Scalable Load Balancing in Networked Systems

Sem Borst

Eindhoven University of Technology (TU/e) & Nokia Bell Labs

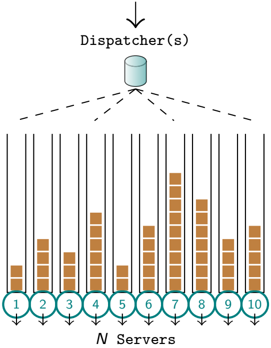
UFRJ, 2 August 2018

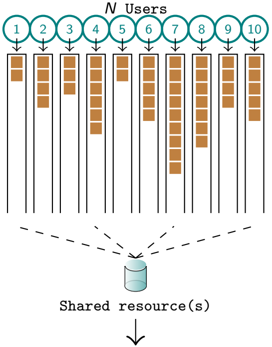
Based on joint work with Mark van der Boor, Johan van Leeuwaarden, Debankur Mukherjee & Phil Whiting

Two Fundamental Problems in Networked Systems

Centralized queue impractical/undesirable in large-scale networked systems

Achieve low delay and maintain low overhead as system grows large (with highly distributed queues)





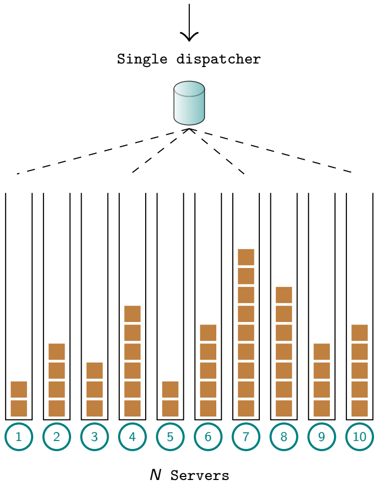
Load balancing/routing

Assign tasks from single arrival stream to be performed by one of several servers, each with their own queue

Resource allocation/scheduling

Allocate shared resource to execute tasks from one of several users, each with their own arrival stream and queue

Load Balancing/Routing in Parallel-Server Systems



Large-Scale Parallel-Server Systems: Some Examples

Supermarket checkout line

Large-Scale Parallel-Server Systems: Some Examples

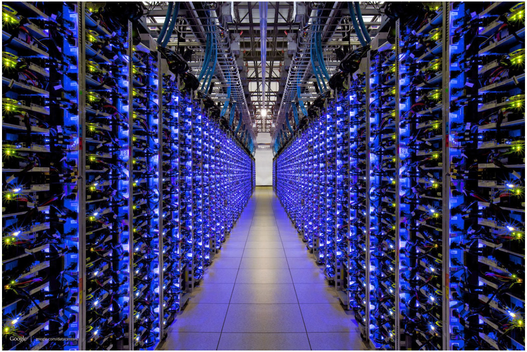
Supermarket checkout line

Road toll plaza

Large-Scale Parallel-Server Systems: Some Examples

Supermarket checkout line

Road toll plaza



Data center

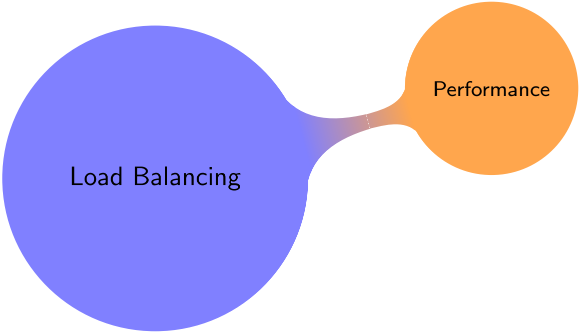
High-Level Outline

I. Scalability challenges and classical results
II. Asymptotic optimality and universality (no memory)
III. Reduction in communication overhead (memory)
IV. Heterogeneity issues and network scenarios

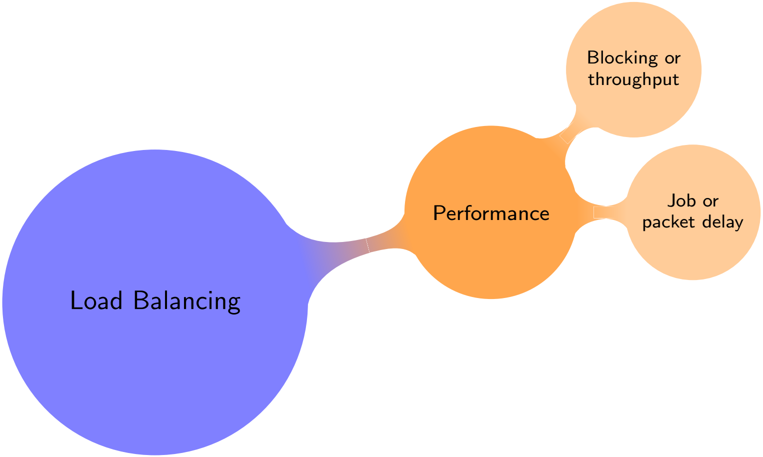
I. Scalability challenges and classical results
II. Asymptotic optimality and universality (no memory)
III. Reduction in communication overhead (memory)
IV. Heterogeneity issues and network scenarios

Load Balancing: Scalability Issues

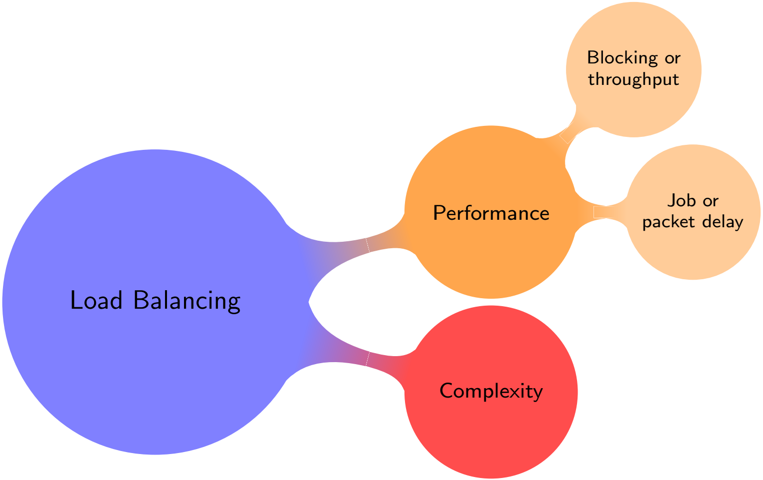
Load Balancing: Scalability Issues



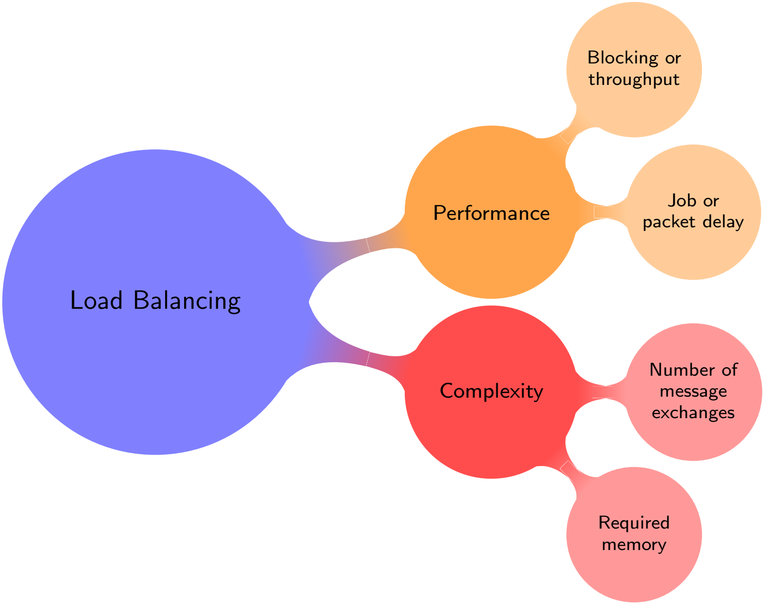
Load Balancing: Scalability Issues



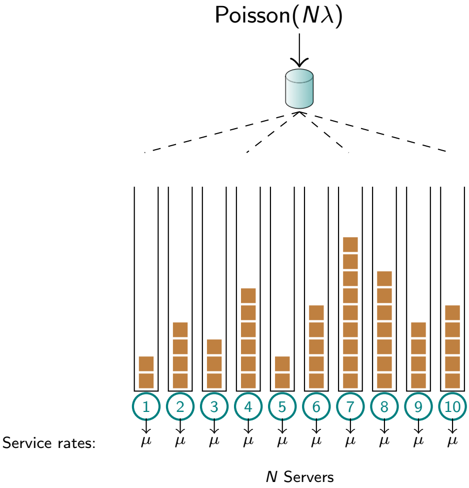
Load Balancing: Scalability Issues



Load Balancing: Scalability Issues

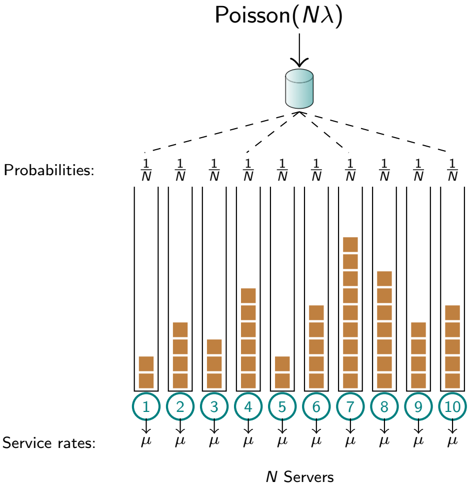


Purely Random Assignment



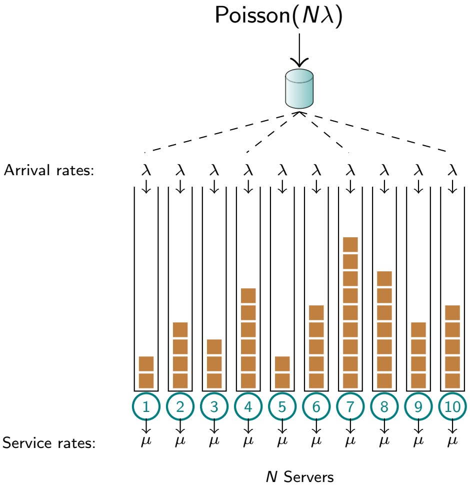
Purely Random Assignment

Assign each task to server selected uniformly at random



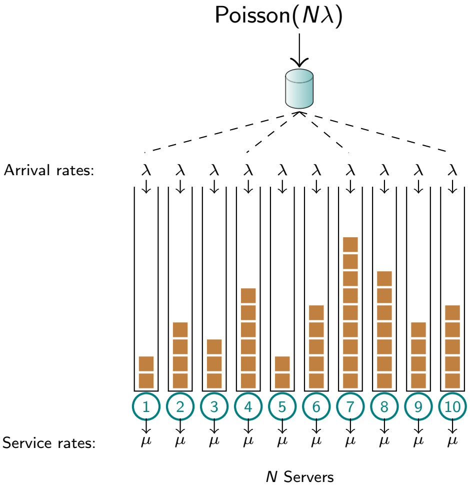
Purely Random Assignment

Assign each task to server selected uniformly at random



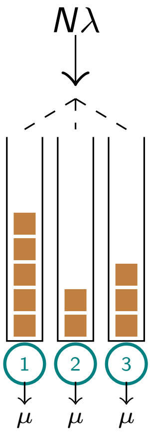
Purely Random Assignment

Assign each task to server selected uniformly at random



N independent M/M/1 queues with arrival rate λ and service rate μ

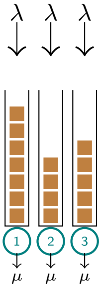
Load Balancing Scenarios



Separate Queues

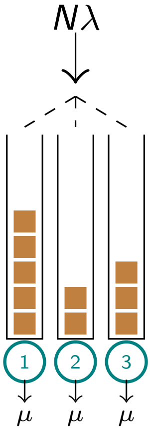
Strictly Random Routing

N $\times$ M/M/1



queue length $N \times \frac{\lambda^2}{\mu(\mu-\lambda)}$

waiting time $\frac{\lambda}{\mu(\mu-\lambda)}$

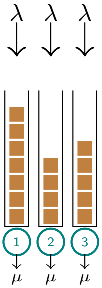


Load Balancing Scenarios

Separate Queues

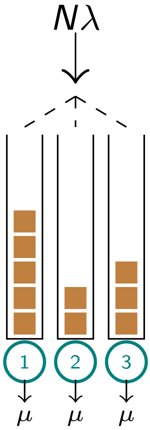
Strictly Random Routing

N $\times$ M/M/1



queue length $N \times \frac{\lambda^2}{\mu(\mu-\lambda)}$

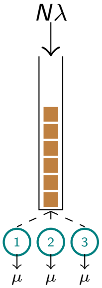
waiting time $\frac{\lambda}{\mu(\mu-\lambda)}$



Centralized Queue

Complete Resource Pooling

M/M/N

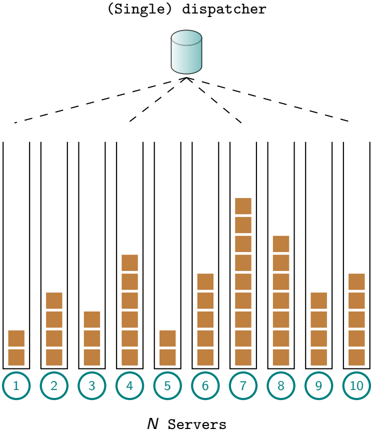


queue length $\Pi_W \frac{\lambda}{\mu-\lambda}$

waiting time $\frac{1}{N} \Pi_W \frac{1}{\mu-\lambda}$

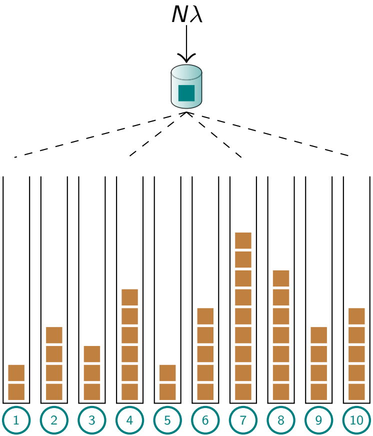
Join the Shortest Queue (JSQ) Policy

Classical natural policy: Join the Shortest Queue (JSQ)



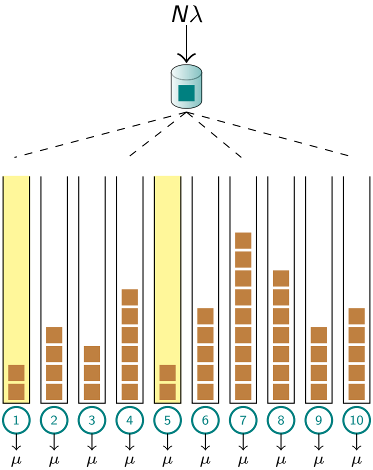
Join the Shortest Queue (JSQ) Policy

Classical natural policy: Join the Shortest Queue (JSQ)



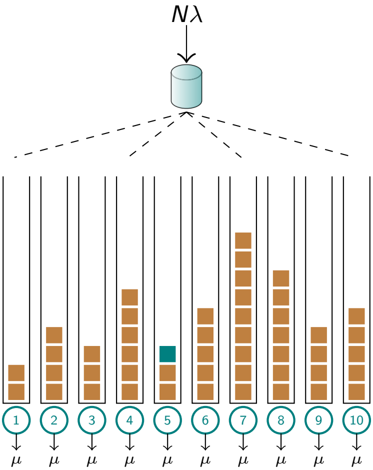
Join the Shortest Queue (JSQ) Policy

Classical natural policy: Join the Shortest Queue (JSQ)



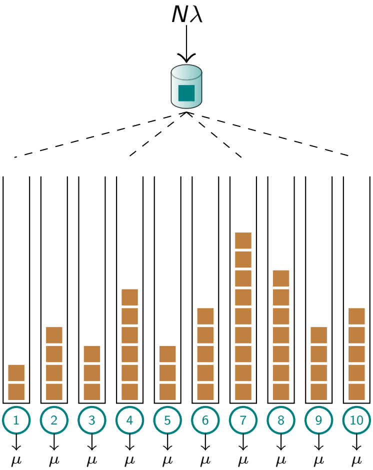
Join the Shortest Queue (JSQ) Policy

Classical natural policy: Join the Shortest Queue (JSQ)



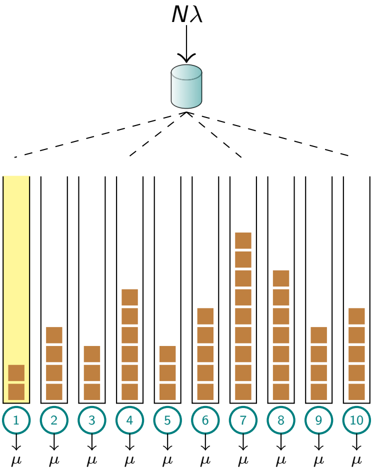
Join the Shortest Queue (JSQ) Policy

Classical natural policy: Join the Shortest Queue (JSQ)



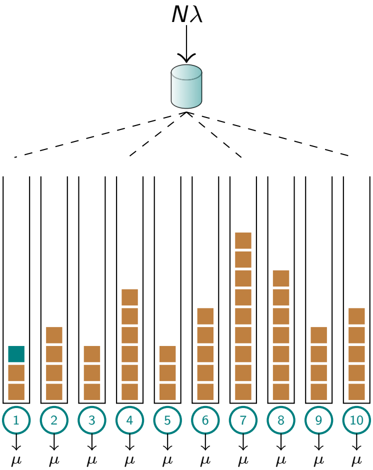
Join the Shortest Queue (JSQ) Policy

Classical natural policy: Join the Shortest Queue (JSQ)



Join the Shortest Queue (JSQ) Policy

Classical natural policy: Join the Shortest Queue (JSQ)



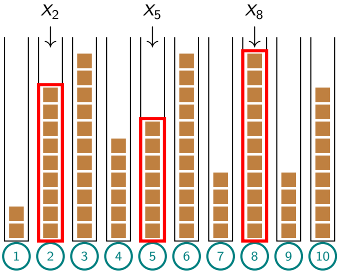
Equivalent State Description

Symmetry among servers allows equivalent state description

Equivalent State Description

Symmetry among servers allows equivalent state description

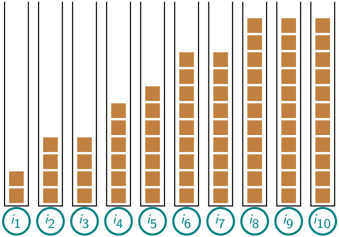
- Denote by X_k queue length at server k, $k = 1, 2, \ldots, N$



Equivalent State Description

Symmetry among servers allows equivalent state description

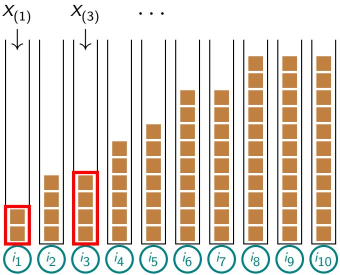
- Denote by X_k queue length at server k, $k = 1, 2, \ldots, N$
- Rearrange servers from left to right in non-decreasing order of X_k's



Equivalent State Description

Symmetry among servers allows equivalent state description

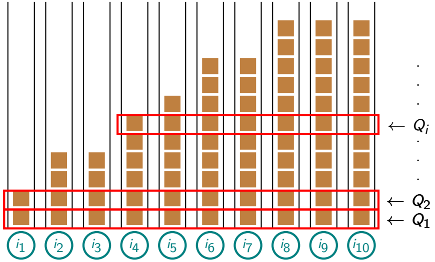
- Denote by X_k queue length at server k, $k = 1, 2, \ldots, N$
- Rearrange servers from left to right in non-decreasing order of X_k's
- $X_{(k)}$ is size of k^{th} shortest queue, i.e., height of k^{th} left-most stack



Equivalent State Description

Symmetry among servers allows equivalent state description

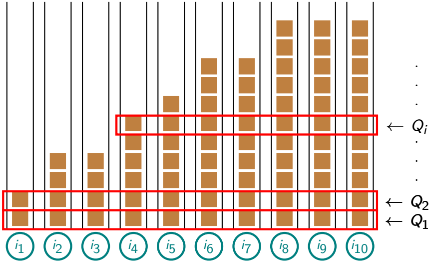
- Denote by X_k queue length at server k, $k = 1, 2, \ldots, N$
- Rearrange servers from left to right in non-decreasing order of X_k's
- $X_{(k)}$ is size of k^{th} shortest queue, i.e., height of k^{th} left-most stack
- Denote by Q_i number of servers with queue length i or larger, $i = 1, 2, \ldots,$ i.e., width of i^{th} horizontal bar



Equivalent State Description

Symmetry among servers allows equivalent state description

- Denote by X_k queue length at server k, $k = 1, 2, \ldots, N$
- Rearrange servers from left to right in non-decreasing order of X_k's
- $X_{(k)}$ is size of k^{th} shortest queue, i.e., height of k^{th} left-most stack
- Denote by Q_i number of servers with queue length i or larger, $i = 1, 2, \ldots$, i.e., width of i^{th} horizontal bar
- Fluid-scaled state variables $q_i^N(t) = Q_i^N(t)/N$ represent fraction of servers with queue length i or larger



Stochastic Optimality of JSQ Policy

- JSQ stochastically minimizes aggregate size of l right-most stacks, i.e., total number of tasks in l largest queues

$$\sum_{k=N-l+1}^{N} X_{(k)}^{\mathrm{JSQ}} \leq_{st} \sum_{k=N-l+1}^{N} X_{(k)}^{\Pi}$$

for all $l = 1, \ldots, N$, for any non-anticipating policy Π

[Towsley-Sparragis-Cassandras 1992; Sparaggis-Towsley-Cassandra 1994]

Stochastic Optimality of JSQ Policy

- JSQ stochastically minimizes aggregate size of l right-most stacks, i.e., total number of tasks in l largest queues

$$\sum_{k=N-l+1}^{N} X_{(k)}^{\mathrm{JSQ}} \leq_{st} \sum_{k=N-l+1}^{N} X_{(k)}^{\Pi}$$

for all $l = 1, \ldots, N$, for any non-anticipating policy Π

[Towsley-Sparragis-Cassandras 1992; Sparaggis-Towsley-Cassandra 1994]

- JSQ stochastically minimizes aggregate size of bars at level j or higher, i.e., total number of tasks in queue position j or higher

$$\sum_{i=j}^{\infty} Q_i^{\mathrm{JSQ}} \leq_{st} \sum_{i=j}^{\infty} Q_i^{\Pi}$$

for all $j = 1, 2, \ldots,$ for any non-anticipating policy Π

[Mukherjee, B, Van Leeuwaarden, Whiting 2016]

Stochastic Optimality of JSQ Policy

- JSQ stochastically minimizes aggregate size of l right-most stacks, i.e., total number of tasks in l largest queues

$$\sum_{k=N-l+1}^{N} X_{(k)}^{\mathrm{JSQ}} \leq_{st} \sum_{k=N-l+1}^{N} X_{(k)}^{\Pi}$$

  for all $l = 1, \ldots, N$, for any non-anticipating policy Π

  [Towsley-Sparragis-Cassandras 1992; Sparaggis-Towsley-Cassandra 1994]

- JSQ stochastically minimizes aggregate size of bars at level j or higher, i.e., total number of tasks in queue position j or higher

$$\sum_{i=j}^{\infty} Q_i^{\mathrm{JSQ}} \leq_{st} \sum_{i=j}^{\infty} Q_i^{\Pi}$$

  for all $j = 1, 2, \ldots,$ for any non-anticipating policy Π

  [Mukherjee, B, Van Leeuwaarden, Whiting 2016]

In particular (take $l = N$ or $j = 1$), JSQ stochastically minimizes total number of tasks in system, and hence overall mean delay

JSQ Policy in Many-Server Regime

JSQ yields dramatic performance improvements as $N \to \infty$

JSQ Policy in Many-Server Regime

JSQ yields dramatic performance improvements as $N \to \infty$

- Eliminates queues

Scheme	Queue length	
Random	$p_i^N = \lambda^i$	
JSQ	$p_1^N \to \lambda$, $p_2^N \to 0$	

p_i^N is stationary probability that queue length at server is $\geq i$ in N^{th} system

JSQ Policy in Many-Server Regime

JSQ yields dramatic performance improvements as $N \to \infty$

- Eliminates queues
- Achieves zero wait

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)	
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	
JSQ	$p_1^N \to \lambda$, $p_2^N \to 0$	$o(1)$	$\Theta(1/\sqrt{N})$	

p_i^N is stationary probability that queue length at server is $\geq i$ in N^{th} system

JSQ yields dramatic performance improvements as $N \to \infty$

- Eliminates queues
- Achieves zero wait

But...

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)	
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	
JSQ	$p_1^N \to \lambda$, $p_2^N \to 0$	$o(1)$	$\Theta(1/\sqrt{N})$	

p_i^N is stationary probability that queue length at server is $\geq i$ in N^{th} system

JSQ Policy in Many-Server Regime

JSQ yields dramatic performance improvements as $N \to \infty$

- Eliminates queues
- Achieves zero wait

But...

- JSQ involves high communication overhead as $N \to \infty$

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)	Overhead
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	0
JSQ	$p_1^N \to \lambda$, $p_2^N \to 0$	$o(1)$	$\Theta(1/\sqrt{N})$	$2N$

p_i^N is stationary probability that queue length at server is $\geq i$ in N^{th} system

JSQ Policy in Many-Server Regime

JSQ yields dramatic performance improvements as $N \to \infty$

- Eliminates queues
- Achieves zero wait

But...

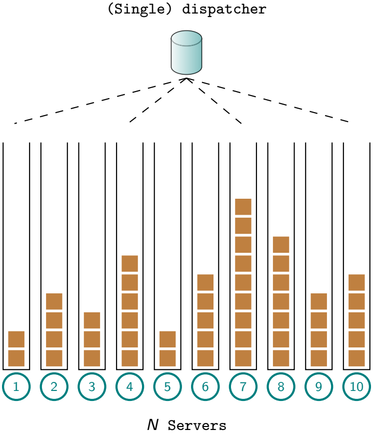
- JSQ involves high communication overhead as $N \to \infty$
- Straightforward implementation of JSQ (no memory at dispatcher) requires queue lengths at all servers to be checked at each arrival, which may be prohibitive in large-scale systems

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)	Overhead
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	0
JSQ	$p_1^N \to \lambda$, $p_2^N \to 0$	o(1)	$\Theta(1/\sqrt{N})$	$2N$

p_i^N is stationary probability that queue length at server is $\geq i$ in N^{th} system

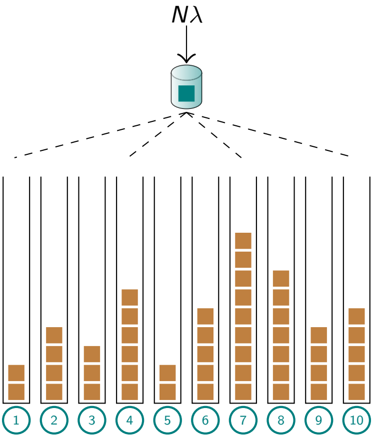
JSQ(d) Scheme: Reduced Communication Overhead

At each arrival, select d servers (e.g. $d = 2$) uniformly at random, and assign task to shortest queue among selected servers



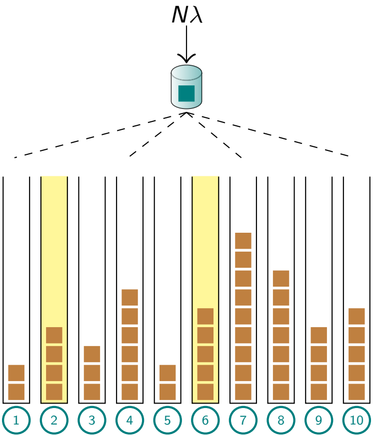
JSQ(d) Scheme: Reduced Communication Overhead

At each arrival, select d servers (e.g. $d = 2$) uniformly at random, and assign task to shortest queue among selected servers



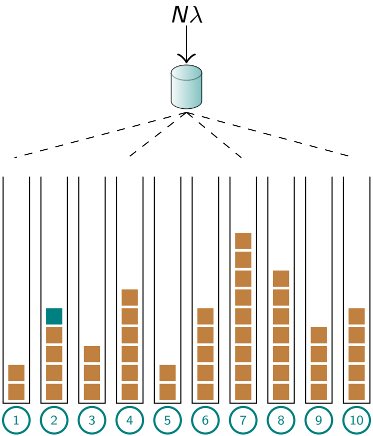
JSQ(d) Scheme: Reduced Communication Overhead

At each arrival, select d servers (e.g. $d = 2$) uniformly at random, and assign task to shortest queue among selected servers



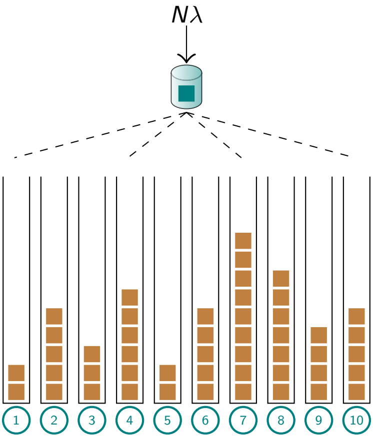
JSQ(d) Scheme: Reduced Communication Overhead

At each arrival, select d servers (e.g. $d = 2$) uniformly at random, and assign task to shortest queue among selected servers



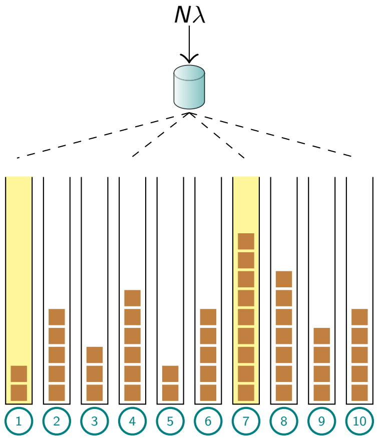
JSQ(d) Scheme: Reduced Communication Overhead

At each arrival, select d servers (e.g. $d = 2$) uniformly at random, and assign task to shortest queue among selected servers



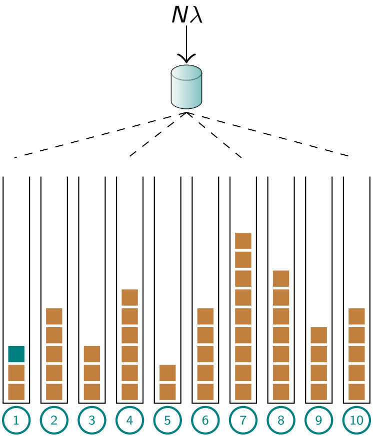
JSQ(d) Scheme: Reduced Communication Overhead

At each arrival, select d servers (e.g. $d = 2$) uniformly at random, and assign task to shortest queue among selected servers



JSQ(d) Scheme: Reduced Communication Overhead

At each arrival, select d servers (e.g. $d = 2$) uniformly at random, and assign task to shortest queue among selected servers



Yet Lower Communication Overhead per Task for Batch Arrivals

For batch arrivals d queue samples can be amortized over several tasks

L. Ying, R. Srikant, X. Kang (2015). The power of slightly more than one sample in randomized load balancing. In: *Proc. IEEE Infocom 2015*.

Connection between JSQ(d) and Redundancy-d Policies

Redundancy-d policy assigns replicas of each arriving task to d servers sampled uniformly at random, and kills $d-1$ replicas as soon as execution of first replica is started/completed: Wait before see rather than see before wait

K.S. Gardner, M. Harchol-Balter, A. Scheller-Wolf, M. Velednitsky, S. Zbarsky (2017). Redundancy-d: The power of d choices for redundancy. *Oper. Res.* **65 (4)**, 1078–1094.

K.S. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, A. Scheller-Wolf (2015). Reducing latency via redundant requests: Exact analysis. In: *ACM SIGMETRICS Perf. Eval. Rev.* **43 (1)**, 347–360.

K.S. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, A. Scheller-Wolf (2016). Queueing with redundant requests: Exact analysis. *Queueing Systems* **83 (3–4)**, 227–259.

K.S. Gardner, S. Zbarsky, M. Harchol-Balter, A. Scheller-Wolf (2016). The power of d choices for redundancy. In: *ACM SIGMETRICS Perf. Eval. Rev.* **44 (1)**, 409–410.

K.S. Gardner, S. Zbarsky, M. Velednitsky, M. Harchol-Balter, A. Scheller-Wolf (2016). Understanding response time in the redundancy-d system. In: *ACM SIGMETRICS Perf. Eval. Rev.* **44 (2)**, 33–35.

Fluid Limit for JSQ(d) Scheme

Fluid limit for JSQ(d) [Mitzenmacher 1996, 2001; Vvedenskaya *et al.* 1996]

If $\mathbf{q}^{JSQ(d)}(0) \to \mathbf{q}^{\infty}$ as $N \to \infty$, then $\{\mathbf{q}^{JSQ(d)}(t)\}_{t \geq 0}$ weakly converges to $\{\mathbf{q}(t)\}_{t \geq 0}$ as $N \to \infty$, with

$$\frac{dq_i(t)}{dt} = \lambda[(q_{i-1}(t))^d - (q_i(t))^d] - \mu[q_i(t) - q_{i+1}(t)],$$

where $\mathbf{q}(0) = \mathbf{q}^{\infty}$

Fluid Limit for JSQ(d) Scheme

Fluid limit for JSQ(d) [Mitzenmacher 1996, 2001; Vvedenskaya *et al.* 1996]

If $\mathbf{q}^{JSQ(d)}(0) \to \mathbf{q}^\infty$ as $N \to \infty$, then $\{\mathbf{q}^{JSQ(d)}(t)\}_{t\geq 0}$ weakly converges to $\{\mathbf{q}(t)\}_{t\geq 0}$ as $N \to \infty$, with

$$\frac{dq_i(t)}{dt} = \lambda[(q_{i-1}(t))^d - (q_i(t))^d] - \mu[q_i(t) - q_{i+1}(t)],$$

where $\mathbf{q}(0) = \mathbf{q}^\infty$

$[(q_{i-1}(t))^d - (q_i(t))^d]$ is (instantaneous) fraction of arriving tasks assigned to servers with queue length $i-1$ in fluid-level state $\mathbf{q}(t)$

Power-of-d Effect for JSQ(d) Scheme

Assuming $\mu = 1$ as before, fixed point of fluid limit is

$$q_i^* := \lim_{t \to \infty} q_i(t) = \lambda^{\frac{d^i - 1}{d-1}}, \qquad i = 1, 2, \ldots$$

Power-of-d Effect for JSQ(d) Scheme

Assuming $\mu = 1$ as before, fixed point of fluid limit is

$$q_i^* := \lim_{t \to \infty} q_i(t) = \lambda^{\frac{d^i - 1}{d-1}}, \qquad i = 1, 2, \ldots$$

Denote $p_i^* := \lim_{N \to \infty} p_i^N$,
with p_i^N stationary probability that queue length at server is $\geq i$ in N^{th} system

Power-of-d Effect for JSQ(d) Scheme

Assuming $\mu = 1$ as before, fixed point of fluid limit is

$$q_i^* := \lim_{t \to \infty} q_i(t) = \lambda^{\frac{d^i - 1}{d - 1}}, \qquad i = 1, 2, \ldots$$

Denote $p_i^* := \lim_{N \to \infty} p_i^N$,
with p_i^N stationary probability that queue length at server is $\geq i$ in N^{th} system

Interchange of large-N and large-t limits: $p_i^* = q_i^*$

Power-of-d Effect for JSQ(d) Scheme

Assuming $\mu = 1$ as before, fixed point of fluid limit is

$$q_i^* := \lim_{t \to \infty} q_i(t) = \lambda^{\frac{d^i - 1}{d - 1}}, \qquad i = 1, 2, \ldots$$

Denote $p_i^* := \lim_{N \to \infty} p_i^N$,
with p_i^N stationary probability that queue length at server is $\geq i$ in N^{th} system

Interchange of large-N and large-t limits: $p_i^* = q_i^*$

Tail of stationary queue length distribution falls off much faster for $d \geq 2$ than for purely random assignment ($d = 1$)

Power-of-d Effect for JSQ(d) Scheme

Assuming $\mu = 1$ as before, fixed point of fluid limit is

$$q_i^* := \lim_{t \to \infty} q_i(t) = \lambda^{\frac{d^i - 1}{d-1}}, \qquad i = 1, 2, \ldots$$

Denote $p_i^* := \lim_{N \to \infty} p_i^N$,
with p_i^N stationary probability that queue length at server is $\geq i$ in N^{th} system

Interchange of large-N and large-t limits: $p_i^* = q_i^*$

Tail of stationary queue length distribution falls off much faster for $d \geq 2$ than for purely random assignment ($d = 1$)

"power-of-two" effect: Even value as small as $d = 2$ yields significant performance improvements over purely random assignment, while drastically reducing communication overhead compared to JSQ ($d = N$)

JSQ(d) Provides Strong Benefits over Random Assignment

JSQ(d) provides doubly-exponential rather than (singly-)exponential decay of stationary queue length tail

Scheme	Queue length		Overhead
Random	$p_i^N = \lambda^i$		0
JSQ(d)	$p_i^* = \lambda^{\frac{d^i-1}{d-1}}$		$2d$
JSQ	$p_1^* = \lambda$, $p_2^* = 0$		$2N$

JSQ(d) Provides Strong Benefits over Random Assignment

JSQ(d) provides doubly-exponential rather than (singly-)exponential decay of stationary queue length tail

But...

Scheme	Queue length		Overhead
Random	$p_i^N = \lambda^i$		0
JSQ(d)	$p_i^* = \lambda^{\frac{d^i-1}{d-1}}$		$2d$
JSQ	$p_1^* = \lambda$, $p_2^* = 0$		$2N$

JSQ(d) Provides Strong Benefits over Random Assignment

JSQ(d) provides doubly-exponential rather than (singly-)exponential decay of stationary queue length tail

But...

- JSQ(d) scheme is suboptimal for any fixed value of d

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)	Overhead
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	0
JSQ(d)	$p_i^* = \lambda^{\frac{d^i-1}{d-1}}$	$\Theta(1)$	$\Omega(\log N)$	$2d$
JSQ	$p_1^* = \lambda$, $p_2^* = 0$	$o(1)$	$\Theta(1/\sqrt{N})$	$2N$

JSQ(d) Provides Strong Benefits over Random Assignment

JSQ(d) provides doubly-exponential rather than (singly-)exponential decay of stationary queue length tail

But...

- JSQ(d) scheme is suboptimal for any fixed value of d
- Waiting time does not vanish as $N \to \infty$

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)	Overhead
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	0
JSQ(d)	$p_i^* = \lambda^{\frac{d^i-1}{d-1}}$	$\Theta(1)$	$\Omega(\log N)$	$2d$
JSQ	$p_1^* = \lambda$, $p_2^* = 0$	$o(1)$	$\Theta(1/\sqrt{N})$	$2N$

JSQ(d) Provides Strong Benefits over Random Assignment

JSQ(d) provides doubly-exponential rather than (singly-)exponential decay of stationary queue length tail

But...

- JSQ(d) scheme is suboptimal for any fixed value of d
- Waiting time does not vanish as $N \to \infty$

In absence of any memory at dispatcher, communication overhead must grow with N in order for zero delay to be achievable [Gamarnik-Tsitsiklis-Zubeldia 2016]

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time $(1 - \lambda \sim 1/\sqrt{N})$	Overhead
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	0
JSQ(d)	$p_i^* = \lambda^{\frac{d^i - 1}{d-1}}$	$\Theta(1)$	$\Omega(\log N)$	$2d$
JSQ	$p_1^* = \lambda$, $p_2^* = 0$	$o(1)$	$\Theta(1/\sqrt{N})$	$2N$

I. Scalability challenges and classical results
II. Asymptotic optimality and universality (no memory)
III. Reduction in communication overhead (memory)
IV. Heterogeneity issues and network scenarios

Performance versus Communication Overhead at a Glance

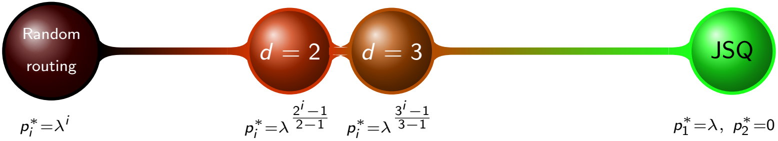
Random routing

$p_i^* = \lambda^i$

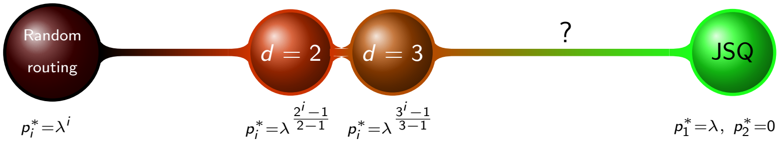
Performance versus Communication Overhead at a Glance

$p_i^* = \lambda^i$ $\quad$ $p_i^* = \lambda^{\frac{2^i - 1}{2-1}}$ $\quad$ $p_i^* = \lambda^{\frac{3^i - 1}{3-1}}$

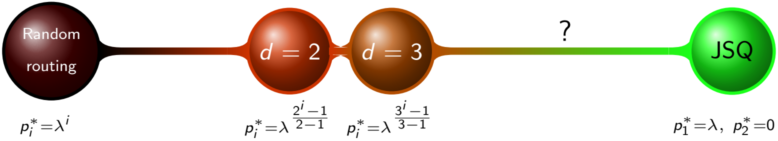
Performance versus Communication Overhead at a Glance



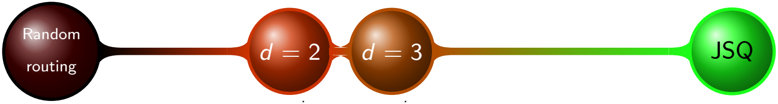
Performance versus Communication Overhead at a Glance



Performance versus Communication Overhead at a Glance



In order to examine performance versus communication trade-off, we allow parameter d to depend on N and write $d(N)$ to reflect that



$p_i^* = \lambda^i$ $\quad$ $p_i^* = \lambda^{\frac{2^i-1}{2-1}}$ $\quad$ $p_i^* = \lambda^{\frac{3^i-1}{3-1}}$ $\quad$ Fluid: $d(N) \gg 1, \; p_1^* = \lambda, \; p_2^* = 0$

In order to examine performance versus communication trade-off, we allow parameter d to depend on N and write $d(N)$ to reflect that

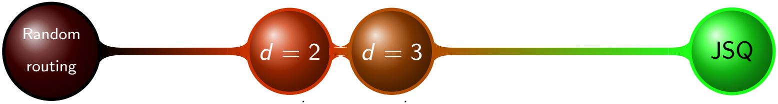
Universality and Asymptotic Optimality Properties

- JSQ($d(N)$) has same fluid limit as JSQ, and achieves fluid-level optimality when

$$d(N) \longrightarrow \infty$$

Performance versus Communication Overhead at a Glance

Diffusion: $d(N) \gg \sqrt{N} \log N$



$p_i^* = \lambda^i$ $\quad$ $p_i^* = \lambda^{\frac{2^i-1}{2-1}}$ $\quad$ $p_i^* = \lambda^{\frac{3^i-1}{3-1}}$

Fluid: $d(N) \gg 1, \; p_1^* = \lambda, \; p_2^* = 0$

In order to examine performance versus communication trade-off, we allow parameter d to depend on N and write $d(N)$ to reflect that

Universality and Asymptotic Optimality Properties

- JSQ($d(N)$) has same fluid limit as JSQ, and achieves fluid-level optimality when

$$d(N) \longrightarrow \infty$$

- JSQ($d(N)$) has same diffusion limit as JSQ, and achieves diffusion-level optimality when

$$\frac{d(N)}{\sqrt{N} \log N} \longrightarrow \infty$$

Universality of Fluid Limit for JSQ($d(N)$) Scheme

- Fluid-scaled state variables $q_i^{d(N)}(t) = Q_i^{d(N)}(t)/N$ represent fraction of servers with queue length i or larger in N^{th} system with JSQ($d(N)$) scheme

Universality of Fluid Limit for JSQ($d(N)$) Scheme

- Fluid-scaled state variables $q_i^{d(N)}(t) = Q_i^{d(N)}(t)/N$ represent fraction of servers with queue length i or larger in N^{th} system with JSQ($d(N)$) scheme

Universality of fluid limit for JSQ($d(N)$) [Mukherjee, B, Van L, W 2016]

If $\mathbf{q}^{d(N)}(0) \to \mathbf{q}^\infty$ and $d(N) \to \infty$ as $N \to \infty$, then $\{\mathbf{q}^{d(N)}(t)\}_{t\geq 0}$ has same weak limit $\{\mathbf{q}(t)\}_{t\geq 0}$ as $N \to \infty$ as ordinary JSQ policy, with

$$\frac{dq_i(t)}{dt} = \lambda p_{i-1}(\mathbf{q}(t)) - \mu[q_i(t) - q_{i+1}(t)],$$

where $\mathbf{q}(0) = \mathbf{q}^\infty$

Universality of Fluid Limit for JSQ($d(N)$) Scheme

- Fluid-scaled state variables $q_i^{d(N)}(t) = Q_i^{d(N)}(t)/N$ represent fraction of servers with queue length i or larger in N^{th} system with JSQ($d(N)$) scheme

Universality of fluid limit for JSQ($d(N)$) [Mukherjee, B, Van L, W 2016]

If $\mathbf{q}^{d(N)}(0) \to \mathbf{q}^{\infty}$ and $d(N) \to \infty$ as $N \to \infty$, then $\{\mathbf{q}^{d(N)}(t)\}_{t \geq 0}$ has same weak limit $\{\mathbf{q}(t)\}_{t \geq 0}$ as $N \to \infty$ as ordinary JSQ policy, with

$$\frac{dq_i(t)}{dt} = \lambda p_{i-1}(\mathbf{q}(t)) - \mu[q_i(t) - q_{i+1}(t)],$$

where $\mathbf{q}(0) = \mathbf{q}^{\infty}$

$p_{i-1}(\mathbf{q}(t))$ is (instantaneous) fraction of arriving tasks assigned to servers with queue length $i-1$ in fluid-level state $\mathbf{q}(t)$

Universality of Fluid Limit for JSQ($d(N)$) Scheme

Observations

- Fluid-level behavior coincides with that of ordinary JSQ policy as long as $d(N) \to \infty$ as $N \to \infty$

Universality of Fluid Limit for JSQ($d(N)$) Scheme

Observations

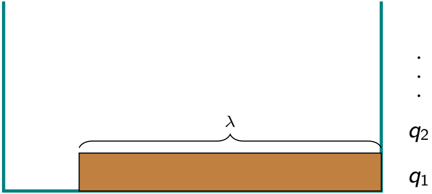
- Fluid-level behavior coincides with that of ordinary JSQ policy as long as $d(N) \to \infty$ as $N \to \infty$
- JSQ($d(N)$) scheme achieves fluid-level optimality while reducing communication overhead by nearly a factor O(N)

Universality of Fluid Limit for JSQ($d(N)$) Scheme

Observations

- Fluid-level behavior coincides with that of ordinary JSQ policy as long as $d(N) \to \infty$ as $N \to \infty$
- JSQ($d(N)$) scheme achieves fluid-level optimality while reducing communication overhead by nearly a factor O(N)
- Fixed point of fluid limit

$$q_1^\star = \lambda, \qquad q_i^\star = 0, \qquad i \geq 2$$



Universality of Diffusion Limit for JSQ($d(N)$) Scheme

- Fluid limit is rather crude, and diffusion limit provides more refined characterization when $\lambda \sim 1$

Universality of Diffusion Limit for JSQ($d(N)$) Scheme

- Fluid limit is rather crude, and diffusion limit provides more refined characterization when $\lambda \sim 1$
- Suppose total arrival rate $\lambda(N)$ satisfies $N - \lambda(N) \sim \beta\sqrt{N}$ as $N \to \infty$, so relative slack capacity is $\beta/\sqrt{N}$ (Halfin-Whitt regime)

Universality of Diffusion Limit for JSQ($d(N)$) Scheme

- Fluid limit is rather crude, and diffusion limit provides more refined characterization when $\lambda \sim 1$
- Suppose total arrival rate $\lambda(N)$ satisfies $N - \lambda(N) \sim \beta\sqrt{N}$ as $N \to \infty$, so relative slack capacity is $\beta/\sqrt{N}$ (Halfin-Whitt regime)
- Introduce diffusion-scaled state variables:

$$\bar{Q}_i^{d(N)}(t) = \begin{cases} \dfrac{Q_i^{d(N)-N}(t)}{\sqrt{N}} \text{ for } i = 1 \\[2ex] \dfrac{Q_i^{d(N)}(t)}{\sqrt{N}} \text{ for } i \geq 2 \end{cases}$$

Universality of Diffusion Limit for JSQ($d(N)$) Scheme

- Fluid limit is rather crude, and diffusion limit provides more refined characterization when $\lambda \sim 1$
- Suppose total arrival rate $\lambda(N)$ satisfies $N - \lambda(N) \sim \beta\sqrt{N}$ as $N \to \infty$, so relative slack capacity is $\beta/\sqrt{N}$ (Halfin-Whitt regime)
- Introduce diffusion-scaled state variables:

$$\bar{Q}_i^{d(N)}(t) = \begin{cases} \dfrac{Q_i^{d(N)-N}(t)}{\sqrt{N}} & \text{for } i = 1 \\ \\ \dfrac{Q_i^{d(N)}(t)}{\sqrt{N}} & \text{for } i \geq 2 \end{cases}$$

Universality of diffusion limit for JSQ($d(N)$) scheme [Mukherjee, B, Van L, W 2016]

Assume $d(N)/(\sqrt{N}\log(N)) \to \infty$ as $N \to \infty$. Then, under suitable initial conditions, $\{\bar{\mathbf{Q}}^{d(N)}(t)\}_{t\geq 0}$ has same weak limit $\{\bar{\mathbf{Q}}(t)\}_{t\geq 0}$ as $N \to \infty$ as ordinary JSQ policy as established by Eschenfeldt & Gamarnik 2015

Universality of Diffusion Limit for JSQ($d(N)$) Scheme

Observations

- Diffusion-level behavior coincides with that of ordinary JSQ policy provided $d(N)/(\sqrt{N}\log(N)) \to \infty$ as $N \to \infty$

Universality of Diffusion Limit for JSQ($d(N)$) Scheme

Observations

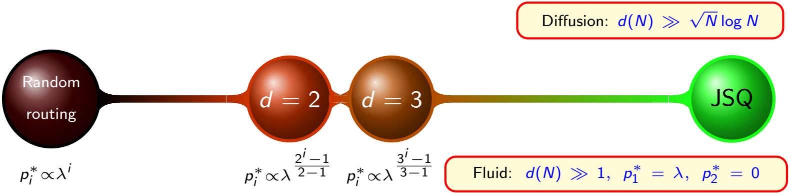
- Diffusion-level behavior coincides with that of ordinary JSQ policy provided $d(N)/(\sqrt{N}\log(N)) \to \infty$ as $N \to \infty$
- JSQ($d(N)$) scheme achieves diffusion-level optimality while reducing communication overhead by nearly a factor $O(\sqrt{N}/\log(N))$

Universality of Diffusion Limit for JSQ($d(N)$) Scheme

Observations

- Diffusion-level behavior coincides with that of ordinary JSQ policy provided $d(N)/(\sqrt{N}\log(N)) \to \infty$ as $N \to \infty$
- JSQ($d(N)$) scheme achieves diffusion-level optimality while reducing communication overhead by nearly a factor $O(\sqrt{N}/\log(N))$
- Latter condition is nearly necessary: if $d(N)/(\sqrt{N}\log(N)) \to 0$ as $N \to \infty$, then diffusion limit of JSQ($d(N)$) scheme differs from that of JSQ policy

High-Level Summary



Universality and Asymptotic Optimality Properties

- JSQ($d(N)$) has same fluid limit as JSQ, and achieves fluid-level optimality when
$$d(N) \longrightarrow \infty$$
- JSQ($d(N)$) has same diffusion limit as JSQ, and achieves diffusion-level optimality when
$$\frac{d(N)}{\sqrt{N} \log N} \longrightarrow \infty$$

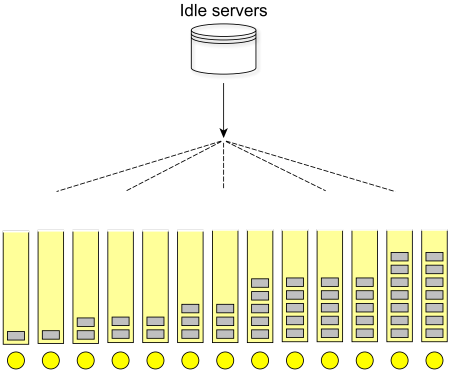
Performance versus Communication Overhead Trade-off

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)	Overhead
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	0
JSQ(d)	$p_i^* = \lambda^{\frac{d^i-1}{d-1}}$	$\Theta(1)$	$\Omega(\log N)$	$2d$
$d(N) \to \infty$	same as JSQ	same as JSQ	??	$2d(N)$
$\frac{d(N)}{\sqrt{N}\log(N)} \to \infty$	same as JSQ	same as JSQ	same as JSQ	$2d(N)$
JSQ	$p_1^* = \lambda$, $p_2^* = 0$	o(1)	$\Theta(1/\sqrt{N})$	$2N$

I. Scalability challenges and classical results
II. Asymptotic optimality and universality (no memory)
III. Reduction in communication overhead (memory)
IV. Heterogeneity issues and network scenarios

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

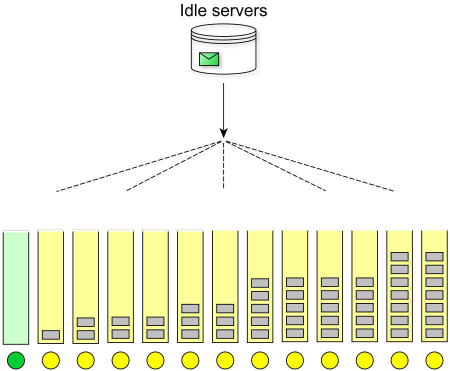
Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

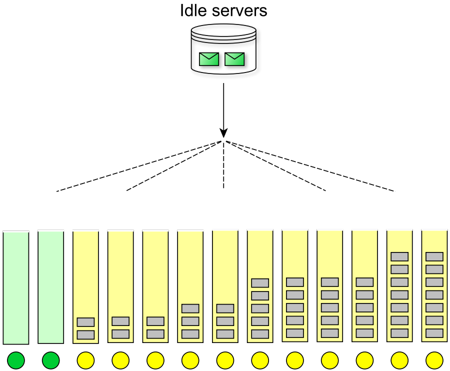
- When server becomes idle, it sends token to dispatcher to advertise its availability



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

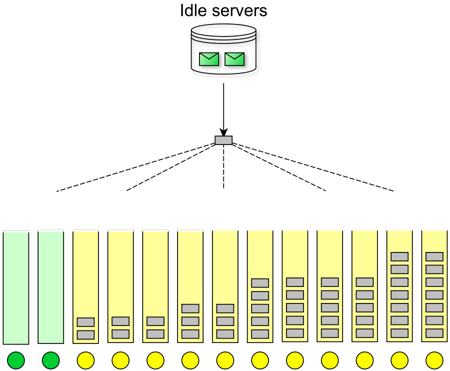
- When server becomes idle, it sends token to dispatcher to advertise its availability



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

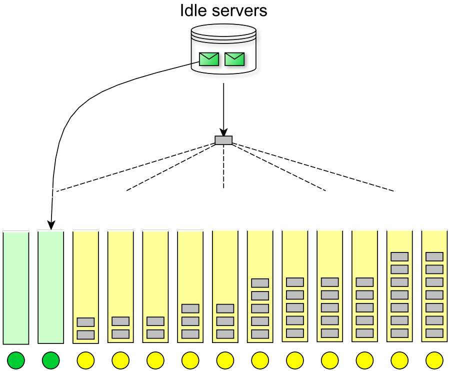
- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

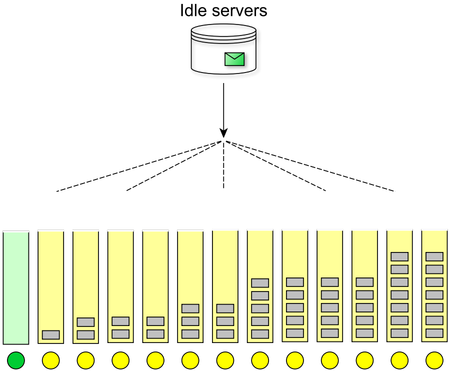
- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

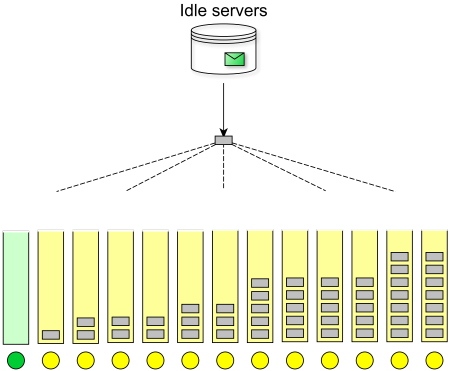
- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

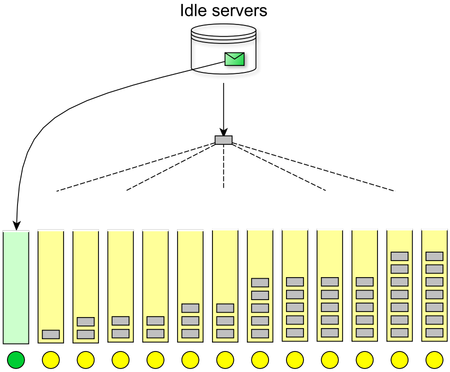
- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

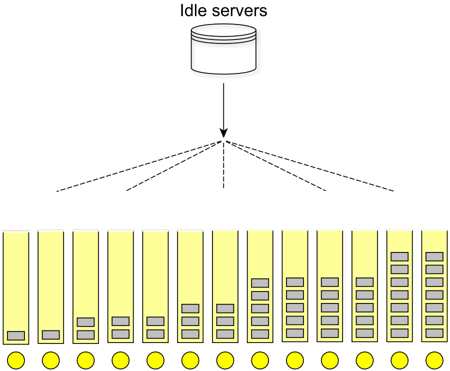
- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

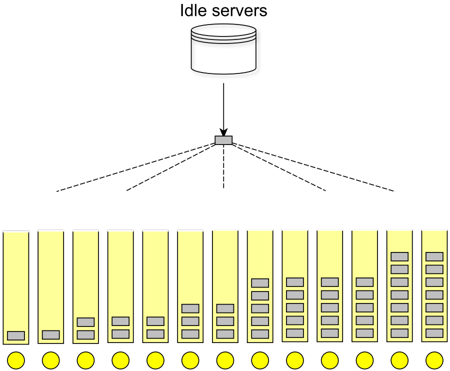
- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

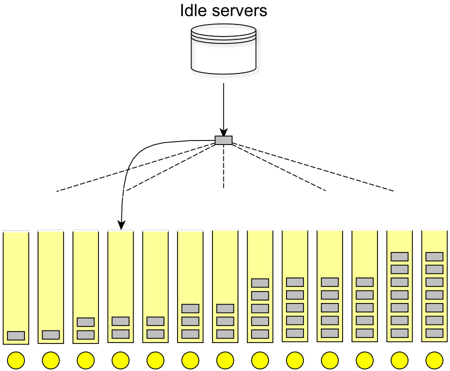
- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)
- When no tokens are available arriving task is forwarded to randomly selected server



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

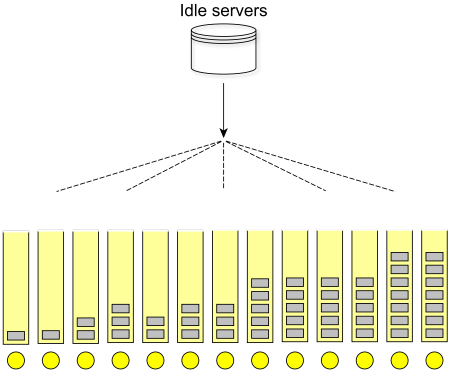
- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)
- When no tokens are available arriving task is forwarded to randomly selected server



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Server-driven rather than dispatcher-driven [Badonell & Burgess 2008, Lu *et al.* 2011]

- When server becomes idle, it sends token to dispatcher to advertise its availability
- When dispatcher has outstanding tokens at time of arrival, it assigns task to one of corresponding servers (and disposes of token)
- When no tokens are available arriving task is forwarded to randomly selected server

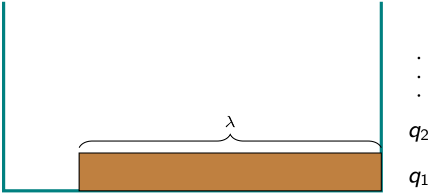


Server only sends token when service completion leaves its queue empty, implying that at most one token is generated per task

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Fixed point of fluid limit for JIQ strategy [Stolyar 2015]

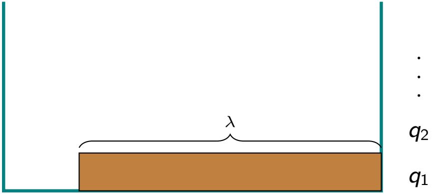
$$q_1^\star = \lambda, \qquad q_i^\star = 0, \qquad i \geq 2$$



Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Fixed point of fluid limit for JIQ strategy [Stolyar 2015]

$$q_1^\star = \lambda, \qquad q_i^\star = 0, \qquad i \geq 2$$



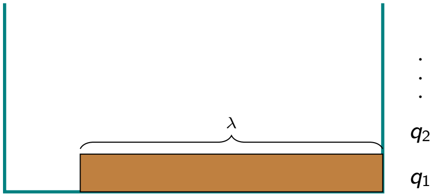
Observations

- Fluid-level behavior of JIQ strategy coincides with that of JSQ policy

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

Fixed point of fluid limit for JIQ strategy [Stolyar 2015]

$$q_1^\star = \lambda, \qquad q_i^\star = 0, \qquad i \geq 2$$



Observations

- Fluid-level behavior of JIQ strategy coincides with that of JSQ policy
- JIQ strategy achieves fluid-level optimality with O(1) communication overhead per task

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

In fact, JIQ strategy provides diffusion-level optimality as well with O(1) communication overhead per task

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

In fact, JIQ strategy provides diffusion-level optimality as well with O(1) communication overhead per task

Diffusion limit for JIQ strategy [Mukherjee, B, Van L, W 2016]

Under suitable initial conditions, $\{\bar{\mathbf{Q}}^{JIQ}(t)\}_{t\geq 0}$ has same weak limit $\{\bar{\mathbf{Q}}(t)\}_{t\geq 0}$ as $N \to \infty$ as ordinary JSQ policy

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

In fact, JIQ strategy provides diffusion-level optimality as well with O(1) communication overhead per task

Diffusion limit for JIQ strategy [Mukherjee, B, Van L, W 2016]

Under suitable initial conditions, $\{\bar{\mathbf{Q}}^{JIQ}(t)\}_{t\geq 0}$ has same weak limit $\{\bar{\mathbf{Q}}(t)\}_{t\geq 0}$ as $N \to \infty$ as ordinary JSQ policy

- JIQ strategy outperforms JSQ(d) schemes in terms of performance and communication overhead

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

In fact, JIQ strategy provides diffusion-level optimality as well with O(1) communication overhead per task

Diffusion limit for JIQ strategy [Mukherjee, B, Van L, W 2016]

Under suitable initial conditions, $\{\bar{\mathbf{Q}}^{JIQ}(t)\}_{t\geq 0}$ has same weak limit $\{\bar{\mathbf{Q}}(t)\}_{t\geq 0}$ as $N \to \infty$ as ordinary JSQ policy

- JIQ strategy outperforms JSQ(d) schemes in terms of performance and communication overhead
- For no single value of d, JSQ(d) scheme can offer both zero delay and constant overhead as $N \to \infty$

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

In fact, JIQ strategy provides diffusion-level optimality as well with O(1) communication overhead per task

Diffusion limit for JIQ strategy [Mukherjee, B, Van L, W 2016]

Under suitable initial conditions, $\{\bar{\mathbf{Q}}^{JIQ}(t)\}_{t\geq 0}$ has same weak limit $\{\bar{\mathbf{Q}}(t)\}_{t\geq 0}$ as $N \to \infty$ as ordinary JSQ policy

- JIQ strategy outperforms JSQ(d) schemes in terms of performance and communication overhead
- For no single value of d, JSQ(d) scheme can offer both zero delay and constant overhead as $N \to \infty$
- Note that JIQ strategy uses $O(N)$ amount of memory at dispatcher

Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead

In fact, JIQ strategy provides diffusion-level optimality as well with O(1) communication overhead per task

Diffusion limit for JIQ strategy [Mukherjee, B, Van L, W 2016]

Under suitable initial conditions, $\{\bar{\mathbf{Q}}^{JIQ}(t)\}_{t\geq 0}$ has same weak limit $\{\bar{\mathbf{Q}}(t)\}_{t\geq 0}$ as $N \to \infty$ as ordinary JSQ policy

- JIQ strategy outperforms JSQ(d) schemes in terms of performance and communication overhead
- For no single value of d, JSQ(d) scheme can offer both zero delay and constant overhead as $N \to \infty$
- Note that JIQ strategy uses $O(N)$ amount of memory at dispatcher
- In order for zero delay to be achievable [Gamarnik-Tsitsiklis-Zubeldia 2016]
  - either overhead per task $\to \infty$ as $N \to \infty$, so overhead per time unit $\gg N$
  - or amount of memory at dispatcher $\to \infty$ as $N \to \infty$

Performance versus Communication Overhead Trade-off

Scheme	Queue length	Waiting time (fixed $\lambda < 1$)	Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)	Overhead
Random	$p_i^N = \lambda^i$	$\frac{\lambda}{1-\lambda}$	$\Theta(\sqrt{N})$	0
JSQ(d)	$p_i^* = \lambda^{\frac{d^i-1}{d-1}}$	$\Theta(1)$	$\Omega(\log N)$	$2d$
$d(N) \to \infty$	same as JSQ	same as JSQ	??	$2d(N)$
$\frac{d(N)}{\sqrt{N}\log(N)} \to \infty$	same as JSQ	same as JSQ	same as JSQ	$2d(N)$
JSQ	$p_1^* = \lambda$, $p_2^* = 0$	o(1)	$\Theta(1/\sqrt{N})$	$2N$
JIQ	same as JSQ	same as JSQ	same as JSQ	≤ 1

Extensions

- Multiple dispatchers
- Joint auto-scaling and load balancing:
  TABS scheme simultaneously achieves zero delay and zero relative energy wastage in many-server regime

Hyper-Scalable JSQ

Each server provides periodic queue status updates to dispatcher [Van der Boor, B, Van Leeuwaarden 2018]

- Either in synchronized or asynchronous manner
- At either exponentially distributed or constant time intervals with (mean) duration T and frequency $\delta = 1/T$, so overhead per task is δ/λ

Hyper-Scalable JSQ

Each server provides periodic queue status updates to dispatcher [Van der Boor, B, Van Leeuwaarden 2018]

- Either in synchronized or asynchronous manner
- At either exponentially distributed or constant time intervals with (mean) duration T and frequency $\delta = 1/T$, so overhead per task is δ/λ
- SUJSQ$^{\mathrm{d}et}$: Synchronized Constant Update Intervals

Hyper-Scalable JSQ

Each server provides periodic queue status updates to dispatcher [Van der Boor, B, Van Leeuwaarden 2018]

- Either in synchronized or asynchronous manner
- At either exponentially distributed or constant time intervals with (mean) duration T and frequency $\delta = 1/T$, so overhead per task is δ/λ
- SUJSQ$^{\mathrm{d}et}$: Synchronized Constant Update Intervals
- AUJSQ$^{\mathrm{ex}p}$: Asynchronous Exponentially Distributed Update Intervals

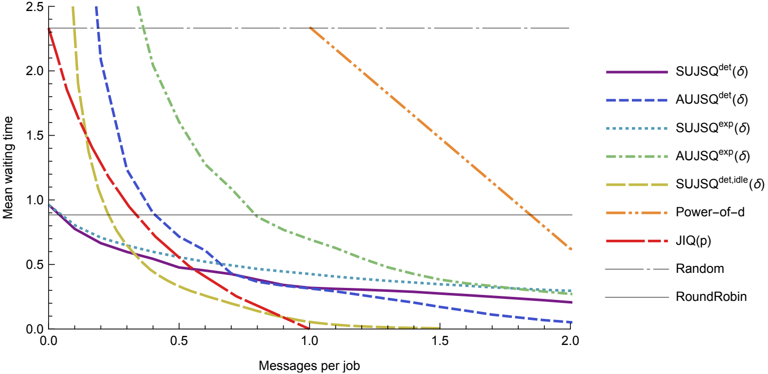
Hyper-Scalable JSQ

Each server provides periodic queue status updates to dispatcher [Van der Boor, B, Van Leeuwaarden 2018]

- Either in synchronized or asynchronous manner
- At either exponentially distributed or constant time intervals with (mean) duration T and frequency $\delta = 1/T$, so overhead per task is δ/λ
- SUJSQ$^{\mathrm{d}et}$: Synchronized Constant Update Intervals
- AUJSQ$^{\mathrm{ex}p}$: Asynchronous Exponentially Distributed Update Intervals

When task arrives, dispatcher assigns it server with minimum value of queue estimate (reset at updates, and incremented for each assigned task)

Hyper-Scalable JSQ



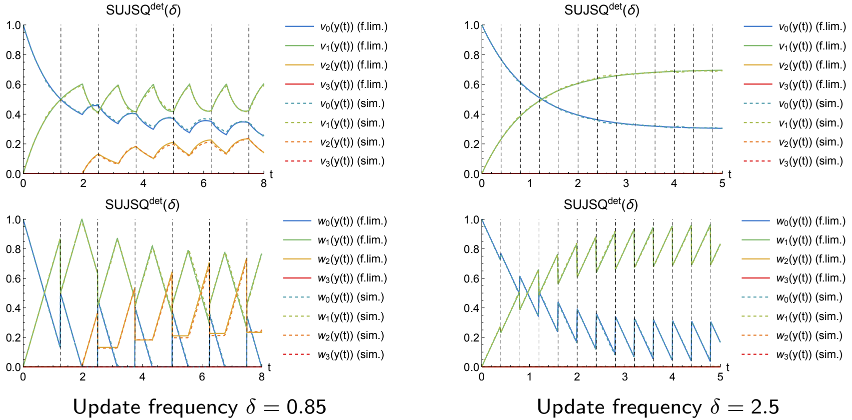
Fluid Limits for Hyper-Scalable JSQ

- $Y^N_{i,j}(t)$: number of servers with actual queue length i and queue estimate $j \geq i$ at time t

Fluid Limits for Hyper-Scalable JSQ

- $Y^N_{i,j}(t)$: number of servers with actual queue length i and queue estimate $j \geq i$ at time t
- $y_{i,j}(t) = \lim_{N\to\infty} Y^N_{i,j}(t)/N$: fluid-limit variables
- $v_i(t) = \sum_{j=i}^{\infty} y_{i,j}(t)$: fraction of servers with queue length i
- $w_j(t) = \sum_{i=0}^{j} y_{i,j}(t)$: fraction of servers with queue estimate j

SUJSQ^{det}: Synchronized Constant Update Intervals



SUJSQ^{det}: Synchronized Constant Update Intervals

Zero-delay threshold SUJSQ$^{det}(\delta)$

If update frequency $\delta > \frac{\lambda}{1-\lambda}$, i.e., at least $\frac{1}{1-\lambda}$ messages per task, then no queues and zero delay in stationarity at fluid level

SUJSQ^{det}: Synchronized Constant Update Intervals

Zero-delay threshold $\text{SUJSQ}^{det}(\delta)$

If update frequency $\delta > \frac{\lambda}{1-\lambda}$, i.e., at least $\frac{1}{1-\lambda}$ messages per task, then no queues and zero delay in stationarity at fluid level

Finite maximum queue length $\text{SUJSQ}^{det}(1/T)$

For any $\lambda < 1$, maximum queue length in stationarity at fluid level is bounded from above by

$$M(\lambda, T) = \max\{L : \lambda T \geq H(L; \lambda, T)\} < \infty,$$

with

$$H(L; \lambda, T) = \left(1 - \frac{\lambda T}{L}\right) \left[\sum_{l=0}^{L} l \mathrm{e}^{-T} \frac{T^l}{l!} + L \sum_{l=L+1}^{\infty} \mathrm{e}^{-T} \frac{T^l}{l!}\right]$$

SUJSQ^{det}: Synchronized Constant Update Intervals

Zero-delay threshold SUJSQ$^{\mathrm{det}}(\delta)$

If update frequency $\delta > \frac{\lambda}{1-\lambda}$, i.e., at least $\frac{1}{1-\lambda}$ messages per task, then no queues and zero delay in stationarity at fluid level

Finite maximum queue length SUJSQ$^{\mathrm{det}}(1/T)$

For any $\lambda < 1$, maximum queue length in stationarity at fluid level is bounded from above by

$$M(\lambda, T) = \max\{L : \lambda T \geq H(L; \lambda, T)\} < \infty,$$

with

$$H(L; \lambda, T) = \left(1 - \frac{\lambda T}{L}\right) \left[\sum_{l=0}^{L} l \mathrm{e}^{-T} \frac{T^l}{l!} + L \sum_{l=L+1}^{\infty} \mathrm{e}^{-T} \frac{T^l}{l!}\right]$$

- Upper bound $M(\lambda, T) \to \infty$ as $T \to \infty$

SUJSQ^{det}: Synchronized Constant Update Intervals

Zero-delay threshold SUJSQ$^{\mathrm{det}}(\delta)$

If update frequency $\delta > \frac{\lambda}{1-\lambda}$, i.e., at least $\frac{1}{1-\lambda}$ messages per task, then no queues and zero delay in stationarity at fluid level

Finite maximum queue length SUJSQ$^{\mathrm{det}}(1/T)$

For any $\lambda < 1$, maximum queue length in stationarity at fluid level is bounded from above by

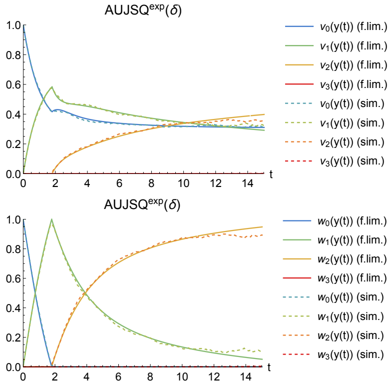
$$M(\lambda, T) = \max\{L : \lambda T \geq H(L; \lambda, T)\} < \infty,$$

with

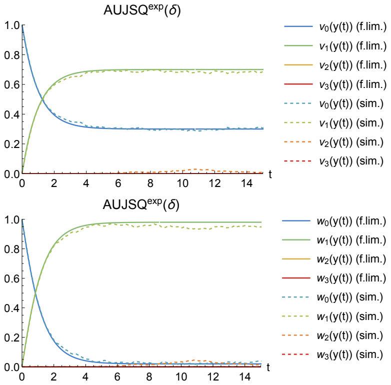
$$H(L; \lambda, T) = \left(1 - \frac{\lambda T}{L}\right) \left[\sum_{l=0}^{L} l \mathrm{e}^{-T} \frac{T^l}{l!} + L \sum_{l=L+1}^{\infty} \mathrm{e}^{-T} \frac{T^l}{l!}\right]$$

- Upper bound $M(\lambda, T) \to \infty$ as $T \to \infty$
- SUJSQ$^{\mathrm{det}}(1/T)$ reduces to Round Robin policy as $T \to \infty$, and each individual server behaves as D/M/1 queue

AUJSQ^{exp}: Asynchronous Exponentially Distributed Update Intervals



Update frequency $\delta = 0.85$



Update frequency $\delta = 2.5$

AUJSQ^{exp}: Asynchronous Exponentially Distributed Update Intervals

Zero-delay threshold AUJSQ$^{exp}(\delta)$

If update frequency $\delta > \frac{\lambda}{1-\lambda}$, i.e., at least $\frac{1}{1-\lambda}$ messages per task, then no queues and zero delay in stationarity at fluid level

AUJSQ^{exp}: Asynchronous Exponentially Distributed Update Intervals

Zero-delay threshold AUJSQ$^{exp}(\delta)$

If update frequency $\delta > \frac{\lambda}{1-\lambda}$, i.e., at least $\frac{1}{1-\lambda}$ messages per task, then no queues and zero delay in stationarity at fluid level

Finite maximum queue length AUJSQ$^{exp}(\delta)$

For any $\lambda < 1$, maximum queue length in stationarity at fluid level is

$$M(\lambda, \delta) = \left\lfloor -\frac{\log(1-\lambda)}{\log(1+\delta)} \right\rfloor < \infty$$

AUJSQ^{exp}: Asynchronous Exponentially Distributed Update Intervals

Zero-delay threshold AUJSQ$^{exp}(\delta)$

If update frequency $\delta > \frac{\lambda}{1-\lambda}$, i.e., at least $\frac{1}{1-\lambda}$ messages per task, then no queues and zero delay in stationarity at fluid level

Finite maximum queue length AUJSQ$^{exp}(\delta)$

For any $\lambda < 1$, maximum queue length in stationarity at fluid level is

$$M(\lambda, \delta) = \left\lfloor -\frac{\log(1-\lambda)}{\log(1+\delta)} \right\rfloor < \infty$$

- Maximum queue length $M(\lambda, \delta) \to \infty$ as $\delta \downarrow 0$

AUJSQ^{exp}: Asynchronous Exponentially Distributed Update Intervals

Zero-delay threshold AUJSQ$^{exp}(\delta)$

If update frequency $\delta > \frac{\lambda}{1-\lambda}$, i.e., at least $\frac{1}{1-\lambda}$ messages per task, then no queues and zero delay in stationarity at fluid level

Finite maximum queue length AUJSQ$^{exp}(\delta)$

For any $\lambda < 1$, maximum queue length in stationarity at fluid level is

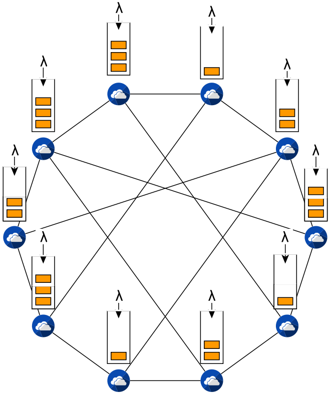
$$M(\lambda, \delta) = \left\lfloor -\frac{\log(1-\lambda)}{\log(1+\delta)} \right\rfloor < \infty$$

- Maximum queue length $M(\lambda, \delta) \to \infty$ as $\delta \downarrow 0$
- Queue length at individual server under SUJSQ$^{\mathrm{det}}(1/T)$ wildly oscillates as $\delta \downarrow 0$,
  rising to $M(\lambda, \delta)$ shortly after each update and then gradually dropping to low value before next update

I. Scalability challenges and classical results
II. Asymptotic optimality and universality (no memory)
III. Reduction in communication overhead (memory)
IV. Heterogeneity issues and network scenarios

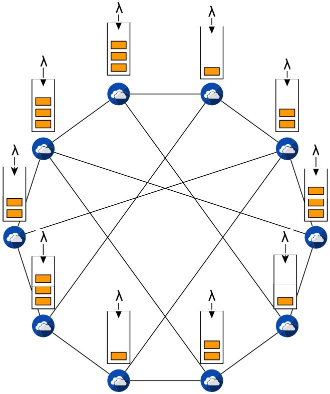
Load Balancing in Network Scenarios

- Suppose N servers are interconnected by underlying graph topology G_N



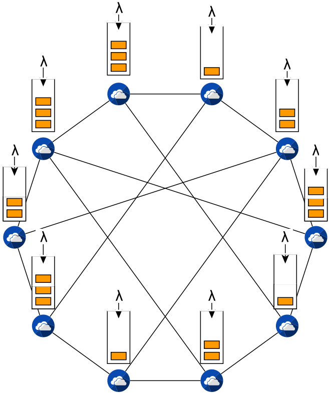
Load Balancing in Network Scenarios

- Suppose N servers are interconnected by underlying graph topology G_N
- Tasks arrive at each server as Poisson process of rate λ and can either be executed locally or forwarded to neighbor in G_N with shorter queue, if any



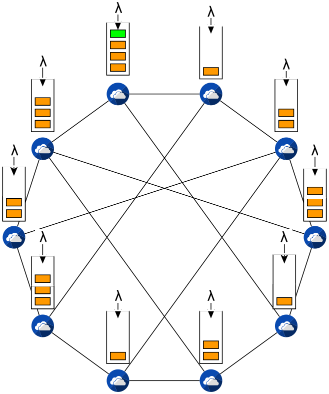
Load Balancing in Network Scenarios

- Suppose N servers are interconnected by underlying graph topology G_N
- Tasks arrive at each server as Poisson process of rate λ and can either be executed locally or forwarded to neighbor in G_N with shorter queue, if any
- Data 'locality', server-task affinity relations, compatibility constraints



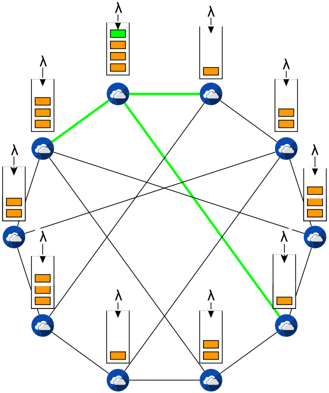
Load Balancing in Network Scenarios

- Suppose N servers are interconnected by underlying graph topology G_N
- Tasks arrive at each server as Poisson process of rate λ and can either be executed locally or forwarded to neighbor in G_N with shorter queue, if any
- Data 'locality', server-task affinity relations, compatibility constraints



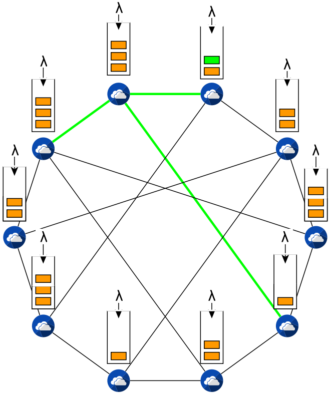
Load Balancing in Network Scenarios

- Suppose N servers are interconnected by underlying graph topology G_N
- Tasks arrive at each server as Poisson process of rate λ and can either be executed locally or forwarded to neighbor in G_N with shorter queue, if any
- Data 'locality', server-task affinity relations, compatibility constraints



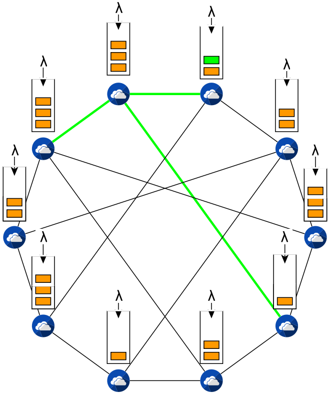
Load Balancing in Network Scenarios

- Suppose N servers are interconnected by underlying graph topology G_N
- Tasks arrive at each server as Poisson process of rate λ and can either be executed locally or forwarded to neighbor in G_N with shorter queue, if any
- Data 'locality', server-task affinity relations, compatibility constraints



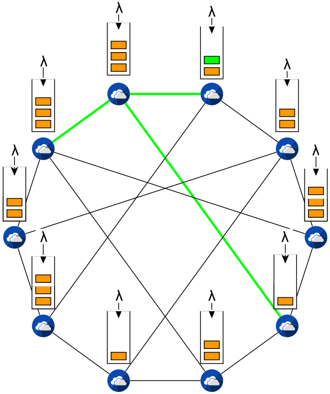
Load Balancing in Network Scenarios

- Suppose N servers are interconnected by underlying graph topology G_N
- Tasks arrive at each server as Poisson process of rate λ and can either be executed locally or forwarded to neighbor in G_N with shorter queue, if any
- Data 'locality', server-task affinity relations, compatibility constraints
- How does graph structure affect performance?

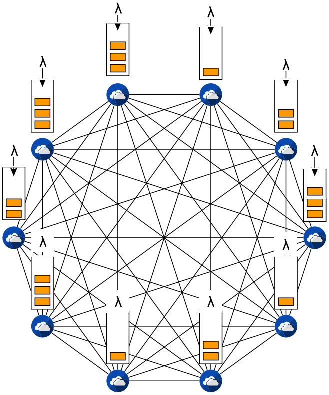


Load Balancing in Network Scenarios

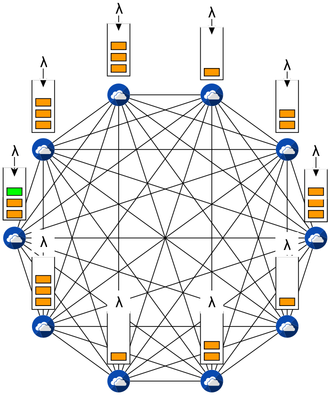
- Suppose N servers are interconnected by underlying graph topology G_N
- Tasks arrive at each server as Poisson process of rate λ and can either be executed locally or forwarded to neighbor in G_N with shorter queue, if any
- Data 'locality', server-task affinity relations, compatibility constraints
- How does graph structure affect performance?
- Lack of exchangeability among servers breaks underpinning for stochastic coupling and fluid and diffusion limits



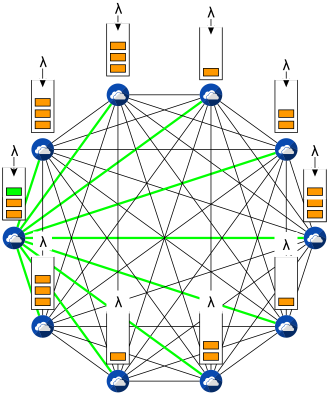
Case of complete graph (clique) corresponds to ordinary supermarket model



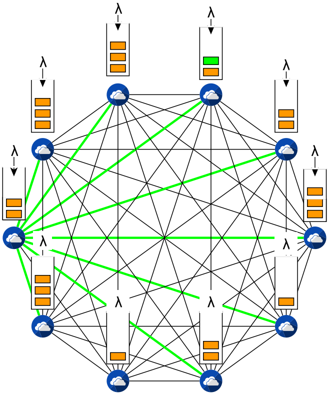
Case of complete graph (clique) corresponds to ordinary supermarket model



Case of complete graph (clique) corresponds to ordinary supermarket model

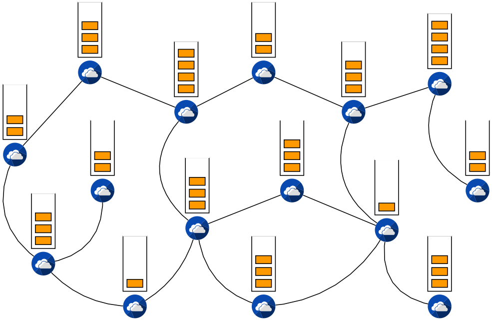


Case of complete graph (clique) corresponds to ordinary supermarket model



Our Perspective: Asymptotic Optimality and Universality

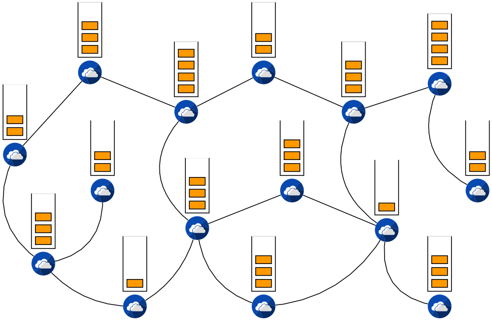
How much connectivity is required in order for JSQ to achieve asymptotically similar performance in G_N as in complete graph as $N \to \infty$?



Our Perspective: Asymptotic Optimality and Universality

How much connectivity is required in order for JSQ to achieve asymptotically similar performance in G_N as in complete graph as $N \to \infty$?

Slightly different criterion: what degree of connectivity is required for JSQ(d) to yield asymptotically similar performance in G_N as in complete graph



JSQ on Deterministic Graphs: Informal Statements

Let $\{G_N\}_{N\geq 1}$ be sequence of graphs

Condition 1

Neighborhood size of any $\Theta(N)$ vertices is $N - o(N)$

Condition 2

Neighborhood size of any $\Theta(\sqrt{N})$ vertices is $N - o(\sqrt{N})$

JSQ on Deterministic Graphs: Informal Statements

Let $\{G_N\}_{N\geq 1}$ be sequence of graphs

Condition 1

Neighborhood size of any $\Theta(N)$ vertices is $N - o(N)$

Condition 2

Neighborhood size of any $\Theta(\sqrt{N})$ vertices is $N - o(\sqrt{N})$

Graph sequence $\{G_N\}_{N\geq 1}$ is said to be fluid-optimal or diffusion-optimal if JSQ in this graph sequence yields same fluid limit and diffusion limit as in classical setup, respectively

JSQ on Deterministic Graphs: Informal Statements

Let $\{G_N\}_{N\geq 1}$ be sequence of graphs

Condition 1

Neighborhood size of any $\Theta(N)$ vertices is $N - o(N)$

Condition 2

Neighborhood size of any $\Theta(\sqrt{N})$ vertices is $N - o(\sqrt{N})$

Graph sequence $\{G_N\}_{N\geq 1}$ is said to be fluid-optimal or diffusion-optimal if JSQ in this graph sequence yields same fluid limit and diffusion limit as in classical setup, respectively

Theorem JSQ on deterministic graphs

Graph sequence $\{G_N\}_{N\geq 1}$ is

(i) fluid-optimal if Condition 1 is satisfied

(ii) diffusion-optimal if Condition 2 is satisfied

Theorem JSQ on random graphs

Sequence of (Erdős-Rényi or random regular) graphs with avg degree $c(N)$ is

- fluid-optimal if $c(N) \to \infty$
- diffusion-optimal if $c(N)/(\sqrt{N} \log N) \to \infty$

Theorem Worst-case scenario

For any graph sequence $\{G_N\}_{N\geq 1}$, if

- $d_{\min}(G_N)/N \to 1$, then sequence must be fluid-optimal
- $d_{\min}(G_N)/N \to c < 1/2$, then sequence may not be fluid-optimal
- $\#$ bounded degree vertices is $\Theta(N)$, then sequence is not fluid-optimal

Conclusion

- Much sparser graphs can asymptotically match optimal performance of complete graph, provided they are suitably random
- In worst-case scenario, performance can be sub-optimal even when graph is sufficiently dense

Some References

M. van der Boor, S.C. Borst, J.S.H. van Leeuwaarden, D. Mukherjee (2018). Scalable load balancing in networked systems: Universality properties and stochastic coupling methods In: *Proc. ICM 2018*, Rio de Janeiro.

M. van der Boor, S.C. Borst, J.S.H. van Leeuwaarden, D. Mukherjee (2018). Scalable load balancing in networked systems: A survey of recent advances.

D. Mukherjee (2018). Scalable load balancing in networked systems. PhD thesis, Eindhoven University of Technology.

Thank You for your Attention!!