Heavy-traffic Delay Optimality in Pull-based Load Balancing Systems: Necessary and Sufficient Conditions

Xingyu Zhou

The Ohio State University

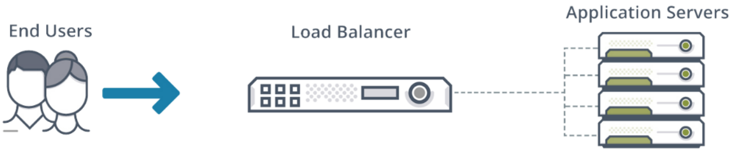
June 25 2019 @ Sigmetrics/Performance'19

Joint work with...

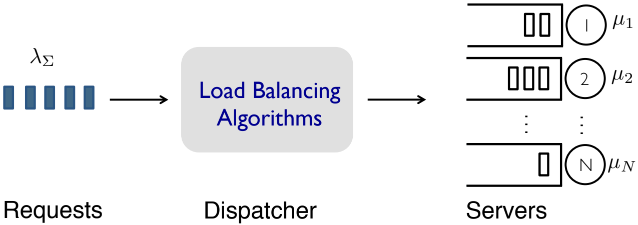
Jian Tan, Alibaba

Ness Shroff, OSU

The Model...

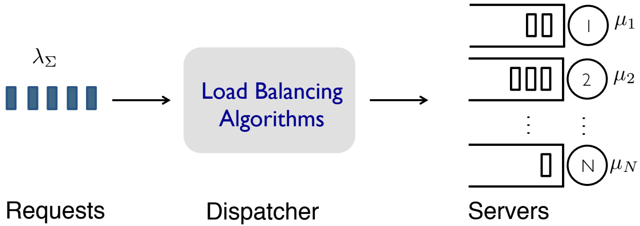


The Model...



λ_Σ
Load Balancing Algorithms
1
μ_1
2
μ_2
N
μ_N
Requests
Dispatcher
Servers

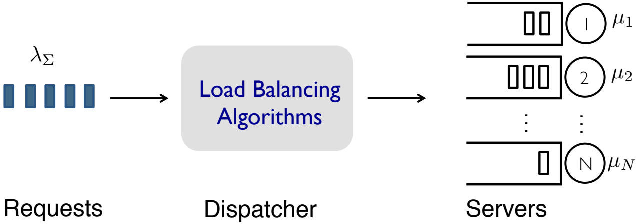
The Model...



- Discrete-time system, i.e, time-slotted.

[1] with all moments bounded

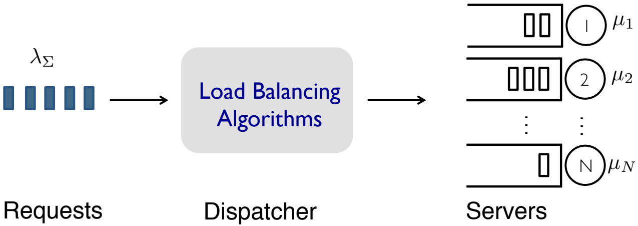
The Model...



- Discrete-time system, i.e, time-slotted.
- Arrival rate at each time slot is λ_Σ, general distribution [1] .

[1] with all moments bounded

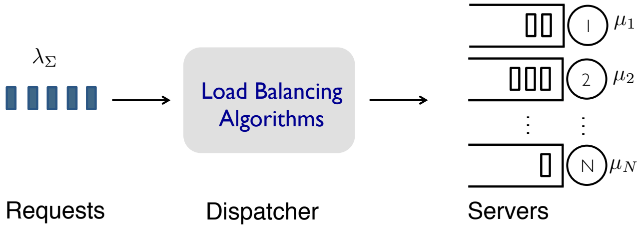
The Model...



- Discrete-time system, i.e, time-slotted.
- Arrival rate at each time slot is λ_Σ, general distribution [1] .
- Service rate at each server k is μ_k, general distribution.

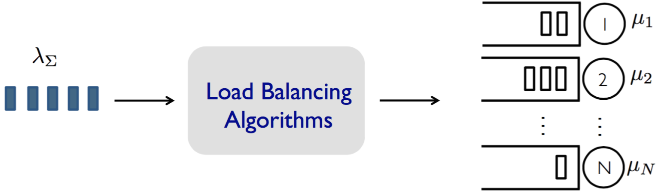
[1] with all moments bounded

The Model...



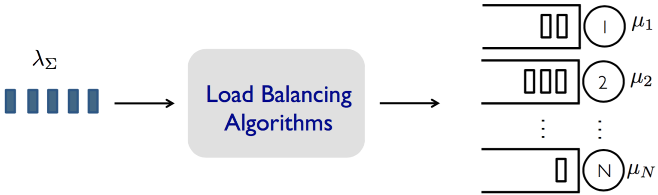
- Discrete-time system, i.e, time-slotted.
- Arrival rate at each time slot is λ_Σ, general distribution [1] .
- Service rate at each server k is μ_k, general distribution.
- Arrival and service are independent.

[1] with all moments bounded



The goal of load balancing:

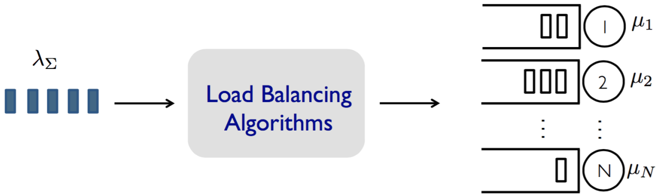
choose the *right* server(s) for each request.



The goal of load balancing:

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

What does *right* mean?

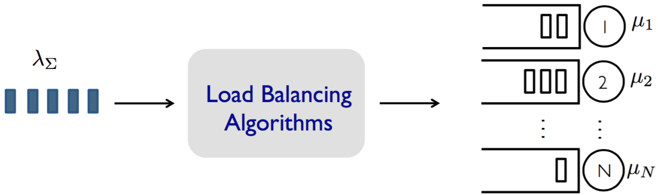


The goal of load balancing:

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

What does *right* mean?

- High throughput

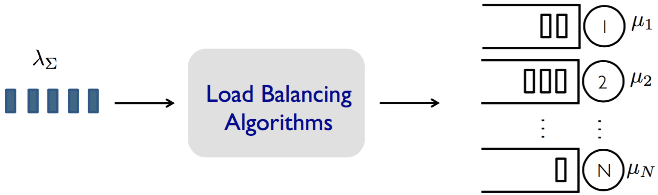


The goal of load balancing:

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

What does *right* mean?

- High throughput
- Low latency

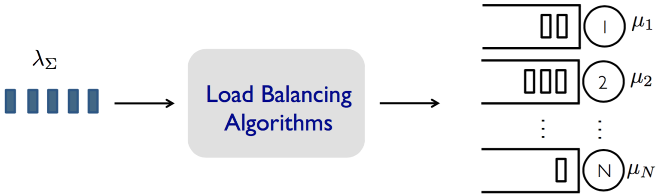


The goal of load balancing:

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

What does *right* mean?

- High throughput
- Low latency
    1. dispatching time (arrive at dispatcher – joining a server).



The goal of load balancing:

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

What does *right* mean?

- High throughput
- Low latency
  1. dispatching time (arrive at dispatcher – joining a server).
  2. response time (joining a server – leaving the server).
     - by Little's law, minimize the mean number of requests in system.

Which load balancing policy is the best?

Which load balancing policy is the best?

Maybe the most intuitive one: Join the Shortest Queue

Looking back...

Latency $=$ dispatching time $+$ response time (delay).

Looking back...

Latency = dispatching time + response time (delay).

- **40 years ago...**
  - Foschini, et. al, proved that JSQ is delay optimal in heavy traffic.

Looking back...

Latency = dispatching time + response time (delay).

- **40 years ago...**
  - Foschini, et. al, proved that JSQ is delay optimal in heavy traffic.
  - Initiated by dispatcher (push-based), non-zero dispatching time.
  - Often impractical to implement due to high message overhead.

Looking back...

Latency = dispatching time + response time (delay).

- **40 years ago...**
  - Foschini, et. al, proved that JSQ is delay optimal in heavy traffic.
  - Initiated by dispatcher (push-based), non-zero dispatching time.
  - Often impractical to implement due to high message overhead.
- **25 years ago...**
  - Frank Kelly, et. al, proposed a threshold type policy JBT.
    - Join-Below-Threshold (r): randomly join a server whose queue length is below r, if any; otherwise, randomly select any one to join.

Looking back...

Latency = dispatching time + response time (delay).

- **40 years ago...**
  - Foschini, et. al, proved that JSQ is delay optimal in heavy traffic.
  - Initiated by dispatcher (push-based), non-zero dispatching time.
  - Often impractical to implement due to high message overhead.
- **25 years ago...**
  - Frank Kelly, et. al, proposed a threshold type policy JBT.
    - Join-Below-Threshold (r): randomly join a server whose queue length is below r, if any; otherwise, randomly select any one to join.
  - Initiated by server (pull-based), zero dispatching time.
  - The message overhead is at most 1 per arrival.

Looking back...

Latency = dispatching time + response time (delay).

- **40 years ago...**
  - Foschini, et. al, proved that JSQ is delay optimal in heavy traffic.
  - Initiated by dispatcher (push-based), non-zero dispatching time.
  - Often impractical to implement due to high message overhead.
- **25 years ago...**
  - Frank Kelly, et. al, proposed a threshold type policy JBT.
    - Join-Below-Threshold (r): randomly join a server whose queue length is below r, if any; otherwise, randomly select any one to join.
  - Initiated by server (pull-based), zero dispatching time.
  - The message overhead is at most 1 per arrival.
  - They **conjectured** that if

    $$r \geq K \log(\text{average number of requests in system})$$

    then JBT is delay optimal in heavy traffic.

Looking back...

Latency = dispatching time + response time (delay).

- **40 years ago...**
  - Foschini, et. al, proved that JSQ is delay optimal in heavy traffic.
  - Initiated by dispatcher (push-based), non-zero dispatching time.
  - Often impractical to implement due to high message overhead.
- **25 years ago...**
  - Frank Kelly, et. al, proposed a threshold type policy JBT.
    - Join-Below-Threshold (r): randomly join a server whose queue length is below r, if any; otherwise, randomly select any one to join.
  - Initiated by server (pull-based), zero dispatching time.
  - The message overhead is at most 1 per arrival.
  - They **conjectured** that if

    $$r \geq K \log(\text{average number of requests in system})$$

    then JBT is delay optimal in heavy traffic.
- **From 1993...**
  - Logarithmic growth rate of threshold guarantees heavy-traffic delay optimality in several scheduling settings [Harrison'98], [Williams, et al'01].
  - However, it is still open in the load balancing setting.

Contributions...

We present necessary and sufficient conditions on the threshold for delay optimality in heavy traffic in **load balancing systems**.

Contributions...

We present necessary and sufficient conditions on the threshold for delay optimality in heavy traffic in **load balancing systems**.

- Theoretical contributions:

Contributions...

We present necessary and sufficient conditions on the threshold for delay optimality in heavy traffic in **load balancing systems**.

- Theoretical contributions:
  - Resolves the long-standing **open problem** mentioned before.
    - The results are more general compared to Kelly's original conjecture.

Contributions...

We present necessary and sufficient conditions on the threshold for delay optimality in heavy traffic in **load balancing systems**.

- Theoretical contributions:
  - Resolves the long-standing **open problem** mentioned before.
    - The results are more general compared to Kelly's original conjecture.
  - Provides new **insights** into heavy-traffic delay optimality.
    - The 'King' equation.

Contributions...

We present necessary and sufficient conditions on the threshold for delay optimality in heavy traffic in **load balancing systems**.

- Theoretical contributions:
  - Resolves the long-standing **open problem** mentioned before.
    - The results are more general compared to Kelly's original conjecture.
  - Provides new **insights** into heavy-traffic delay optimality.
    - The 'King' equation.
  - Develops new **techniques** for the analysis of load balancing policies.
    - New type of *state-space collapse*.

Queueing Systems 13 (1993) 47–86 47

Dynamic routing in open queueing networks: Brownian models, cut constraints and resource pooling

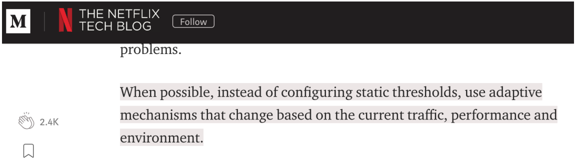
F.P. Kelly and C.N. Laws*

Statistical Laboratory, University of Cambridge, 16 Mill Lane, Cambridge CB2 1SB, England

Contributions...

The necessary and sufficient conditions on the threshold have...

- Practical contributions:
  - Provides a simple **guideline** for practical systems, e.g., Netflix Zuul.
  - Sheds light on the **design** of new pull-based algorithms.


THE NETFLIX TECH BLOG Follow

problems.

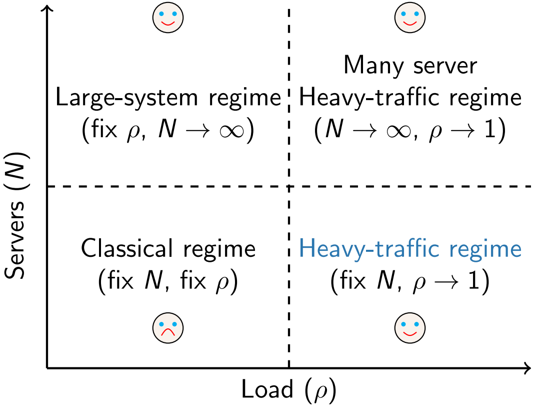
When possible, instead of configuring static thresholds, use adaptive mechanisms that change based on the current traffic, performance and environment.

2.4K

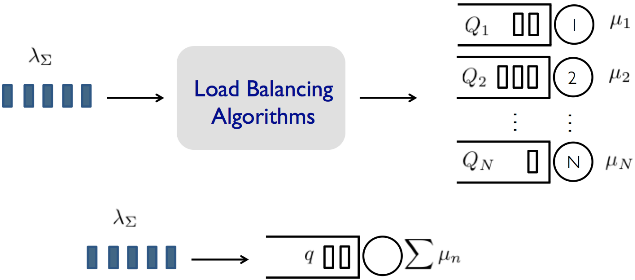
Part I: Background

Low delay...

- Closed-form formula in classical regime is very difficult.
- Turn to asymptotic regimes for insights.



Heavy-traffic Delay Optimality

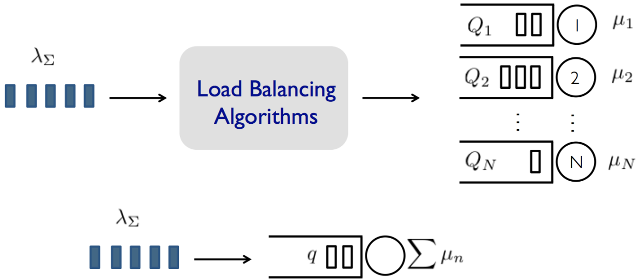


Fact: $\mathbb{E}\left[\sum Q_n\right] \geq \mathbb{E}\left[q\right]$, since one service process is stochastically larger.

Heavy-traffic Delay Optimality

Definition

It can achieve the lower bound on delay when $\epsilon \to 0$ ($\epsilon = \sum \mu_n - \lambda_\Sigma$), that is, $\lim_{\epsilon \downarrow 0} \epsilon \mathbb{E}\left[\sum Q_n\right] = \lim_{\epsilon \downarrow 0} \epsilon \mathbb{E}\left[q\right]$ (the queue length is on the order $O(1/\epsilon)$)



Fact: $\mathbb{E}\left[\sum Q_n\right] \geq \mathbb{E}\left[q\right]$, since one service process is stochastically larger.

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)

- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)
  - delay optimal in heavy traffic.
- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)
  - ☺ delay optimal in heavy traffic.

- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)
  - ☹ very poor delay in heavy traffic.

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)
  - ☺ delay optimal in heavy traffic.
  - ☹ non-zero dispatching delay.

- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)
  - ☹ very poor delay in heavy traffic.

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)
  - ☺ delay optimal in heavy traffic.
  - ☹ non-zero dispatching delay.

- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)
  - ☹ very poor delay in heavy traffic.
  - ☺ zero dispatching delay.

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)
  - ☺ delay optimal in heavy traffic.
  - ☹ non-zero dispatching delay.
  - ☹ high (relatively) message overhead.

- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)
  - ☹ very poor delay in heavy traffic.
  - ☺ zero dispatching delay.

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)
  - delay optimal in heavy traffic.
  - non-zero dispatching delay.
  - high (relatively) message overhead.

- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)
  - very poor delay in heavy traffic.
  - zero dispatching delay.
  - low message overhead.

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)
  - ☺ delay optimal in heavy traffic.
  - ☹ non-zero dispatching delay. (☹ by nature of push algorithms)
  - ☹ high (relatively) message overhead.

- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)
  - ☹ very poor delay in heavy traffic.
  - ☺ zero dispatching delay.
  - ☺ low message overhead.

Big picture...

- Push algorithms, e.g., JSQ and Pod (*sample and join the shorter queues*)
  - delay optimal in heavy traffic.
  - non-zero dispatching delay. (by nature of push algorithms)
  - high (relatively) message overhead.

- Pull algorithm, e.g., JIQ (*idle server pulls jobs from dispatcher*)
  - very poor delay in heavy traffic. (some hope here?)
  - zero dispatching delay.
  - low message overhead.

Can we design a heavy-traffic delay optimal pull-based policy?

Can we design a heavy-traffic delay optimal pull-based policy?

Main idea: A dynamic threshold!

Can we design a heavy-traffic delay optimal pull-based policy?

Main idea: A dynamic threshold!

The hope is that:

- instead of only storing idle servers.
- the dispatcher stores servers with queue lengths being **less than** a dynamic threshold!

Part II: Necessary Conditions

The general pull-based policy...

The JBT(r) (Join-Below-Threshold(r)) algorithm:

The general pull-based policy...

The JBT(r) (Join-Below-Threshold(r)) algorithm:

1. the memory at the dispatcher stores IDs of servers with queue lengths being less than r. (mechanism will be introduced later)

The general pull-based policy...

The JBT(r) (Join-Below-Threshold(r)) algorithm:

1. the memory at the dispatcher stores IDs of servers with queue lengths being less than r. (mechanism will be introduced later)
2. if the memory is non-empty, randomly picks a ID and joins the server.

The general pull-based policy...

The JBT(r) (Join-Below-Threshold(r)) algorithm:

1. the memory at the dispatcher stores IDs of servers with queue lengths being less than r. (mechanism will be introduced later)
2. if the memory is non-empty, randomly picks a ID and joins the server.
3. otherwise, randomly picks a queue to join.

The general pull-based policy...

The JBT(r) (Join-Below-Threshold(r)) algorithm:

1. the memory at the dispatcher stores IDs of servers with queue lengths being less than r. (mechanism will be introduced later)
2. if the memory is non-empty, randomly picks a ID and joins the server.
3. otherwise, randomly picks a queue to join.

Remark:

- JIQ is just a special case of JBT(r) with constant $r = 1$.

The general pull-based policy...

The JBT(r) (Join-Below-Threshold(r)) algorithm:

1. the memory at the dispatcher stores IDs of servers with queue lengths being less than r. (mechanism will be introduced later)
2. if the memory is non-empty, randomly picks a ID and joins the server.
3. otherwise, randomly picks a queue to join.

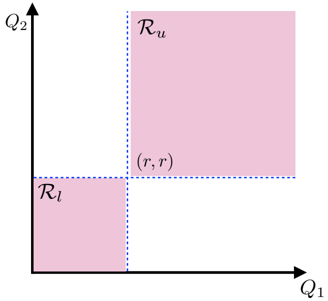
Remark:

- JIQ is just a special case of JBT(r) with constant $r = 1$.
- For heterogeneous servers, we can just replace random selection with selection in proportion to the service rate.

Geometry of JBT...

The JBT(r) (Join-Below-Threshold(r)) algorithm:

1. the memory at the dispatcher stores IDs of servers with queue lengths being less than r. (mechanism will be introduced later)
2. if the memory is non-empty, randomly picks an ID and join the server.
3. otherwise, randomly picks a queue to join.

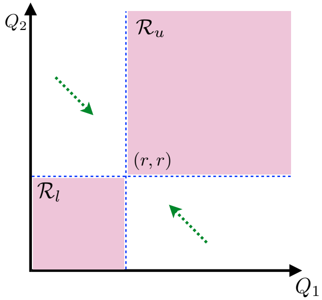


$\mathbf{Q} \in \mathcal{R}_l$: Random (full memory) $\mathbf{Q} \in \mathcal{R}_u$: Random (empty memory)

Geometry of JBT...

The JBT(r) (Join-Below-Threshold(r)) algorithm:

1. the memory at the dispatcher stores IDs of servers with queue lengths being less than r. (mechanism will be introduced later)
2. if the memory is non-empty, randomly picks a ID and join the server.
3. otherwise, randomly picks a queue to join.



$\mathbf{Q} \notin \mathcal{R}_l \cup \mathcal{R}_u$: shorter queues are preferred.

Grow, but not too fast...

Theorem (Necessary Conditions)

Consider the JBT(r) policy.

1. *For any constant threshold r, we have the following average delay ordering in heavy traffic:*

$$\overline{D}_{JSQ} < \overline{D}_{JBT(r)} < \overline{D}_{Rand}$$

2. *For $r = \Omega((1/\epsilon)^{1+\alpha})$ for any $\alpha > 0$, we have that in heavy traffic:*

$$\overline{D}_{JSQ} < \overline{D}_{JBT(r)} = \overline{D}_{Rand}$$

Before the proof, any intuitions?

The 'King' equation...

The sufficient and necessary condition for HT-optimality:

$$\lim_{\epsilon \downarrow 0} \mathbb{E}\left[\left\| \overline{\mathbf{Q}}^{(\epsilon)}(t+1) \right\|_1 \left\| \overline{\mathbf{U}}^{(\epsilon)}(t) \right\|_1 \right] = 0.$$

where the unused service vector $\mathbf{U}(t) = \max\{\mathbf{S}(t) - \mathbf{Q}(t) - \mathbf{A}(t), \mathbf{0}\}$.

The ‘King’ equation...

The sufficient and necessary condition for HT-optimality:

$$\lim_{\epsilon \downarrow 0} \mathbb{E}\left[\left\| \overline{\mathbf{Q}}^{(\epsilon)}(t+1) \right\|_1 \left\| \overline{\mathbf{U}}^{(\epsilon)}(t) \right\|_1 \right] = 0.$$

where the unused service vector $\mathbf{U}(t) = \max\{\mathbf{S}(t) - \mathbf{Q}(t) - \mathbf{A}(t), \mathbf{0}\}$.

- Note that $Q_n(t+1)U_n(t) = 0$ for all n and t.

The 'King' equation...

The sufficient and necessary condition for HT-optimality:

$$\lim_{\epsilon \downarrow 0} \mathbb{E}\left[\left\|\overline{\mathbf{Q}}^{(\epsilon)}(t+1)\right\|_1 \left\|\overline{\mathbf{U}}^{(\epsilon)}(t)\right\|_1\right] = 0.$$

where the unused service vector $\mathbf{U}(t) = \max\{\mathbf{S}(t) - \mathbf{Q}(t) - \mathbf{A}(t), \mathbf{0}\}$.

- Note that $Q_n(t+1)U_n(t) = 0$ for all n and t.

IMPLICATIONS: No server is idle while others with high loads.

The 'King' equation...

The sufficient and necessary condition for HT-optimality:

$$\lim_{\epsilon \downarrow 0} \mathbb{E}\left[\left\|\overline{\mathbf{Q}}^{(\epsilon)}(t+1)\right\|_1 \left\|\overline{\mathbf{U}}^{(\epsilon)}(t)\right\|_1\right] = 0.$$

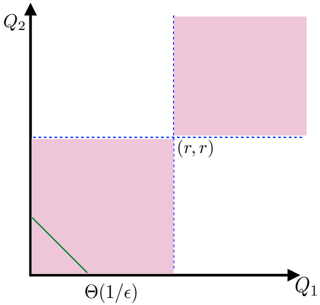
where the unused service vector $\mathbf{U}(t) = \max\{\mathbf{S}(t) - \mathbf{Q}(t) - \mathbf{A}(t), \mathbf{0}\}$.

- Note that $Q_n(t+1)U_n(t) = 0$ for all n and t.

IMPLICATIONS: No server is idle while others with high loads.

"*Probability theory is nothing but common sense reduced to calculation.*"

— Pierre Laplace

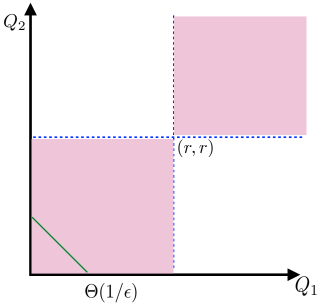


$(a) \quad r = \Omega((1/\epsilon)^{1+\alpha}), \alpha > 0$

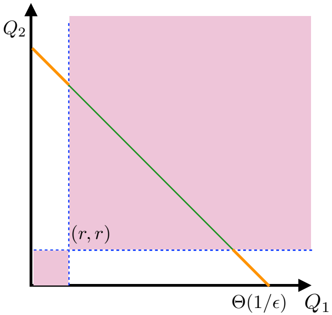
Completely degenerate to random.

$$\overline{D}_{JSQ} < \overline{D}_{JBT(r)} = \overline{D}_{Rand}$$

$\mathbf{Q} \in \mathcal{R}_I$: Random (full memory)



$(a) \quad r = \Omega((1/\epsilon)^{1+\alpha}), \alpha > 0$



$(b) \quad r$ is a constant in $[1, \infty)$

Completely degenerate to random.

Always has 'nice' things happen.

$$\overline{D}_{JSQ} < \overline{D}_{JBT(r)} = \overline{D}_{Rand}$$

$\mathbf{Q} \in \mathcal{R}_l$: Random (full memory)

$$\overline{D}_{JSQ} < \overline{D}_{JBT(r)} < \overline{D}_{Rand}$$

$\mathbf{Q} \notin \mathcal{R}_l \cup \mathcal{R}_u$: 'nice' things.

The Universal Equality...

$$\mathbb{E}\left[\left\|\overline{\mathbf{Q}}^{(\epsilon)}(t+1)\right\|_1\left\|\overline{\mathbf{U}}^{(\epsilon)}(t)\right\|_1\right] = \mathcal{T}_1^{(\epsilon)} + \mathcal{T}_2^{(\epsilon)} - \mathcal{T}_3^{(\epsilon)}$$

where

$$\mathcal{T}_1^{(\epsilon)} \triangleq 2\sum_{i=1}^{N}\sum_{j>i}^{N}\mathbb{E}\left[\left(\overline{Q}_i^{(\epsilon)} - \overline{Q}_j^{(\epsilon)}\right)\left(\overline{A}_i^{(\epsilon)} - \overline{A}_j^{(\epsilon)} - \overline{S}_i^{(\epsilon)} + \overline{S}_j^{(\epsilon)}\right)\right]$$

$$\mathcal{T}_2^{(\epsilon)} \triangleq \sum_{i=1}^{N}\sum_{j>i}^{N}\mathbb{E}\left[\left(\overline{A}_i^{(\epsilon)} - \overline{A}_j^{(\epsilon)} - \overline{S}_i^{(\epsilon)} + \overline{S}_j^{(\epsilon)}\right)^2\right]$$

$$\mathcal{T}_3^{(\epsilon)} \triangleq \sum_{i=1}^{N}\sum_{j>i}^{N}\mathbb{E}\left[\left(\overline{U}_i^{(\epsilon)} - \overline{U}_j^{(\epsilon)}\right)^2\right]$$

$$\overline{\mathbf{Q}}^{+} \triangleq \overline{\mathbf{Q}}(t+1)$$

The first unified method to show a policy is NOT optimal.

Then...how fast should it grow?

Part III: Sufficient Conditions

Conjecture time....

Consider the JBT(r) policy with load parameter $\epsilon = \sum \mu_n - \lambda_\Sigma$.

Conjecture time....

Consider the JBT(r) policy with load parameter $\epsilon = \sum \mu_n - \lambda_\Sigma$.

Question: Which of the following r value guarantees 'optimality'?

(A). $r = 100$
(B). $r = \theta\left(\log(1/\epsilon)\right)$
(C). $r = \theta(\log^2(1/\epsilon))$
(D). $r = \theta((1/\epsilon)^{1.01})$

Hint: The average number of tasks is on the order of $1/\epsilon$.

Conjecture time....

Consider the JBT(r) policy with load parameter $\epsilon = \sum \mu_n - \lambda_\Sigma$.

Question: Which of the following r value guarantees 'optimality'?

(A). $\cancel{r = 100}$
(B). $r = \theta\left(\log(1/\epsilon)\right)$
(C). $r = \theta(\log^2(1/\epsilon))$
(D). $r = \theta((1/\epsilon)^{1.01})$

Hint: The average number of tasks is on the order of $1/\epsilon$.

Conjecture time....

Consider the JBT(r) policy with load parameter $\epsilon = \sum \mu_n - \lambda_\Sigma$.

Question: Which of the following r value guarantees 'optimality'?

(A). $\cancel{r = 100}$
(B). $r = \theta\left(\log(1/\epsilon)\right)$
(C). $r = \theta(\log^2(1/\epsilon))$
(D). $\cancel{r = \theta((1/\epsilon)^{1.01})}$

Hint: The average number of tasks is on the order of $1/\epsilon$.

25 years ago...

25 years ago...

Queueing Systems 13 (1993) 47–86 47

Dynamic routing in open queueing networks: Brownian models, cut constraints and resource pooling

F.P. Kelly and C.N. Laws*

Statistical Laboratory, University of Cambridge, 16 Mill Lane, Cambridge CB2 1SB, England

Conjecture: 'optimality' is guaranteed if $r \geq K \log(1/\epsilon)$.[2]

[2] Two-server case and diffusion approximation optimality

Logarithmic growth is sufficient...

Theorem (Sufficient Conditions)

Consider the JBT(r) policy with threshold r. Suppose $r \geq K \log(1/\epsilon)$ and $r = o(1/\epsilon)$, for some constant K, then it is heavy-traffic delay optimal in steady-state.

Logarithmic growth is sufficient...

Theorem (Sufficient Conditions)

Consider the JBT(r) policy with threshold r. Suppose $r \geq K \log(1/\epsilon)$ and $r = o(1/\epsilon)$, for some constant K, then it is heavy-traffic delay optimal in steady-state.

Remark:

Logarithmic growth is sufficient...

Theorem (Sufficient Conditions)

Consider the JBT(r) policy with threshold r. Suppose $r \geq K \log(1/\epsilon)$ and $r = o(1/\epsilon)$, for some constant K, then it is heavy-traffic delay optimal in steady-state.

Remark:

- This holds for any fixed finite number of servers, $2 \leq N < \infty$.

Logarithmic growth is sufficient...

Theorem (Sufficient Conditions)

Consider the JBT(r) policy with threshold r. Suppose $r \geq K \log(1/\epsilon)$ and $r = o(1/\epsilon)$, for some constant K, then it is heavy-traffic delay optimal in steady-state.

Remark:

- This holds for any fixed finite number of servers, $2 \leq N < \infty$.
- This holds for general arrival and service distributions (with finite moment bounds).

Logarithmic growth is sufficient...

Theorem (Sufficient Conditions)

Consider the JBT(r) policy with threshold r. Suppose $r \geq K \log(1/\epsilon)$ and $r = o(1/\epsilon)$, for some constant K, then it is heavy-traffic delay optimal in steady-state.

Remark:

- This holds for any fixed finite number of servers, $2 \leq N < \infty$.
- This holds for general arrival and service distributions (with finite moment bounds).
- The optimality is directly in steady-state.

The challenge to prove it...

The challenge to prove it...

(a) Diffusion approximations method.

- ☹ multi-dimensional Brownian motion is difficult to analyze.

The challenge to prove it...

(a) Diffusion approximations method.

- ☹ multi-dimensional Brownian motion is difficult to analyze.
- ☹ optimality often only exists in finite time, not steady-state result.

The challenge to prove it...

(a) Diffusion approximations method.

- ☹ multi-dimensional Brownian motion is difficult to analyze.
- ☹ optimality often only exists in finite time, not steady-state result.

(b) Lyapunov drift-based method.

The challenge to prove it...

(a) Diffusion approximations method.

- multi-dimensional Brownian motion is difficult to analyze.
- optimality often only exists in finite time, not steady-state result.

(b) Lyapunov drift-based method.

- standard techniques fail in our case.
  - since the *state-space collapse* is neither a line nor a cone.
  - instead, it is even non-convex.

Our methods:

The challenge to prove it...

(a) Diffusion approximations method.

- ☹ multi-dimensional Brownian motion is difficult to analyze.
- ☹ optimality often only exists in finite time, not steady-state result.

(b) Lyapunov drift-based method.

- ☹ standard techniques fail in our case.
  - since the *state-space collapse* is neither a line nor a cone.
  - instead, it is even non-convex.

Our methods:

- we use drift-based analysis to establish a new type of state-space collapse.

The challenge to prove it...

(a) Diffusion approximations method.

- ☹ multi-dimensional Brownian motion is difficult to analyze.
- ☹ optimality often only exists in finite time, not steady-state result.

(b) Lyapunov drift-based method.

- ☹ standard techniques fail in our case.
  - since the *state-space collapse* is neither a line nor a cone.
  - instead, it is even non-convex.

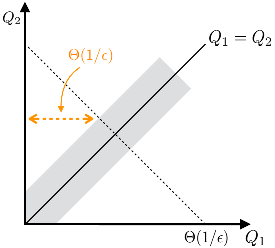
Our methods:

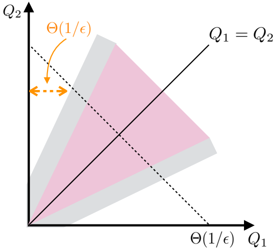
- we use drift-based analysis to establish a new type of state-space collapse.
- we combine this state-space collapse with Chernoff bound.

What is *state-space collapse?*

Informally, it means steady state *'concentrates'* around a subspace.

- the distance between the steady state and a subspace has bounded moment upper bounds.
- the subspace could be a line or a cone in previous works.





Why is *state-space collapse* important?

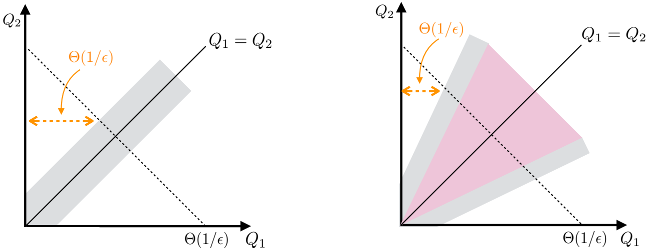
Recall the ‘King’ equation...

$$\lim_{\epsilon \downarrow 0} \mathbb{E}\left[\left\| \overline{\mathbf{Q}}^{(\epsilon)}(t+1) \right\|_1 \left\| \overline{\mathbf{U}}^{(\epsilon)}(t) \right\|_1 \right] = 0.$$

where the unused service vector $\mathbf{U}(t) = \max\{\mathbf{S}(t) - \mathbf{Q}(t) - \mathbf{A}(t), \mathbf{0}\}$.

- Note that $Q_n(t+1)U_n(t) = 0$ for all n and t.

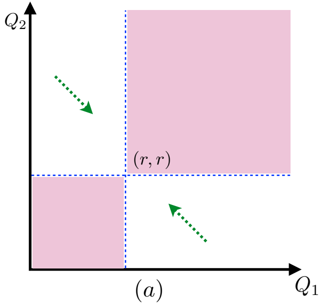
IMPLICATIONS: No server is idle while others with high loads.



In heavy traffic ($\epsilon \to 0$), steady state lies far away from boundary.

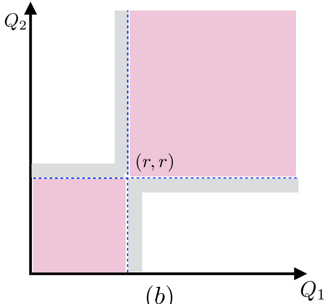
What would be the *state-space collapse* in our case?

From drift to Chernoff bound...



Drift towards the pink region due to preference of shorter queues.

From drift to Chernoff bound...



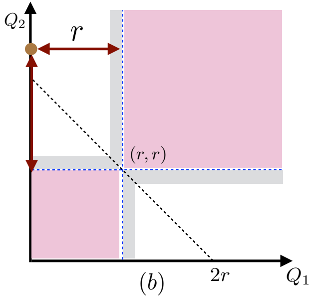
(b)

Then, ALL the moments of the distance to pink region are bounded!

$$\mathbb{E}\left[e^{\theta^* d_{\mathcal{R}(r)}\left(\overline{\mathbf{Q}}^{(\epsilon)}\right)}\right] \leq C^*,$$

where both θ^* and C^* are independent of ϵ.

From drift to Chernoff bound...

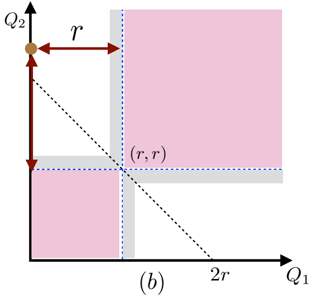


$$\mathbb{E}\left[\overline{Q}_2^{(\epsilon)}(t+1)\overline{U}_1^{(\epsilon)}\right]$$

$$=\mathbb{E}\left[\overline{Q}_2(t+1)\overline{U}_1\mathcal{I}\left(\overline{Q}_2(t+1)\leq 2r,\overline{Q}_1(t+1)=0\right)\right] \quad (1)$$

$$+\mathbb{E}\left[\overline{Q}_2(t+1)\overline{U}_1\mathcal{I}\left(\overline{Q}_2(t+1)> 2r,\overline{Q}_1(t+1)=0\right)\right] \quad (2)$$

From drift to Chernoff bound...



$$
\begin{aligned}
& \mathbb{E}\left[\overline{Q}_2^{(\epsilon)}(t+1)\overline{U}_1^{(\epsilon)}\right] \\
=& \mathbb{E}\left[\overline{Q}_2(t+1)\overline{U}_1 \mathcal{I}\left(\overline{Q}_2(t+1) \leq 2r, \overline{Q}_1(t+1) = 0\right)\right] && (1) \\
& + \mathbb{E}\left[\overline{Q}_2(t+1)\overline{U}_1 \mathcal{I}\left(\overline{Q}_2(t+1) > 2r, \overline{Q}_1(t+1) = 0\right)\right] && (2)
\end{aligned}
$$

$(1) \leq 2r\epsilon,$ since $\mathbb{E}\left[\overline{U}_1\right] \leq \epsilon.$

$$(2) \leq C\frac{1}{\epsilon^2}\mathbb{P}\left(\overline{Q}_2(t+1) > 2r, \overline{Q}_1(t+1) = 0\right) \leq C\frac{1}{\epsilon^2}\frac{1}{e^{\theta r}}$$

Implementation...

The key requirement: set of servers whose queue lengths are below the threshold are known at the dispatcher

Implementation...

The key requirement: set of servers whose queue lengths are below the threshold are known at the dispatcher

Definition

JBT(r) is composed of the following components:

Implementation...

The key requirement: set of servers whose queue lengths are below the threshold are known at the dispatcher

Definition

JBT(r) is composed of the following components:

- Each server is initialized with an empty queue, and a corresponding ID in the local memory of the dispatcher.

Implementation...

The key requirement: set of servers whose queue lengths are below the threshold are known at the dispatcher

Definition

JBT(r) is composed of the following components:

- Each server is initialized with an empty queue, and a corresponding ID in the local memory of the dispatcher.
- Upon new arrivals at the beginning of each time-slot, the dispatcher checks the available IDs in memory.

Implementation...

The key requirement: set of servers whose queue lengths are below the threshold are known at the dispatcher

Definition

JBT(r) is composed of the following components:

- Each server is initialized with an empty queue, and a corresponding ID in the local memory of the dispatcher.
- Upon new arrivals at the beginning of each time-slot, the dispatcher checks the available IDs in memory.
  - If one or more IDs exist, it removes one uniformly at random, and sends all the new arrivals to the corresponding server.

Implementation...

The key requirement: set of servers whose queue lengths are below the threshold are known at the dispatcher

Definition

JBT(r) is composed of the following components:

- Each server is initialized with an empty queue, and a corresponding ID in the local memory of the dispatcher.
- Upon new arrivals at the beginning of each time-slot, the dispatcher checks the available IDs in memory.
  - If one or more IDs exist, it removes one uniformly at random, and sends all the new arrivals to the corresponding server.
  - Otherwise, all the new arrivals are dispatched uniformly at random to one of the servers in the system.

Implementation...

The key requirement: set of servers whose queue lengths are below the threshold are known at the dispatcher

Definition

JBT(r) is composed of the following components:

- Each server is initialized with an empty queue, and a corresponding ID in the local memory of the dispatcher.
- Upon new arrivals at the beginning of each time-slot, the dispatcher checks the available IDs in memory.
  - If one or more IDs exist, it removes one uniformly at random, and sends all the new arrivals to the corresponding server.
  - Otherwise, all the new arrivals are dispatched uniformly at random to one of the servers in the system.
- Each server reports its ID to the dispatcher at the end of each time-slot if
  - its queue length is below the threshold
  - AND the dispatcher does not contain its ID (how?)

How to determine the threshold?

How to determine the threshold?

- If we can estimate the traffic load, then we can directly apply the sufficient condition.

How to determine the threshold?

- If we can estimate the traffic load, then we can directly apply the sufficient condition.
- If not, we can use sampling every T time-slots.
  - In particular, randomly sample d queues and take the minimum as threshold.
  - We can prove the following result.

How to determine the threshold?

- If we can estimate the traffic load, then we can directly apply the sufficient condition.
- If not, we can use sampling every T time-slots.
  - In particular, randomly sample d queues and take the minimum as threshold.
  - We can prove the following result.

Theorem

For any finite T and $d \geq 1$, the policy is throughput and delay optimal in heavy traffic.

Then...is the **logarithmic** growth rate also necessary?

Then...is the **logarithmic** growth rate also necessary?

We conjecture so!

Conclusion...

Theorem (Necessary Conditions)

- The threshold r should grow with the traffic load.
- But, it can not grow too fast.
- It provides a sharp characterization of JIQ policy.

Theorem (Sufficient Conditions)

- It is sufficient to have a logarithmic growth rate.
- This resolves a long-standing open problem.
- It provides a useful guideline for practical systems.

Thank you!

Q & A