Load balancing and auto-scaling for large-scale parallel-server systems

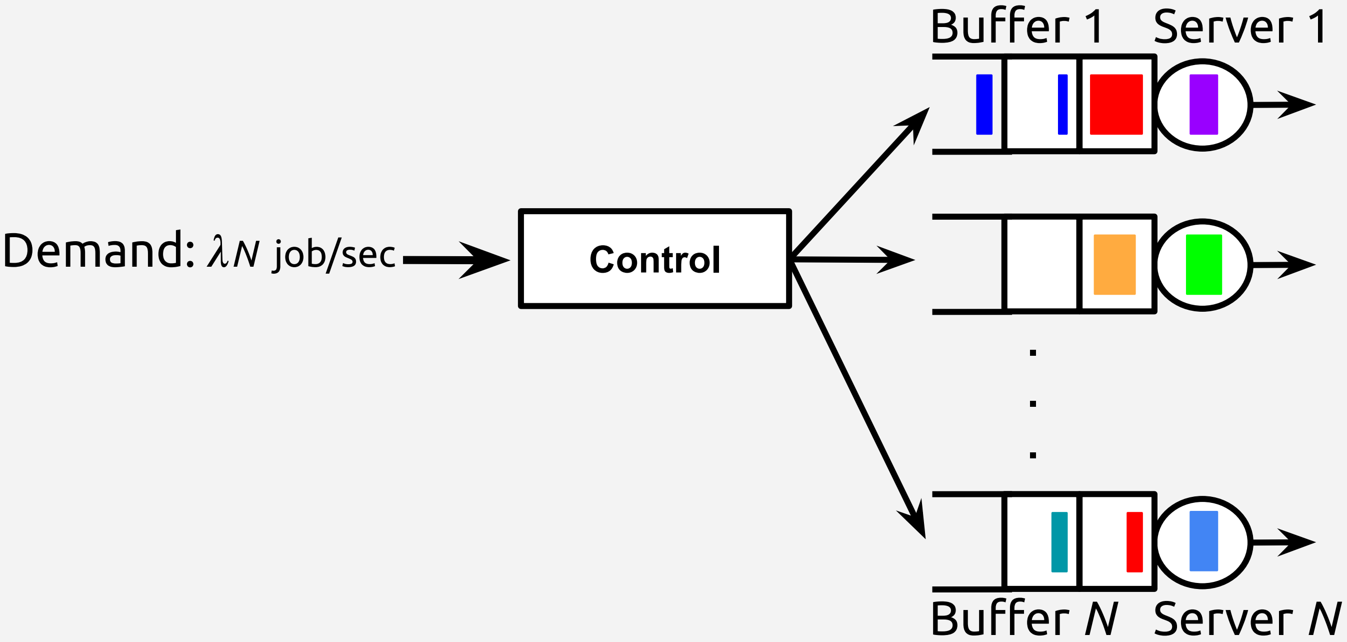
Jonatha ANSELMI, Inria, Polaris team

July 5, 2022

A/E/P/12
12ème Atelier d'Evaluation de Performances, 4-5 juillet 2022, Grenoble

Fundamental Problem

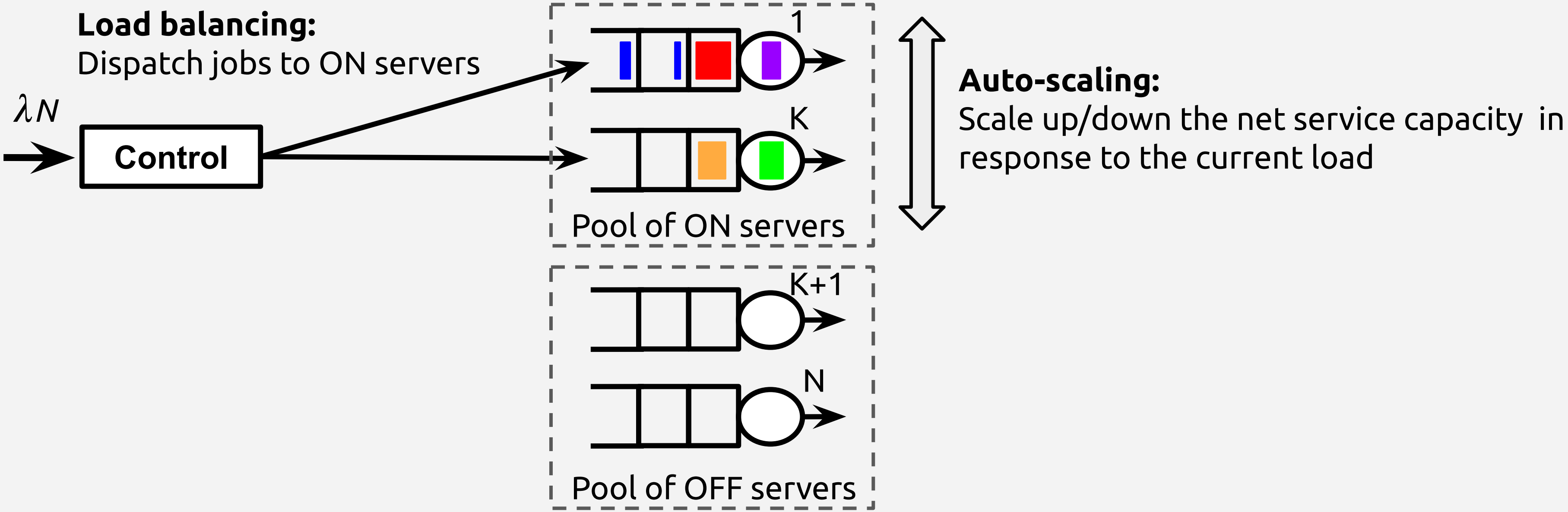
Decentralized control of large-scale parallel-server systems



Design objectives:

- ➔ Minimization of congestion (eg, delays) and operational costs (eg, energy usage)
- ➔ Simple policies: low complexity, scalability

Load Balancing and Auto-scaling



Challenge: Design algorithms that achieve low wait and energy consumption for large N

In math: Can we make latency go to zero with no waste of energy in the limit where $N \to \infty$?

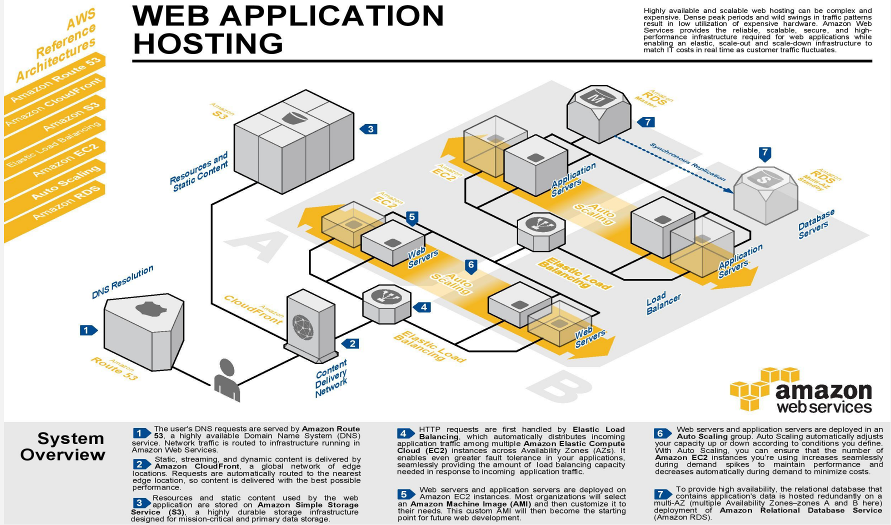
Assumptions: - The mean demand λN is proportional to the nominal service capacity N
- Load balancing and auto-scaling operate within the same timescale

Some Examples

Supermarket checkout lines

Call centers

Data centers



In France, 10% of the electricity produced is consumed only to meet the needs of data centres [source: https://corporate.ovhcloud.com]

Outline

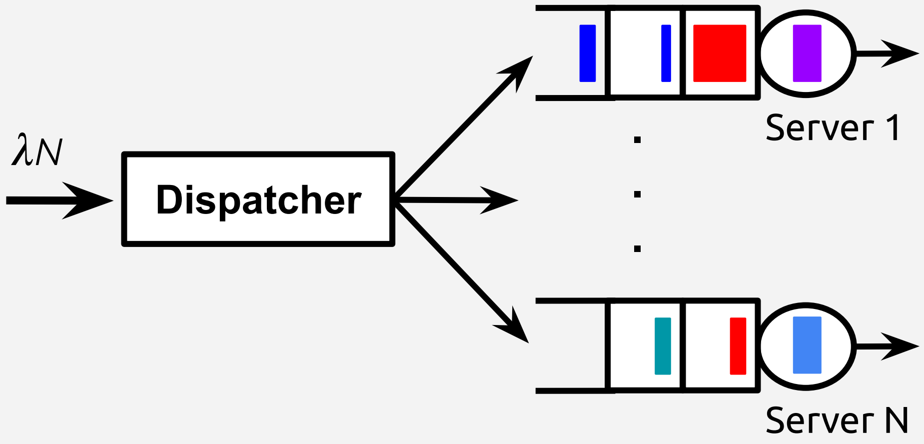
- Load balancing
  - Classic algorithms: quick review, fundamental question
  - Recent approaches: replication vs speculation
- Auto-scaling
  - Quick state of the art
  - A new framework

Outline

- **Load balancing**
  - **Classic algorithms: quick review, fundamental question**
  - **Recent approaches: replication vs speculation**
- Auto-scaling
  - Quick state of the art
  - A new framework

Standard Load Balancing

Classic algorithms: each incoming job is dispatched to a (unique) queue



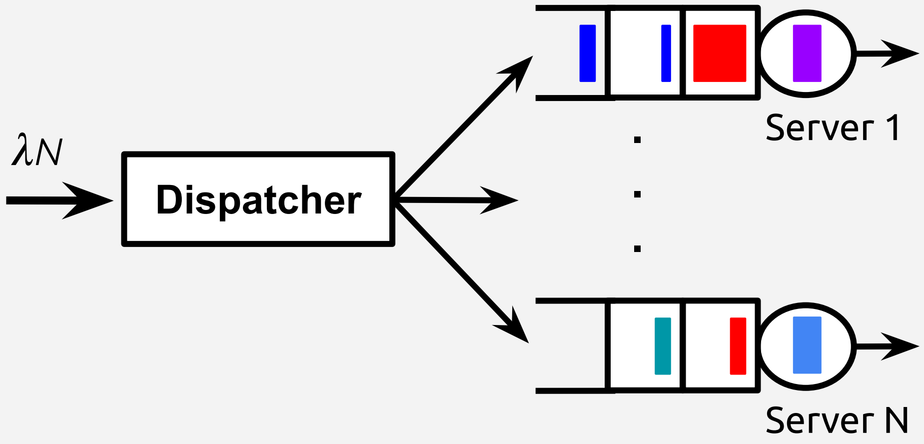
Huge Literature

Random
Round-Robin, RR
Join-the-shortest-queue, JSQ(N)
Power-of-d, JSQ(d)
Join-the-idle-queue
Least Left Workload
Size Interval Task Allocation, SITA

… and a lot more

Standard Load Balancing

Classic algorithms: each incoming job is dispatched to a (unique) queue



Huge Literature

Random
Round-Robin, RR
Join-the-shortest-queue, JSQ(N)
Power-of-d, JSQ(d)
Join-the-idle-queue
Least Left Workload
Size Interval Task Allocation, SITA

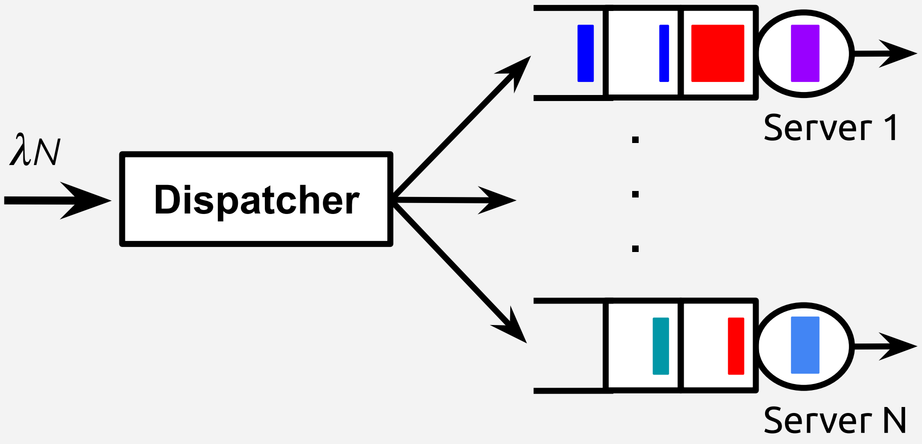
… and a lot more

JSQ: Zero wait but high complexity

Power-of-d : Non-zero wait but low complexity

Standard Load Balancing

Classic algorithms: each incoming job is dispatched to a (unique) queue



Huge Literature

Random

Round-Robin, RR

Join-the-shortest-queue, JSQ(N)

Power-of-*d*, JSQ(*d*)

Join-the-idle-queue

Least Left Workload

Size Interval Task Allocation, SITA

… and a lot more

JSQ: Zero wait but high complexity

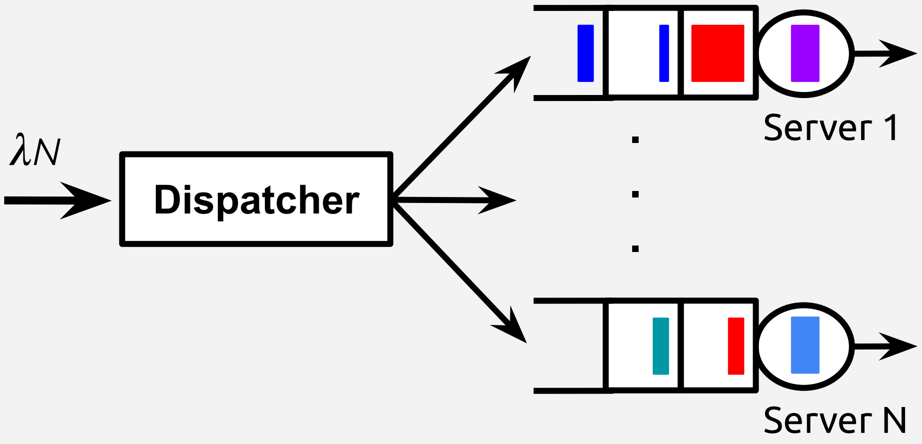
Power-of-*d*: Non-zero wait but low complexity

Power-of-*d* "with memory": Low complexity and zero wait if λ<1-1/d, excellent performance otherwise

[J. Anselmi, F. Dufour *Power-of-d-Choices with Memory: Fluid Limit and Optimality*, Mathematics of Operations Research, 2020]

Standard Load Balancing

Classic algorithms: each incoming job is dispatched to a (unique) queue



Huge Literature

Random

Round-Robin, RR

Join-the-shortest-queue, JSQ(*N*)

Power-of-*d*, JSQ(*d*)

Join-the-idle-queue

Least Left Workload

Size Interval Task Allocation, SITA

… and a lot more

JSQ: Zero wait but high complexity

Power-of-*d* : Non-zero wait but low complexity

Power-of-*d* "with memory": Low complexity and zero wait if λ<1-1/d, excellent performance otherwise

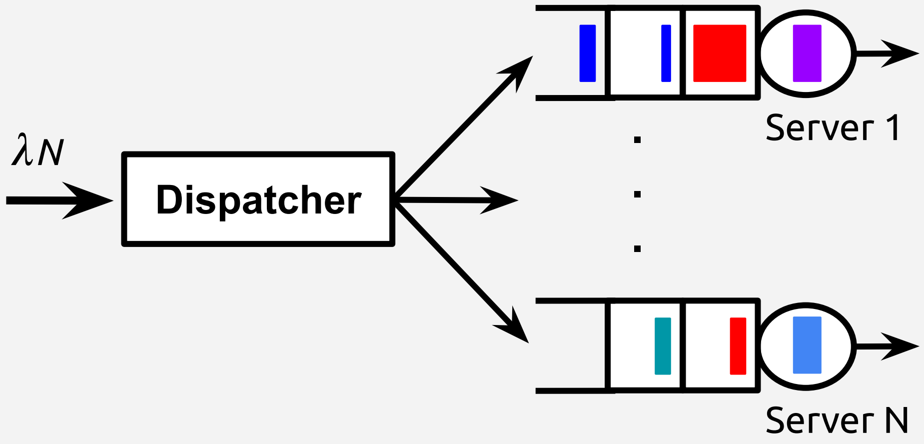
[J. Anselmi, F. Dufour *Power-of-d-Choices with Memory: Fluid Limit and Optimality*, Mathematics of Operations Research, 2020]

RR+SITA: Low complexity and zero wait if job sizes are known

[J. Anselmi *Combining Size-Based Load Balancing with Round-Robin for Scalable Low Latency*, IEEE Transactions on Parallel and Distributed Systems, 2019]

Standard Load Balancing

Classic algorithms: each incoming job is dispatched to a (unique) queue



Huge Literature

Random

Round-Robin, RR

Join-the-shortest-queue, JSQ(N)

Power-of-*d*, JSQ(*d*)

Join-the-idle-queue

Least Left Workload

Size Interval Task Allocation, SITA

… and a lot more

Remark: All these load balancing algorithms are *stable* if and only if $\lambda<1$. Can we do better?

Recent Approach: Replicate

[The Tail at Scale, Google Research]

Motivation: to mitigate the effect of *stragglers*

Two underlying principles

Either **replicate** (as in Google's BigTable):

1) *"replicate a job upon its arrival and use the results from whichever replica responds first"*

or **speculate** (as in Apache Spark and Hadoop MapReduce):

2) *"replicate a job as soon as the system detects it as a straggler"*

Recent Approach: Replicate

[The Tail at Scale, Google Research]

Motivation: to mitigate the effect of *stragglers*

Two underlying principles

Either **replicate** (as in Google's BigTable):

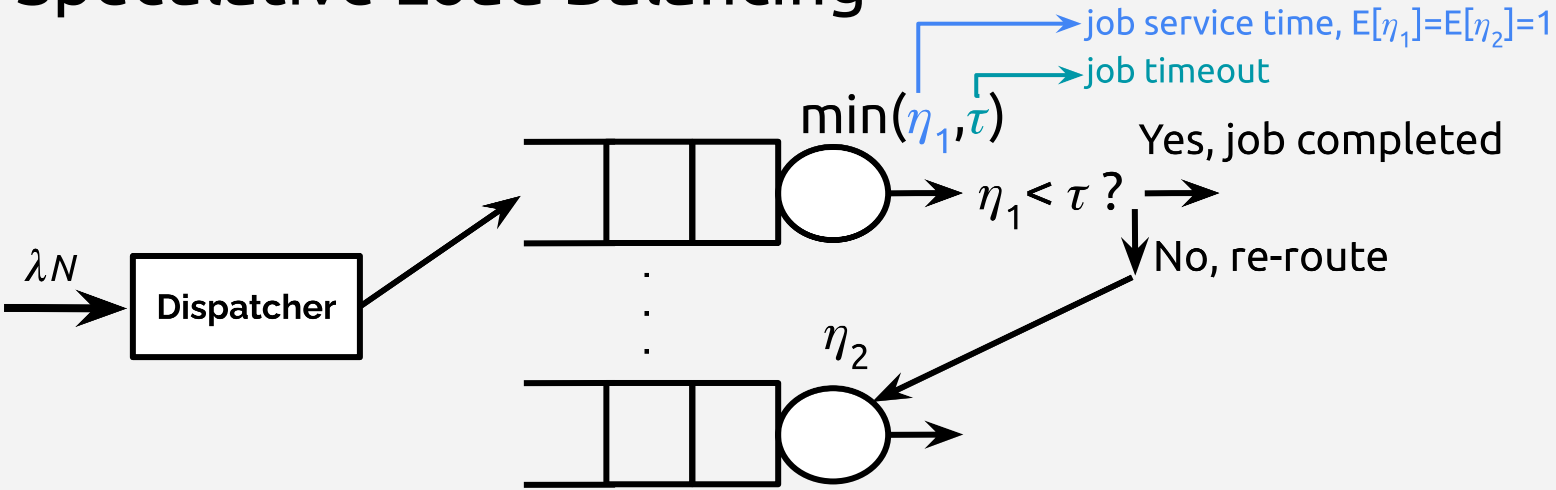
1) *"replicate a job upon its arrival and use the results from whichever replica responds first"*

or **speculate** (as in Apache Spark and Hadoop MapReduce):

2) *"replicate a job as soon as the system detects it as a straggler"*

Which approach provides the best results?

Speculative Load Balancing

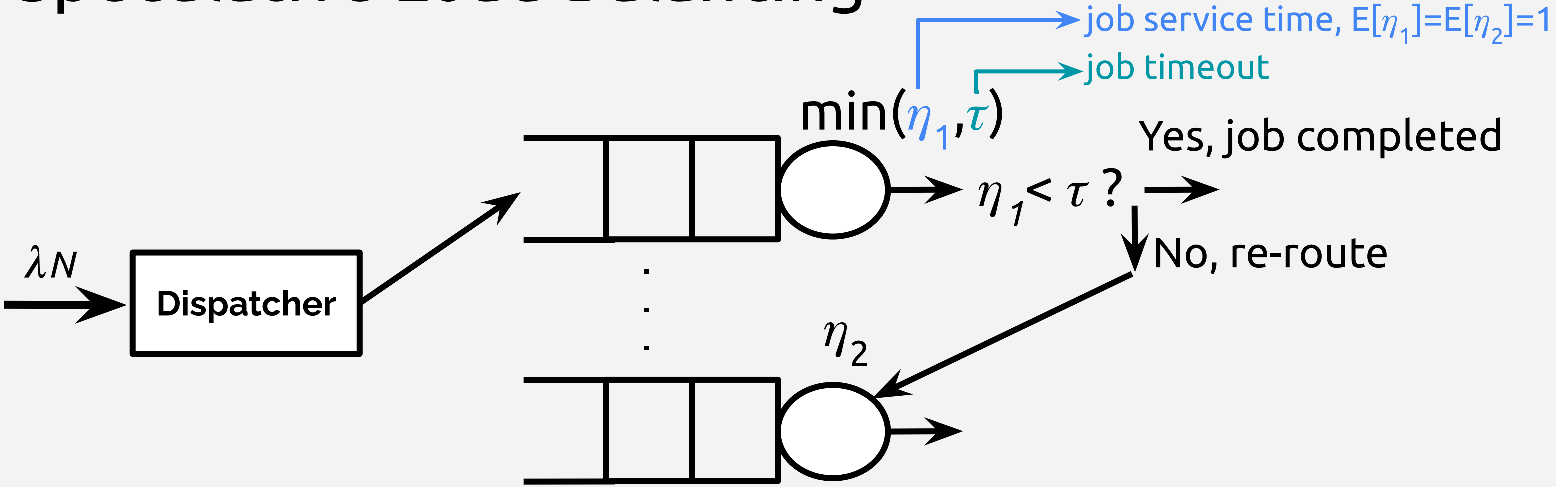


Theorem. The system is stable iff

$$\rho(\tau) := \mathbb{E}[\min(\eta_1, \tau)] + \mathbb{P}(\eta_1 > \tau)\mathbb{E}[\eta_2 \mid \eta_1 > \tau] < 1$$

[J. Anselmi and N. Walton *Stability and Optimization of Speculative Queueing Networks*, IEEE Transactions on Networking (to appear)]

Speculative Load Balancing



Theorem. The system is stable iff

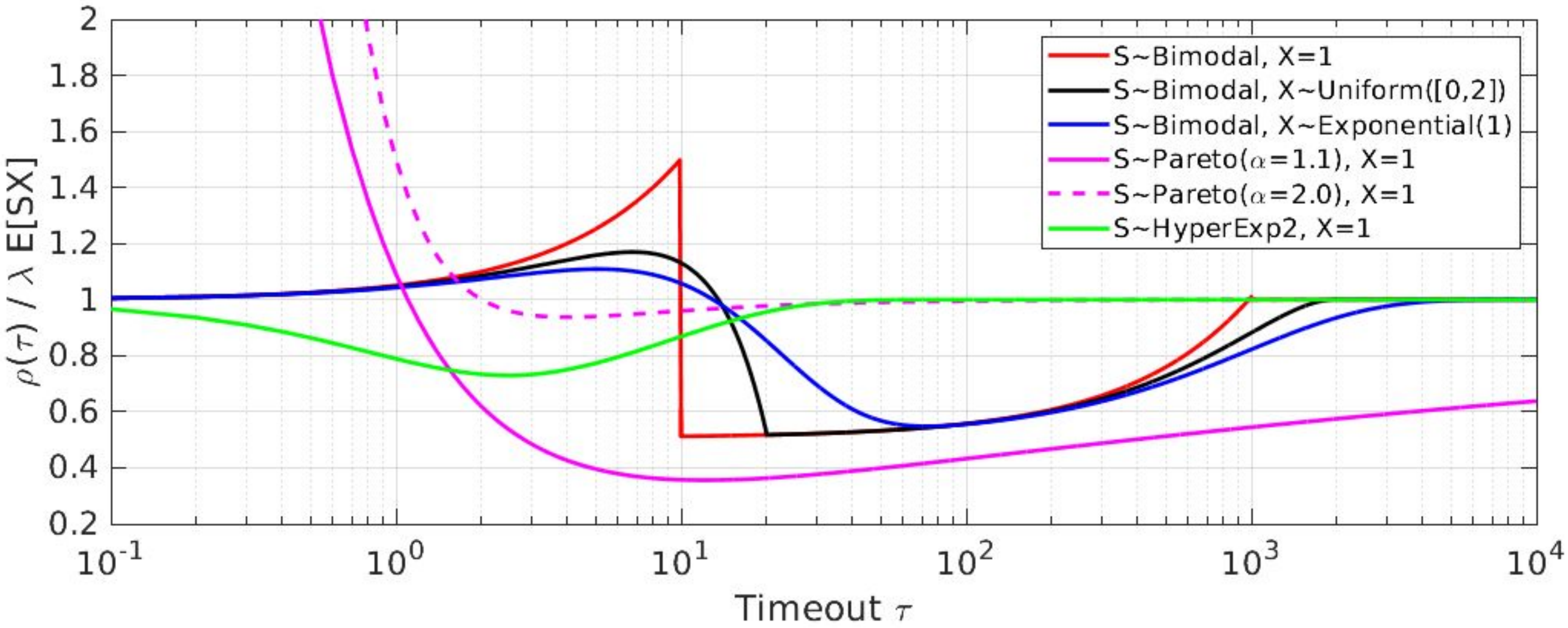
$$\rho(\tau) := \mathbb{E}[\min(\eta_1, \tau)] + \mathbb{P}(\eta_1 > \tau)\mathbb{E}[\eta_2 \mid \eta_1 > \tau] < 1$$

[J. Anselmi and N. Walton *Stability and Optimization of Speculative Queueing Networks*, IEEE Transactions on Networking (to appear)]

Remark. The stability regions of speculative load balancing, $\rho(\tau)<1$, and standard load balancing, $\lambda<1$, are different!

Speculation vs Standard Load Balancing

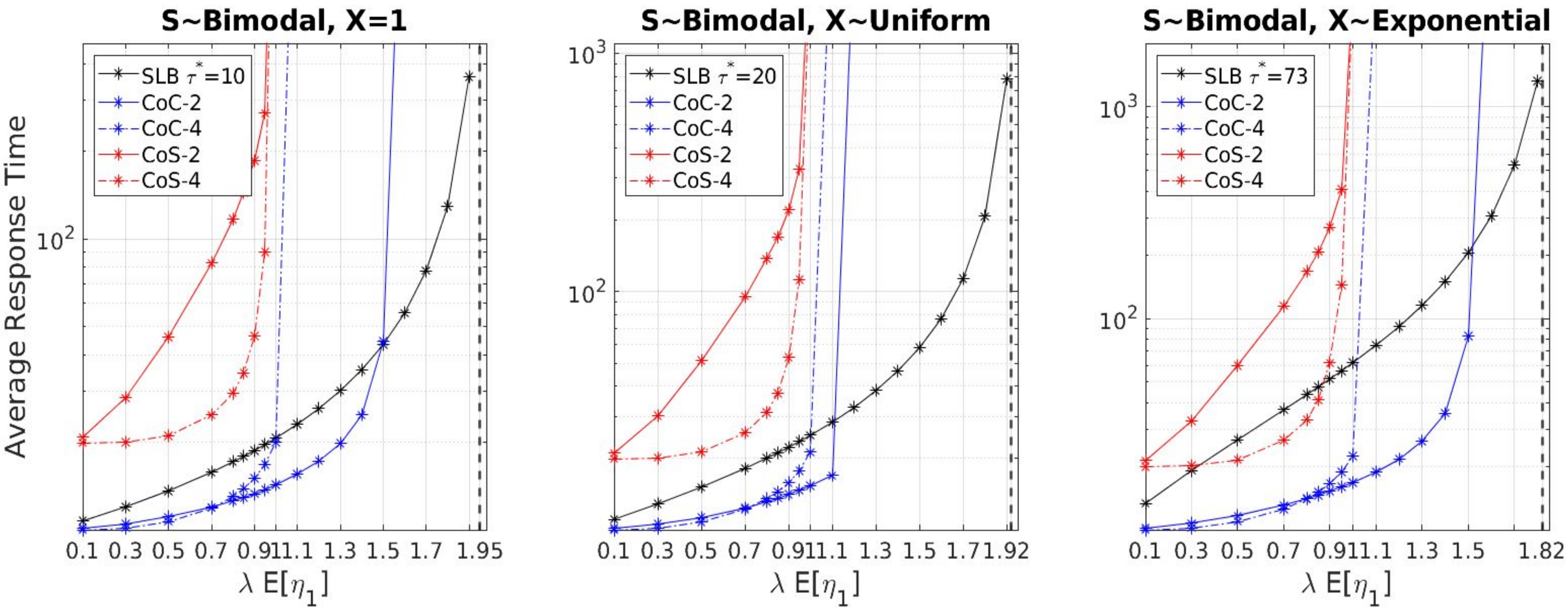
Service times at server *i*: $\eta_i = S_i X$ — where S_i ="server slowdown" and X="job intrinsic size"



$$\text{Bimodal:} \quad S = 10 \text{ w.p. } 0.99, S = 10^3 \text{ w.p. } 0.01.$$

Speculation vs Replication

Replication strategies: Cancel-on-Complete-*d* (CoC-*d*) and Cancel-on-Start-*d* (CoS-*d*)

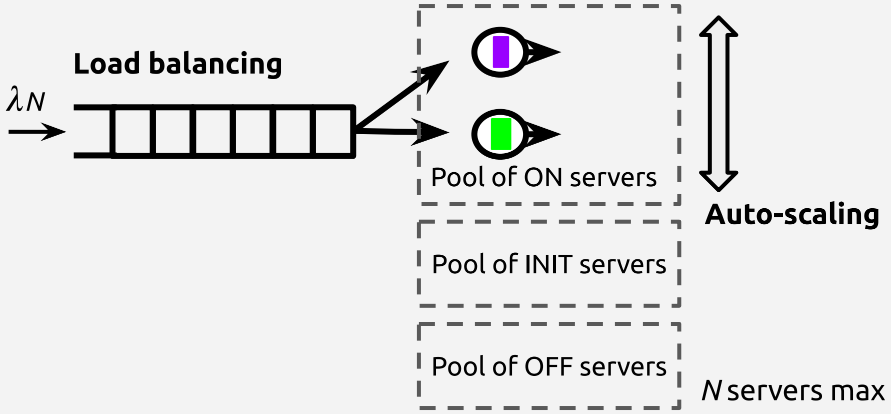


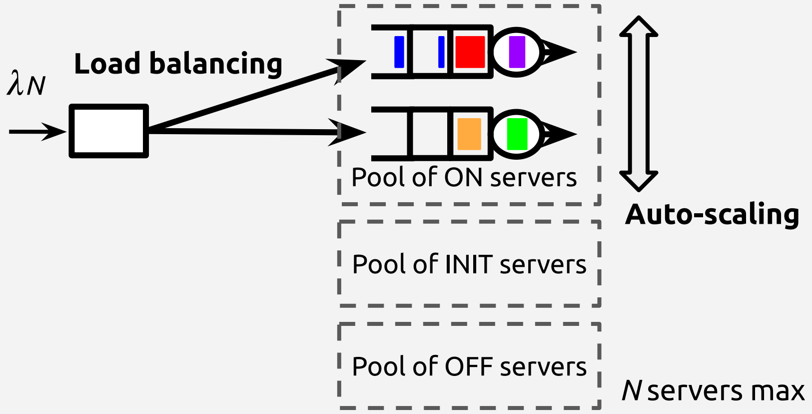
⇒ Speculation provides a larger stability region!

Outline

- Load balancing
  - Classic algorithms: quick review, fundamental question
  - Recent approaches: replication vs speculation
- **Auto-scaling**
  - **Quick state of the art**
  - **A new framework**

Load Balancing and Auto-scaling



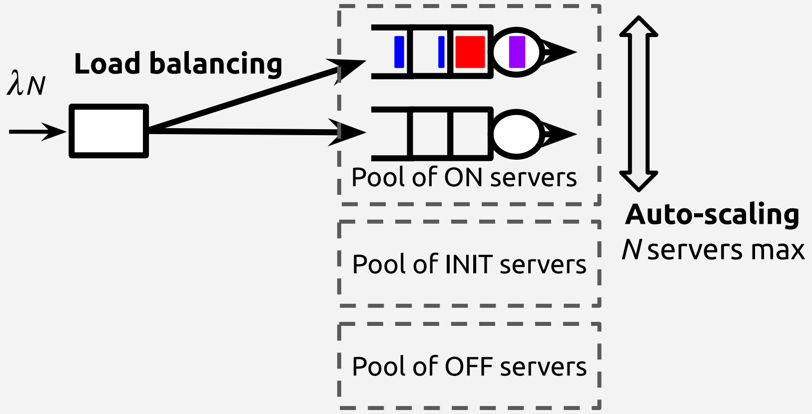


	Centralized	**Decentralized**
Synchronous	AWS Lambda, Azure Functions, IBM Cloud Functions, Apache OpenWhisk ⇒ several research works	Some theoretical work [Borst et al. 2017, Clausen et al. 2021]
Asynchronous	?	**Knative** (Google Cloud Run) ⇒ no theoretical work (AFAIK!)

Asynchronous Load Balancing and Auto-scaling

Challenge

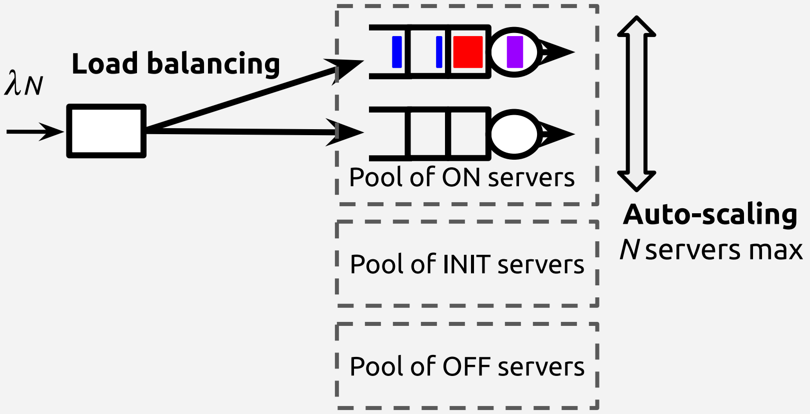
To investigate the dynamics of the serverless platform Knative to help the platform user to design and efficiently evaluate the performance of different scaling rules.



DREO: Delay and Relative Energy Optimality

User-perceived delay and the relative energy wastage induced by idle servers vanish as $N \longrightarrow \infty$

Asynchronous Load Balancing and Auto-scaling



Challenge

To investigate the dynamics of the serverless platform Knative to help the platform user to design and efficiently evaluate the performance of different scaling rules.

DREO: Delay and Relative Energy Optimality

User-perceived delay and the relative energy wastage induced by idle servers vanish as $N \longrightarrow \infty$

Theorem (Optimal Design). DREO is guaranteed by using Join-the-Idle-Queue and a scale-up rate that is zero if and only if λ exceeds the overall rate at which servers become idle-on, i.e., idle and active.

[J. Anselmi *Asynchronous Load Balancing and Auto-scaling: Mean-field Limit and Optimal Design* (submitted) https://arxiv.org/abs/2204.02352]