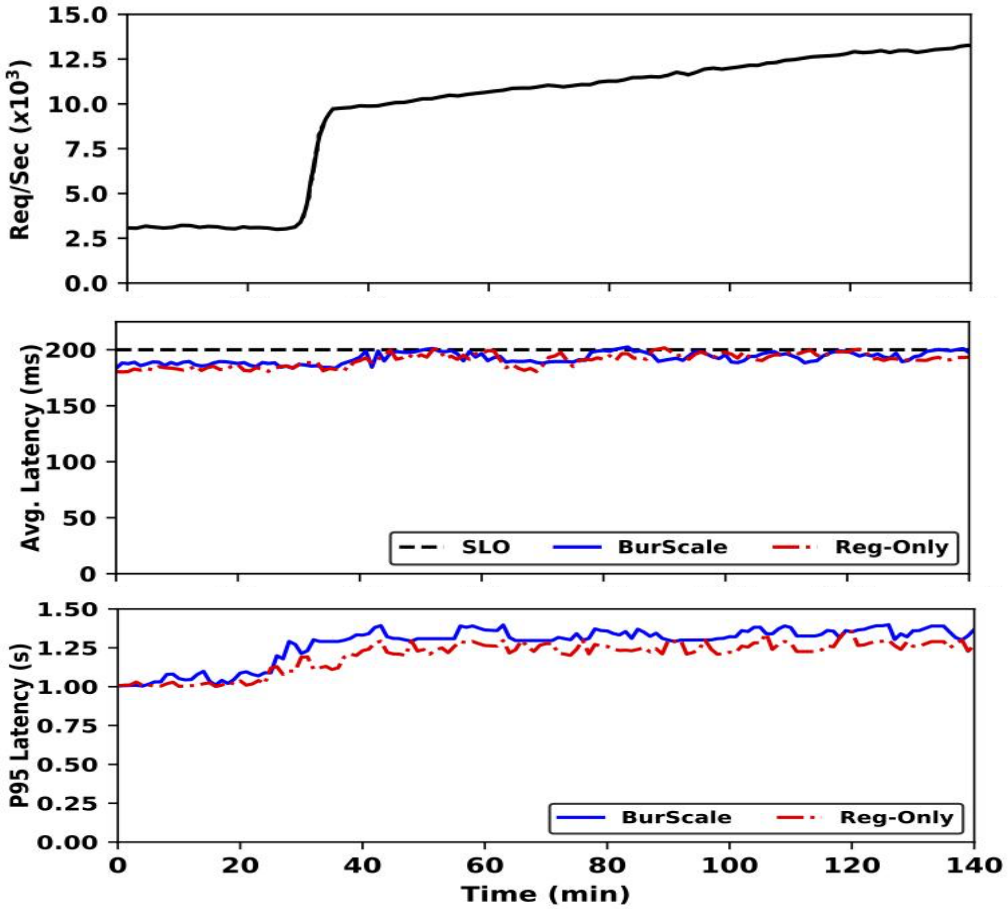


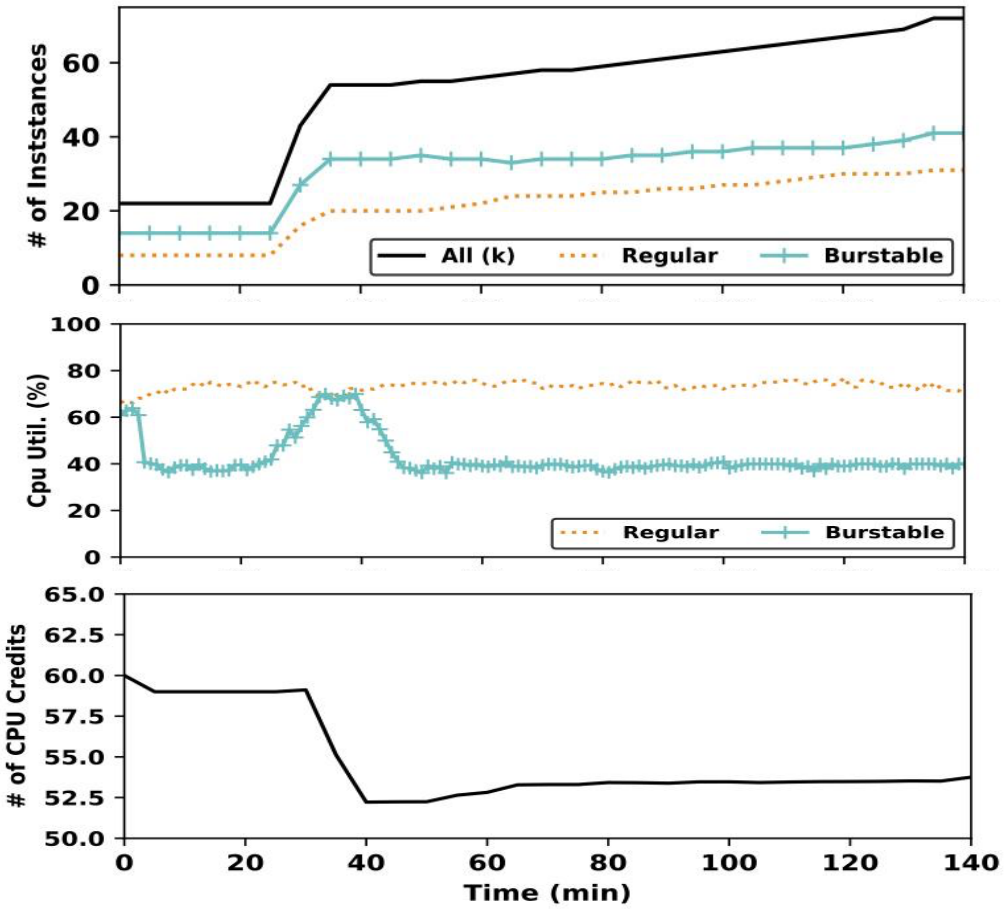


BurScale saves 16.8% in costs

PennState

Handling Flash Crowds



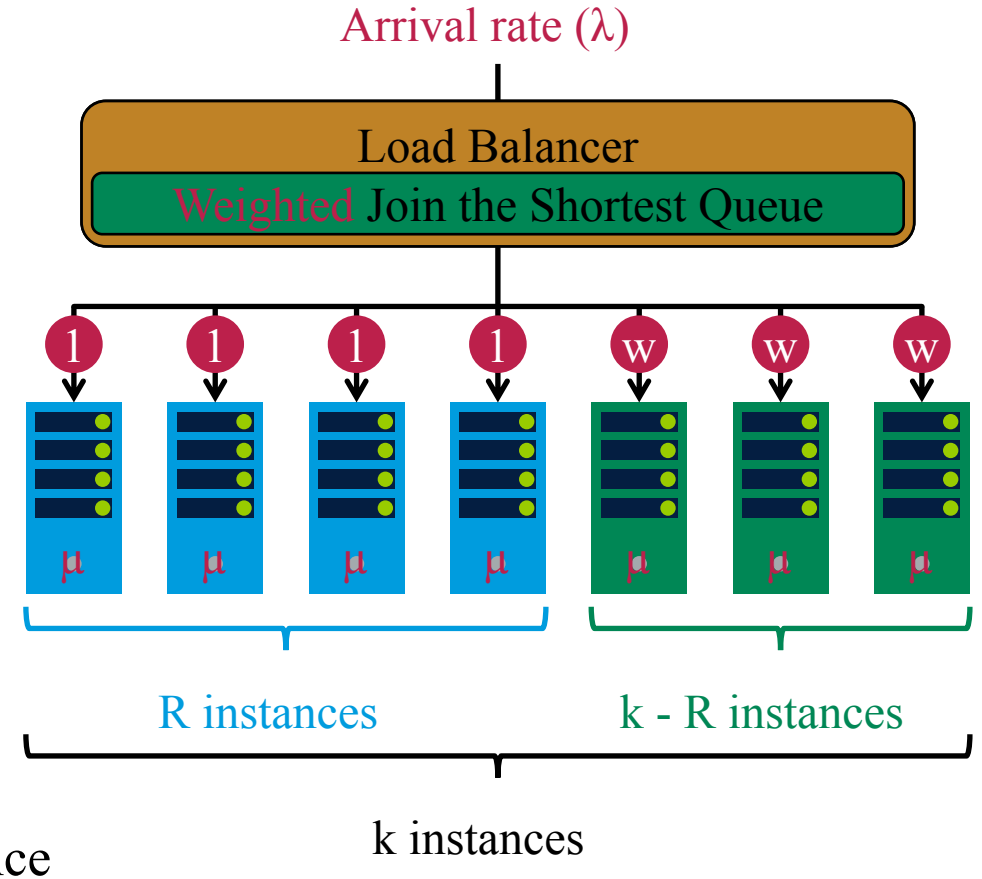


BurScale saves 46.3% in costs

PennState

Conclusion

- Goal: Cost-effective autoscaling using burstable instances
- Challenge: avoid running out of CPU credits
- Solution: BurScale
  - Selects appropriate number of burstable instances
  - Dynamically adjusts load balancer weights
- Results: BurScale saves cost while maintaining performance
  - Evaluated under web applications, flash crowds, and stateful caches



BurScale is open-sourced at: https://github.com/psu-cloud/BurScale

PennState