Grid Brokering of Batch Allocation using Indexes

Vandy BERTEN[1] Bruno GAUJAL[2]

[1]Université Libre de Bruxelles and FNRS
vandy.berten@ulb.ac.be

[2]INRIA and LIG (CNRS, INPG, UJF)
bruno.gaujal@imag.fr

Bordeaux, Jan 31, 2008

Outline

Motivations

On a methodological point of view :

- Use statistical data and predictions based on a stochastic model to propose an alternative to usual Grid brokering strategies.

Motivations

On a methodological point of view :

- Use statistical data and predictions based on a stochastic model to propose an alternative to usual Grid brokering strategies.
- Evaluate the impact of information.

Motivations

On a methodological point of view :

- Use statistical data and predictions based on a stochastic model to propose an alternative to usual Grid brokering strategies.
- Evaluate the impact of information.
- Assess the robustness of index policies in the context of grid brokering.

Motivations

On a methodological point of view :

- Use statistical data and predictions based on a stochastic model to propose an alternative to usual Grid brokering strategies.
- Evaluate the impact of information.
- Assess the robustness of index policies in the context of grid brokering.

On a technical point of view :

Motivations

On a methodological point of view :

- Use statistical data and predictions based on a stochastic model to propose an alternative to usual Grid brokering strategies.
- Evaluate the impact of information.
- Assess the robustness of index policies in the context of grid brokering.

On a technical point of view :

- Propose an alternative algorithm allowing to compute indexes.

Motivations

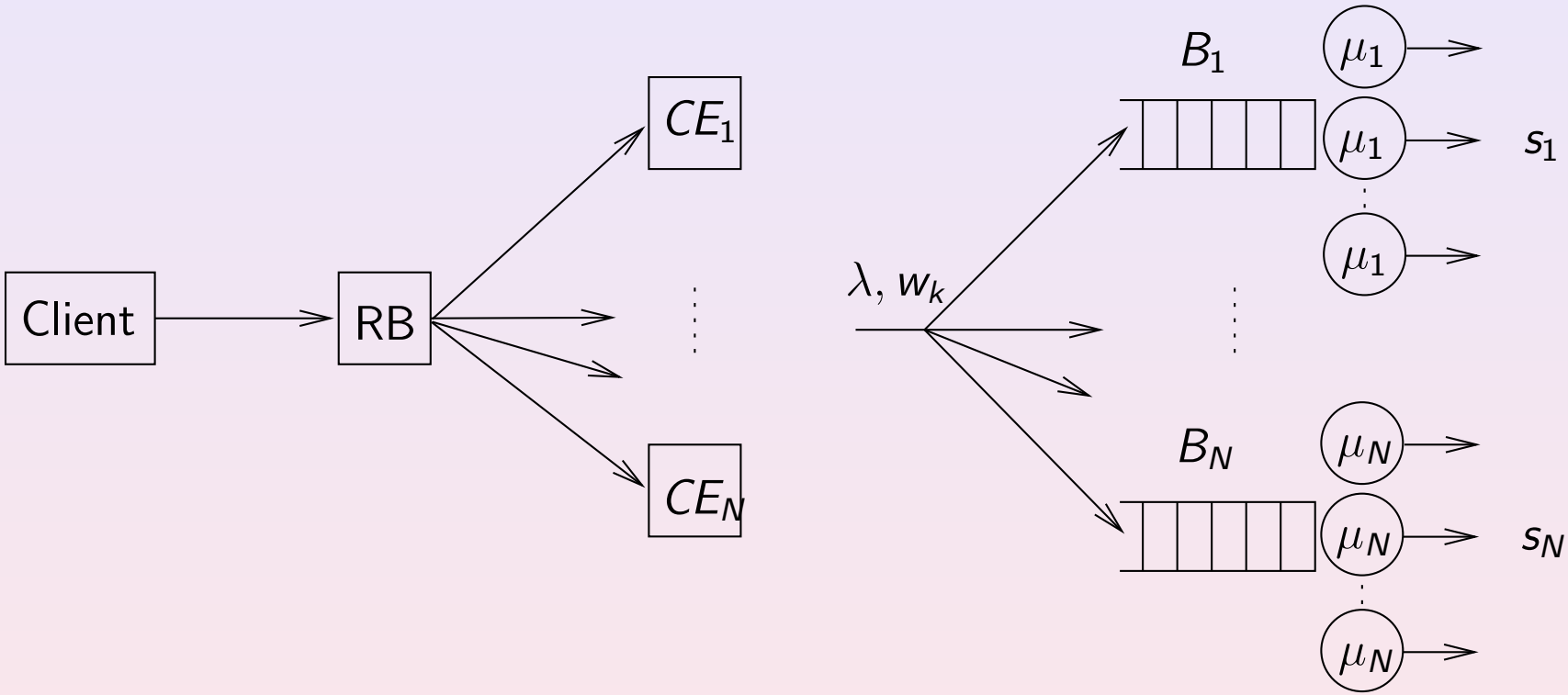
On a methodological point of view :

- Use statistical data and predictions based on a stochastic model to propose an alternative to usual Grid brokering strategies.
- Evaluate the impact of information.
- Assess the robustness of index policies in the context of grid brokering.

On a technical point of view :

- Propose an alternative algorithm allowing to compute indexes.
- Extend the classical models to batch jobs, with or without information on the size of the batches.

Model



Brokering

- According to the state (the number of processes in every queue), choose a destination for an incoming job.

Brokering

- According to the state (the number of processes in every queue), choose a destination for an incoming job.
- If the size of a batch is known at the brokering time, this information can be taken into account.

Brokering

- According to the state (the number of processes in every queue), choose a destination for an incoming job.
- If the size of a batch is known at the brokering time, this information can be taken into account.
- Using Index heuristics (Gittins, Whittle, Mitrani, Nino-Mora, ...), we send a job to the "cheapest" queue.

Markovian Model

The jobs are made of several parallel processes, to be executed on the same cluster. We do not model precisely the synchronizations and communications between the processes belonging to the same job.

- The state $(x_1, \ldots, x_N)$ in S is the number of processes in clusters $1, \ldots, N$ ($x_i \in \{0, \ldots, B_i\}$).
- An action (u in U) is either i (cluster i is chosen), or 0 (rejection of a job). Rejection is only allowed when all queues are "full".
- The only possible events are job arrivals and process departures.

Markovian Model

The jobs are made of several parallel processes, to be executed on the same cluster. We do not model precisely the synchronizations and communications between the processes belonging to the same job.

- The state $(x_1, \ldots, x_N)$ in S is the number of processes in clusters $1, \ldots, N$ ($x_i \in \{0, \ldots, B_i\}$).
- An action (u in U) is either i (cluster i is chosen), or 0 (rejection of a job). Rejection is only allowed when all queues are "full".
- The only possible events are job arrivals and process departures.

We look for a routing policy minimizing the expected discounted workload in infinite horizon.

Markov Decision Process

This can be seen as a Markov Decision Problem after uniformization by $\Lambda = \lambda + \max_{i \in \{1,\ldots,N\}} s_i \mu_i$.

Markov Decision Process

This can be seen as a Markov Decision Problem after uniformization by $\Lambda = \lambda + \max_{i \in \{1,\ldots,N\}} s_i \mu_i$.
The cost under policy $\pi = (u_0, u_1, \ldots)$ with initial state $x^{(0)} = (x_1^{(0)}, \ldots, x_N^{(0)})$ is

$$J_\pi(x^{(0)}) = \limsup_{n \to \infty} \mathbb{E} \sum_{m=0}^{n-1} \alpha^m h(x^{(m)}, u_m(x^{(m)})),$$

with discounting factor α and immediate cost

$$h \stackrel{def}{=} \sum_{i=1}^{N} c_i((x_i^{(m)} + \delta_{\{\text{arrival at step } m\}} \delta_{\{u_m(x^{(m)})=i\}})),$$

where c_i is the cost in the i-th cluster per customer per time unit.

Markov Decision Process

This can be seen as a Markov Decision Problem after uniformization by $\Lambda = \lambda + \max_{i \in \{1,\ldots,N\}} s_i \mu_i$.
The cost under policy $\pi = (u_0, u_1, \ldots)$ with initial state $x^{(0)} = (x_1^{(0)}, \ldots, x_N^{(0)})$ is

$$J_\pi(x^{(0)}) = \limsup_{n \to \infty} \mathbb{E} \sum_{m=0}^{n-1} \alpha^m h(x^{(m)}, u_m(x^{(m)})),$$

with discounting factor α and immediate cost

$$h \stackrel{def}{=} \sum_{i=1}^{N} c_i((x_i^{(m)} + \delta_{\{\text{arrival at step } m\}} \delta_{\{u_m(x^{(m)})=i\}})),$$

where c_i is the cost in the i-th cluster per customer per time unit.
As the cost is uniformly bounded over the state space, we consider time and initial state independent routing policies : u says which action to take in each state ($u : S \to U$).

Markov Decision Process

This can be seen as a Markov Decision Problem after uniformization by $\Lambda = \lambda + \max_{i \in \{1,\ldots,N\}} s_i \mu_i$.
The cost under policy $\pi = (u_0, u_1, \ldots)$ with initial state $x^{(0)} = (x_1^{(0)}, \ldots, x_N^{(0)})$ is

$$J_\pi(x^{(0)}) = \limsup_{n \to \infty} \mathbb{E} \sum_{m=0}^{n-1} \alpha^m h(x^{(m)}, u_m(x^{(m)})),$$

with discounting factor α and immediate cost

$$h \stackrel{def}{=} \sum_{i=1}^{N} c_i((x_i^{(m)} + \delta_{\{\text{arrival at step } m\}} \delta_{\{u_m(x^{(m)})=i\}})),$$

where c_i is the cost in the i-th cluster per customer per time unit.
As the cost is uniformly bounded over the state space, we consider time and initial state independent routing policies : u says which action to take in each state ($u : S \to U$).
This problem is P-space hard in general.

Routing based on a Local Criterion

Routing policies based on a Local Criterion (LCR) are a subset of routing policies. They are defined as follows :

- Each queue has a Local index function : $L_i : \{0, \ldots, B_i - 1\} \rightarrow \mathbb{R}^+$.

Routing based on a Local Criterion

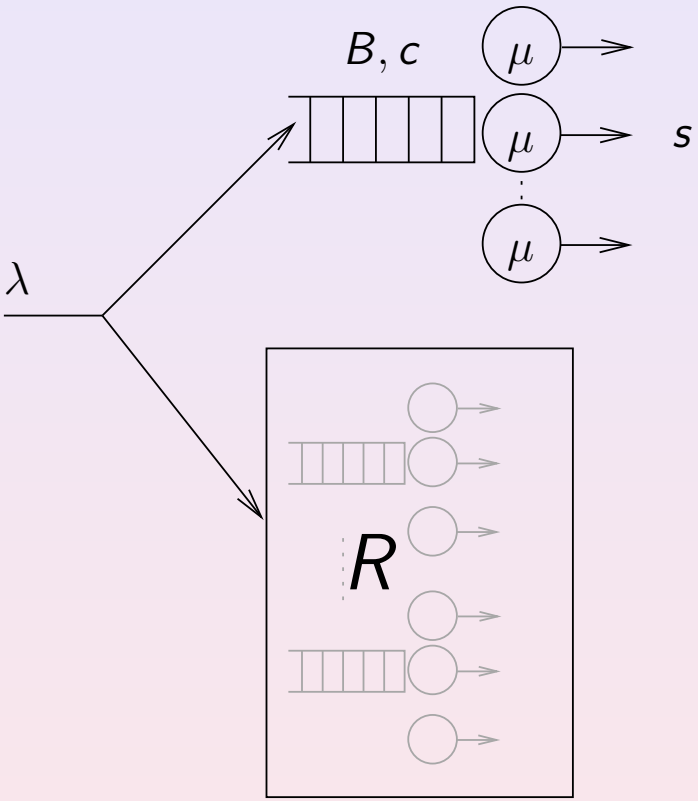
Routing policies based on a Local Criterion (LCR) are a subset of routing policies. They are defined as follows :

- Each queue has a Local index function : $L_i : \{0, \ldots, B_i - 1\} \rightarrow \mathbb{R}^+$.
- The routing policy is the following :

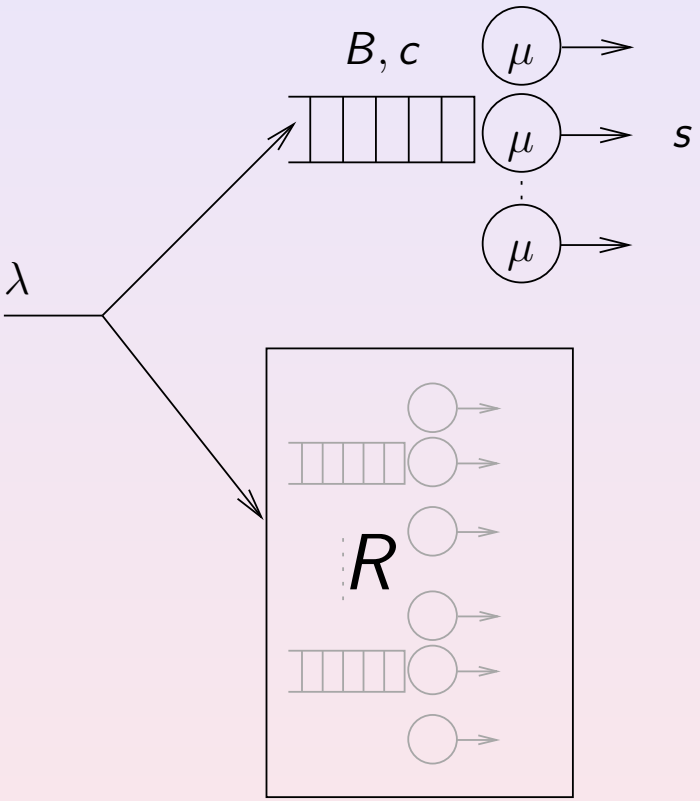
$$u(x_1, \ldots, x_N) = \underset{x_i < B_i}{argmin} \{L_1(x_1), \ldots, L_N(x_N)\}$$

An arriving job is sent towards the queue with the smallest current local index. If the results of the argmin is empty, the incoming job is rejected.

Index heuristics

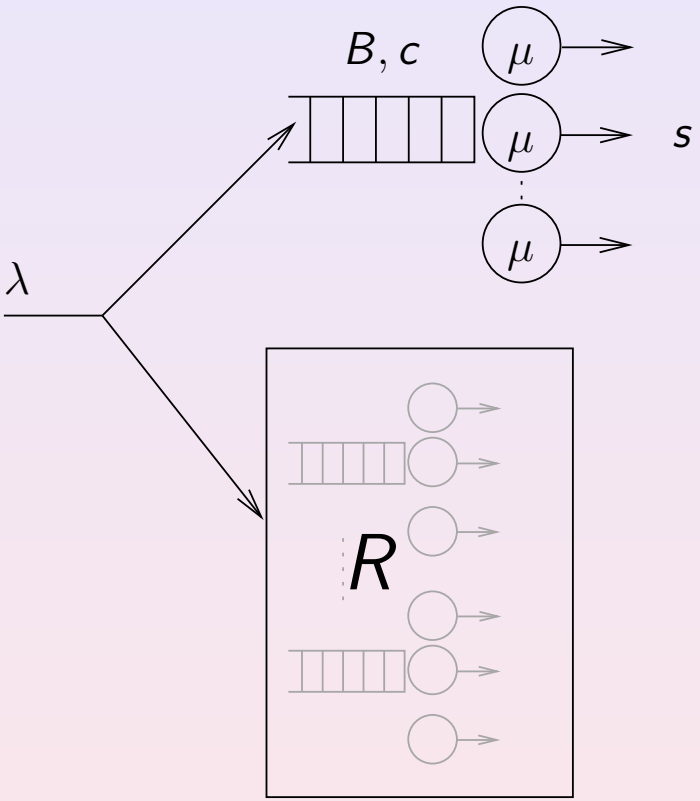


Index heuristics



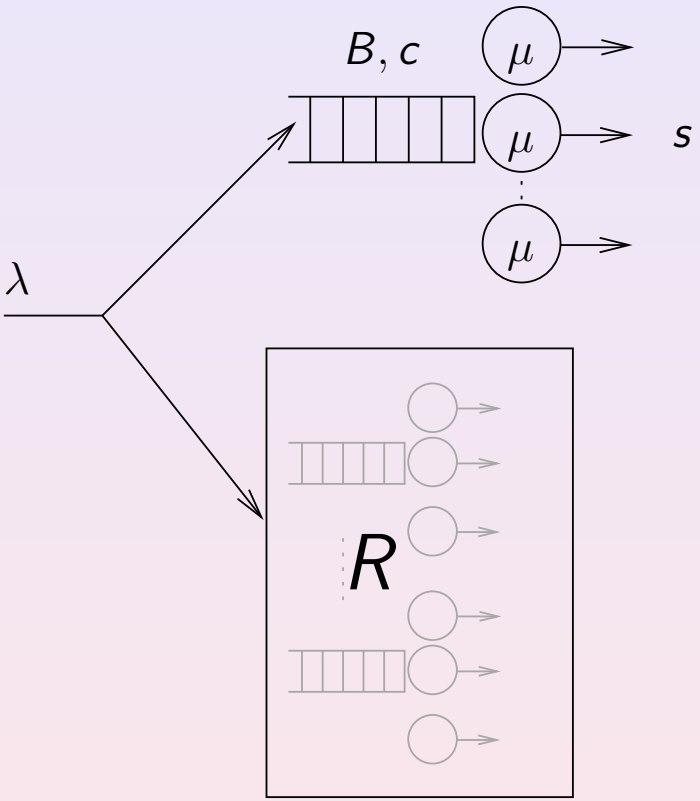
- We consider that the cost of accepting a job on CE_i only depends upon the state of CE_i.

Index heuristics



- We consider that the cost of accepting a job on CE_i only depends upon the state of CE_i.
- The effect of routing a job to queue i is rather intricate to measure in the long term. The idea is to come up with a real valued cost of this decision.

Index heuristics



- We consider that the cost of accepting a job on CE_i only depends upon the state of CE_i.
- The effect of routing a job to queue i is rather intricate to measure in the long term. The idea is to come up with a real valued cost of this decision.
- To do so, we concentrate on one queue, other ones being considered as a black box costing R (degree of freedom).

Computing the Local Index

The cost in one queue under policy u is :

$$J_u(x^0) = \limsup_{n\to\infty} \mathbb{E} \sum_{m=0}^{n-1} \alpha^m (cx^{(m)} + kR\delta_{\{u_m=0\}}\delta_{\{\text{arrival at step } m \text{ of a job of size } k\}}).$$

Computing the Local Index

The cost in one queue under policy u is :

$$J_u(x^0) = \limsup_{n\to\infty} \mathbb{E} \sum_{m=0}^{n-1} \alpha^m (cx^{(m)} + kR\delta_{\{u_m=0\}}\delta_{\{\text{arrival at step } m \text{ of a job of size } k\}}).$$

The optimal cost J^* is the unique solution of the Bellman Equation,

$$\begin{aligned} & J^*(x) \\ = \; & \alpha\Big(\lambda' \textstyle\sum_k P_k \min\{kR + J^*(x), J^*(x+k)\} \\ & \quad + \mu' \min\{s, x\} J^*(x-1) \\ & \quad + (1 - \lambda' - \mu' \min\{s, x\}) J^*(x) \Big) \\ & + cx. \end{aligned} \tag{1}$$

Computing the Local Index

The cost in one queue under policy u is :

$$J_u(x^0) = \limsup_{n\to\infty} \mathbb{E} \sum_{m=0}^{n-1} \alpha^m (cx^{(m)} + kR\delta_{\{u_m=0\}}\delta_{\{\text{arrival at step } m \text{ of a job of size } k\}}).$$

The optimal cost J^* is the unique solution of the Bellman Equation,

$$\begin{aligned} & J^*(x) \\ = \; & \alpha\Big(\lambda' \textstyle\sum_k P_k \min\{kR + J^*(x), J^*(x+k)\} \\ & \quad + \mu' \min\{s,x\} J^*(x-1) \\ & \quad + (1 - \lambda' - \mu' \min\{s,x\}) J^*(x) \Big) \\ & + cx. \end{aligned} \tag{1}$$

There is an optimal policy minimizing the α-discounted cost in infinite horizon for one queue of threshold type : $u^*(x) = 1$ if $x < \Theta(R)$ and $u^*(x) = 0$ otherwise.

Algorithms

Under threshold θ, the cost verifies a matrix equation $J_\theta = (\alpha F_\theta J_\theta + S_\theta)$.

Algorithms

Under threshold θ, the cost verifies a matrix equation $J_\theta = (\alpha F_\theta J_\theta + S_\theta)$.
The optimal threshold $\Theta(R)$ and its cost verify
$J^* = \min_{\theta \in \{0,\ldots,B\}}(\alpha F_\theta J^* + S_\theta)$.

Algorithms

Under threshold θ, the cost verifies a matrix equation $J_\theta = (\alpha F_\theta J_\theta + S_\theta)$.
The optimal threshold $\Theta(R)$ and its cost verify
$J^* = \min_{\theta \in \{0,\ldots,B\}}(\alpha F_\theta J^* + S_\theta)$.
Computing the cost can be done by using the cost equation and its derivatives [Palmer, Mitrani 2005] or using polytope problems [Nio-Mora 2002]

Algorithms

Under threshold θ, the cost verifies a matrix equation $J_\theta = (\alpha F_\theta J_\theta + S_\theta)$.
The optimal threshold $\Theta(R)$ and its cost verify
$J^* = \min_{\theta \in \{0,\ldots,B\}}(\alpha F_\theta J^* + S_\theta)$.
Computing the cost can be done by using the cost equation and its derivatives [Palmer, Mitrani 2005] or using polytope problems [Nio-Mora 2002]
We use an alternative approach, based on policy iteration.
Let $J_{\theta,\theta'} \stackrel{def}{=} (\alpha F_{\theta'} J_\theta + S_{\theta'})$ be the cost obtained by using values found for J_θ under policy $u_{\theta'}$.

1. choose θ
2. solve $J_\theta = (\alpha F_\theta J_\theta + S_\theta)$.
3. find θ^* such as $\theta^* = argmax_{\theta'} \sum_{x=0}^{B} J_\theta(x) - J_{\theta,\theta'}(x)$.
4. restart in 2 with θ^* as long as $\theta \neq \theta^*$.

Index heuristics

The local index L is defined as follows (inverse of Θ) :
$L(x) = \{\sup R | \Theta(R) \leq x\}$.

- $L(x)$ (index) represents the long run cost of accepting a job on the queue if the state is x.

Index heuristics

The local index L is defined as follows (inverse of Θ) :
$L(x) = \{\sup R | \Theta(R) \leq x\}$.

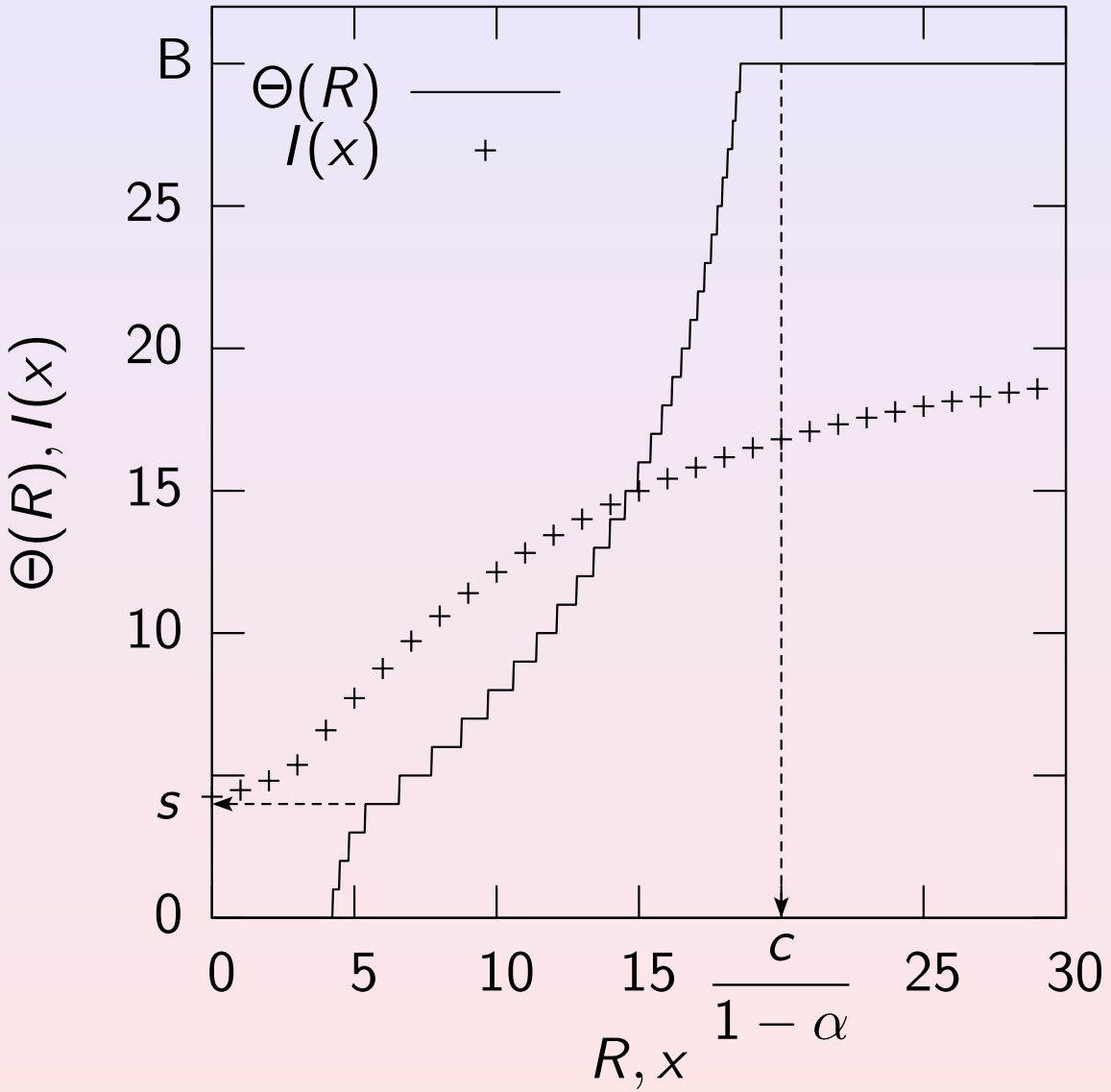
- $L(x)$ (index) represents the long run cost of accepting a job on the queue if the state is x.
- An incoming job is sent to the CE with the smallest expected cost, $\min\{L_1(x_1), \ldots, L_N(x_N)\}$.

Index heuristics

The local index L is defined as follows (inverse of Θ) :
$L(x) = \{\sup R | \Theta(R) \leq x\}$.

- $L(x)$ (index) represents the long run cost of accepting a job on the queue if the state is x.
- An incoming job is sent to the CE with the smallest expected cost, $\min\{L_1(x_1), \ldots, L_N(x_N)\}$.
- Computing L can be done in a few seconds in the mono-dimensional case : $O(eB(B+K)K + (B+K)K \log B)$, where $e = -log(\epsilon)$, the precision chosen in the previous algorithm.

Threshold and Local Index



Strategies

- `Random` : Bernoulli routing (weight $\mu_i s_i$)

Strategies

- `Random` : Bernoulli routing (weight $\mu_i s_i$)
- `JSQ` : Join the Smallest State (x)

Strategies

- `Random` : Bernoulli routing (weight $\mu_i s_i$)
- `JSQ` : Join the Smallest State (x)
- `JSQ2` : Join the Shortest Queue ($\max\{0, x - s_i\}$)

Strategies

- `Random` : Bernoulli routing (weight $\mu_i s_i$)
- `JSQ` : Join the Smallest State (x)
- `JSQ2` : Join the Shortest Queue ($\max\{0, x - s_i\}$)
- `JSQ-mu` : `JSQ`$/(\mu_i s_i)$

Strategies

- `Random` : Bernoulli routing (weight $\mu_i s_i$)
- `JSQ` : Join the Smallest State (x)
- `JSQ2` : Join the Shortest Queue ($\max\{0, x - s_i\}$)
- `JSQ-mu` : `JSQ`$/(\mu_i s_i)$
- `JSQ2-mu` : `JSQ2`$/(\mu_i s_i)$

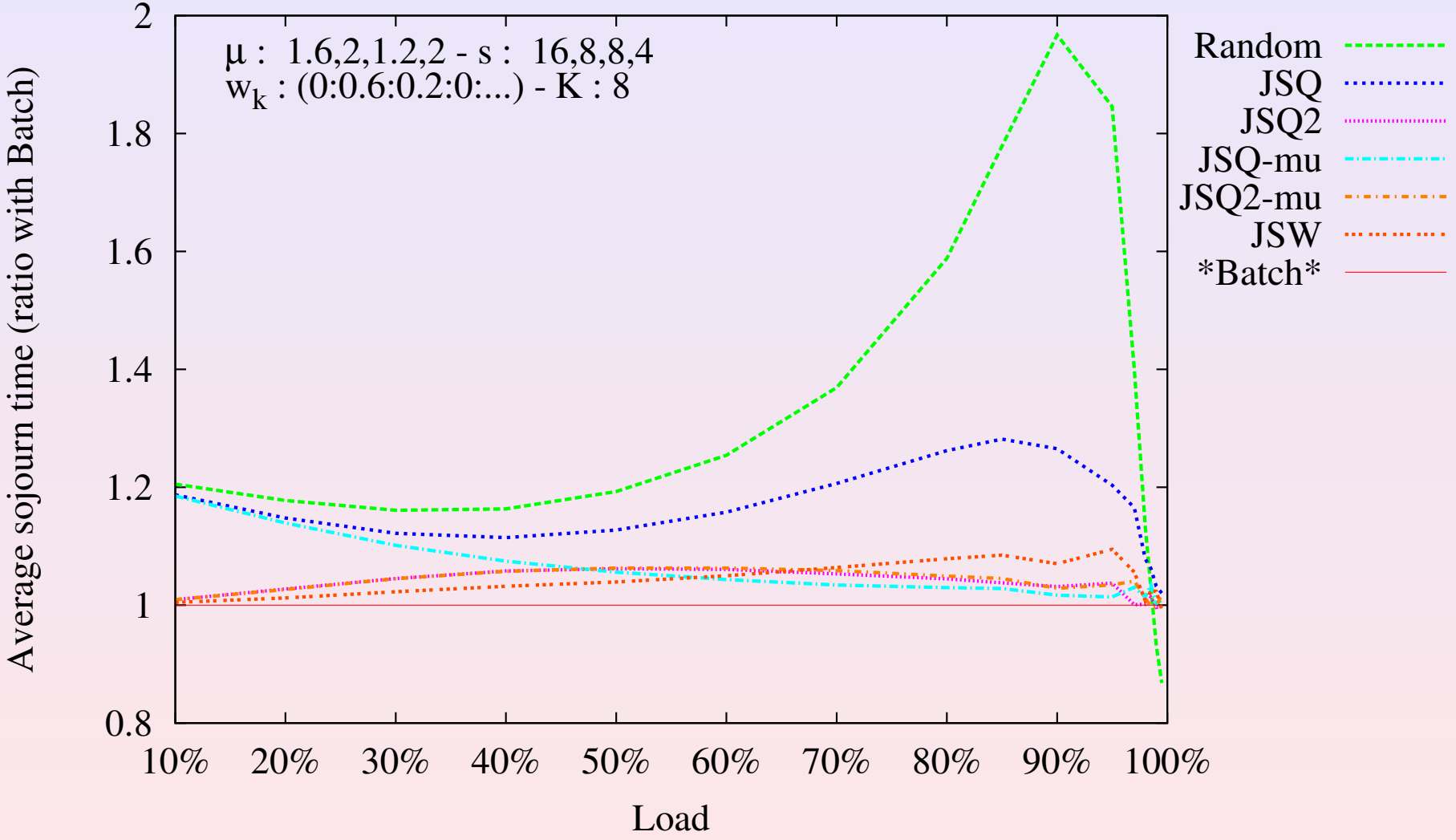
Strategies

- `Random` : Bernoulli routing (weight $\mu_i s_i$)
- `JSQ` : Join the Smallest State (x)
- `JSQ2` : Join the Shortest Queue ($\max\{0, x - s_i\}$)
- `JSQ-mu` : `JSQ`$/(\mu_i s_i)$
- `JSQ2-mu` : `JSQ2`$/(\mu_i s_i)$
- `JSW` : Join the Smallest Waiting time ($\min\left\{\mu_i^{-1}, \frac{x+1}{\mu_i s_i}\right\}$)

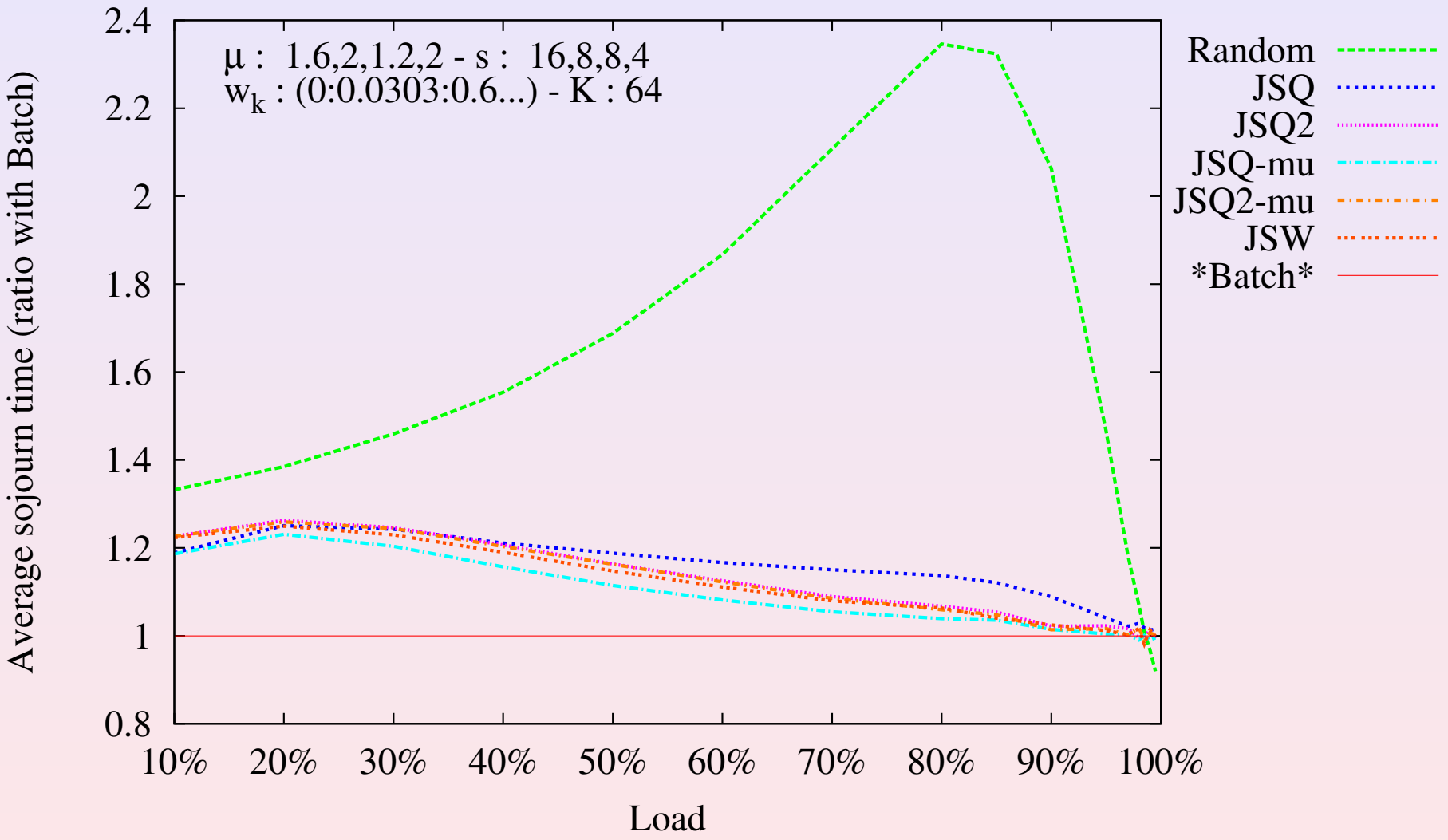
Strategies

- `Random` : Bernoulli routing (weight $\mu_i s_i$)
- `JSQ` : Join the Smallest State (x)
- `JSQ2` : Join the Shortest Queue ($\max\{0, x - s_i\}$)
- `JSQ-mu` : `JSQ`$/(\mu_i s_i)$
- `JSQ2-mu` : `JSQ2`$/(\mu_i s_i)$
- `JSW` : Join the Smallest Waiting time ($\min\left\{\mu_i^{-1}, \frac{x+1}{\mu_i s_i}\right\}$)
- `Batch` : index without width knowledge.

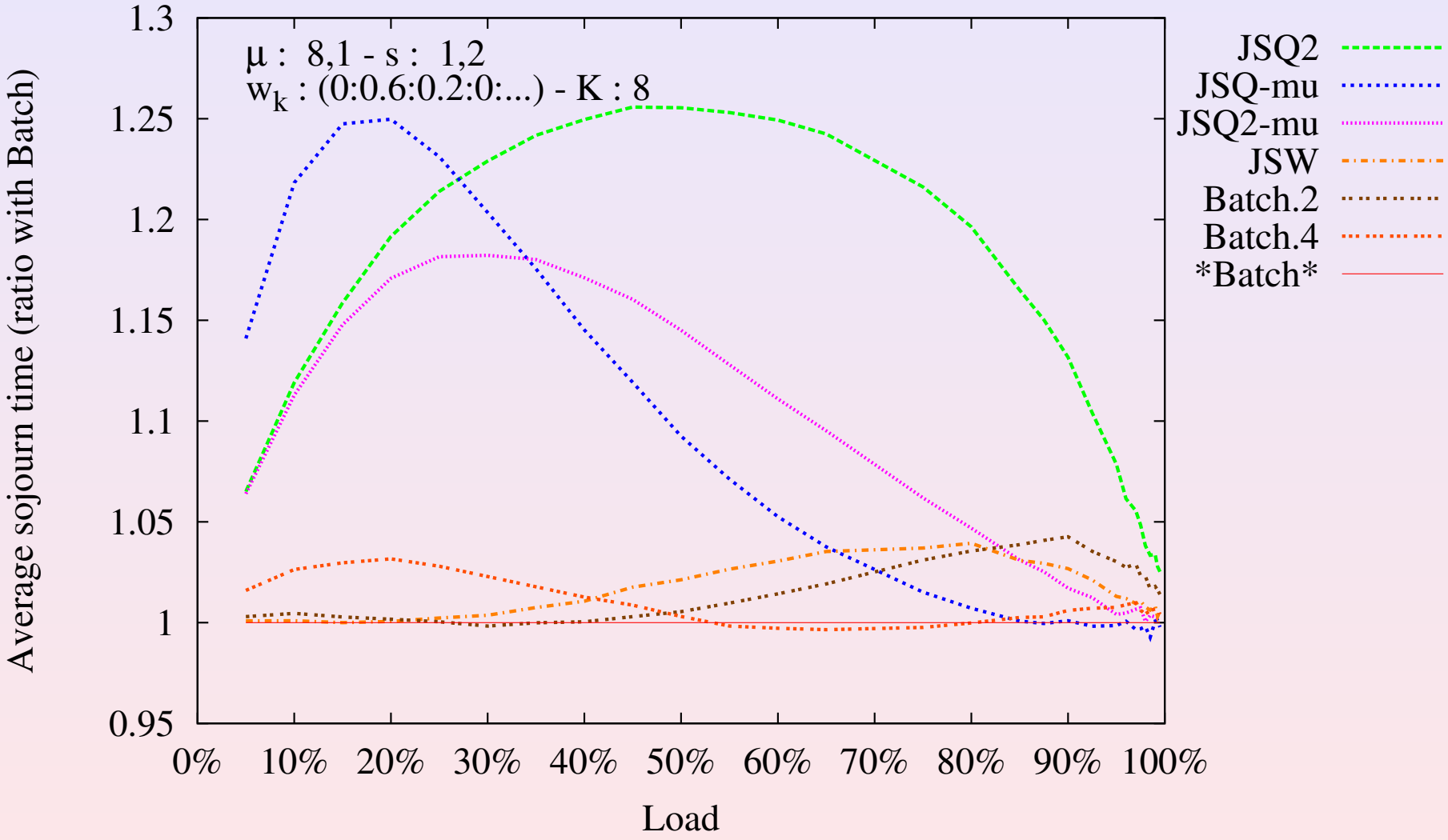
Impact of job width distribution (K=8)



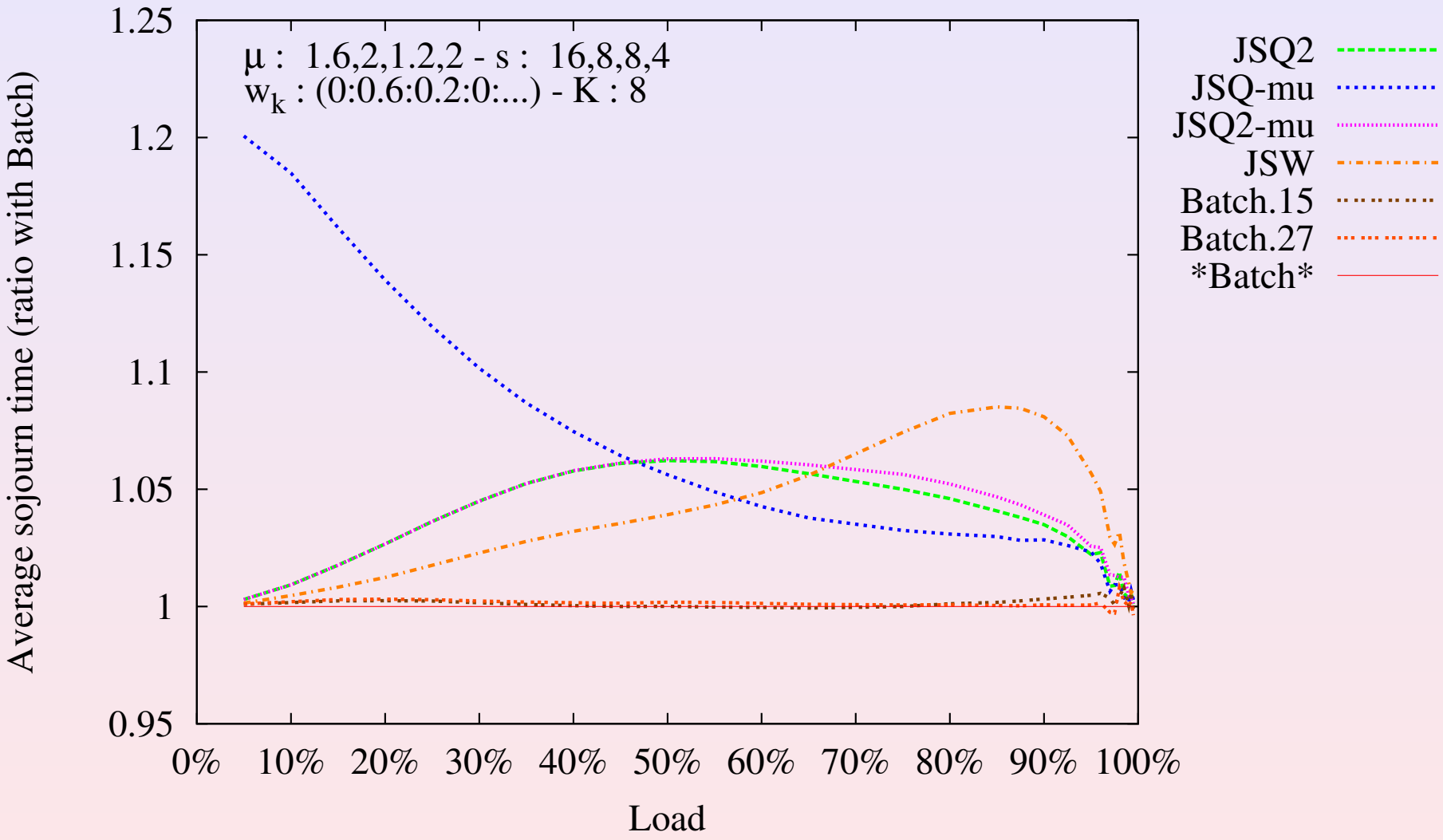
Impact of job width distribution (K=64)



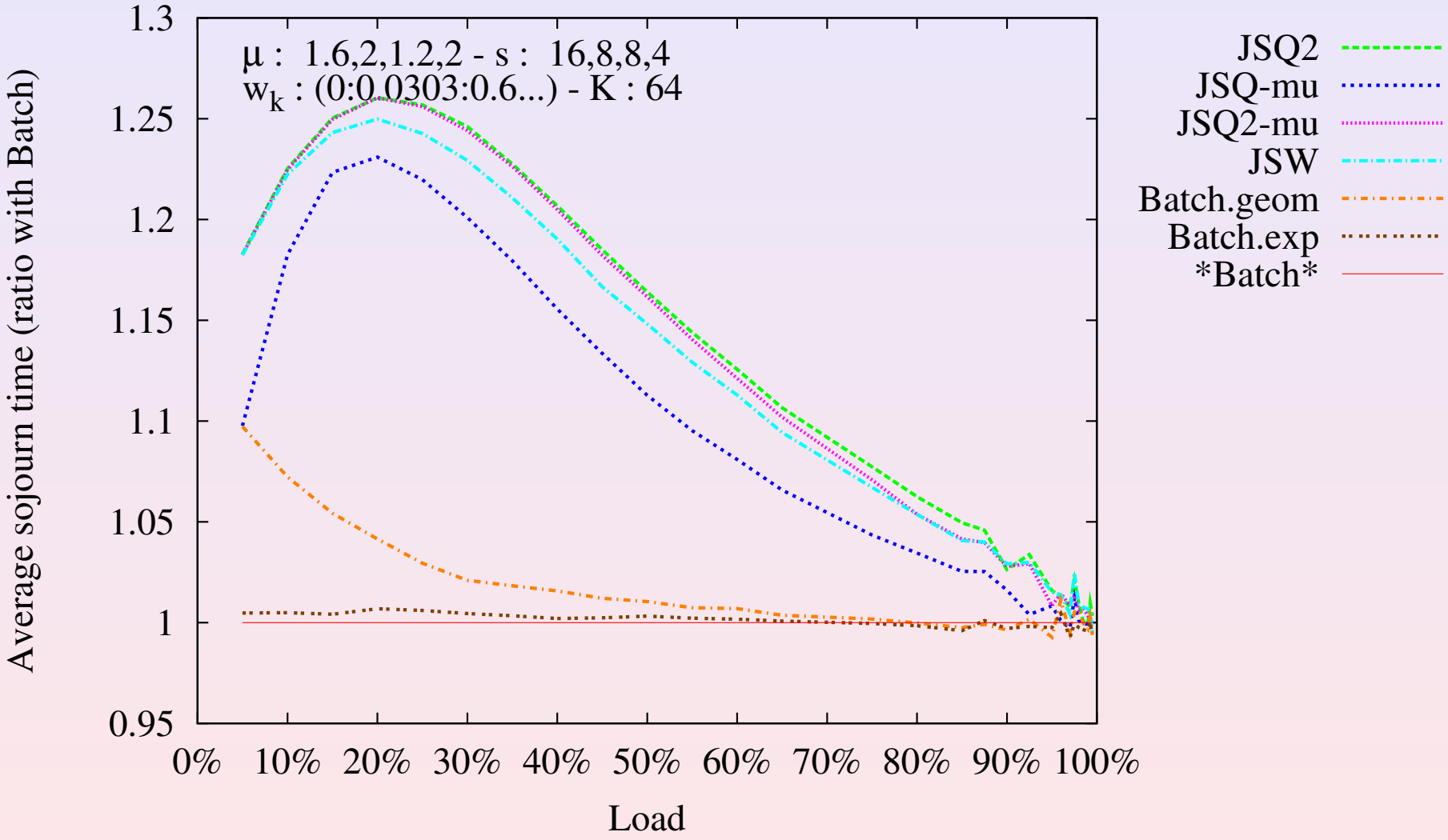
Robustness on load variations



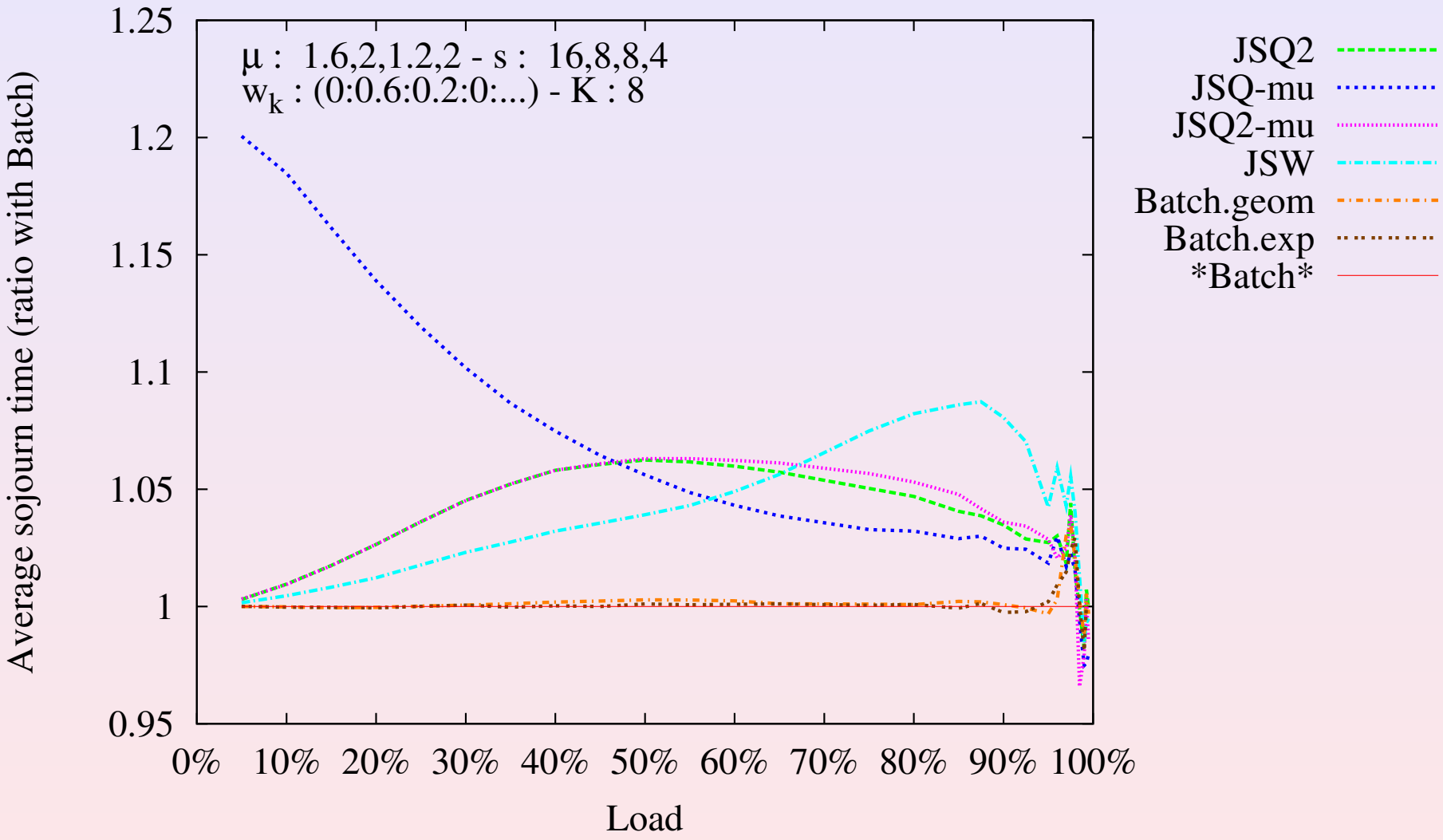
Robustness on load variations



Robustness on job width distribution



Robustness on job width distribution



Index routing, conclusion

- Very efficient strategy, especially with heterogeneous systems, and when $w_k \gg 0$ for large k

Index routing, conclusion

- Very efficient strategy, especially with heterogeneous systems, and when $w_k \gg 0$ for large k
- Easy to implement

Index routing, conclusion

- Very efficient strategy, especially with heterogeneous systems, and when $w_k \gg 0$ for large k
- Easy to implement
- Do not require too much computational power for index generation

Index routing, conclusion

- Very efficient strategy, especially with heterogeneous systems, and when $w_k \gg 0$ for large k
- Easy to implement
- Do not require too much computational power for index generation
- Large robustness to parameters variation

Batches are known by the broker

- Job width must be taken into account by the resource broker, as any additional information.

Batches are known by the broker

- Job width must be taken into account by the resource broker, as any additional information.
- Index becomes *bi-dimensional* : they depend on the state and the job size.

Batches are known by the broker

- Job width must be taken into account by the resource broker, as any additional information.
- Index becomes *bi-dimensional* : they depend on the state and the job size.
- As seen later, computations will become much more complex.

Index brokering of known batches

Once again, we focus on local index based policies.

The optimal policy is of threshold type, depending on k the size of the incoming job : If $x < \theta(k)$ the job is rejected and accepted otherwise.

Index brokering of known batches

Once again, we focus on local index based policies.

The optimal policy is of threshold type, depending on k the size of the incoming job : If $x < \theta(k)$ the job is rejected and accepted otherwise.

With unknown batches, the optimal cost is

$$J(x) = \min_{\theta \in \{0,\dots B\}} \alpha\Big(\lambda' \sum_k P_k J(x + k\delta_{x<\theta}) + \mu' \min\{x, s\} J(x-1)$$
$$+(1 - \lambda' - \mu' \min\{x, s\})J(x)\Big) + cx + \lambda' RW\delta_{x \geq \theta},$$

Index brokering of known batches

Once again, we focus on local index based policies.
The optimal policy is of threshold type, depending on k the size of the incoming job : If $x < \theta(k)$ the job is rejected and accepted otherwise.
With unknown batches, the optimal cost is

$$J(x) = \min_{\theta \in \{0,\dots B\}} \alpha\Big(\lambda' \sum_k P_k J(x + k\delta_{x<\theta}) + \mu' \min\{x, s\} J(x-1) + (1 - \lambda' - \mu' \min\{x, s\}) J(x)\Big) + cx + \lambda' RW \delta_{x \geq \theta},$$

Here the optimal cost is

$$J(x,k) = \min_{\theta \in \{B\}^{\{K\}}} \alpha\Big(\lambda' \sum_{k'=1}^{K} P_{k'} J(x + k\delta_{x<\theta(k)}, k') + \mu' \min\{x, s\} J(x-1,k) + (1 - \lambda' - \mu' \min\{x, s\}) J(x,k)\Big) + cx + \lambda' Rk \delta_{x \geq \theta(k)}$$

This makes the computation of the optimal index possible (but much harder than in the unknown case).

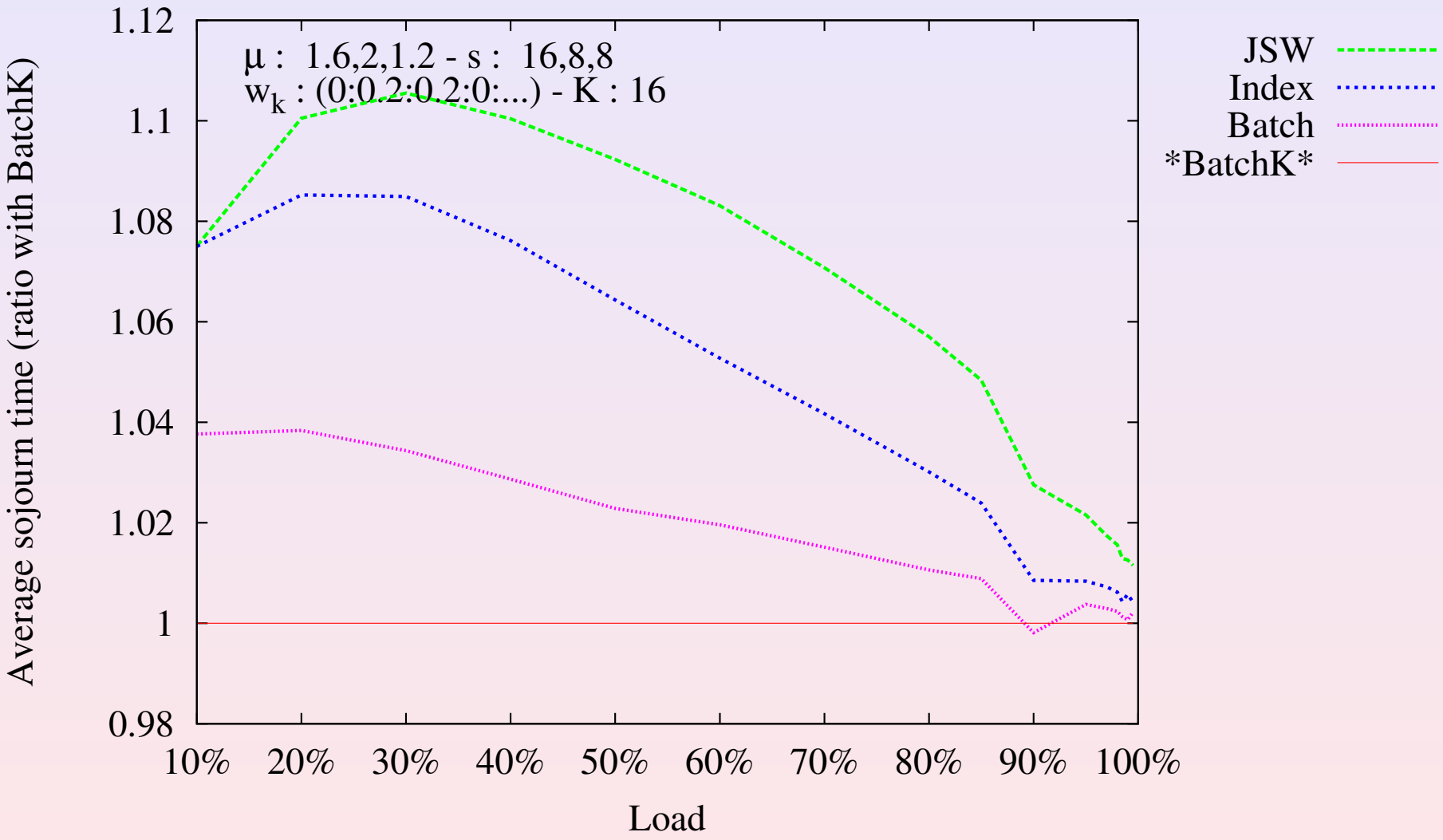
Bi-dimensional index

$\Theta(R, k)$ is the optimal threshold given the rejection cost R, when the size of an incoming job is k.

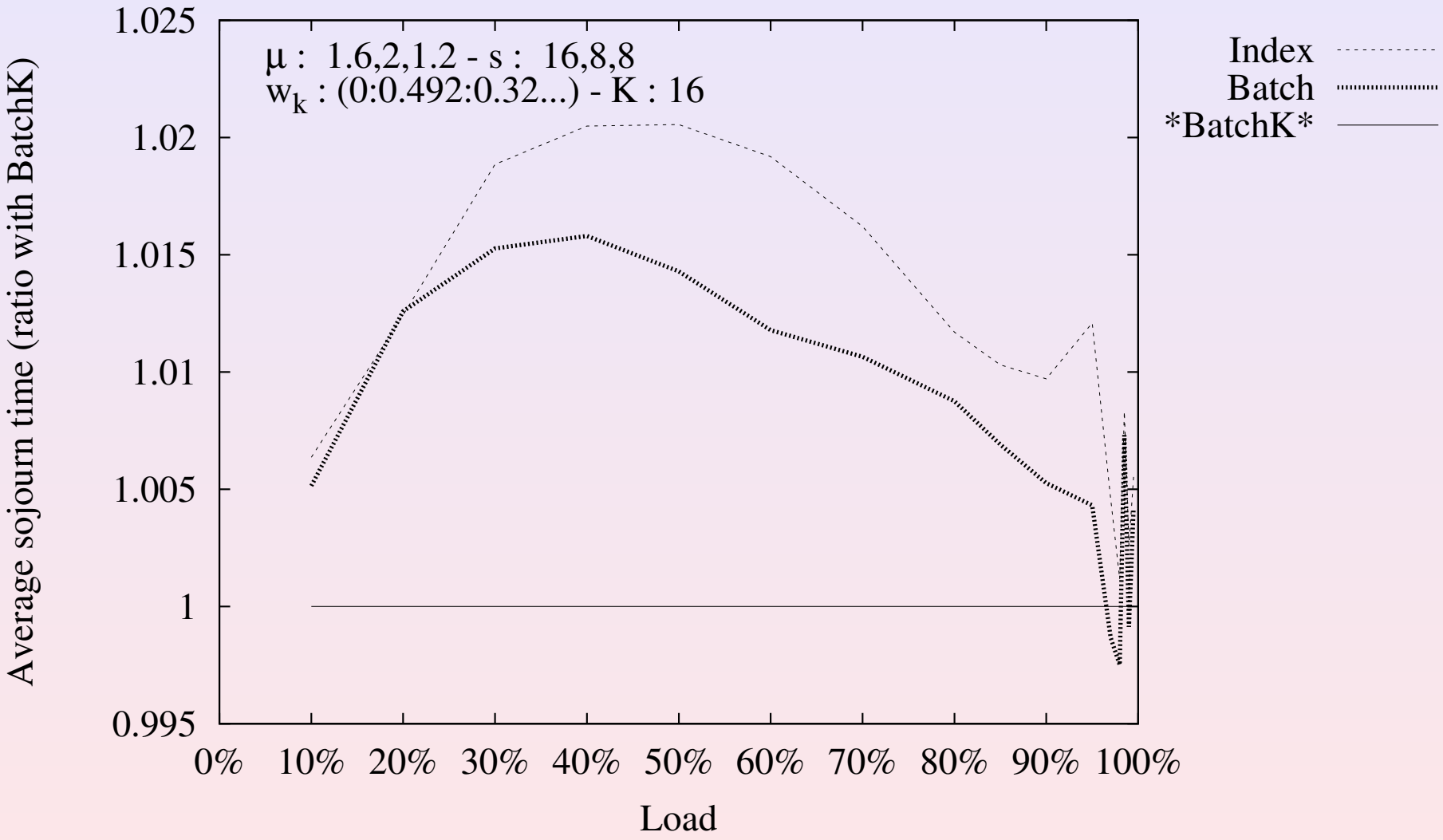
Bi-dimensional index

$\Theta(R, k)$ is the optimal threshold given the rejection cost R, when the size of an incoming job is k.
The local index is : $\forall k, \quad L(x, k) = \sup\{R | \Theta(R, k) \leq x\}$.
$L(x, k)$ is the cost of accepting a job of size k in the queue if the state is x.
Now, the LCR policy is : an upcoming job of size k is sent to the queue with the smallest index, $L_i(x_i, k)$.
This new policy should improve on all policies discussed before. The question is by how much. This is usually called the value of information.

Bi-dimensional indexes



Bi-dimensional indexes



Conclusions

- Experimental validation of the usability of indexes to grid brokering

Conclusions

- Experimental validation of the usability of indexes to grid brokering
- Extension of the classical index model to batch arrivals, and bi-dimensional indexes

Conclusions

- Experimental validation of the usability of indexes to grid brokering
- Extension of the classical index model to batch arrivals, and bi-dimensional indexes
- BatchK : more efficient if $P_k \gg 0$ for large k, similar to Batch otherwise

The End