Stochastic Bin Packing with Time-Varying Item Sizes

Joint work with Yige Hong (CMU) and Qiaomin Xie (UW–Madison)

Weina Wang
Carnegie Mellon University
The problem

items

bins
The problem

• Each arriving item needs to be assigned to a bin
The problem

• Each arriving item needs to be assigned to a bin
• Infinite # bins
The problem

- Each arriving item needs to be assigned to a bin
- Infinite # bins
- Each bin has a capacity $M$
The problem

• Each arriving job needs to be assigned to a bin
• Infinite # bins
• Each bin has a capacity $M$
The problem

• Each arriving job needs to be assigned to a server
• Infinite # servers
• Each server has a resource capacity $M$
The problem

• Each arriving job needs to be assigned to a server
• Infinite # servers
• Each server has a resource capacity $M$

Traditional job model:
• Each job has a fixed resource requirement
• Each job departs after a random time

servers

jobs

$M$
The problem

• Each arriving job needs to be assigned to a server
• Infinite # servers
• Each server has a resource capacity $M$

Traditional job model:
• Each job has a fixed resource requirement
• Each job departs after a random time
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity $M$

Traditional job model:
- Each job has a fixed resource requirement
- Each job departs after a random time
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity $M$

Traditional job model:
- Each job has a fixed resource requirement
- Each job departs after a random time
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity $M$

Traditional job model:
- Each job has a fixed resource requirement
- Each job departs after a random time
The problem

• Each arriving job needs to be assigned to a server
• Infinite # servers
• Each server has a resource capacity $M$

Traditional job model:
• Each job has a fixed resource requirement
• Each job departs after a random time
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity $M$

Traditional job model:
- Each job has a fixed resource requirement
- Each job departs after a random time
The problem

• Each arriving job needs to be assigned to a server
• Infinite # servers
• Each server has a resource capacity $M$

Traditional job model:
• Each job has a fixed resource requirement
• Each job departs after a random time
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity $M$

Traditional job model:
- Each job has a fixed resource requirement
- Each job departs after a random time
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity $M$

Traditional job model:
- Each job has a fixed resource requirement
- Each job departs after a random time

**Goal:** minimize $\mathbb{E} \left[ \text{# active servers} \right]$ job assigning policy
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity $M$

Traditional job model:

- Each job has a fixed resource requirement
- Each job departs after a random time

**Goal:** minimize $E$ [# active servers]

Prior work: algorithms with asymptotic optimality

[Stolyar and Zhong 2013, 2015], [Stolyar 2017], [Stolyar and Zhong 2021], …
The problem

- Each arriving **job** needs to be assigned to a **server**
- Infinite # **servers**
- Each **server** has a resource capacity $M$

A new job model:
- Each job has a fixed resource requirement
- Each job departs after a random time

**Goal:** minimize $\mathbb{E} [\# \text{ active servers}]$
The problem

• Each arriving job needs to be assigned to a server
• Infinite # servers
• Each server has a resource capacity $M$

A new job model:
• Each job has a fixed resource requirement
• Each job departs after a random time

Goal: minimize $\mathbb{E} [\# \text{ active servers}]$ job assigning policy
The problem

• Each arriving job needs to be assigned to a server
• Infinite # servers
• Each server has a resource capacity $M$

A new job model:
• Each job has a fixed resource requirement
• Each job departs after a random time

Goal: minimize $\mathbb{E} [\# \text{ active servers}]$
The problem

- Each arriving job needs to be assigned to a server
- Infinite # servers
- Each server has a resource capacity $M$

A new job model:
- Each job has a fixed resource requirement
- Each job departs after a random time

Goal: minimize $\mathbb{E}$ [number of active servers] subject to cost (resource contention) $\leq$ budget
Why does time-varying matter?

resource requirement of a job

10 CPUs

1 CPU

time
Why does time-varying matter?

- Reserve resources based on peak requirement

![Graph showing resource requirement of a job over time with peak requirements at 10 CPUs and 1 CPU.](image)
Why does time-varying matter?

- Reserve resources based on peak requirement
  - low resource utilization on a server
Why does time-varying matter?

- Reserve resources based on peak requirement
  - low resource utilization on a server
  - larger # active servers
Why does time-varying matter?

- Reserve resources based on peak requirement
  - low resource utilization on a server
  - larger # active servers
- Overcommit resources on a server
Why does time-varying matter?

• Reserve resources based on peak requirement
  ➡ low resource utilization on a server
  ➡ larger # active servers

• Overcommit resources on a server
  ➡ possible resource contention
Why does time-varying matter?

- Reserve resources based on peak requirement
  - low resource utilization on a server
  - larger # active servers
- Overcommit resources on a server
  - possible resource contention

Our formulation captures:

- utilization
- resource contention
More details on the job model

Example MC

Phase $L$ $\mu_{HL}$ $\mu_{LH}$

Phase $H$

Phase $\perp$

(completion)

Example MC
More details on the job model

- Resource requirement of a job evolves over time following a Markov chain.
More details on the job model

- Resource requirement of a job evolves over time following a Markov chain

- Initial job type follows an initial distribution
More details on the job model

• Resource requirement of a job evolves over time following a Markov chain

• Initial job type follows an initial distribution

• MCs of jobs are independent of each other, and they are exogenous (not affected by resource contention)

Example MC
More details on the job model

- Resource requirement of a job evolves over time following a Markov chain
- Initial job type follows an initial distribution
- MCs of jobs are independent of each other, and they are exogenous (not affected by resource contention)
- Jobs arrive following a Poisson process
jobs

servers
state: # jobs of each type on each server

servers

jobs
Weina Wang (CMU)

Stochastic Bin Packing with Time-Varying Item Sizes

state space is large!

state: # jobs of each type on each server

servers

jobs
Reducing dimensionality

state space is large!

state: # jobs of each type on each server

```
jobs
```

```
servers
```
Reducing dimensionality

- Server-by-server evaluation:
Reducing dimensionality

- Server-by-server evaluation:
  - How to evaluate each server?

![Diagram showing state space](image)

state space is large!

state: # jobs of each type on each server
Reducing dimensionality

- Server-by-server evaluation:
  - How to evaluate each server?
  - How to relate to $E[\# \text{ active servers}]$?

state space is large!

state: # jobs of each type on each server

servers

jobs

state: # jobs of each type on each server
A policy-conversion framework

Policies in the $\infty$-server system

Policies in a single-server system
A policy-conversion framework

Policies in the infinite-server system

\[ \sigma \leftrightarrow \overline{\sigma} \]

Policies in a single-server system

achievability
A policy-conversion framework

- Use $\sigma$ to tell how to evaluate each server
- Performance of $\sigma$ is related to properties of $\sigma$
A policy-conversion framework

- Use $\overline{\sigma}$ to tell how to evaluate each server
- Performance of $\sigma$ is related to properties of $\overline{\sigma}$
A policy-conversion framework

- Use $\bar{\sigma}$ to tell how to evaluate each server
- Performance of $\sigma$ is related to properties of $\bar{\sigma}$

- Allows us to obtain lower bound on $\mathbb{E}[\# \text{active servers}]$
A policy-conversion framework

Policies in the $\infty$-server system

Policies in a single-server system

Single-server system
A policy-conversion framework

Policies in the $\infty$-server system

Policies in a single-server system

Single-server system

Infinite supply of jobs of all types

requests

jobs
A policy-conversion framework

Policies in the infinite-server system ↔ Policies in a single-server system

Single-server system

Infinite supply of jobs of all types

requests

jobs

A policy decides when to request what types of jobs to:

- maximize throughput
- subject to cost (resource contention) ≤ budget
Policies in the $\infty$-server system $\longleftrightarrow \bar{\sigma}$ Policies in a single-server system
- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
Policies in the $\infty$-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$

Policies in a single-server system

Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
Policies in the $\infty$-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$

Policies in a single-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$
Policies in the $\infty$-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$

Policies in a single-server system

Arrival rates: $r \cdot (\lambda_L, \lambda_H)$

Asymptotic regime: $r \to +\infty$

Phase $L$ ↔ Phase $H$

Phase $\perp$ (completion)
- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$
Policies in the $\infty$-server system

- Arrival rates: $r \cdot (\lambda_L, \lambda_H)$
- Asymptotic regime: $r \to +\infty$

Policies in a single-server system

Policy $\bar{\sigma}$

\[
\text{throughput} \cdot \bar{N} = r \cdot (\lambda_L, \lambda_H)
\]

\text{cost (resource contention)} \leq \text{budget}
Policies in the \( \infty \)-server system

- Arrival rates: \( r \cdot (\lambda_L, \lambda_H) \)
- Asymptotic regime: \( r \to +\infty \)

Policies in a single-server system

Policy \( \bar{\sigma} \)

\[
\text{throughput} \cdot N = r \cdot (\lambda_L, \lambda_H)
\]

\[
\text{cost (resource contention)} \leq \text{budget}
\]

Policy \( \sigma \)

\[
\mathbb{E} [\text{# active servers}] \leq \left( 1 + O(r^{-0.5}) \right) \cdot N
\]

\[
\text{cost (resource contention)} \leq \left( 1 + O(r^{-0.5}) \right) \cdot \text{budget}
\]
Policies in the \( \infty \)-server system

- Arrival rates: \( r \cdot (\lambda_L, \lambda_H) \)
- Asymptotic regime: \( r \to +\infty \)

Policies in a single-server system

\[ \sigma \leftrightarrow \overline{\sigma} \]

Policy \( \overline{\sigma} \)

- Throughput: \( \text{throughput} \cdot \overline{N} = r \cdot (\lambda_L, \lambda_H) \)
- Cost (resource contention) \( \leq \) budget

Policy \( \sigma \)

- \( \text{E [\# active servers]} \leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \overline{N} \)
- \( \text{Cost (resource contention)} \leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \text{budget} \)

Main Result: We design a policy for the original \( \infty \)-server system that is asymptotically optimal.
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\sigma$)
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

- For each server, run a single-server policy $\overline{\sigma}$
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\sigma$)

- For each server, run a single-server policy $\bar{\sigma}$
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\sigma$)

- For each server, run a single-server policy $\sigma$
- If $\sigma$ requests a job of type $i$, generate a token of type $i$
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\overline{\sigma}$)

- For each server, run a single-server policy $\overline{\sigma}$
- If $\overline{\sigma}$ requests a job of type $i$, generate a token of type $i$
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\bar{\sigma}$)

- For each server, run a single-server policy $\bar{\sigma}$
- If $\bar{\sigma}$ requests a job of type $i$, generate a token of type $i$
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\sigma$)

- For each server, run a single-server policy $\sigma$
- If $\sigma$ requests a job of type $i$, generate a token of type $i$
- When a job arrives, it checks tokens of its type and joins one uniformly at random
Policy conversion: single-server to \( \infty \)-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER (\( \sigma \))

- For each server, run a single-server policy \( \sigma \)
- If \( \sigma \) requests a job of type \( i \), generate a token of type \( i \)
- When a job arrives, it checks tokens of its type and joins one uniformly at random

```
jobs  \( \rightarrow \)  Single-server system running policy \( \sigma \)  
```

```
\( \Rightarrow \)  Request a job of type L
```

![Diagram showing single-server and \( \infty \)-server systems](image)

Request a job of type L

Single-server system running policy \( \sigma \)

Stochastic Bin Packing with Time-Varying Item Sizes

Weina Wang (CMU)
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\bar{\sigma}$)

- For each server, run a single-server policy $\bar{\sigma}$
- If $\bar{\sigma}$ requests a job of type $i$, generate a token of type $i$
- When a job arrives, it checks tokens of its type and joins one uniformly at random

Request a job of type L

Single-server system running policy $\bar{\sigma}$
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\sigma$)

- For each server, run a single-server policy $\sigma$
- If $\sigma$ requests a job of type $i$, generate a token of type $i$
- When a job arrives, it checks tokens of its type and joins one uniformly at random
- If no tokens, go to an inactive server
Policy conversion: single-server to $\infty$-server

Meta-algorithm: JOIN-THE-RECENTLY-REQUESTING-SERVER ($\sigma$)

- For each server, run a single-server policy $\sigma$
- If $\sigma$ requests a job of type $i$, generate a token of type $i$
- When a job arrives, it checks tokens of its type and joins one uniformly at random
- If no tokens, go to an inactive server

How is the throughput related to # active servers via tokens?
Policy conversion: more details

jobs

servers
Policy conversion: more details

Run single-server policy $\bar{\sigma}$ for only

$$\bar{N} = \frac{\text{arrival rate}}{\text{throughput}(\bar{\sigma})}$$

servers
Policy conversion: more details

Run single-server policy $\bar{\sigma}$ for only

$$\bar{N} = \frac{\text{arrival rate}}{\text{throughput}(\bar{\sigma})} \text{ servers}$$

Recall that we aim to show

$$\mathbb{E}[\text{# active servers}] \leq \left(1 + O(r^{-0.5})\right) \cdot \bar{N}$$
Policy conversion: more details

Run single-server policy $\bar{\sigma}$ for only servers

$$\bar{N} = \frac{\text{arrival rate}}{\text{throughput}(\bar{\sigma})}$$

Recall that we aim to show

$$\mathbb{E} \left[ \text{# active servers} \right] \leq \left( 1 + O \left( r^{-0.5} \right) \right) \cdot \bar{N}$$

When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs
Policy conversion: more details

Run single-server policy $\bar{\sigma}$ for only $\bar{N}$ servers

$$\bar{N} = \frac{\text{arrival rate}}{\text{throughput}(\sigma)}$$

Recall that we aim to show

$$E[\# \text{ active servers}] \leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \bar{N}$$

When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs
Policy conversion: more details

Run single-server policy $\bar{\sigma}$ for only
arrival rate
throughput($\bar{\sigma}$) servers

Recall that we aim to show
$E[\# \text{ active servers}] \leq \left(1 + O\left(r^{-0.5}\right)\right) \cdot \bar{N}$

We can prove that $E[\# \text{ virtual jobs}] = O\left(r^{0.5}\right)$
Key proof idea 1

jobs

servers
Key proof idea 1

Single-server system running policy $\bar{\sigma}$
Key proof idea 1

Will show that each server in the original system \( \approx \) an independent single-server system.

Single-server system running policy \( \sigma \)
Key proof idea 1

Will show that each server in the original system $\approx$ an independent single-server system

If only each token were replaced by a job immediately …
Key proof idea 1

Will show that each server in the original system $\approx$ an independent single-server system.

Single-server system running policy $\sigma$

Difficult: the dynamics of a server in the original system depends on other servers through arrivals & token overflows.
Key proof idea 1

Will show that each server in the original system ≈ an independent single-server system

Idea: for each type $i$, consider

$$\tilde{K}_i = \# \text{jobs} + \# \text{virtual jobs} + \# \text{tokens}$$
Key proof idea 1

Will show that each server in the original system
\approx an independent single-server system

Single-server system
running policy \sigma

**Idea:** for each type \( i \), consider
\[ \tilde{K}_i = \text{# jobs} + \text{# virtual jobs} + \text{# tokens} \]

**Difficulty:** the dynamics of a server in the original system depends on other servers through arrivals & token overflows

Why does considering \( \tilde{K}_i \) help decouple servers?
Key proof idea 1

Will show that each server in the original system
\( \approx \) an independent single-server system

**Idea:** for each type \( i \), consider
\( \tilde{K}_i = \# \text{ jobs} + \# \text{ virtual jobs} + \# \text{ tokens} \)

- Arrivals & token overflows do not affect \( \tilde{K}_i \)

**Difficulty:** the dynamics of a server in the original system depends on other servers through arrivals & token overflows

Why does considering \( \tilde{K}_i \) help decouple servers?
Key proof idea 1

Will show that each server in the original system \( \approx \) an independent single-server system

**Idea:** for each type \( i \), consider

\[
\hat{K}_i = \text{# jobs} + \text{# virtual jobs} + \text{# tokens} \quad \text{v.s.} \quad K_i = \text{# jobs of type } i
\]

- Arrivals & token overflows do not affect \( \hat{K}_i \)

Single-server system running policy \( \sigma \)
Will show that each server in the original system
\( \approx \) an independent single-server system

**Key proof idea 1**

**Idea:** for each type \( i \), consider

\[
\tilde{K}_i = \text{# jobs} + \text{# virtual jobs} + \text{# tokens} \quad \text{v.s.} \quad \bar{K}_i = \text{# jobs of type } i
\]

- Arrivals & token overflows do not affect \( \tilde{K}_i \)
- Requests by \( \bar{\sigma} \) change \( \tilde{K}_i \) and \( \bar{K}_i \) in the same way, difference bounded by \# tokens
Key proof idea 1

Will show that each server in the original system
\[ \approx \text{an independent single-server system} \]

Idea: for each type $i$, consider
\[ \tilde{K}_i = \# \text{jobs} + \# \text{virtual jobs} + \# \text{tokens} \quad \text{v.s.} \quad K_i = \# \text{jobs of type } i \]

- Arrivals & token overflows do not affect $\tilde{K}_i$
- Requests by $\bar{\sigma}$ change $\tilde{K}_i$ and $K_i$ in the same way, difference bounded by $\# \text{tokens}$
- Job phase transitions in $\tilde{K}_i$ and $K_i$ differ by $\# \text{tokens}$
Key proof idea 1

Will show that each server in the original system
≈ an independent single-server system

Idea: for each type $i$, consider

$$\tilde{K}_i = \#\text{jobs} + \#\text{virtual jobs} + \#\text{tokens} \quad \text{v.s.} \quad \bar{K}_i = \#\text{jobs of type } i$$

- Arrivals & token overflows do not affect $\tilde{K}_i$
- Requests by $\bar{\sigma}$ change $\tilde{K}_i$ and $\bar{K}_i$ in the same way, difference bounded by $\#\text{tokens}$
- Job phase transitions in $\tilde{K}_i$ and $\bar{K}_i$ differ by $\#\text{tokens}$

Using Stein’s method, we show

$$d_W \left( \tilde{K}_1^{1:N}, \bar{K}_1^{1:N} \right) = O(r^{0.5})$$
Key proof idea 2

jobs

servers
Key proof idea 2

When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs.
Key proof idea 2

When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs.
Key proof idea 2

When the # tokens of a type $> \sqrt{r}$, remove the overflow tokens and generate virtual jobs.
Key proof idea 2

Will show that \# virtual jobs = \( o(\sqrt{r}) \),
and \# backup servers = \( o(\sqrt{r}) \)
Weina Wang (CMU)

Stochastic Bin Packing with Time-Varying Item Sizes

Key proof idea 2

Will show that \# virtual jobs = \( O(\sqrt{r}) \),
and \# backup servers = \( O(\sqrt{r}) \)
Key proof idea 2

Will show that the number of virtual jobs is $O\left(\sqrt{r}\right)$, and the number of backup servers is $O\left(\sqrt{r}\right)$.

A server requests a type L job.
Key proof idea 2

Will show that # virtual jobs = \( O(\sqrt{r}) \), and # backup servers = \( O(\sqrt{r}) \).
Key proof idea 2

Will show that \# virtual jobs = \(O\left(\sqrt{r}\right)\),
and \# backup servers = \(O\left(\sqrt{r}\right)\).

What happens when # tokens hits \(\sqrt{r}\)?
Key proof idea 2

Will show that \# virtual jobs = \( O\left(\sqrt{r}\right) \), and \# backup servers = \( O\left(\sqrt{r}\right) \)

What happens when \# tokens hits \( \sqrt{r} \)?
Generate a virtual job

a server requests a type L job

a type L job arrives

\[ L \ L \ L \ L \]

\[ \sqrt{r} \]

backup servers

jobs
Key proof idea 2

Will show that \# virtual jobs = \( O(\sqrt{r}) \), and \# backup servers = \( O(\sqrt{r}) \)

What happens when # tokens hits 0?
Generate a virtual job

What happens when # tokens hits \( \sqrt{r} \)?
Key proof idea 2

Will show that \# virtual jobs = \( O(\sqrt{r}) \),
and \# backup servers = \( O(\sqrt{r}) \)

What happens when \# tokens hits 0?
Generate a job to backup servers

What happens when \# tokens hits \( \sqrt{r} \)?
Generate a virtual job

a type L job arrives
a server requests a type L job

L L L L

backup servers

jobs

servers
Key proof idea 2

Will show that \# virtual jobs = \(O\left(\sqrt{r}\right)\),
and \# backup servers = \(O\left(\sqrt{r}\right)\)

---

Weina Wang (CMU)

Stochastic Bin Packing with Time-Varying Item Sizes
Key proof idea 2

Will show that \# virtual jobs = \( O\left(\sqrt{r}\right) \),
and \# backup servers = \( O\left(\sqrt{r}\right) \)

a type L job arrives
a server requests a type L job

serves
Key proof idea 2

Will show that \# virtual jobs = \( O\left(\sqrt{r}\right) \),
and \# backup servers = \( O\left(\sqrt{r}\right) \)

- An almost balanced random walk
Key proof idea 2

Will show that \( \# \) virtual jobs = \( O\left(\sqrt{r}\right) \),
and \( \# \) backup servers = \( O\left(\sqrt{r}\right) \)

- An almost balanced random walk
- Stationary distribution \( \approx \) uniform on \( \{0, 1, \ldots, \sqrt{r}\} \)
Key proof idea 2

Will show that \# virtual jobs $= O\left(\sqrt{r}\right)$,
and \# backup servers $= O\left(\sqrt{r}\right)$

- An almost balanced random walk
- Stationary distribution $\approx$ uniform on \{0, 1, ..., $\sqrt{r}$\}
- Rate of generating virtual jobs $\approx$ rate of sending jobs to backup servers
  $\approx$ arrival rate $/ \sqrt{r} = O\left(\sqrt{r}\right)$
Summary

Phase $L$ $\rightarrow$ Phase $H$ $\rightarrow$ Phase $\perp$

$\mu_{LL}$ $\rightarrow$ $\mu_{HL}$ $\rightarrow$ $\mu_{HH}$

(job completion)

servers

jobs
Summary

• We considered the problem of assigning jobs to servers when jobs have time-varying resource requirements.
Summary

• We considered the problem of assigning jobs to servers when jobs have time-varying resource requirements

• We designed an asymptotically optimal policy
Summary

• We considered the problem of assigning jobs to servers when jobs have time-varying resource requirements.
• We designed an asymptotically optimal policy.
• We proposed a policy-conversion framework that allows us to reduce the policy-design problem to that in a single-server system.
We considered the problem of assigning jobs to servers when jobs have time-varying resource requirements.

We designed an asymptotically optimal policy.

We proposed a policy-conversion framework that allows us to reduce the policy-design problem to that in a single-server system.

A highlight of the framework is the meta-algorithm, JOIN-THE-RECENTLY-REQUESTING-SERVER (JRSS), that converts a single-server policy to a policy in the original system.