Scalable Load Balancing in Networked Systems

Sem Borst

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

ICM 2018, Rio de Janeiro, 3 August 2018

Based on joint work with Mark van der Boor, Johan van Leeuwaarden, Debankur Mukherjee & Phil Whiting
Load Balancing/Routing in Parallel-Server Systems

Single dispatcher

N Servers
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
Purely Random Assignment

Poisson($N\lambda$)

Service rates: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

$N$ Servers
**Purely Random Assignment**

Assign each task to server selected uniformly at random

$\text{Poisson}(N \lambda)$

Probabilities:

\[
\frac{1}{N}, \frac{1}{N}, \frac{1}{N}, \frac{1}{N}, \frac{1}{N}, \frac{1}{N}, \frac{1}{N}, \frac{1}{N}, \frac{1}{N}, \frac{1}{N}
\]

Service rates:

\[\mu, \mu, \mu, \mu, \mu, \mu, \mu, \mu, \mu, \mu\]

$N$ Servers
Purely Random Assignment

Assign each task to server selected uniformly at random

Poisson($N\lambda$)

Arrival rates:

Service rates:

$N$ Servers
PURELY RANDOM ASSIGNMENT

Assign each task to server selected uniformly at random

Poisson($N\lambda$)

Arrival rates: $\lambda$ $\lambda$ $\lambda$ $\lambda$ $\lambda$ $\lambda$ $\lambda$ $\lambda$ $\lambda$ $\lambda$

Service rates: $\mu$ $\mu$ $\mu$ $\mu$ $\mu$ $\mu$ $\mu$ $\mu$ $\mu$ $\mu$

$N$ Servers

$N$ independent M/M/1 queues with arrival rate $\lambda$ and service rate $\mu$
Load Balancing Scenarios

- Separate Queues
- Strictly Random Routing

\[ N \times \frac{\lambda}{\mu} \]
- Queue length

\[ \frac{\lambda \mu}{\mu - \lambda} \]
- Waiting time

Centralized Queue
- Complete Resource Pooling

\[ N \times \frac{\lambda}{\mu} \]
- Queue length

\[ \frac{\lambda \mu}{\mu - \lambda} \]
- Waiting time
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

\(N \times M/M/1\)

\[\begin{align*}
\text{queue length} &= N \times \frac{\lambda^2}{\mu(\mu - \lambda)} \\
\text{waiting time} &= \frac{\lambda}{\mu(\mu - \lambda)}
\end{align*}\]

Centralized Queue
Complete Resource Pooling

\(M/M/N\)

\[\begin{align*}
\text{queue length} &= \Pi W \frac{\lambda}{\mu - \lambda} \\
\text{waiting time} &= \frac{1}{N} \Pi W \frac{1}{\mu - \lambda}
\end{align*}\]
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)
Classical natural policy: Join the Shortest Queue (JSQ)
JOIN THE SHORTEST QUEUE (JSQ) POLICY

Classical natural policy: Join the Shortest Queue (JSQ)
Classical natural policy: Join the Shortest Queue (JSQ)
**Join the Shortest Queue (JSQ) Policy**

Classical natural policy: Join the Shortest Queue (JSQ)

\[ N \lambda \]

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\mu & \mu & \mu & \mu & \mu & \mu & \mu & \mu & \mu & \mu \\
\end{array}
\]
Symmetry among servers allows equivalent state description
Symmetry among servers allows equivalent state description

- Denote by $X_k$ queue length at server $k$, $k = 1, 2, \ldots, N$
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
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
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
JSQ stochastically minimizes aggregate size of \( l \) right-most stacks, i.e., total number of tasks in \( l \) largest queues

\[
\sum_{k=N-l+1}^{N} X_{(k)}^{\text{JSQ}} \leq_{st} \sum_{k=N-l+1}^{N} X_{(k)}^{\Pi}
\]

for all \( l = 1, \ldots, N \), for any non-anticipating policy \( \Pi \)

[Towsley-Sparagis-Cassandras 1992; Sparagis-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

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

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

[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

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

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

[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

$$\sum_{k=N-l+1}^{N} X^{\text{JSQ}}_{(k)} \leq_{st} \sum_{k=N-l+1}^{N} X^{\Pi}_{(k)}$$

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

[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

$$\sum_{i=j}^{\infty} Q^{\text{JSQ}}_{i} \leq_{st} \sum_{i=j}^{\infty} Q^{\Pi}_{i}$$

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

[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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p_i^N = \lambda^i$</td>
</tr>
<tr>
<td>JSQ</td>
<td>$p_1^N \to \lambda$, $p_2^N \to 0$</td>
</tr>
</tbody>
</table>

$p_i^N$ 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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Waiting time (fixed $\lambda &lt; 1$)</th>
<th>Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p_i^N = \lambda^i$</td>
<td>$\frac{\lambda}{1-\lambda}$</td>
<td>$\Theta(\sqrt{N})$</td>
</tr>
<tr>
<td>JSQ</td>
<td>$p_1^N \to \lambda$, $p_2^N \to 0$</td>
<td>$o(1)$</td>
<td>$\Theta(1/\sqrt{N})$</td>
</tr>
</tbody>
</table>

$p_i^N$ 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...

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Waiting time (fixed $\lambda &lt; 1$)</th>
<th>Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p_i^N = \lambda^i$</td>
<td>$\frac{\lambda}{1-\lambda}$</td>
<td>$\Theta(\sqrt{N})$</td>
</tr>
<tr>
<td>JSQ</td>
<td>$p_1^N \to \lambda$, $p_2^N \to 0$</td>
<td>$o(1)$</td>
<td>$\Theta(1/\sqrt{N})$</td>
</tr>
</tbody>
</table>

$p_i^N$ 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 $N \to \infty$

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Waiting time (fixed $\lambda &lt; 1$)</th>
<th>Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p_i^N = \lambda^i$</td>
<td>$\frac{\lambda}{1-\lambda}$</td>
<td>$\Theta(\sqrt{N})$</td>
<td>0</td>
</tr>
<tr>
<td>JSQ</td>
<td>$p_1^N \to \lambda$, $p_2^N \to 0$</td>
<td>$o(1)$</td>
<td>$\Theta(1/\sqrt{N})$</td>
<td>2$N$</td>
</tr>
</tbody>
</table>

$p_i^N$ 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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Waiting time (fixed $\lambda &lt; 1$)</th>
<th>Waiting time $(1 - \lambda \sim 1/\sqrt{N})$</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p_i^N = \lambda^i$</td>
<td>$\frac{\lambda}{1-\lambda}$</td>
<td>$\Theta(\sqrt{N})$</td>
<td>0</td>
</tr>
<tr>
<td>JSQ</td>
<td>$p_1^N \to \lambda$, $p_2^N \to 0$</td>
<td>$o(1)$</td>
<td>$\Theta(1/\sqrt{N})$</td>
<td>$2N$</td>
</tr>
</tbody>
</table>

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

$$
\bar{Q}_i^{(N)}(t) = \begin{cases} 
Q_i^{(N)}(t) - N & \text{for } i = 1 \\
\frac{Q_i^{(N)}(t)}{\sqrt{N}} & \text{for } i \geq 2
\end{cases}
$$
Diffusion limit for JSQ policy [Eschenfeldt & Gamarnik 2015]

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

$$\bar{Q}_1(t) = \bar{Q}_1(0) + \sqrt{2}W(t) - \beta t + \int_0^t (-\bar{Q}_1(s) + \bar{Q}_2(s))ds - U_1(t)$$

$$\bar{Q}_2(t) = \bar{Q}_2(0) + U_1(t) - \int_0^t \bar{Q}_2(s)ds$$

for $t \geq 0$ where $W$ is standard Brownian motion and $U_1$ is unique non-decreasing non-negative process in $D$ satisfying $\int_0^\infty \mathbb{1}_{[\bar{Q}_1(t) < 0]} dU_1(t) = 0$
Diffusion limit for JSQ policy [Eschenfeldt & Gamarnik 2015]

Under suitable initial conditions, \( \{ \tilde{Q}^{JSQ}(t) \}_{t \geq 0} \) has weak limit \( \{ \tilde{Q}(t) \}_{t \geq 0} \) as \( N \to \infty \), where

\[
\tilde{Q}_1(t) = \tilde{Q}_1(0) + \sqrt{2} W(t) - \beta t + \int_0^t ( - \tilde{Q}_1(s) + \tilde{Q}_2(s) ) ds - U_1(t)
\]

\[
\tilde{Q}_2(t) = \tilde{Q}_2(0) + U_1(t) - \int_0^t \tilde{Q}_2(s) ds
\]

for \( t \geq 0 \) where \( W \) is standard Brownian motion and \( U_1 \) is unique non-decreasing non-negative process in \( D \) satisfying \( \int_0^\infty \mathbb{1}_{[\tilde{Q}_1(t) < 0]} dU_1(t) = 0 \)

Interchange of limits established in [Braverman 2018]
**Diffusion Limit for JSQ Policy**

Diffusion limit for JSQ policy [Eschenfeldt & Gamarnik 2015]

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

$$\bar{Q}_1(t) = \bar{Q}_1(0) + \sqrt{2}W(t) - \beta t + \int_0^t (-\bar{Q}_1(s) + \bar{Q}_2(s))ds - U_1(t)$$

$$\bar{Q}_2(t) = \bar{Q}_2(0) + U_1(t) - \int_0^t \bar{Q}_2(s)ds$$

for $t \geq 0$ where $W$ is standard Brownian motion and $U_1$ is unique non-decreasing non-negative process in $D$ satisfying $\int_0^\infty 1_{[\bar{Q}_1(t)<0]}dU_1(t) = 0$

Interchange of limits established in [Braverman 2018]

Steady-state properties of diffusion limit characterized in [Banerjee & Mukherjee 2018]

- Tail asymptotics for any $\beta > 0$ (Gaussian for $\bar{Q}_1$ and exponential for $\bar{Q}_2$)
- Behavior for sufficiently small and large $\beta$
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.
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.
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.
Fluid Limit for JSQ($d$) Scheme

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

If $q^{JSQ(d)}(0) \to q^\infty$ as $N \to \infty$, then $\{q^{JSQ(d)}(t)\}_{t \geq 0}$ weakly converges to $\{q(t)\}_{t \geq 0}$ 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 $q(0) = q^\infty$
Fluid Limit for JSQ(d) Scheme

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

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

$[(q_{i-1}(t))^d - (q_i(t))^d]$ is (instantaneous) fraction of arriving tasks assigned to servers with queue length $i - 1$ in fluid-level state $q(t)$
Assuming $\mu = 1$ as before, fixed point of fluid limit is

$$q_i^* := \lim_{t \to \infty} q_i(t) = \lambda \frac{d^i - 1}{d - 1}, \quad i = 1, 2, \ldots$$
Power-of-$d$ Effect for JSQ($d$) Scheme

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

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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p_i^N = \lambda^i$</td>
<td>0</td>
</tr>
<tr>
<td>JSQ($d$)</td>
<td>$p_i^* = \lambda^{\frac{d^i-1}{d-1}}$</td>
<td>$2d$</td>
</tr>
<tr>
<td>JSQ</td>
<td>$p_1^* = \lambda$, $p_2^* = 0$</td>
<td>$2N$</td>
</tr>
</tbody>
</table>
JSQ\((d)\) Provides Strong Benefits over Random Assignment

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

But...

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>(p_i^N = \lambda^i)</td>
<td>0</td>
</tr>
<tr>
<td>JSQ((d))</td>
<td>(p_i^* = \lambda^{d^i-1/(d-1)})</td>
<td>2d</td>
</tr>
<tr>
<td>JSQ</td>
<td>(p_1^* = \lambda, \quad p_2^* = 0)</td>
<td>2N</td>
</tr>
</tbody>
</table>
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 $d$

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Waiting time (fixed $\lambda &lt; 1$)</th>
<th>Waiting time ($1 - \lambda \sim 1/\sqrt{N}$)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p_i^N = \lambda^i$</td>
<td>$\frac{\lambda}{1-\lambda}$</td>
<td>$\Theta(\sqrt{N})$</td>
<td>0</td>
</tr>
<tr>
<td>JSQ($d$)</td>
<td>$p_i^* = \lambda^{\frac{d^i-1}{d-1}}$</td>
<td>$\Theta(1)$</td>
<td>$\Omega(\log N)$</td>
<td>$2d$</td>
</tr>
<tr>
<td>JSQ</td>
<td>$p_1^* = \lambda$, $p_2^* = 0$</td>
<td>$o(1)$</td>
<td>$\Theta(1/\sqrt{N})$</td>
<td>$2N$</td>
</tr>
</tbody>
</table>
**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 \(N \to \infty\)

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Waiting time (fixed (\lambda &lt; 1))</th>
<th>Waiting time ((1 - \lambda \sim 1/\sqrt{N}))</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>(p_i^N = \lambda^i)</td>
<td>(\frac{\lambda}{1-\lambda})</td>
<td>(\Theta(\sqrt{N}))</td>
<td>0</td>
</tr>
<tr>
<td>JSQ((d))</td>
<td>(p_i^* = \lambda^{\frac{d^i-1}{d-1}})</td>
<td>(\Theta(1))</td>
<td>(\Omega(\log N))</td>
<td>(2d)</td>
</tr>
<tr>
<td>JSQ</td>
<td>(p_1^* = \lambda,) (&lt;p_2^* = 0)</td>
<td>(o(1))</td>
<td>(\Theta(1/\sqrt{N}))</td>
<td>(2N)</td>
</tr>
</tbody>
</table>
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]

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Waiting time (fixed (\lambda &lt; 1))</th>
<th>Waiting time ((1 - \lambda \sim 1/\sqrt{N}))</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>(p_i^N = \lambda^i)</td>
<td>(\frac{\lambda}{1-\lambda})</td>
<td>(\Theta(\sqrt{N}))</td>
<td>0</td>
</tr>
<tr>
<td>JSQ((d))</td>
<td>(p_i^* = \lambda^{\frac{d^i - 1}{d-1}})</td>
<td>(\Theta(1))</td>
<td>(\Omega(\log N))</td>
<td>(2d)</td>
</tr>
</tbody>
</table>
| 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

Random routing

\[ p_i^* = \lambda^i \]

\[ d = 2 \]

\[ p_i^* = \lambda \frac{2^i - 1}{2 - 1} \]

\[ d = 3 \]

\[ p_i^* = \lambda \frac{3^i - 1}{3 - 1} \]
Performance versus Communication Overhead at a Glance

\[ p_i^* = \lambda^i \]

\[ p_i^* = \lambda \frac{2^i - 1}{2 - 1} \]

\[ p_i^* = \lambda \frac{3^i - 1}{3 - 1} \]

\[ p_1^* = \lambda, \ p_2^* = 0 \]
Performance versus Communication Overhead at a Glance

Random routing

$p_i^* = \lambda^i$

$d = 2$

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

$d = 3$

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

JSQ

$p_1^* = \lambda, \ p_2^* = 0$
In order to examine performance versus communication trade-off, we allow parameter $d$ to depend on $N$ and write $d(N)$ to reflect that
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**

- $\text{JSQ}(d(N))$ has same fluid limit as JSQ, and achieves fluid-level optimality when $d(N) \to \infty$
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

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

$$d(N) \rightarrow \infty$$

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

$$\frac{d(N)}{\sqrt{N \log N}} \rightarrow \infty$$
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 \(q_i^{d(N)}(t) = \frac{Q_i^{d(N)}(t)}{N}\) 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 \(q^{d(N)}(0) \rightarrow q^\infty\) and \(d(N) \rightarrow \infty\) as \(N \rightarrow \infty\), then \(\{q^{d(N)}(t)\}_{t \geq 0}\) has same weak limit \(\{q(t)\}_{t \geq 0}\) as \(N \rightarrow \infty\) as ordinary JSQ policy, with

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

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

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

where $q(0) = q^\infty$

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

$$q_1^* = \lambda, \quad q_i^* = 0, \quad i \geq 2$$
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:

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

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

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

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

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

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

- JSQ($d(N)$) scheme achieves diffusion-level optimality while reducing communication overhead by nearly a factor $O(\sqrt{N}/\log(N))$
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

Random routing

$p_i^* \propto \lambda^i$

$d = 2$

$p_i^* \propto \lambda \frac{2^i-1}{2-1}$

$d = 3$

$p_i^* \propto \lambda \frac{3^i-1}{3-1}$

JSQ

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

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

Universality and Asymptotic Optimality Properties

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

$$d(N) \rightarrow \infty$$

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

$$\frac{d(N)}{\sqrt{N} \log N} \rightarrow \infty$$
### Performance versus Communication Overhead Trade-off

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

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


- When server becomes idle, it sends token to dispatcher to advertise its availability
Join the Idle Queue (JIQ) Strategy: Yet Lower Overhead


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

- 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 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


- 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


- 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 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


- 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


- 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


- 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

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

\[ q_1^* = \lambda, \quad q_i^* = 0, \quad i \geq 2 \]
Fixed point of fluid limit for JIQ strategy [Stolyar 2015]

\[ q_1^* = \lambda, \quad q_i^* = 0, \quad i \geq 2 \]

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]

\[ q_1^* = \lambda, \quad q_i^* = 0, \quad i \geq 2 \]

**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, $\{\bar{Q}^{JIQ}(t)\}_{t \geq 0}$ has same weak limit $\{\bar{Q}(t)\}_{t \geq 0}$ 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, $\{\bar{Q}^{JIQ}(t)\}_{t \geq 0}$ has same weak limit $\{\bar{Q}(t)\}_{t \geq 0}$ as $N \to \infty$ as ordinary JSQ policy.

- JIQ strategy outperforms JSQ($d$) schemes in terms of performance and communication overhead.
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, $\{\bar{Q}^{JIQ}(t)\}_{t\geq 0}$ has same weak limit $\{\bar{Q}(t)\}_{t\geq 0}$ 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 $N \to \infty$. 

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, $\{\bar{Q}^{JIQ}(t)\}_{t \geq 0}$ has same weak limit $\{\bar{Q}(t)\}_{t \geq 0}$ 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, \( \{\bar{Q}^{JIQ}(t)\}_{t \geq 0} \) has same weak limit \( \{\bar{Q}(t)\}_{t \geq 0} \) 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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Queue length</th>
<th>Waiting time (fixed $\lambda &lt; 1$)</th>
<th>Waiting time $(1 - \lambda \sim 1/\sqrt{N})$</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p_i^N = \lambda^i$</td>
<td>$\frac{\lambda}{1-\lambda}$</td>
<td>$\Theta(\sqrt{N})$</td>
<td>0</td>
</tr>
<tr>
<td>JSQ($d$)</td>
<td>$p_i^* = \lambda \frac{d_i-1}{d-1}$</td>
<td>$\Theta(1)$</td>
<td>$\Omega(\log N)$</td>
<td>$2d$</td>
</tr>
<tr>
<td>$d(N)$ $\rightarrow \infty$</td>
<td>same as JSQ</td>
<td>same as JSQ</td>
<td>??</td>
<td>$2d(N)$</td>
</tr>
<tr>
<td>$\frac{d(N)}{\sqrt{N} \log(N)} \rightarrow \infty$</td>
<td>same as JSQ</td>
<td>same as JSQ</td>
<td>same as JSQ</td>
<td>$2d(N)$</td>
</tr>
<tr>
<td>JSQ</td>
<td>$p_1^* = \lambda$, $p_2^* = 0$</td>
<td>$o(1)$</td>
<td>$\Theta(1/\sqrt{N})$</td>
<td>$2N$</td>
</tr>
<tr>
<td>JIQ</td>
<td>same as JSQ</td>
<td>same as JSQ</td>
<td>same as JSQ</td>
<td>$\leq 1$</td>
</tr>
</tbody>
</table>
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 $\lambda$ and can either be executed locally or forwarded to neighbor in $G_N$ with shorter queue, if any

![Diagram of load balancing in network scenarios]
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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $N \to \infty$? [Mukherjee, B, Van Leeuwaarden 2018]
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 $N \to \infty$? [Mukherjee, B, Van Leeuwaarden 2018]

Or equivalently, how much sparsity can be allowed while retaining asymptotically similar performance under JSQ in $G_N$ as in complete graph?
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 $N \to \infty$? [Mukherjee, B, Van Leeuwaarden 2018]

Or equivalently, how much sparsity can be allowed while retaining asymptotically similar performance under JSQ in $G_N$ as in complete graph?

Slightly different criterion: what degree of connectivity is required for $\text{JSQ}(d)$ to yield asymptotically similar performance in $G_N$ as in complete graph [Budhiraja, Mukherjee, Wu 2018]
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 \( N - o(\sqrt{N}) \)
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.
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
- \( d_{\min}(G_N)/N \to c < \frac{1}{2} \), 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


Thank You for your Attention!!