Dispatching Problem with Fixed Size Jobs and Processor Sharing Discipline

E. Hyytiä, A. Penttinen, S. Aalto and J. Virtamo

Department of Communications and Networking
Aalto University, School of Electrical Engineering, Finland

7.9.2011
Dispatching problem to parallel queues

Upon arrival a job is routed to one of the $m$ servers
Dispatching problem to parallel queues

Upon arrival a job is routed to one of the $m$ servers

Each server processes jobs according to a certain scheduling discipline (e.g., PS)
Dispatching problem to parallel queues

Upon arrival a job is routed to one of the $m$ servers

Each server processes jobs according to a certain scheduling discipline (e.g., PS)

Objective: minimize the mean delay (mean sojourn time)
Dispatching problem to parallel queues

- Upon arrival a job is routed to one of the $m$ servers
- Each server processes jobs according to a certain scheduling discipline (e.g., PS)
- Objective: minimize the mean delay (mean sojourn time)
- Examples:
  - job assignment in supercomputing
  - traffic routing
  - web-server farms, and
  - other distributed computing systems
Heuristic policies

State-independent Policies:

1. **Bernoulli splitting (RND):**
   Choose queue in random using probabilities $p_i$:
   
   i) RND-U splits the arrival stream uniformly, $p_i = 1/m$
   ii) RND-$\rho$ balances the load, $p_i = c_i / \sum_j c_j$
   iii) RND-opt uses the $p_i$ that minimize the mean sojourn time
Heuristic policies

State-independent Policies:

1. **Bernoulli splitting (RND):**
   Choose queue in random using probabilities $p_i$:
   i) RND-U splits the arrival stream uniformly, $p_i = 1/m$
   ii) RND-$\rho$ balances the load, $p_i = c_i / \sum_j c_j$
   iii) RND-opt uses the $p_i$ that minimize the mean sojourn time

State-dependent Policies:

1. **Join-the-Shortest-Queue (JSQ):**
   Optimal when Poisson arrivals, Exponential jobs, identical servers, and only the occupancy is known (Winston, 1977).
Heuristic policies

State-independent Policies:

1. Bernoulli splitting (RND):
   Choose queue in random using probabilities $p_i$:
   i) RND-U splits the arrival stream uniformly, $p_i = 1/m$
   ii) RND-$\rho$ balances the load, $p_i = c_i/\sum_j c_j$
   iii) RND-opt uses the $p_i$ that minimize the mean sojourn time

State-dependent Policies:

1. Join-the-Shortest-Queue (JSQ):
   Optimal when Poisson arrivals, Exponential jobs, identical servers, and only the occupancy is known (Winston, 1977).

2. Round-robin (RR):
   Optimal with identical servers that were initially in a same state (Ephremides et. al, 1980).
Heuristic policies

State-independent Policies:

1. **Bernoulli splitting (RND):**
   Choose queue in random using probabilities $p_i$:
   i) RND-U splits the arrival stream uniformly, $p_i = 1/m$
   ii) RND-$\rho$ balances the load, $p_i = c_i / \sum_j c_j$
   iii) RND-opt uses the $p_i$ that minimize the mean sojourn time

State-dependent Policies:

1. **Join-the-Shortest-Queue (JSQ):**
   Optimal when Poisson arrivals, Exponential jobs, identical servers, and only the occupancy is known (Winston, 1977).

2. **Round-robin (RR):**
   Optimal with identical servers that were initially in a same state (Ephremides et. al, 1980).

3. **Least-Work-Left (LWL):**
   Pick the queue with the shortest backlog (Sharifnia, 1997).
State-aware dispatching with constant job size

- Poisson arrival process, rate $\lambda$
State-aware dispatching with constant job size

- Poisson arrival process, rate $\lambda$
- Fixed job size $d$
State-aware dispatching with constant job size

- Poisson arrival process, rate $\lambda$
- Fixed job size $d$
- $m$ parallel heterogeneous servers:

![Diagram showing Poisson arrival process, dispatching, and PS-queues]
State-aware dispatching with constant job size

- Poisson arrival process, rate $\lambda$
- Fixed job size $d$
- $m$ parallel heterogeneous servers:
  - Server specific processing rates $c_i$
State-aware dispatching with constant job size

- Poisson arrival process, rate $\lambda$
- Fixed job size $d$
- $m$ parallel heterogeneous servers:
  - Server specific processing rates $c_i$
  - Processor Sharing (PS) scheduling discipline
State-aware dispatching with constant job size

- Poisson arrival process, rate $\lambda$
- Fixed job size $d$
- $m$ parallel heterogeneous servers:
  - Server specific processing rates $c_i$
  - Processor Sharing (PS) scheduling discipline
- Queue states (remaining service times) are known to the dispatcher
State-aware dispatching with constant job size

- Poisson arrival process, rate $\lambda$
- Fixed job size $d$
- $m$ parallel heterogeneous servers:
  - Server specific processing rates $c_i$
  - Processor Sharing (PS) scheduling discipline
- Queue states (remaining service times) are known to the dispatcher
- Objective: minimize the mean delay
Delay costs and relative value

Delay costs are accrued at rate

\[ N_z(t) \triangleq \text{"the number of jobs in the system"}, \]

where \( z \) denotes the initial state at time \( t = 0 \).
Delay costs and relative value

Delay costs are accrued at rate

\[ N_z(t) \triangleq \text{"the number of jobs in the system"}, \]

where \( z \) denotes the initial state at time \( t = 0 \).

Delay costs accrued during \( (0, t) \):

\[ V_z(t) \triangleq \int_0^t N_z(s) \, ds. \]
Delay costs and relative value

Delay costs are accrued at rate

\[ N_z(t) \triangleq \text{"the number of jobs in the system"}, \]

where \( z \) denotes the initial state at time \( t = 0 \).

Delay costs accrued during \((0, t)\):

\[ V_z(t) \triangleq \int_0^t N_z(s) \, ds. \]

Relative value: the expected difference in the cumulative costs between a system initially in state \( z \) and a system in equilibrium,

\[
\nu_z \triangleq \lim_{t \to \infty} E[V_z(t) - r \, t]
\]

\[
= \lim_{t \to \infty} \left( E\left[ \int_0^t N_z(s) \, ds \right] - E[N] \, t \right).
\]
Approach: MDP and first policy iteration (FPI)

- Size- and state-aware setting; future arrivals not known
Approach: MDP and first policy iteration (FPI)

- Size- and state-aware setting; future arrivals not known
- Idea: start with a reasonable basic dispatching policy, and carry out the first policy iteration (FPI) step
Approach: MDP and first policy iteration (FPI)

- Size- and state-aware setting; future arrivals not known
- Idea: start with a reasonable basic dispatching policy, and carry out the first policy iteration (FPI) step
- Policy iteration finds the optimal policy, and the FPI step typically yields the highest improvement.
Approach: MDP and first policy iteration (FPI)

- Size- and state-aware setting; future arrivals not known
- Idea: start with a reasonable basic dispatching policy, and carry out the first policy iteration (FPI) step
- Policy iteration finds the optimal policy, and the FPI step typically yields the highest improvement.
- Requires the relative values of states $v_z$
Approach: MDP and first policy iteration (FPI)

- Size- and state-aware setting; future arrivals not known
- Idea: start with a reasonable basic dispatching policy, and carry out the first policy iteration (FPI) step
- Policy iteration finds the optimal policy, and the FPI step typically yields the highest improvement.
- Requires the relative values of states $v_z$
- However, our state-space is extremely complex (remaining service requirements at each queue)
Decomposition to independent M/D/1-PS queues

- Deriving a relative value is generally a difficult task.
Decomposition to independent M/D/1-PS queues

- Deriving a relative value is generally a difficult task.
- However, any state-independent policy feeds each server jobs according to a Poisson process (cf. Bernoulli split)

![Diagram of M/D/1-PS queues with lambda and RND nodes]
Decomposition to independent M/D/1-PS queues

- Deriving a relative value is generally a difficult task.
- However, any **state-independent policy** feeds each server jobs according to a Poisson process (cf. Bernoulli split)

Analyze single M/D/1-PS queues instead?
FPI of state-independent basic policy

Figure: FPI considers a single decision, after which one falls back to the basic policy. Can we solve the latter exactly?
FPI of state-independent basic policy

\[ \lambda_1 + \lambda_2 \]

arrivals

\[ ? \]

Dispatch

\[ c_1 \rightarrow \]

queue states

\[ c_2 \rightarrow \]

\[ \Rightarrow \]

later arrivals

\[ \lambda_1 \]

arrivals

\[ ? \]

this task

\[ \lambda_2 \]

later arrivals

\[ ? \]

queue states

\[ \Rightarrow \]

Figure: FPI considers a single decision, after which one falls back to the basic policy.
FPI of state-independent basic policy

Figure: FPI considers a single decision, after which one falls back to the basic policy.

Can we solve the latter exactly?
Figure: FPI of state-independent basic policy: later arrivals are dispatched according to the basic policy, isolating the queues.
FPI of state-independent basic policy

Figure: FPI of state-independent basic policy: later arrivals are dispatched according to the basic policy, isolating the queues.

The relative values $v_{z_1}$ and $v_{z_2}$ tell us which is the better option!
FPI of state-independent basic policy

State of only one queue changes.

Figure: Comparison between two states in each queue.
FPI of state-independent basic policy

Later arrivals \( \lambda_1 \) later arrivals

PS

Queue states

Later arrivals \( \lambda_2 \)

State of only one queue changes.

Figure: Comparison between two states in each queue.

Increments in the queue specific relative values \( \nu_z^{(1)} \) and \( \nu_z^{(2)} \) tell us which queue to choose!
Roadmap

1. Assume a state-independent basic policy.
Roadmap

1. Assume a state-independent basic policy.
2. Derive relative values for an “isolated queue”.

\[ v_z = \sum_i v_{z_i} \]

4. Carry out FPI ⇒ new efficient dispatching policy.

In practice, it is sufficient to know, e.g., \( v_z - v_0 \).

Next step: Derive \( v_z - v_0 \) for an M/D/1-PS queue.
Roadmap

1. Assume a state-independent basic policy.
2. Derive relative values for an “isolated queue”.
3. Relative value of the whole system is the sum of the queue specific relative values:
   \[ v_z = \sum_i v_{z_i}. \]
Roadmap

1. Assume a state-independent basic policy.
2. Derive relative values for an “isolated queue”.
3. Relative value of the whole system is the sum of the queue specific relative values:
\[ v_z = \sum_i v_{z_i} \]
4. Carry out FPI \( \Rightarrow \) new efficient dispatching policy.
Roadmap

1. Assume a state-independent basic policy.
2. Derive relative values for an “isolated queue”.
3. Relative value of the whole system is the sum of the queue specific relative values:

\[ v_z = \sum_i v_{zi}. \]

4. Carry out FPI ⇒ new efficient dispatching policy.

In practice, it is sufficient to know, e.g., \( v_z - v_0 \).

Next step: Derive \( v_z - v_0 \) for an M/D/1-PS queue.
Relative value for an M/D/1-PS queue

**Notation:**
- $\lambda$ is the Poisson arrival rate.
Relative value for an M/D/1-PS queue

Notation:

- $\lambda$ is the Poisson arrival rate.
- $\rho = \lambda d$ and $d$ denotes the fixed job size.

Proposition: The size-aware relative value of state $z = (\Delta_1, \ldots, \Delta_n)$ with respect to the delay in an M/D/1-PS queue is given by

$$v(\Delta_1, \ldots, \Delta_n) - v_0 = \lambda 1 - \rho u z - u z + 2 \sum_{i=1}^{n} i \Delta_i.$$ (1)

where $v_0$ denotes the relative value of an empty system, and $u z = \sum_{i} \Delta_i$ the backlog in the queue.
Relative value for an M/D/1-PS queue

Notation:

- $\lambda$ is the Poisson arrival rate.
- $\rho = \lambda d$ and $d$ denotes the fixed job size.
- $z = (\Delta_1; ..; \Delta_n)$ are the remaining service times, $\Delta_i > \Delta_{i+1}$
Relative value for an M/D/1-PS queue

**Notation:**

- $\lambda$ is the Poisson arrival rate.
- $\rho = \lambda d$ and $d$ denotes the fixed job size.
- $z = (\Delta_1; ..; \Delta_n)$ are the remaining service times, $\Delta_i > \Delta_{i+1}$

**Proposition:** The size-aware relative value of state $z$ with respect to the delay in an M/D/1-PS queue is given by

$$v(\Delta_1;..;\Delta_n) - v_0 = \frac{\lambda}{1 - \rho} u_z^2 - u_z + 2 \sum_{i=1}^{n} i \Delta_i. \quad (1)$$

where $v_0$ denotes the relative value of an empty system, and $u_z = \sum_i \Delta_i$ the backlog in the queue.
Proof sketched

▷ Consider two systems under the same arrivals:
  ▷ S1 initially in state \( z = (\Delta_1; \ldots; \Delta_n) \) with \( \Delta_1 \geq \ldots \geq \Delta_n \).
  ▷ S2 initially empty.

Each arrival increases the total delay (immediate cost) \( z = \tau (\Delta_1; \ldots; \Delta_n) - \tau (\Delta_1; \ldots; \Delta_n) = 2u_z + d \). (3)
Proof sketched

- Consider two systems under the same arrivals:
  - S1 initially in state $z = (\Delta_1; \ldots; \Delta_n)$ with $\Delta_1 \geq \ldots \geq \Delta_n$.
  - S2 initially empty.
- Once S1 is empty, the two systems behave equivalently.
Consider two systems under the same arrivals:
- S1 initially in state \( z = (\Delta_1; \ldots; \Delta_n) \) with \( \Delta_1 \geq \ldots \geq \Delta_n \).
- S2 initially empty.
Once S1 is empty, the two systems behave equivalently.
Without new arrivals, the total delay accrued in S1 is
\[
\tau_z = \Delta_n \cdot n^2 + (\Delta_{n-1} - \Delta_n) \cdot (n - 1)^2 + \ldots + (\Delta_1 - \Delta_2),
\]
\[
= \sum_{i=1}^{n} (2i - 1) \Delta_i. \tag{2}
\]
Proof sketched

Consider two systems under the same arrivals:
- S1 initially in state $\mathbf{z} = (\Delta_1; \ldots; \Delta_n)$ with $\Delta_1 \geq \ldots \geq \Delta_n$.
- S2 initially empty.

Once S1 is empty, the two systems behave equivalently.

Without new arrivals, the total delay accrued in S1 is

$$
\tau_\mathbf{z} = \Delta_n \cdot n^2 + (\Delta_{n-1} - \Delta_n) \cdot (n - 1)^2 + \ldots + (\Delta_1 - \Delta_2),
$$

$$
= \sum_{i=1}^{n} (2i - 1)\Delta_i. \tag{2}
$$

Each arrival increases the total delay (immediate cost)

$$
\mathbf{s}_\mathbf{z} = \tau(d; \Delta_1; \ldots; \Delta_n) - \tau(\Delta_1; \ldots; \Delta_n) = 2u_\mathbf{z} + d. \tag{3}
$$
Proof sketched

- Consider two systems under the same arrivals:
  - S1 initially in state $\mathbf{z} = (\Delta_1; \ldots; \Delta_n)$ with $\Delta_1 \geq \ldots \geq \Delta_n$.
  - S2 initially empty.
- Once S1 is empty, the two systems behave equivalently.
- Without new arrivals, the total delay accrued in S1 is
  \[
  \tau_{\mathbf{z}} = \Delta_n \cdot n^2 + (\Delta_{n-1} - \Delta_n) \cdot (n-1)^2 + \ldots + (\Delta_1 - \Delta_2),
  \]
  \[
  = \sum_{i=1}^{n} (2i - 1)\Delta_i. \tag{2}
  \]
- Each arrival increases the total delay (immediate cost)
  \[
  s_{\mathbf{z}} = \tau(d; \Delta_1; \ldots; \Delta_n) - \tau(\Delta_1; \ldots; \Delta_n) = 2u_{\mathbf{z}} + d. \tag{3}
  \]
- Utilize the lack of memory of Poisson arrivals.
- Virtual busy periods similar (S1 has an offset in backlog)
  ⇒ the mean contribution of a busy period.
Proof sketched

- Consider two systems under the same arrivals:
  - S1 initially in state $z = (\Delta_1; \ldots; \Delta_n)$ with $\Delta_1 \geq \ldots \geq \Delta_n$.
  - S2 initially empty.
- Once S1 is empty, the two systems behave equivalently.
- Without new arrivals, the total delay accrued in S1 is
  \[
  \tau_z = \Delta_n \cdot n^2 + (\Delta_{n-1} - \Delta_n) \cdot (n-1)^2 + \ldots + (\Delta_1 - \Delta_2),
  \]
  \[
  = \sum_{i=1}^{n} (2i - 1) \Delta_i. \tag{2}
  \]
- Each arrival increases the total delay (immediate cost)
  \[
  s_z = \tau(d; \Delta_1; \ldots; \Delta_n) - \tau(\Delta_1; \ldots; \Delta_n) = 2u_z + d. \tag{3}
  \]
- Utilize the lack of memory of Poisson arrivals.
- Virtual busy periods similar (S1 has an offset in backlog),
  \Rightarrow the mean contribution of a busy period.
- Details in the paper.
Cost of a new task in M/D/1-PS

**Corollary:** The expected cost due to accepting a new task to an M/D/1-PS queue at state \( z = (\Delta_1; ..; \Delta_n) \) is given by

\[
    w_z = v(d;\Delta_1;..;\Delta_n) - v(\Delta_1;..;\Delta_n) = \frac{2u_z + d}{1 - \rho}.
\]

(4)
Cost of a new task in M/D/1-PS

**Corollary:** The expected cost due to accepting a new task to an M/D/1-PS queue at state $z = (\Delta_1; ..; \Delta_n)$ is given by

$$w_z = v(d; \Delta_1; ..; \Delta_n) - v(\Delta_1; ..; \Delta_n) = \frac{2u_z + d}{1 - \rho}. \quad (4)$$

That is, the immediate cost divided by $1 - \rho$. 

---

Preemptive M/G/1-LIFO: Immediate cost in an M/G/1-LIFO is $(n + 1)x$, where $x$ is the size of the new task. Similarly, the expected cost due to accepting a new task with size $x$ is $w_z = (n + 1)x / (1 - \rho)$, i.e., the immediate cost divided by $1 - \rho$. 

---

Aalto University School of Electrical Engineering

7.9.2011 23rd ITC, San Francisco, USA 14/22
Cost of a new task in M/D/1-PS

**Corollary:** The expected cost due to accepting a new task to an M/D/1-PS queue at state \( z = (\Delta_1; \ldots; \Delta_n) \) is given by

\[
    w_z = v(d; \Delta_1; \ldots; \Delta_n) - v(\Delta_1; \ldots; \Delta_n) = \frac{2 u_z + d}{1 - \rho}.
\]

That is, the immediate cost divided by \( 1 - \rho \).

**Preemptive M/G/1-LIFO:** Immediate cost in an M/G/1-LIFO is \((n + 1)x\), where \( x \) is the size of the new task.
Cost of a new task in M/D/1-PS

**Corollary:** The expected cost due to accepting a new task to an M/D/1-PS queue at state $\mathbf{z} = (\Delta_1;..;\Delta_n)$ is given by

$$w_{\mathbf{z}} = v(d;\Delta_1;..;\Delta_n) - v(\Delta_1;..;\Delta_n) = \frac{2\,u_{\mathbf{z}} + d}{1 - \rho}. \quad (4)$$

That is, the immediate cost divided by $1 - \rho$.

**Preemptive M/G/1-LIFO:** Immediate cost in an M/G/1-LIFO is $(n+1)x$, where $x$ is the size of the new task. Similarly, the expected cost due to accepting a new task with size $x$ is

$$w_{\mathbf{z}} = \frac{(n+1)x}{1 - \rho},$$

i.e., the immediate cost divided by $1 - \rho$. 
First policy iteration (FPI) with M/D/1-PS

- Assume: relative values $v_z$ are available for basic policy

\[
\alpha(z) \equiv \arg\min_i (v_{z'}(i) - v_z) = \arg\min_i w_z(i)
\]

where $z'(i)$ is the new state if the job is added to queue $i$.

"Choose the action with the smallest expected future cost"

Basic policy $\rho$ balances load, $\rho_i = \rho_j$, and FPI reduces to

\[
\alpha(z) = \arg\min_i (u_i(z) + 0.5d_i)
\]
First policy iteration (FPI) with M/D/1-PS

- Assume: relative values \( v_z \) are available for basic policy.
- Improved decision according to FPI at state \( z \):

\[
\alpha(z) \triangleq \arg\min_i \left(v_{z'(i)} - v_z\right) = \arg\min_i w_{z(i)}
\]

where \( z'(i) \) is the new state if the job is added to queue \( i \).
First policy iteration (FPI) with M/D/1-PS

- Assume: relative values $v_z$ are available for basic policy
- Improved decision according to FPI at state $z$:

$$
\alpha(z) \triangleq \arg\min_i \left( v_{z'(i)} - v_z \right) = \arg\min_i w_{z(i)}
$$

where $z'(i)$ is the new state if the job is added to queue $i$.

"Choose the action with the smallest expected future cost"
First policy iteration (FPI) with M/D/1-PS

- Assume: relative values $v_z$ are available for basic policy
- Improved decision according to FPI at state $z$:

$$\alpha(z) \triangleq \arg\min_i (v_{z'}(i) - v_z) = \arg\min_i w_{z(i)}$$

where $z'(i)$ is the new state if the job is added to queue $i$.

"Choose the action with the smallest expected future cost"

- Basic policy RND-$\rho$ balances load, $\rho_i = \rho_j$, and FPI reduces to

$$\alpha(z) = \arg\min_i (u_i(z) + 0.5 d_i).$$
Policy family $\mathcal{P}(\beta)$

Policy family $\mathcal{P}(\beta)$ with policy parameter $\beta$ is defined by

$$\arg\min_i u_i(z) + \beta \cdot d_i.$$ 

| LWL$^-\ $ | $\beta = 0$ | “smallest backlog before” |
| LWL$^+\ $ | $\beta = 1$ | “smallest backlog afterwards” |
| FPI-$\rho\ $ | $\beta = 0.5$ | “compromise between the above” |
Policy family $\mathcal{P}(\beta)$

Policy family $\mathcal{P}(\beta)$ with policy parameter $\beta$ is defined by

$$\arg\min_{i} u_i(z) + \beta \cdot d_i.$$ 

| LWL$^-$ | $\beta = 0$ | “smallest backlog before” |
| LWL$^+$ | $\beta = 1$ | “smallest backlog afterwards” |
| FPI-$\rho$ | $\beta = 0.5$ | “compromise between the above” |

State-dependent policies in $\mathcal{P}(\beta)$ are of the switch-over type:
Numerical examples

Performance metrics:

1. Absolute mean delay (sojourn time)
2. Relative delay when compared to FPI policy
Numerical examples

Performance metrics:
1. Absolute mean delay (sojourn time)
2. Relative delay when compared to FPI policy

Scenarios:
1. Symmetric case with two identical servers
2. Asymmetric case with two heterogeneous servers

Additionally, policy optimization within $\mathcal{P}$
Identical servers

Two policies, i) RND-U, ii) LWL/FPI/RR, and single server

- Left: resulting mean sojourn time
- Right: relative performance against the LWL
Identical servers

- Two policies, i) RND-U, ii) LWL/FPI/RR, and single server
- Left: resulting mean sojourn time
- Right: relative performance against the LWL
- Optimal state-independent policy: RND-U
Two policies, i) RND-U, ii) LWL/FPI/RR, and single server

Left: resulting mean sojourn time

Right: relative performance against the LWL

Optimal state-independent policy: RND-U

Optimal state-dependent policy: LWL/FPI-U/RR,

“Choose the queue with a smaller backlog”
Asymmetric servers: $d_1 = 1$ and $d_2 = 4$

- Left: mean sojourn time
- Right: relative performance against the FPI-$\rho$ policy
Asymmetric servers: \( d_1 = 1 \) and \( d_2 = 4 \)

- Left: mean sojourn time
- Right: relative performance against the FPI-\( \rho \) policy
- Both LWL policies are clearly suboptimal
Asymmetric servers: \( d_1 = 1 \) and \( d_2 = 4 \)

- Left: mean sojourn time
- Right: relative performance against the FPI-\( \rho \) policy
- Both LWL policies are clearly suboptimal
- FPI-\( \rho \) makes very good dispatching decisions for all \( \rho \)
Asymmetric servers: $d_1 = 1$ and $d_2 = 4$

- Left: mean sojourn time
- Right: relative performance against the FPI-$\rho$ policy
- Both LWL policies are clearly suboptimal
- FPI-$\rho$ makes very good dispatching decisions for all $\rho$
- Gray area: optimal policy from $\mathcal{P}(\beta)$, defined by

$$u_i(z) + \beta \cdot d_i.$$
Policy optimization in $\mathcal{P}(\beta)$

- Two servers, $d_1 = 1$ and $d_2 = 4$
Policy optimization in $\mathcal{P}(\beta)$

- Two servers, $d_1 = 1$ and $d_2 = 4$
- $x$-axis: policy parameter $\beta$
- $y$-axis: arrival rate $\lambda$
- $z$-axis: mean delay relative to the optimal at given $\lambda$
Policy optimization in $\mathcal{P}(\beta)$

- Two servers, $d_1 = 1$ and $d_2 = 4$
- x-axis: policy parameter $\beta$
- y-axis: arrival rate $\lambda$
- z-axis: mean delay relative to the optimal at given $\lambda$
- Valley: delay is within 1% from the minimum at given $\lambda$
Policy optimization in $\mathcal{P}(\beta)$

- Two servers, $d_1 = 1$ and $d_2 = 4$
- $x$-axis: policy parameter $\beta$
  $y$-axis: arrival rate $\lambda$
  $z$-axis: mean delay relative to the optimal at given $\lambda$
- Valley: delay is within 1% from the minimum at given $\lambda$
- FPI-$\rho$ ($\beta = 0.5$) close to optimal optimal (within $\mathcal{P}$)
Conclusions

- Size- and state-aware dispatching problem can be approached in MDP framework.
Conclusions

- Size- and state-aware dispatching problem can be approached in MDP framework
- FPI requires the relative values of the basic policy

Thanks!
Conclusions

- Size- and state-aware dispatching problem can be approached in MDP framework
- FPI requires the relative values of the basic policy
- For a state-independent basic policy, sufficient to analyze M/D/1-PS queue in isolation
Conclusions

- Size- and state-aware dispatching problem can be approached in MDP framework
- FPI requires the relative values of the basic policy
- For a state-independent basic policy, sufficient to analyze M/D/1-PS queue in isolation
- We give the relative value for a size-aware M/D/1-PS

General case of M/G/1-PS seems to be difficult, however, exact result for a size-aware M/M/1-PS is also available (Hyytiä et. al, Performance 2011)

- For FCFS, LCFS, SPT and SRPT, the size-aware relative values are available for M/G/1 (submitted)

Thanks!
Conclusions

- Size- and state-aware dispatching problem can be approached in MDP framework
- FPI requires the relative values of the basic policy
- For a state-independent basic policy, sufficient to analyze M/D/1-PS queue in isolation
- We give the relative value for a size-aware M/D/1-PS
- General case of M/G/1-PS seems to be difficult, however, exact result for a size-aware M/M/1-PS is also available (Hyytiä et. al, Performance 2011)

Thanks!
Conclusions

▶ Size- and state-aware dispatching problem can be approached in MDP framework
▶ FPI requires the relative values of the basic policy
▶ For a state-independent basic policy, sufficient to analyze M/D/1-PS queue in isolation
▶ We give the relative value for a size-aware M/D/1-PS
▶ General case of M/G/1-PS seems to be difficult, however, exact result for a size-aware M/M/1-PS is also available (Hyytiä et. al, Performance 2011)
▶ For FCFS, LCFS, SPT and SRPT, the size-aware relative values are available for M/G/1 (submitted)
Conclusions

▶ Size- and state-aware dispatching problem can be approached in MDP framework
▶ FPI requires the relative values of the basic policy
▶ For a state-independent basic policy, sufficient to analyze M/D/1-PS queue in isolation
▶ We give the relative value for a size-aware M/D/1-PS
▶ General case of M/G/1-PS seems to be difficult, however, exact result for a size-aware M/M/1-PS is also available (Hyytiä et. al, Performance 2011)
▶ For FCFS, LCFS, SPT and SRPT, the size-aware relative values are available for M/G/1 (submitted)

Thanks!
References:

1. E. Hyytiä, A. Penttinen and S. Aalto, 
   *Size- and State-Aware Dispatching Problem with Queue-Specific Job Sizes*, 

2. E. Hyytiä, A. Penttinen, S. Aalto and J. Virtamo, 
   *Dispatching problem with fixed size jobs and processor sharing discipline*, 
   in 23rd International Teletraffic Congress (ITC’23), September 2011, San 
   Fransisco, USA.

3. E. Hyytiä, J. Virtamo, S. Aalto and A. Penttinen, 
   *M/M/1-PS Queue and Size-Aware Task Assignment*, 
   in IFIP PERFORMANCE, October 2011, Amsterdam, Netherlands, (to appear).