Designing Low-Complexity Heavy-Traffic Delay-Optimal Load Balancing Schemes: Theory to Algorithms

Xingyu Zhou

The Ohio State University

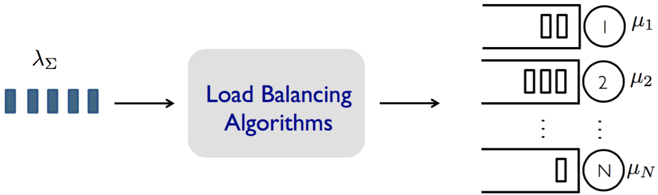
Joint work with...

Fei Wu, OSU

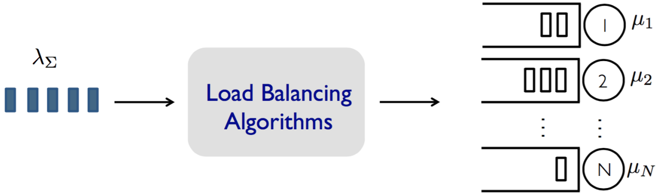
Jian Tan, OSU

Yin Sun, Auburn

Ness Shroff, OSU

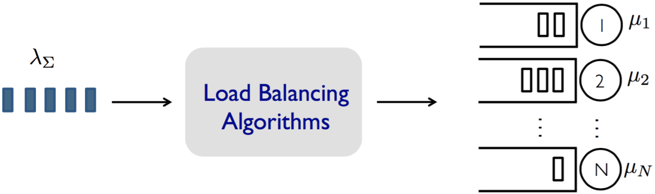


λ_Σ
Load Balancing Algorithms
1
2
N
μ_1
μ_2
μ_N



The goal of load balancing:

choose the *right* server(s) for each request.



The goal of load balancing:

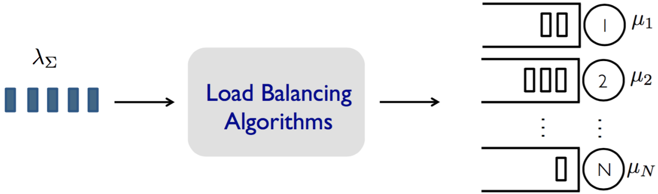
choose the *right* server(s) for each request.

What does *right* mean?

Define "optimal" algorithm

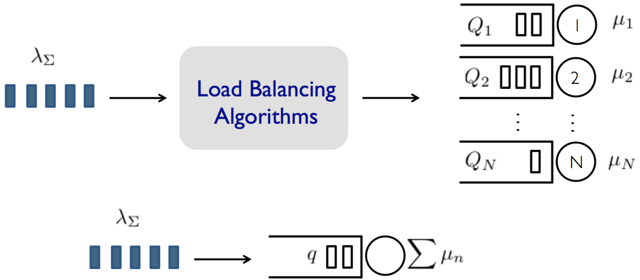
Definition (Throughput Optimal)

It can stabilize the system for any arrival rate in capacity region, i.e, for any $\epsilon > 0$ where $\epsilon = \sum \mu_n - \lambda_\Sigma$.



Definition (Heavy-traffic Delay Optimal)

It can achieve the lower bound on delay when $\epsilon \to 0$, that is, $\lim_{\epsilon \downarrow 0} \epsilon \mathbb{E}\left[\sum Q_n\right] = \lim_{\epsilon \downarrow 0} \epsilon \mathbb{E}\left[q\right]$

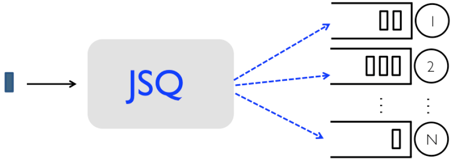


Fact: $\mathbb{E}\left[\sum Q_n\right] \geq \mathbb{E}\left[q\right]$, since packet remains in the queue until finished.

Push VS. Pull

Push algorithm: Join-shortest-queue (JSQ)

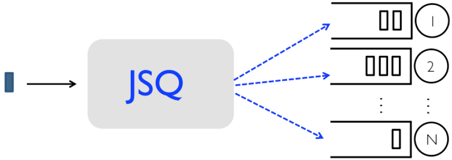
- sample each queue length
- join the shortest one



Push VS. Pull

Push algorithm: Join-shortest-queue (JSQ)

- sample each queue length
- join the shortest one



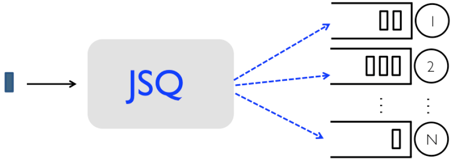
Pros:

- Delay optimal in a stochastic order sense. [Weber'78]
- *Heavy-traffic delay optimal.* [Foschini and Salz'78], [Eryilmaz and Srikant'12]

Push VS. Pull

Push algorithm: Join-shortest-queue (JSQ)

- sample each queue length
- join the shortest one



Pros:

- Delay optimal in a stochastic order sense. [Weber'78]
- *Heavy-traffic delay optimal.* [Foschini and Salz'78], [Eryilmaz and Srikant'12]

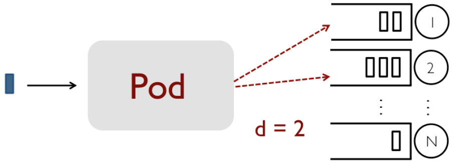
Cons:

- Message overhead is undesriable ($2N$ per arrival).
- Non-zero dispatching delay.

Push VS. Pull

Push algorithm: Power-of-d (Pod)

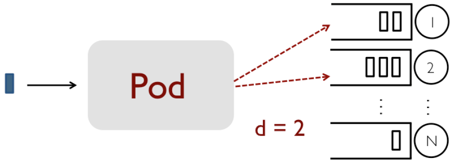
- randomly sample the queue lengths of d servers.
- join the shortest one among them.



Push VS. Pull

Push algorithm: Power-of-*d* (Pod)

- randomly sample the queue lengths of *d* servers.
- join the shortest one among them.



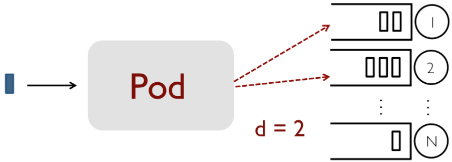
Pros:

- Double exponential decay when N is large. [Mitzenmacher'96]
- *Heavy-traffic delay optimal* for homogeneous servers. [Chen and Ye'12], [Maguluri, et al'14]
- Improved message overhead (2*d* per arrival)

Push VS. Pull

Push algorithm: Power-of-d (Pod)

- randomly sample the queue lengths of d servers.
- join the shortest one among them.



Pros:

- Double exponential decay when N is large. [Mitzenmacher'96]
- *Heavy-traffic delay optimal* for homogeneous servers. [Chen and Ye'12], [Maguluri, et al'14]
- Improved message overhead ($2d$ per arrival)

Cons:

- Non-zero dispatching delay.

Push VS. Pull

Pull algorithm: Join-idle-queue (JIQ)

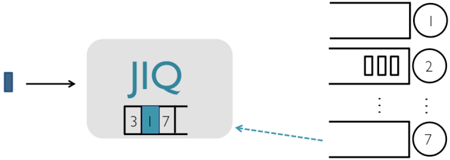
- if possible, join an idle queue randomly.
- otherwise, join an arbitrary queue randomly.

Push VS. Pull

Pull algorithm: Join-idle-queue (JIQ)

- if possible, join an idle queue randomly.
- otherwise, join an arbitrary queue randomly.

It is a pull algorithm since it behaves like the idle server pulls tasks from the dispathcer.

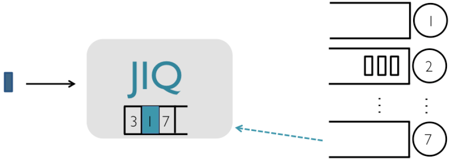


Push VS. Pull

Pull algorithm: Join-idle-queue (JIQ)

- if possible, join an idle queue randomly.
- otherwise, join an arbitrary queue randomly.

It is a pull algorithm since it behaves like the idle server pulls tasks from the dispathcer.



Pros:

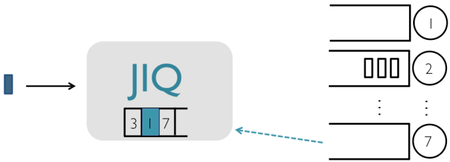
- Better delay performance than Pod with a lower message overhead (at most 1 per arrival), when traffic is not heavy. [Lu, et al'11], [Stolyar'15]
- Zero dispatching delay

Push VS. Pull

Pull algorithm: Join-idle-queue (JIQ)

- if possible, join an idle queue randomly.
- otherwise, join an arbitrary queue randomly.

It is a pull algorithm since it behaves like the idle server pulls tasks from the dispathcer.



Pros:

- Better delay performance than Pod with a lower message overhead (at most 1 per arrival), when traffic is not heavy. [Lu, et al'11], [Stolyar'15]
- Zero dispatching delay

Cons:

- Delay performance downgrades substantially under heavy traffic.

Motivation

The main problem:

- push algorithms are *heavy-traffic delay optimal*, but non-zero dispatching delay and relatively high message overhead.

- pull algorithm (JIQ) has *zero dispatching delay* and low message overhead, but very poor delay in heavy traffic.

Is it possible to attain both benefits at the same time?

Part I: Algorithms that attain both benefits

What we want...

What we want...

- "Optimal": throughput and heavy-traffic delay

What we want...

- "Optimal": throughput and heavy-traffic delay
- Zero dispatching delay

What we want...

- "Optimal": throughput and heavy-traffic delay
- Zero dispatching delay
- Low message overhead

What we want...

- "Optimal": throughput and heavy-traffic delay
- Zero dispatching delay
- Low message overhead
- Good performance over a large range of traffic

How can a single algorithm achieve all of these?

Show me your result...

The solution is our JBT-d (Join-Below-Threshold) algorithm:

Show me your result...

The solution is our JBT-d (Join-Below-Threshold) algorithm:

1. every T time-slots, randomly sample d servers and take the minimum queue length as threshold.

Show me your result...

The solution is our JBT-d (Join-Below-Threshold) algorithm:

1. every T time-slots, randomly sample d servers and take the minimum queue length as threshold.
2. each server report its ID when its queue length is below or equal to the threshold for the first time.

Show me your result...

The solution is our JBT-d (Join-Below-Threshold) algorithm:

1. every T time-slots, randomly sample d servers and take the minimum queue length as threshold.
2. each server report its ID when its queue length is below or equal to the threshold for the first time.
3. if possible, randomly picks a ID in memory and join the server.

Show me your result...

The solution is our JBT-d (Join-Below-Threshold) algorithm:

1. every T time-slots, randomly sample d servers and take the minimum queue length as threshold.
2. each server report its ID when its queue length is below or equal to the threshold for the first time.
3. if possible, randomly picks a ID in memory and join the server.
4. otherwise, randomly picks a queue to join.

Show me your result...

The solution is our JBT-d (Join-Below-Threshold) algorithm:

1. every T time-slots, randomly sample d servers and take the minimum queue length as threshold.
2. each server report its ID when its queue length is below or equal to the threshold for the first time.
3. if possible, randomly picks a ID in memory and join the server.
4. otherwise, randomly picks a queue to join.

Remark:

- static vs. dynamic: JIQ is just a special case of our JBT-d with $T = \infty$ and $th = 0$, thus static.

Show me your result...

The solution is our JBT-d (Join-Below-Threshold) algorithm:

1. every T time-slots, randomly sample d servers and take the minimum queue length as threshold.
2. each server report its ID when its queue length is below or equal to the threshold for the first time.
3. if possible, randomly picks a ID in memory and join the server.
4. otherwise, randomly picks a queue to join.

Remark:

- static vs. dynamic: JIQ is just a special case of our JBT-d with $T = \infty$ and $th = 0$, thus static.
- if servers are heterogeneous, report μ and pick ID with proportional probability in step 3 and 4.

Universal Optimal...

Theorem

For any finite T and $d \geq 1$, JBT-d is throughput and heavy-traffic delay optimal.

Universal Optimal...

Theorem

For any finite T and $d \geq 1$, JBT-d is throughput and heavy-traffic delay optimal.

In contrast...

Theorem

JIQ is not heavy-traffic delay optimal even for homogeneous servers.

Quiz time....

In the heavy-traffic limit: is the delay under JIQ be the same as that under Random?

(A). Yes

(B). No

Quiz time....

In the heavy-traffic limit: is the delay under JIQ be the same as that under Random?

(A). Yes (B). No

The answer is NO!

Quiz time....

In the heavy-traffic limit: is the delay under JIQ be the same as that under Random?

(A). Yes (B). No

The answer is NO!

We know that $Delay_{\text{rand}} = 2Delay_{\text{JSQ}}$ for two-server case.

In fact, in the heavy-traffic limit:

heavy-traffic optimal = JSQ < JIQ < Rand

What we achieve

- "Optimal": throughput and heavy-traffic delay ☺
- Zero dispatching delay
- Low message overhead
- Good performance over a large range of traffic

Immediately dispatched

In contrast to push algorithms, JSQ and Pod, where each arrival has to wait for sampling information, JBT-d dispatches arrival immediately:

- memory ID is non-empty: randomly choose one ID in memory to join.
- memory ID is empty: randomly choose one from all servers to join.

What we achieve

- “Optimal”: throughput and heavy-traffic delay
- Zero dispatching delay
- Low message overhead
- Good performance over a large range of traffic

Low message overhead

A crude upper bound on message overhead per arrival approaches one:

- Push-messages: $2d$ every T time-slots.
- Pull-messages:
  - at most 1 for each arrival
  - due to threshold update, it will discard at most N pull-messages every T time-slots.

Thus,

Upper bound on message per arrival is $1 + (2d + N)/T$

What we achieve

- "Optimal": throughput and heavy-traffic delay
- Zero dispatching delay
- Low message overhead
- Good performance over a large range of traffic

Simulations...

We have conducted a comprehensive set of simulations (32 figures!)

For now, you can temporarily trust me.

What we achieve

- "Optimal": throughput and heavy-traffic delay ☺
- Zero dispatching delay ☺
- Low message overhead ☺
- Good performance over a large range of traffic ☺

JBT-d is just *an* example:

We identify a “bag” of heavy-traffic delay optimal algorithms

Part II: Theory behind the 'bag'

The “bag” Π

Definition

A load balancing algorithm is in Π if

- the dispatching distribution $\mathbf{P}(t)$ is tilted for any t.
- every T time-slots, there exits a slot t' such that $\mathbf{P}(t')$ is δ-tilted.

The "bag" Π

Definition

A load balancing algorithm is in Π if

- the dispatching distribution $\mathbf{P}(t)$ is tilted for any t.
- every T time-slots, there exits a slot t' such that $\mathbf{P}(t')$ is δ-tilted.

Theorem

Any load balancing policy in Π is throughput optimal and heavy-traffic delay optimal.

The "bag" Π

Definition

A load balancing algorithm is in Π if

- the dispatching distribution $\mathbf{P}(t)$ is tilted for any t.
- every T time-slots, there exits a slot t' such that $\mathbf{P}(t')$ is δ-tilted.

Theorem

Any load balancing policy in Π is throughput optimal and heavy-traffic delay optimal.

Key notions: dispatching distribution $\mathbf{P}(t)$

- tilted
- δ-tilted

Dispatching distribution and preference

The nth component of dispatching distribution $\mathbf{P}(t)$ is the *probability* of dispatching arrival to the nth *shortest* queue.

- let $\sigma_t(\cdot)$ be the permutation of queues in increasing order.
- $P_n(t)$ is then the probability for dispatching to the server $\sigma_t(n)$.

Dispatching distribution and preference

The nth component of dispatching distribution $\mathbf{P}(t)$ is the *probability* of dispatching arrival to the nth *shortest* queue.

- let $\sigma_t(\cdot)$ be the permutation of queues in increasing order.
- $P_n(t)$ is then the probability for dispatching to the server $\sigma_t(n)$.

We also define dispatching preference

$$\boxed{\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)}$$

where $\mathbf{P}_{\text{rand}}(t)$ is the dispatching distribution under random routing, i.e,

- homogeneous servers: the nth component of $\mathbf{P}_{\text{rand}}(t)$ is $1/N$.
- heterogeneous servers: the nth component of $\mathbf{P}_{\text{rand}}(t)$ is $\mu_{\sigma_{t(n)}}/\mu_{\Sigma}$.

Example

Let consider a homogeneous case with 4 servers.

- Random: randomly joins one
  - $\mathbf{P}_{\text{rand}}(t) = (1/4, 1/4, 1/4, 1/4)$
  - $\Delta(t) = (0, 0, 0, 0)$

Example

Let consider a homogeneous case with 4 servers.

- Random: randomly joins one
  - $\mathbf{P}_{\mathrm{rand}}(t) = (1/4, 1/4, 1/4, 1/4)$
  - $\Delta(t) = (0, 0, 0, 0)$
- JSQ: always join the shortest one
  - $\mathbf{P}_{\mathrm{JSQ}}(t) = (1, 0, 0, 0)$
  - $\Delta_{JSQ}(t) = (3/4, -1/4, -1/4, -1/4)$

Example

Let consider a homogeneous case with 4 servers.

- Random: randomly joins one
  - $\mathbf{P}_{\text{rand}}(t) = (1/4, 1/4, 1/4, 1/4)$
  - $\Delta(t) = (0, 0, 0, 0)$
- JSQ: always join the shortest one
  - $\mathbf{P}_{\text{JSQ}}(t) = (1, 0, 0, 0)$
  - $\Delta_{JSQ}(t) = (3/4, -1/4, -1/4, -1/4)$
- Power of 2: randomly picks two and joins the shorter one
  - $\mathbf{P}_{\text{Po2}}(t) = (1/2, 1/3, 1/6, 0)$
  - $\Delta_{Po2}(t) = (1/4, 1/12, -1/12, -1/4)$

Example

Let consider a homogeneous case with 4 servers.

- Random: randomly joins one
  - $\mathbf{P}_{\text{rand}}(t) = (1/4, 1/4, 1/4, 1/4)$
  - $\Delta(t) = (0, 0, 0, 0)$
- JSQ: always join the shortest one
  - $\mathbf{P}_{\text{JSQ}}(t) = (1, 0, 0, 0)$
  - $\Delta_{JSQ}(t) = (3/4, -1/4, -1/4, -1/4)$
- Power of 2: randomly picks two and joins the shorter one
  - $\mathbf{P}_{\text{Po2}}(t) = (1/2, 1/3, 1/6, 0)$
  - $\Delta_{Po2}(t) = (1/4, 1/12, -1/12, -1/4)$

Any Observations?

- positive values in Δ indicates preference.

Let’s partition them by the extent of preference?

Let's partition them by the extent of preference?

Solution: tilted and δ-tilted

Tilted and δ-tilted distribution

$$\boxed{\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)}$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

Tilted $\mathbf{P}(t)$ is called 'okay'

Tilted and δ-tilted distribution

$$\boxed{\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)}$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'

Quiz time...

$$\boxed{\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)}$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'

Quiz time...

$$\boxed{\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)}$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

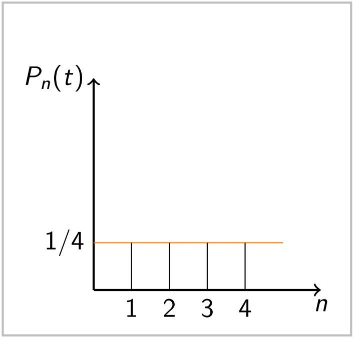
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\boxed{\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)}$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

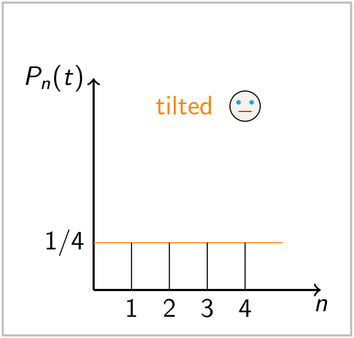
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\boxed{\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)}$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

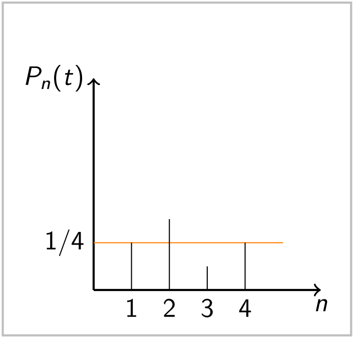
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

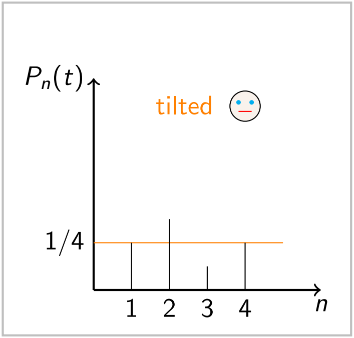
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

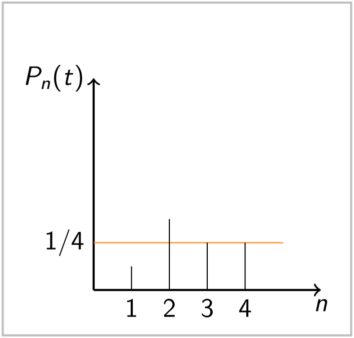
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\boxed{\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)}$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

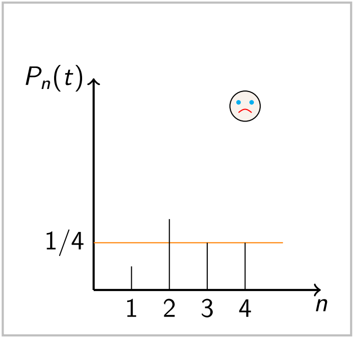
Tilted $\mathbf{P}(t)$ *is called 'okay'*

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ*-tilted* $\mathbf{P}(t)$ *is called 'Good'*



Quiz time...

$$\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

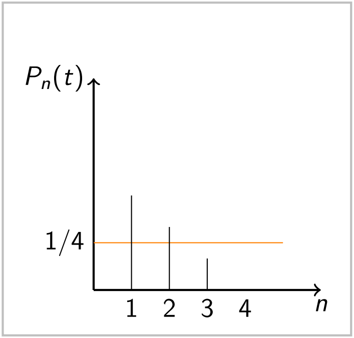
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\mathrm{rand}}(t)$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

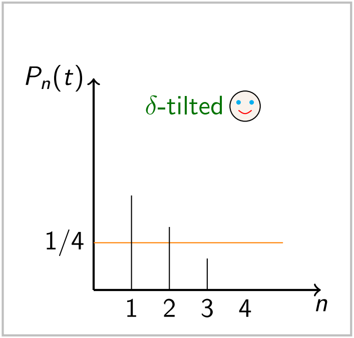
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

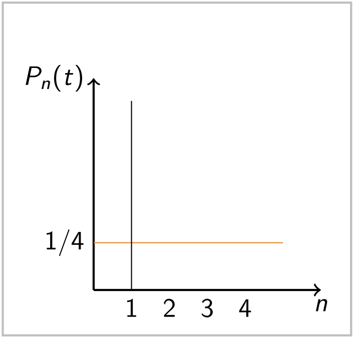
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

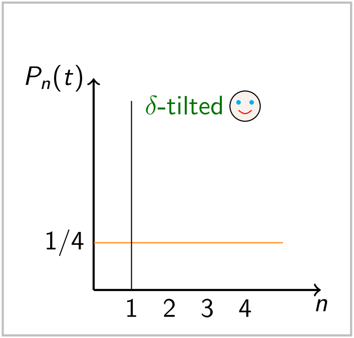
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

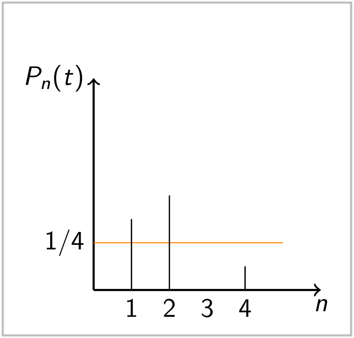
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Quiz time...

$$\Delta(t) \triangleq \mathbf{P}(t) - \mathbf{P}_{\text{rand}}(t)$$

Definition

A $\mathbf{P}(t)$ is tilted if, for some $2 \leq k \leq N$

- $\Delta_n(t) \geq 0$ for all $n < k$.
- $\Delta_n(t) \leq 0$ for all $n \geq k$

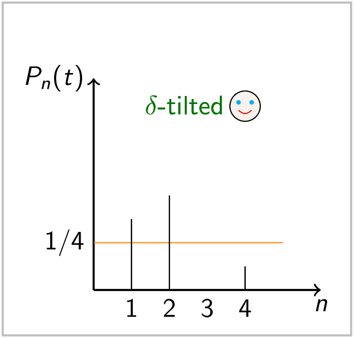
Tilted $\mathbf{P}(t)$ is called 'okay'

Definition

A $\mathbf{P}(t)$ is δ-tilted if

- $\Delta_n(t)$ is tilted.
- $\Delta_1(t) \geq \delta$, $\Delta_N(t) \leq -\delta$

δ-tilted $\mathbf{P}(t)$ is called 'Good'



Well, nice examples, but then?

Well, nice examples, but then?

Congratulations! You have MASTERED a class of 'optimal' policies!

All the items in “bag” are optimal

Recall that...

A load balancing algorithm is in Π if

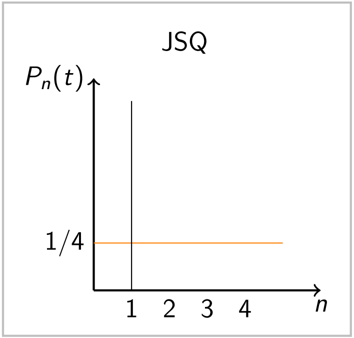
- the dispatching distribution $\mathbf{P}(t)$ is tilted for any t, i.e.,
- every T time-slots, there exits a slot t' such that $\mathbf{P}(t')$ is δ-tilted, i.e.,

Theorem

Any load balancing policy in Π is throughput optimal and heavy-traffic delay optimal.

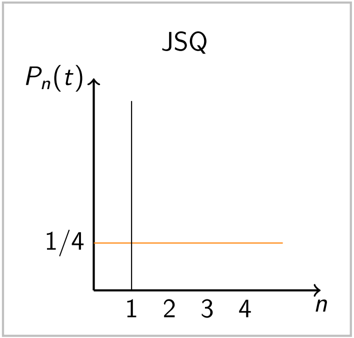
Previous optimal policies are in Π...

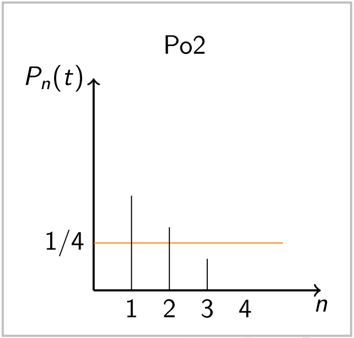
- JSQ is in bag Π
  - $T = 1$
  - $\mathbf{P}_{JSQ}(t) = (1, 0, \dots, 0)$, which is δ-tilted for all t.



Previous optimal policies are in Π...

- JSQ is in bag Π
  - $T = 1$
  - $\mathbf{P}_{JSQ}(t) = (1, 0, \ldots, 0)$, which is δ-tilted for all t.
- Power-of-d is in bag Π for *homogeneous servers and* $d \geq 2$
  - $T = 1$
  - $\mathbf{P}_{Pod}(t) = \binom{N-n}{d-1} / \binom{N}{d}, 1 \leq n \leq N - d + 1$, which is δ-tilted for all t.





New policy is in Π too...

Recall that our JBT-d is...

1. every T time-slots, randomly sample d servers and take the minimum queue length as threshold.
2. each server report its ID when its queue length is not larger than the threshold for the first time.
3. if possible, randomly picks a ID and join the server.
4. otherwise, randomly picks a queue to join.

We can show...

(a) Every t, $\mathbf{P}(t)$ is tilted.

(b) Every $t = kT + 1, k = 0, 1, \ldots,$ $\mathbf{P}(t)$ is δ-tilted.

Umm...please don't scare me with the proof!

Umm...please don't scare me with the proof!

OK, I promise

Here is the intuition...

Key idea: queues in the memory have higher preference.

(a) Every t, $\mathbf{P}(t)$ is tilted.

(b) Every $t = kT + 1, k = 0, 1, \ldots$, $\mathbf{P}(t)$ is δ-tilted.

Here is the intuition...

Key idea: queues in the memory have higher preference.

(a) Every t, $\mathbf{P}(t)$ is tilted.

- in the worst case, random routing is adopted.
- random routing is tilted.

(b) Every $t = kT + 1, k = 0, 1, \ldots,$ $\mathbf{P}(t)$ is δ-tilted.

Here is the intuition...

Key idea: queues in the memory have higher preference.

(a) Every t, $\mathbf{P}(t)$ is tilted.

- in the worst case, random routing is adopted.
- random routing is tilted.

(b) Every $t = kT + 1, k = 0, 1, \ldots,$ $\mathbf{P}(t)$ is δ-tilted.

- after sampling, the shorter the queue is, the more likely it is in the memory.
- as a result, the time-slot immediate after sampling, has preference of shorter queues over longer queues.
- thus, it is δ-tilted.

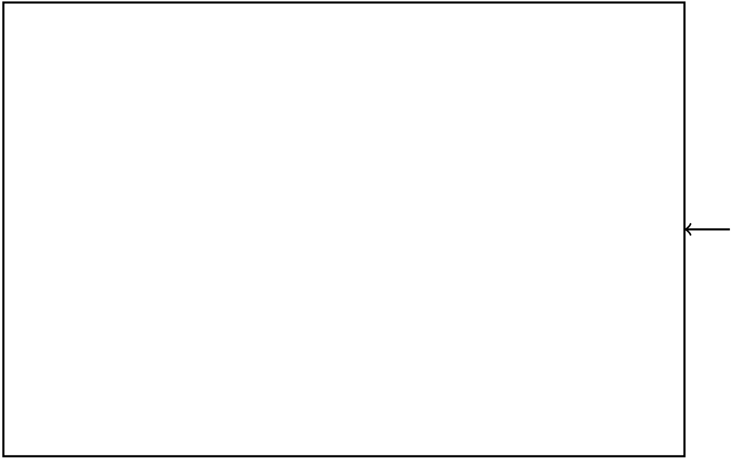
Many more policies deserves your discovery....

Could you give me a big picture now?

Could you give me a big picture now?

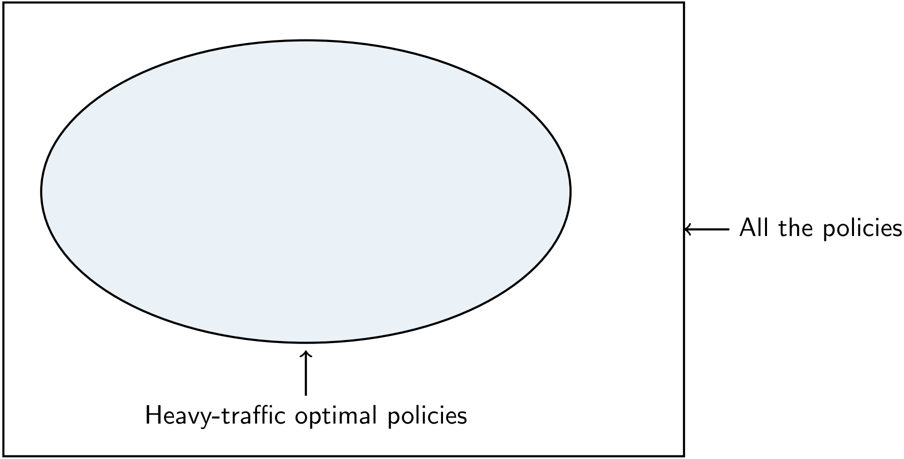
OK, why not!

Big picture

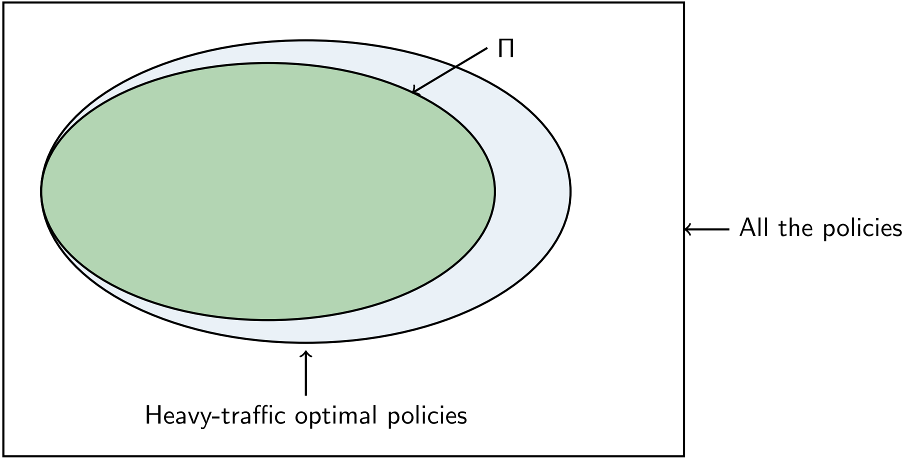


All the policies

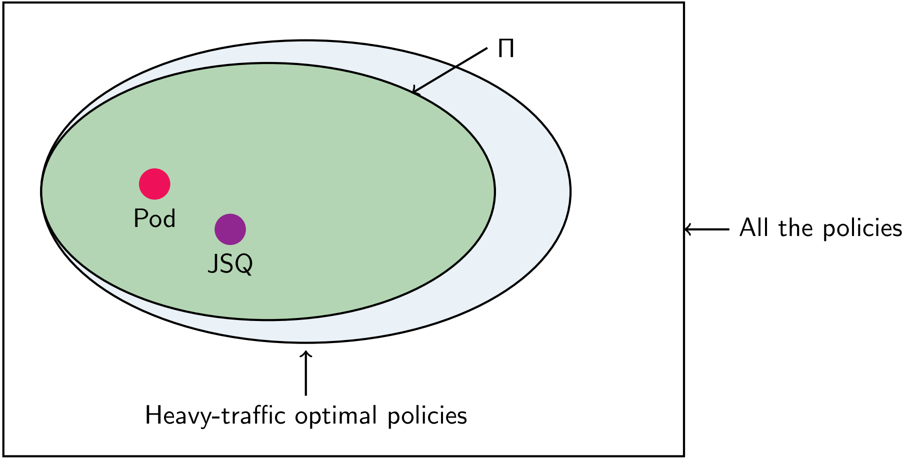
Big picture



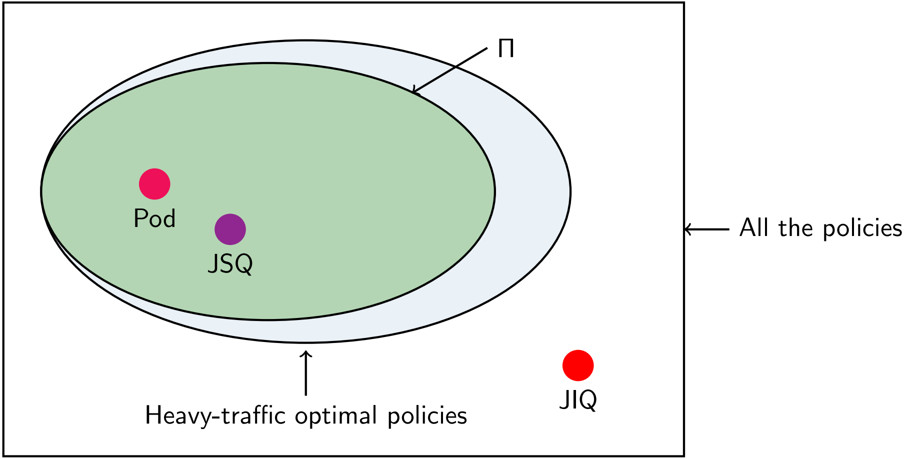
Big picture



Big picture

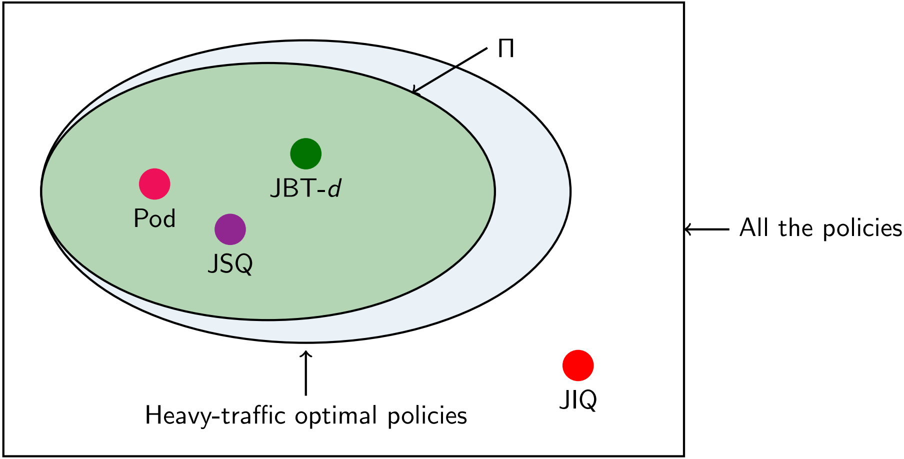


Big picture



Π
Pod
JSQ
All the policies
Heavy-traffic optimal policies
JIQ

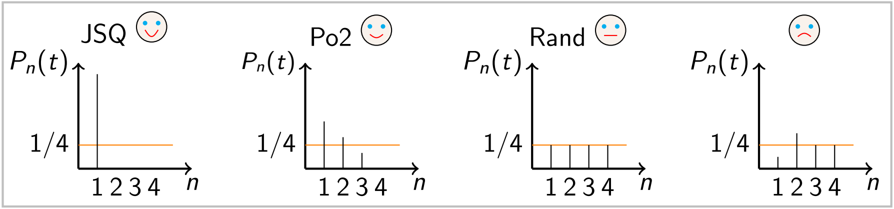
Big picture



Π
JBT-d
Pod
JSQ
All the policies
Heavy-traffic optimal policies
JIQ

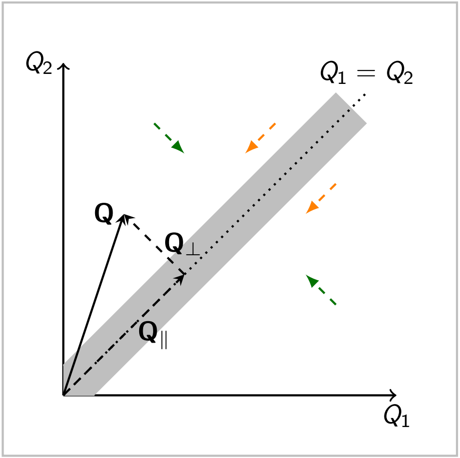
In summary, we go beyond previous ‘optimal’ policies:

1. we identify a class (bag Π) of ‘optimal’ policies.
2. we prove that pull-based JIQ is not ‘optimal’ even for homogeneous case.
3. we design a new ‘optimal’ pull-based policy, which enjoys all the nice features of JIQ.



Thank you!

From drift to optimality...



- The drift  is positive since

  $$\mathbb{E}\left[\langle \mathbf{Q}, \mathbf{A} - \mathbf{S} \rangle \mid \mathbf{Q}\right] \approx -\epsilon \left\| \mathbf{Q} \right\|$$

  under tilted $\mathbf{P}(t)$.

- The drift  is positive since

  $$\mathbb{E}\left[\langle \mathbf{Q}_\perp, \mathbf{A} - \mathbf{S} \rangle \mid \mathbf{Q}\right] \approx -\delta \left\| \mathbf{Q}_\perp \right\|$$

  under δ-tilted $\mathbf{P}(t)$ when $\epsilon \le \epsilon_0$

Backup

- Is the class Π tight?
- Is there any other possible useful policy in Π?
- Can you characterize the growth rate of threshold to guarantee optimality?

Backup

Lemma (Throughput optimal)

If there exist $T_1 > 0$, $K_1 \geq 0$, *and* $\gamma > 0$ *such that for all* $t_0 = 1, 2, \ldots$, *all* $Z \in \mathcal{Z}$ *and* $\lambda_\Sigma < \mu_\Sigma$

$$\mathbb{E}\left[\sum_{t=t_0}^{t_0+T_1-1} \langle \mathbf{Q}(t), \mathbf{A}(t) - \mathbf{S}(t) \rangle \mid Z(t_0) = Z\right] \leq -\gamma \left\|\mathbf{Q}\right\| + K_1, \qquad (1)$$

then the system is throughput-optimal.

Lemma (Heavy-traffic optimal)

Under the assumptions of the above lemma, if there further exist $T_2 > 0$, $K_2 \geq 0$ *and* $\eta > 0$ *that are independent of* ϵ, *such that for all* $t_0 = 1, 2, \ldots$ *and all* $Z \in \mathcal{Z}$

$$\mathbb{E}\left[\sum_{t=t_0}^{t_0+T_2-1} \langle \mathbf{Q}_\perp(t), \mathbf{A}(t) - \mathbf{S}(t) \rangle \mid Z(t_0) = Z\right] \leq -\eta \left\|\mathbf{Q}_\perp\right\| + K_2 \qquad (2)$$

holds for all $\epsilon \in (0, \epsilon_0)$, $\epsilon_0 > 0$, *then the system is heavy-traffic delay optimal.*