Using Priorities and Selfish Scheduling to Obtain Global Optima

Rhonda Righter, IEOR, UC Berkeley

Joint work with

Osman Akgun, Bailard Inc.
Doug Down, McMaster University
Ron Wolff, IEOR, UC Berkeley

Two Views of

"Scheduling and priorities in queueing systems"

- First view to come to mind:
  - Multi-class queueing systems with class-dependent priorities (either given or decided, e.g., by an index rule)
  - Scheduling subject to those priorities
- The view for this talk:
  - Generally a single class of customers
  - Defining priorities to induce a global optimum when customers act selfishly, subject to those priorities
  - As a simpler approach to find the global optimum

Overview of the Second View

- Determining and characterizing **GO** = globally optimal policies is **often hard**.
- Determining and characterizing **IO** = individually optimal (selfish) policies, holding others' policies fixed, is **often easy**.
- If we can set individual priorities to make the two coincide (**IO = GO**), we have an easy route to solve a hard problem.
- The IO = GO approach is often easy to generalize,
- And it provides a simple means for (decentralized) implementation and computation.

Outline – 3 Examples

I. Very simple example: The stochastic sequential assignment problem
   - Without arrivals
   - With arrivals

II. Energy-aware scheduling (e.g., in web server farms)

III. Diminishing marginal returns to flexibility in a routing problem (e.g., in call centers)

I: Stochastic Sequential Assignment

- Worker i has value w_i, $\mathbf{w_1 \geq w_2 \geq \ldots \geq w_n}$
- Poisson job arrivals, job values X_i, i.i.d.
- Assigning a job of value **x** (job x) to a worker of value **w** earns a reward of **wx**
- Decision: Assign an arriving job to a particular worker or reject it (**no recall**)
- Objective: Maximize the total expected discounted reward, W_n
- Derman, Lieberman and Ross, 1972 (and many others)

Applications

- Kidney transplants
  - David and Yechiali (‘90,’95), Su and Zenios (2005)
- Aviation security
  - Nikolaev, Jacobson, and McLay (2010)
- Asset selling
  - Saario (1985)
- Flexible service
  - Akçay, Balakrishnan, and Xu (2010)
- Buying decisions in supply chains
  - You (2000)

Globally Optimal (GO) Policy

Worker weights: $w_1 \geq w_2 \geq \ldots \geq w_n$

Theorem: There exist thresholds

$$\infty =: v_0 > v_1 \geq v_2 \geq \ldots \geq v_n \geq v_{n+1} := 0$$

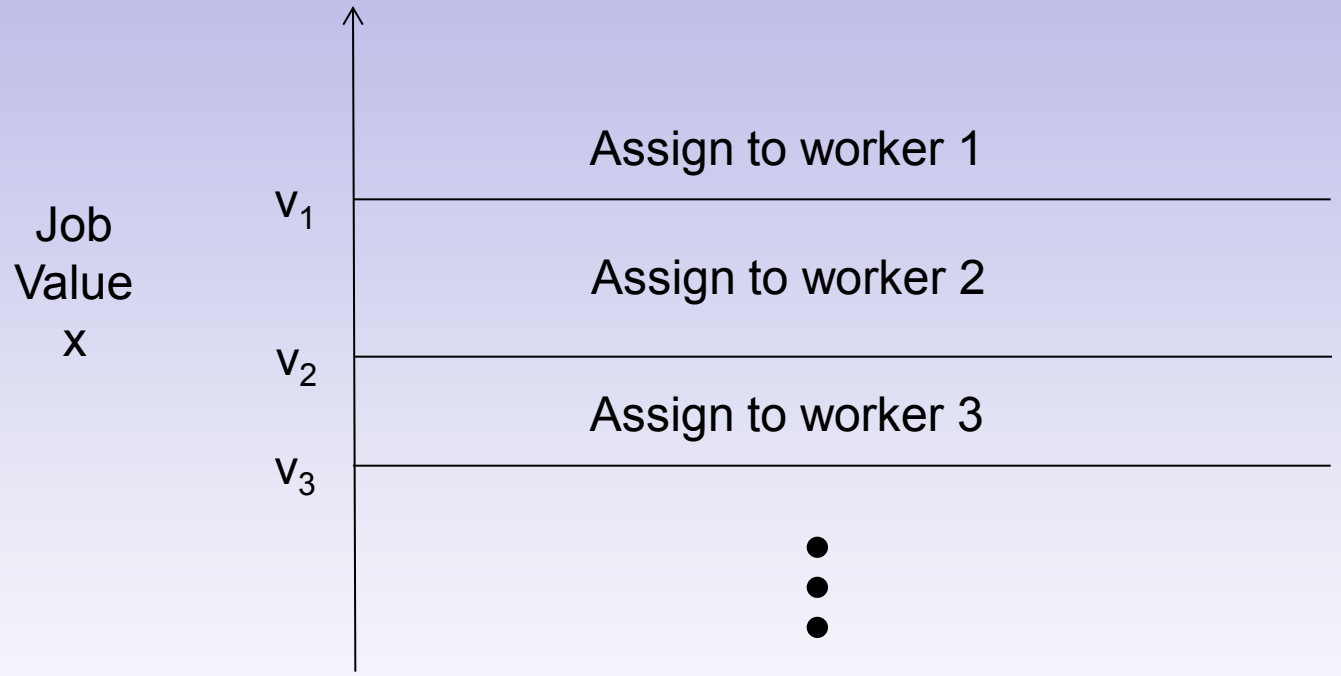
such that the optimal policy is to **assign job x** (an arriving job of value x) **to worker i** if

$$v_{i-1} > x \geq v_i$$

(assignment to n+1 = rejection)

regardless of w_i and of n.

Globally Optimal (GO) Policy



Workers “move up” after each assignment.

We’ll prove the GO structure by way of the selfish, individually optimal (IO) policy.

Selfish, or IO, Policy

- Worker i is given i'th priority
- Jobs are offered to workers **in priority order**
- Each worker tries to maximize its own discounted assigned job value, given all other workers are doing the same

IO Policy – Key Observation

Observation: Workers i+1,…,n have no effect on worker i for all i

Corollary: **Worker n imposes no externality** (worker n only gets the rejects)

Lemma: **The IO policy** for worker i is to **accept job x if $x \geq v_i$**, if x is offered to i, where **v_i = worker i's expected discounted job value for waiting under IO, for any $n \geq i$**

Proof: Almost immediate (stopping problem)

Theorem: IO = GO when $w_i \equiv 1$

$w_i \equiv 1$ ➔ workers are indistinguishable
➔ the GO decision: accept/reject job

For the IO policy, worker priorities are arbitrary

Theorem: When $w_i \equiv 1$, the individually optimal policy is globally optimal

➔The GO policy is to accept a job if $x \geq v_n$, and

V_n := optimal total expected discounted reward
$= \Sigma v_i$

Proof Sketch: IO = GO when $w_i \equiv 1$

(Backward) induction on finite horizon of N jobs.

N=1 is easy.

Suppose IO = GO for N-1, and suppose that for N and for the first job, with value x, GO = $\pi \neq$ IO (and π = IO for future jobs/decisions, by induction)

That is, suppose

π = GO

$\pi \neq$ IO for **first** decision, for job x, and

π = IO for **future** jobs/decisions

Proof Sketch: IO = GO when $w_i \equiv 1$

(Backward) induction on finite horizon of N jobs.

Suppose

π = GO

- $\pi \neq$ IO for **first** decision, for job x, and

π = IO for **future** jobs/decisions

We'll show $\pi \neq$ GO

- **Case 1:** IO accepts and π rejects job x
- **Case 2:** IO rejects and π accepts job x

Proof Sketch: IO = GO when $w_i \equiv 1$

π = GO

- π ≠ IO for **first** decision, for job x, and

π = IO for **future** jobs/decisions

- **Case 1:** IO accepts and π rejects job x ($x > v_i$):

Let **π'** assign x **to worker n** and then agree with IO.

Then **π' = π for workers 1,…,n-1**,

(under IO, **worker n has no externality**) and

π' ≻ π for worker n,

(worker n wants job x under IO)

So **π' ≻ π** for every worker, i.e., **π ≠ GO**.

Proof Sketch: IO = GO when $w_i \equiv 1$

Case 1: IO accepts and π rejects job x: **π ≠ GO**.

Case 2: IO rejects and π accepts job x ($x < v_i$):
(**π** assigns x **to worker n**, wlog: $w_i \equiv 1$)

Let **π' reject x** and then agree with IO. (So π' = IO)
Then **π' = π for workers 1,…,n-1**,
(under IO, **worker n has no externality**) and
π' ≻ π for worker n,
(worker n does not want job x under IO)

So **π' ≻ π** for every worker, i.e., **π ≠ GO**. ■

Proof Summary: IO = GO when $w_i \equiv 1$

Deviating from IO for the first time step (with the lowest priority worker) and then following IO

- hurts the lowest priority worker, and
- has no effect on the other workers
- because the lowest priority worker has no externality

➔ Such a deviation cannot be GO

$w_i \equiv 1$ ➔ Multiple GO Implementations

GO policy: Accept job x iff $x \geq v_n$.

Accepted jobs can be assigned to **any** worker, because workers are indistinguishable when $w_i \equiv 1$.

For example, a worker could be chosen at random (for fairness).

The IO implementation with priorities is only needed for the proof of GO = IO = threshold.

Corollary: IO = GO for any w_i's

Proof Sketch: $(w_1 \geq w_2 \geq \ldots \geq w_n)$

For an **arbitrary policy**, let

u_i = E[discounted job value assigned to worker i]

$U_i = \sum_{j=1}^{i} u_j$ (partial sum)

W_n = the total expected discounted reward

Then $W_n = \sum_{i=1}^{n} w_i u_i = \sum_{i=1}^{n} (w_i - w_{i+1}) \sum_{j=1}^{i} u_j = \sum_{i=1}^{n} (w_i - w_{i+1}) U_i$

IO = GO when $w_i \equiv 1$ ➔ IO maximizes U_i **for all i**

(job i imposes no externality on jobs 1,…,i-1)

➔ IO maximizes W_n because $w_i - w_{i+1} \geq 0$ ■

Corollary: IO = GO for any w_i's

Note that when the w_i's are not identical, we must maintain the IO priorities in the GO policy.

That is, workers with higher values must get higher priority.

Arriving Workers, $w_i \equiv 1$

New extension to the classic problem

Workers arrive according to a Poisson process with rate γ

IO policy: Each worker tries to maximize its own discounted assigned job value, given all other workers are doing the same, and given **LCFP** (**last-come-first-priority**), i.e., the most recently arriving worker has highest priority

Again, with LCFP, **the lowest priority worker imposes no externality** (e.g., Hassin, 1986)

Theorem: IO = GO when $w_i \equiv 1$

The **same IO proof** gives us (Righter, 2011):

Theorem: For $w_i \equiv 1$ and arriving workers,

IO = GO, and, under IO,

Worker i, i = 1,…,n, will **accept (offered) job x if $x \geq v_i$**, where v_i = worker i's expected discounted job value under the IO policy

➔ the GO policy is a **threshold policy**:
accept job x if $x \geq v_n$ (assign it to **any** worker) when n workers are in the system.

Or, equivalently, accept job x if $n \geq t(x)$, $t(x) \downarrow$ in x.

$w_i \equiv 1$ ➔ Multiple GO Implementations

For the IO policy to be globally optimal
(to produce incentive compatibility),
we need it to be implemented with LCFP.

However, because workers are identical,
once we have the thresholds from IO-LCFP,
we can implement the globally optimal policy
by assigning workers arbitrarily (e.g., FCFS)
to acceptable (above threshold v_n) jobs.

Arriving Workers and Arbitrary w_i's

Priorities for IO Policy?

Most reasonable candidate:

Higher priority for higher w_i's

LCFP within a class (with the same w_i)

Theorem: IO ≠ GO

Indeed, now the GO policy depends on the w_i's. The lowest priority worker in a given class poses an externality on workers in lower classes

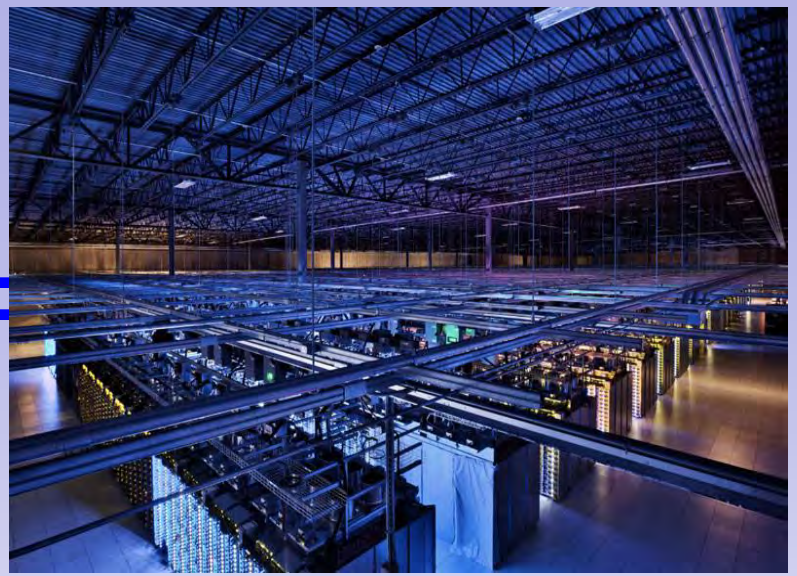
Extensions of IO = GO, threshold policy

- More general arrival processes
- More general discounting
- Finite horizon
- Randomly varying, possibly unknown job value distributions (with, e.g., Bayesian updating)
- Impatient workers

IO policy is still a (state-dependent) threshold policy, and IO = GO

Example II: Energy-Aware Scheduling

Motivation



- Server farms and cloud computing networks
- Energy consumption and greenhouse gas emission have become major concerns
- Servers vary in their processing speeds and energy consumption
- How do we schedule our work to **balance energy consumption and congestion?**

II. Energy-Aware Scheduling – Model

- n identical jobs with **holding cost rate h**
- m servers, exponential service times with **service rate μ_i** and **server usage cost β_i**
- Objective: Minimize total expected cost
- (Global) decision: Assign a job to an available server or leave the server idle
- Extension of "Slow-Server Problem," which has no server usage costs, $\beta_i = 0$

II. Comparison with SSAP (Example 1)

- Now the roles of **jobs** and **workers** (servers) are **interchanged**

I. Sequential assignment, basic case:
  - Fixed set of workers assigned to arriving jobs

II. Energy-aware scheduling:
  - Fixed set of jobs assigned to "arriving" servers

- The energy-aware scheduling problem is harder; now have "recall" of idle servers

II. Energy-Aware Scheduling

- n identical jobs with **holding cost rate h**
- m servers, exponential service times with **service rate μ_i** and **server usage cost β_i**
- Two cases:
  (i) Jobs can be **reassigned** to a new server at any time
  (ii) Jobs cannot be reassigned

  Surprisingly, (ii) is easier, so start with (ii)

(ii) No Reassignment – IO Policy

Theorem: IO policy is a **threshold policy**:

- Servers (j) are ordered in increasing order of

$$\frac{h+\beta_j}{\mu_j}$$ = E[cost for job 1 to use j]

- Job i will accept the lowest indexed available server, j, if it is offered to it and if

$$\frac{h+\beta_j}{\mu_j} \le v_i(j-1)$$ E[server j cost] ≤ E[cost to wait]

$v_i(j-1)$:= expected cost for job i if it waits for one of the servers 1,…,j-1

IO policy – “No recall”

Corollary 1: Under the IO policy, rejected servers will never be used

➔ Under the IO policy, lower priority jobs have no effect on higher priority jobs

➔Job n has no externality

Corollary 2:

The IO thresholds are easily computed

Theorem: IO = GO

Theorem: The individually optimal policy is also globally optimal

Proof outline. Same idea:

Deviating from IO for the first time step (with the lowest priority job) and then following IO

- hurts the lowest priority job, and
- has no effect on the other jobs
- because the lowest priority job has no externality

Corollary for GO policy

The GO policy is a threshold policy:
There are thresholds, t_j,
that are **easily computed via the IO policy**,
s.t. if j is the lowest indexed available server,
and if $n \geq t_j$,
the GO policy **assigns any job to server j regardless of the states of less preferred servers**

E-A Scheduling with Reassignment

- **Reassignment** – can move jobs from one server to another at any time, without penalty
- GO = IO = threshold policy (similar proof)
- Preference order of servers is **more difficult** (no longer an index)
- Threshold computations are **more difficult**
- Versus the **slow server problem** (no usage cost), optimal to use the fastest min{m,n} servers (m servers, n jobs) **easy**

E-A Scheduling – Arrivals, Reassignment

- **Arrivals** of new jobs
- **Reassignment** (easier with arrivals)
- Now the **priorities** for the IO policy must be **LCFP-P** (Preemptive LCFP)
  - Arriving (higher priority) jobs can preempt jobs in service and **can move** to different servers

➔ lower priority jobs have no effect on higher priority jobs, i.e.,

- **Lowest priority job has no externality**

E-A Scheduling – Arrivals, Reassignment

Theorem: With reassignment and arrivals,
IO = GO, a threshold policy

- Same proof
- Policy even more complicated: e.g., preference order for servers depends on arrival rate

E-A Scheduling – Arrivals, No Reassignment

- **Arrivals** of new jobs
- **No reassignment** and **Two** servers
- **IO Priorities**: **LCFP-P** (Preemptive LCFP)
  - Arriving (higher priority) jobs can preempt jobs in service, **but cannot move** (unless preempted)

➔ With **two servers**, lower priority jobs have no effect on higher priority jobs, i.e.,

  - Lowest priority job has no externality

E-A Scheduling – Arrivals, No Reassignment

Theorem: **Without reassignment** and **two** servers, **IO = GO, a threshold policy**

- Same proof
- Can implement GO with FCFS, without moving jobs

Theorem: Without reassignment and **more than two** servers, **IO ≠ GO**

- Now the lowest priority job in queue imposes an externality on other jobs

Energy-Aware Scheduling – Summary

Theorem: The IO policy is a threshold policy, and IO = GO, when

- No arrivals, with or without reassignment
- Arrivals, and jobs can be reassigned
- Arrivals, no reassignment, and 2 servers

IO ≠ GO with arrivals, no reassignment, > 2 servers

Conjecture: (arrivals, no reassignment, > 2 servers) GO is a threshold policy, server j threshold will depend on states of servers j+1,…m (like Weber's slow-server conjecture, but server preference order complicated)

III. Marginal Returns to Routing Flexibility

Motivation: Multi-lingual call center

with bilingual and monolingual customers:

Press 1 for English.
Press 2 for Spanish.

What is the benefit of inserting

Press 0 for bilingual?

How does it depend on

p: the proportion who are bilingual?

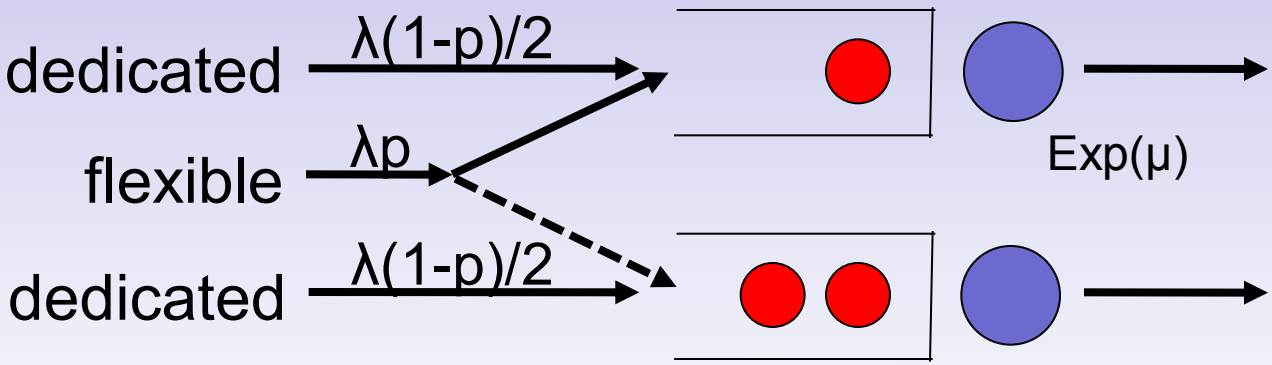
Mobile Millennium project (cell phones and traffic)

Product substitution in make-to-order systems

Internet packet routing

III. Model with Customer Flexibility

A proportion **p** of arrivals are **flexible**,
the rest are **dedicated** to a particular server

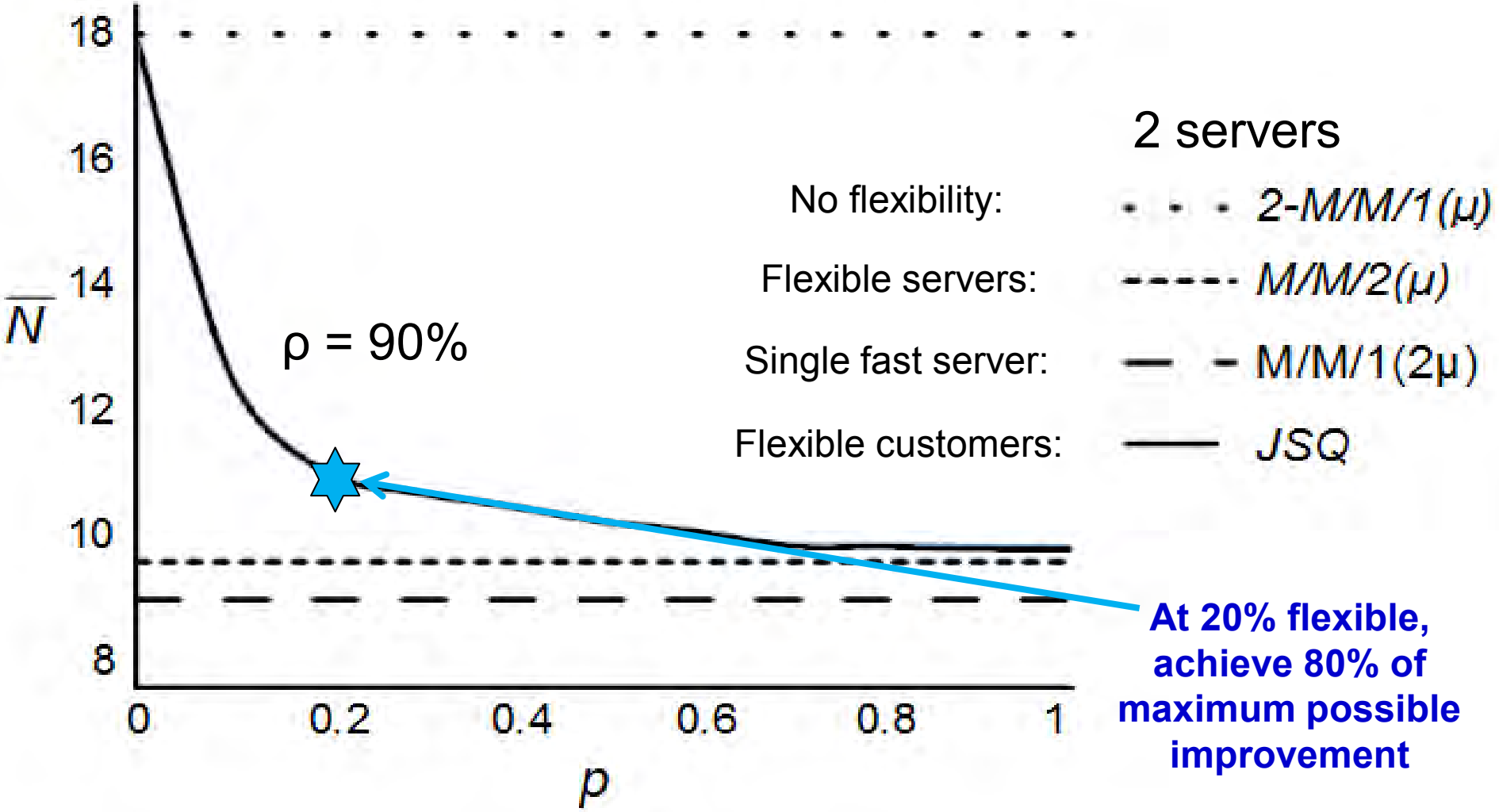


Flexible customers follow JSQ = Join the Shortest Queue

Total Arrival Process: Strongly mixing, state-independent, Bernoulli splitting

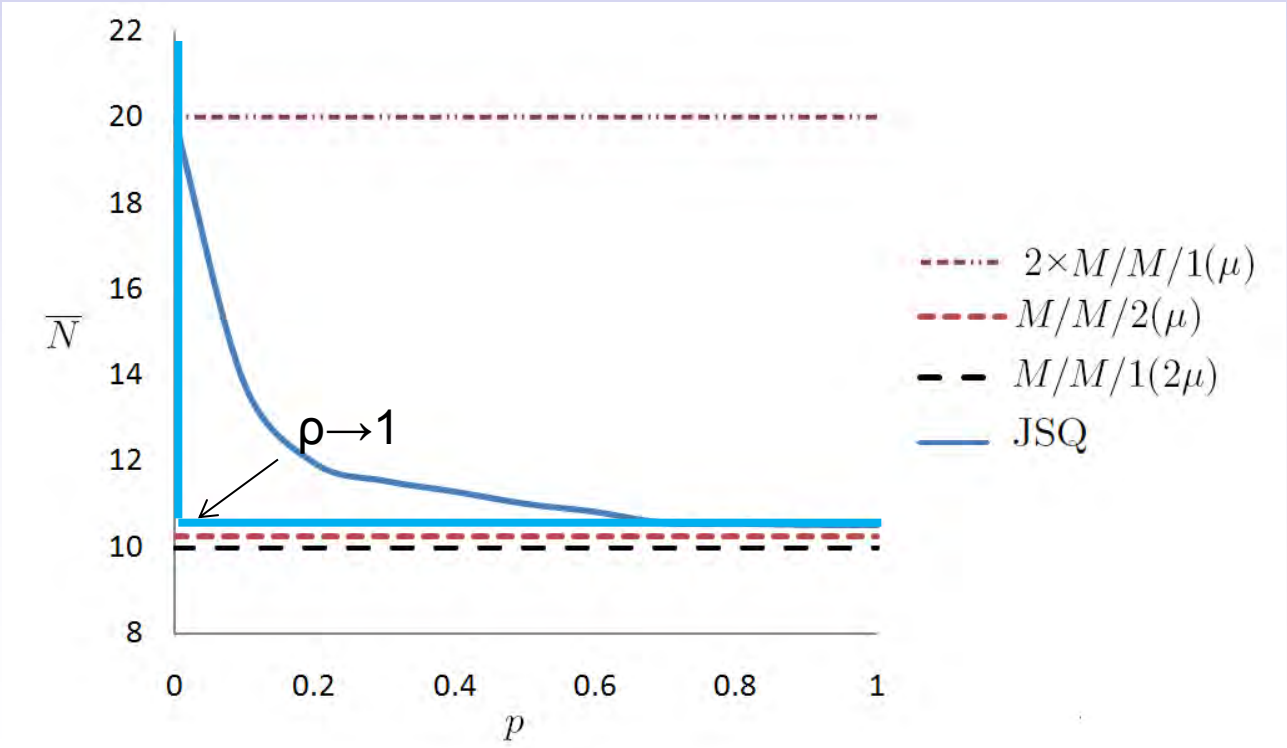
Stationary and ergodic

Mean # in system as a function of p



A little flexibility goes a long way

He and Down (2009): In the diffusion limit, the full benefit of customer flexibility is achieved for p arbitrarily close to 0



Steady-State Mean Waiting Time, W(p)

That **W(p) is decreasing in p** follows from the optimality of **JSQ = Join the shortest queue**.

Winston (1977), Weber (1978),
Ephremides, Varaiya and Walrand (1980),
Johri (1989), Hordijk and Koole (1990),
Menich and Serfozo (1991),
Sparaggis et al. (1990, 1993, 1994,1999),
Bambos and Michailidis (2002),
Movaghar (2005)

We will show monotonicity as the first step to convexity, by way of IO = GO.

Marginal Behavior of W(p)

Consider a single **tagged "ε" customer** that has **lowest preemptive priority**, and that may be flexible or dedicated.

➔ The ε customer has **no externality** on the other customers

➔ The ε customer "**embodies**" the marginal difference of more flexibility in the mean waiting time

IO for ε customer = GO

Y(p) := the **difference** in mean waiting time for the ε customer if it goes to the **long** queue rather than the **short** queue upon arrival

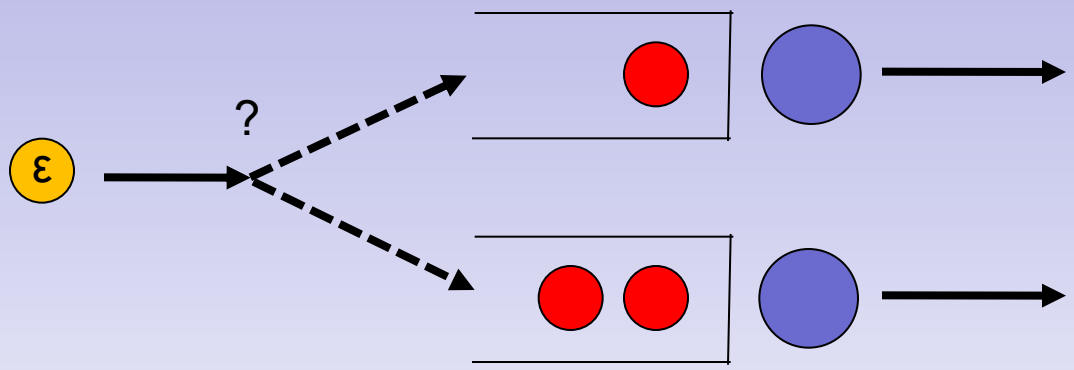
= "**the marginal (IO) flexibility advantage**"

Theorem: The ε customer embodies the GO flexibility advantage:

$$W'(p) = \lim_{\varepsilon \to 0} \frac{W(p+\varepsilon) - W(p)}{\varepsilon} = -Y(p)/2$$

The factor of 2 is because a dedicated customer will go to the short queue with probability ½.

The ε customer



Note that under **FCFS**:

JSQ = IO for the ε customer – immediate

Theorem: Given the ε customer has lowest preemptive priority, then JSQ = IO for the ε customer, i.e., $Y(p) \geq 0$.

➔ JSQ = GO and $W(p) \downarrow$ in p.

Y(p) ≥ 0, i.e., JSQ is IO – Proof Idea

Consider a sample path from the time the ε customer arrives until the first of

- The two queues (excluding the ε customer) are equal ➔ Y(p) = 0
- The ε customer leaves the short queue ➔ Y(p) ≥ 0 ■

➔ **JSQ is IO for the ε customer**

➔ **JSQ is GO (IO = GO) ➔ W(p) decreasing**

Convexity of W(p): Y'(p) ≤ 0

$$Y'(p) = \lim_{\delta \to 0} \frac{Y(p+\delta) - Y(p)}{\delta} = -W''(p)/2$$

Theorem: Y'(p) ≤ 0 (so W(p) is convex in p)

Proof Idea: In addition to the ε customer,

let there be a single "**δ**" customer

that is flexible in system 1 and dedicated in system 2,

and that has lowest preemptive priority, except with respect to the ε customer.

The δ customer "embodies" the marginal impact of more flexibility on the ε customer.

Proof Idea Continued

The "**δ**" customer is
flexible in system 1 and dedicated in system 2.

Consider coupled sample paths from the time the ε customer arrives until the first of

- The two queues (excluding the ε customer) are equal
  ➔ Y'(p) = 0
- The δ customer arrives and goes to the short queue in system 1, coin flip in system 2

  ε customer with JSQ may be delayed in system 1
  ➔ Y'(p) ≤ 0 ■

Extensions

- More than two queues
- Multiple-server stations
- Customer impatience (abandonment, reneging)
- Common random service rate (e.g., due to preventive maintenance)

Aside on Customer Flexibility

- Using **customer flexibility** may be much cheaper than creating **server flexibility** (cross training servers)
- The model and results also apply to cross training, p = proportion of customers that servers are cross trained to serve (e.g., easy or general queries)
- **Dedicated customers are better off** with more flexible customers: $W_d(p) \downarrow p$

Aside on Customer Flexibility

- JSQ alternative: Flexible customers "virtually" join all queues, and start service as early as possible
  ⇔ **JSW = join the smallest work**
- Or, for maximum efficiency,
  queue flexible customers separately and schedule optimally:
  DCF: Serve Dedicated Customers First
  Lose incentive compatibility
- **JSW is the overall best**:
  - W(p) is (empirically) almost as low as DCF
  - Incentive compatible

Conclusions

- Characterizing GO policies is often hard.
- Characterizing IO policies is often easy.
- To make the two coincide, we define a lowest priority worker/customer/job that “embodies” the (global) marginal effect of an extra worker, extra job, or extra flexibility, and has no externality.
- The IO perspective is often easier to generalize and it gives a simple means for implementation and computation.