Index routing for task allocation in Grids

Vandy Berten, Bruno Gaujal

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Index routing for task allocation in Grids

Vandy Berten — Bruno Gaujal

N° 5892

Avril 2006

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--5892--FR+ENG

INRIA
RHÔNE-ALPES

Index routing for task allocation in Grids

Vandy Berten [*] , Bruno Gaujal [†]

Thème NUM — Systèmes numériques
Projet MESCAL

Rapport de recherche n° 5892 — Avril 2006 — 20 pages

Abstract: In this paper we show how index routing policies can be used in practice for task allocation in computational grids. We provide a fast algorithm which can be used off-line or even on-line to compute the index tables. We also report numerous simulations providing numerical evidence of the great efficiency of our index routing policy as well as its robustness with respect to parameter changes.

Key-words: Index Policy, Optimal Routing, Markov Decision Process, Grids

[*] FNRS, Département d'Informatique, Université Libre de Bruxelles, Belgium (Email: Vandy.Berten@ulb.ac.be)
[†] INRIA, Lab. ID-IMAG (CNRS, INPG, INRIA, UJF), 51, Av. J. Kunztmann, Montbonnot, France (Email: Bruno.Gaujal@imag.fr)

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Allocations de tâches sur grilles par politiques d'index

Résumé : Dans cet article, nous montrons comment des politiques de routage utilisant des critères de coûts locaux (ou index) peuvent être utilisées en pratique pour faire de l'allocation de tâches dans des grilles de calcul. Nous fournissons un algorithme rapide de calcul des tables d'index, qui peut être utilisé hors-ligne mais aussi en-ligne. Cette technique est ensuite validée par de nombreuses simulations qui montrent la grande efficacité de notre politique d'index ainsi que sa grande robustesse aux changements de paramètres.

Mots-clés : Politiques d'index, Stratégie de Routage Optimal, Processus de Décision Markoviens, Grilles

1 Introduction

Grids are becoming popular infrastructures for a large part of the research world for intensive computational tasks. The whole scientific community agrees that physicists, biologists, or even mathematicians, won't be able to tackle tomorrow's problems without such computational systems. Grids are on the way of being efficient and usable systems, but computer scientists and engineers have still a huge amount of work to improve their efficiency. Amongst a large number of problems to be solved or to improved upon, the problem of scheduling the work and balancing the load is of first importance.

The aim of this paper is to propose an efficient but simple routing (or meta-scheduling, or brokering) strategy for computational grids having a centralized resource broker based architecture, such as EGEE (Enabling Grids for E-sciencE [1]), GridPP [2] or Grid 5000 [3]. The functioning of such a system is rather simple: a set of resources (clusters, ...) is available to a *resource broker* (RB). When a user wants some work to be executed, (s)he sends its job to this resource broker, which has in charge to choose a resource, and to send (or route) the job to the selected resource. A job then never waits in the RB (except the time needed for choosing the resource), but will eventually be queued into the local resource, where it will be scheduled thanks to classical well known cluster scheduling techniques.

As it is known that finding an optimal scheduling (for the main common criterions) is a difficult problem, heuristic methods are often required. For instance, in EGEE [1], the user can define a *rank*, which expresses preferences between resources. If this rank is the number of free CPUs (not running any job), the job will be sent to the eligible resource having the largest number of free CPUs. The rank can be based on the number of jobs waiting in the queue, or (by default) on an estimation of the sojourn time, based on the current queue size, and the average sojourn time observed recently. In this paper we propose an adapted version of Whittle indices as local cost functions and assess its performances.

Indeed, this class of problems can be seen as *restless Bandit Problems* introduced by Whittle [4]. Unfortunately, Restless Bandit Problems are known to be P-space hard [5] in general, while the precise complexity of optimal routing to several parallel queues, is still open and is acknowledged to be difficult. The research focus has shifted to tractable sub-optimal policies such as in [6]. Whittle has shown that *index policies* are good candidates as solutions to this problem [4] because they have several optimality properties. Whittle indices have been used successfully in [7] to control queues with breakdowns. This approach has also been extended in various directions in [8, 9, 10], for example.

The goal of this paper is to show that index policies can be used in practice because they are very efficient and very easy to compute. The first goal of this paper is to design fast algorithms to compute the indices so that they can be used on-line (*i.e.* recomputed whenever major changes occur in the system). The naive approach (given in the paper) has a $O(eB^4)$ complexity while our most advanced algorithm has a $O(eB^2 + B^2 \log B)$ complexity (here B is the buffer size and e is the required number of precision digits). This is done by showing several properties of our index policy (threshold optimality as well as monotony, univoque and convexity properties).

Second, we test the efficiency of index policies for multi-server queues as a well founded alternative to intuitive policies such as JSQ for task allocation in grids. The numerical experiments provided in Section 4 show how well it behaves in terms of performance, and robustness. In all our experiments, the performance (average sojourn time) of our index based routing policy stays within 2% from the optimal policy and is always better than all classical policies (such as Join the Shortest Queue). Furthermore, our policy is very robust to parameter changes. Combining our index policies computed for loads 0.5 and 0.9 is enough to achieve very good performances (less than 2% loss) over the whole range of loads from 0 to 1.

Our paper is structured as follows. Section 2 presents the overall problem and its model as parallel Markovian queues. It also introduces the index policy and shows its threshold structure and presents a first algorithm to compute the index. Section 3 focuses on algorithmic issues, presenting several improvements leading to a quadratic complexity on average. Finally Section 4

displays several simulations comparing the index policy with the optimal policy for small systems as well as with several classical policies such as JSQ or several types of dual μc rules.

2 Index policies and Bellman equations

2.1 Grid system models

In this paper a grid computing systems is seen as an heterogeneous set of independent clusters. All computing nodes in the same cluster are identical. In one cluster, all tasks are queued in FIFO order into one finite buffer and are allocated to free nodes in an arbitrary fashion.

This is an approximation of the actual behavior of current grid computing systems. The main simplifications used here are the independence between all tasks (they usually have dependences) and the FIFO queueing policy in each cluster (they usually have priority and/or reservation features).

Now, we consider a computational grid as being a set of finite capacity multi-server queues linked together by a *router* or a *resource broker*, through which clients send their jobs. Figure 1 shows our model of computational grid.

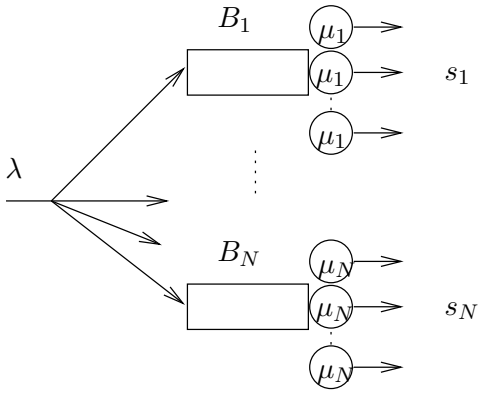

Figure 1: Queuing model of a Computational Grid with Resource Broker. The Grid is composed of N *clusters* (or queue), the i^{th} queue being composed of s_i *CPUs* (or servers) of speed (or rate) μ_i. The system input of rate λ is routed amongst the N queues.

More formally, our system can be described as follows: a router gets a Poisson stream of jobs at rate of λ. It chooses one queue amongst N, and sends the job to the chosen queue, or reject it, if all queues are full.

Each queue is of type $\cdot/M/s_i/B_i/$ FIFO, meaning that the service rate is exponential (with rate μ_i), there are several servers (s_i), the buffer as a finite capacity (B_i) and jobs are served in each queue using the FIFO policy.

In addition, the following notations will be used throughout the paper.

- x_i is the *queue i state*, or the number of jobs currently present in the queue (waiting and running). $x_i \in \{0, \dots, B_i\}$.
- $x = \{x_1, \dots, x_N\}$ is the *system state*.
- S is the *state space*:
  $S = \{0, \dots, B_1\} \times \cdots \times \{0, \dots, B_N\}$.
- U is the *action space*, or the set of actions that the router can choose. An action is either i (if the queue i is chosen), or 0 (if the action is the rejection of a job). Rejection is only allowed when all queues are full. Then, $U = \{1, \dots, N\} \cup 0$ iff $x = \{B_1, \dots, B_N\}$.

In the following, we will focus on routing policies minimizing the expected discounted workload in infinite horizon[1].

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

After uniformization, the cost under policy $\pi = (u_0, u_1, \dots)$ with initial state $x^{(0)} = (x_1^{(0)}, \dots, x_N^{(0)})$ is

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

where α is the discounting factor and

$$h(x^{(k)}, u_k(x^{(k)})) = \sum_{i=1}^{N} c_i((x_i^{(k)} + \delta_{\{u_k(x^{(k)})=i\}})),$$

where c_i is the unit cost in the i-th queue per customer per step.

As the cost is uniformly bounded over the state space, we will only consider time independent routing policies. A *routing policy* u is a function which gives which action to take in each state. Then it is a function $u : S \to U$.

This optimal routing problem can be solved in a very classical way using dynamic programming techniques. Computing the optimal policy boils down to solving a Bellman fixed point equation, which can be proved to have a single solution using general techniques (see [11]). Solving Bellman equations for MDPs is usually more efficient using policy iteration techniques for which the average complexity is often polynomial in the size of the state space.

The problem here is that the size of the state space increases exponentially with the number of queues. This problem, often called the "curse of dimensionality", is acute here. Indeed, the problem becomes too large to be handled by modern computers as soon as $N \geq 4$ (for queues of size 100).

This is why one needs to tackle the problem using a scalable approaches.

2.2 Local Criterion based Routing

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

- For each queue, we define a function which associates a real number to each state of this queue. This function is called the *local cost function* of the queue: $L_i : \{0, \dots, B_i\} \to \mathbb{R}^+$. This function can be considered as an vector of dimension $B_i + 1$.
- We define the *current local cost* of the queue i as $L_i(x_i)$, where x_i is the current state of the queue i.
- The *routing policy* is the following:

  $$u(x_1, \dots, x_N) = argmin\{L_1(x_1), \dots, L_N(x_N)\}.$$

  An arriving job is then sent towards the queue which has the smallest current local cost.

Such policies have several advantages in practice. They scale with the number of parallel queues (the only global operation is the argmin operation. The rest of the computation can be done locally). They are also amenable to perfect simulation [12] as soon as the ranks L_i are non-decreasing in x_i. Most classical policies such as Bernoulli routing, JSQ or JSW (Join the Shortest Waiting time) are LCR. On the rest of the paper, we propose a new LCR policy based on local costs which are Whittle indices adapted to multi-server queues.

[1] A similar question without discounting can also be considered. The approach used the rest of the paper also applies for long run average costs with minor adaptations. These extensions will not be discussed further in this paper.

2.3 Optimal index policy is of threshold type

Here is the way to construct the local cost function for each queue. We consider each cluster (*i.e.* a multi-server queue) in isolation and use a free parameter R, as a rejection cost, as seen in Figure 2.

The first step to construct optimal admission policy in each queue which is of threshold type. The optimal threshold is a function of the rejection cost, $\Theta_i(R)$. Then, the local cost or *index* for this queue is the inverse function of $\Theta_i(R)$:

$$\mathtt{I}_i(x_i) = \sup\{R | \Theta_i(R) \leq x_i\}.$$

In the following sections, devoted to the computation of $\Theta(R)$, we will focus on one specific queue. In order to simplify our notations, the subscript i of the queue will be dropped.

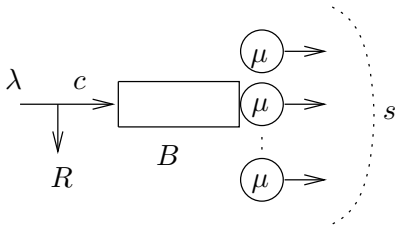

Figure 2: Optimization problem to get the index for each queue, with the additional rejection cost, R.

For one queue, the optimal control problem is to find the optimal control u (accept or reject each incoming customer) in order to minimize the long run discounted cost with initial state x_0, after uniformization by this total rate $\Lambda = \lambda + \max_i \mu_i s_i$,

$$J_u(x_0) = \limsup_{n\to\infty} \mathbb{E} \sum_{k=0}^{n-1} \alpha^k (cx^{(k)} + R\delta_{\{u_k=0\}}\delta_{\{\text{arrival at } k\}}).$$

The α-discounted cost for the optimal policy with initial state x will be denoted $J^*(x)$.

Theorem 2.1. *The optimal policy minimizing the α-discounted cost in infinite horizon $J^*(x)$ for one $M/M/s/B$ queue (or one $M/M/s/\infty$ queue) is of threshold type.*

Proof. This theorem is rather classical (see for example [11]). We provide our own proof for sake of completeness but also because the proof will be useful in Theorem 4 for computation purposes. For the infinite capacity queue, define J^n by $J^0(x) = 0$ for all x and

$$\begin{aligned} J^{n+1}(x) = \quad & \alpha\Big(\quad \lambda' \min\{R + J^n(x), J^n(x+1)\} \\ & \quad + \quad \mu' \min\{s, x\} J^n(x-1) \\ & \quad + \quad (1 - \lambda' - \mu' \min\{s, x\}) J^n(x) \Big) \\ & + cx \end{aligned}$$

where $\lambda' = \dfrac{\lambda}{\Lambda}$, and $\mu' = \dfrac{\mu}{\Lambda}$.

The first step is to show that $J^n(x)$ is convex in x. This is done by induction on n: for all $x \geq 0$, we show in the appendix A that $2J^n(x+1) \leq J^n(x) + J^n(x+2)$, using a case analysis and direct computations.

Now, $J^n(x)$ being convex in x, there exists an optimal policy for each n which is of threshold type as shown in [11].

For any $0 < \alpha < 1$, it is well known that $J^n(x) \to J^*(x)$ when n goes to infinity and J^* is the unique solution of the Bellman Equation,

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

So that the discounted cost is also minimized by a threshold policy.
Now consider the finite capacity case. The equations are the same for all $x < B$
As for $x = B$, the next arrival is surely rejected which incurs a cost of R. Hence

$$\begin{aligned} J^{n+1}(B) = \quad & \alpha\Big(\quad \lambda'(R + J^n(B)) \\ & \quad + \quad \mu' \min\{s, B\} J^n(B-1) \\ & \quad + \quad (1 - \lambda' - \mu' \min\{s, B\}) J^n(B) \Big) \\ & + cB. \end{aligned}$$

The function J^n verifies $2J^n(x+1) \leq J^n(x) + J^n(x+2)$, and the optimal policy is of threshold type for each n. Taking the limit when n goes to infinity in the previous inequality shows that the optimal policy is also of threshold type. □

Actually, we will see later (in Lemma 4) how the thresholds for the infinite and the finite cases are related.

Partial optimality of a LCR using our indices as local criterion functions can already be stated.

Proposition 1. *A Local Criterion Routing based on the indices described in 2.2 is optimal in either of the two following cases,*

i when all queues have the same number of servers, all with the same service rate;

ii when the buffer sizes are smaller than the number of servers in each queue ($B_i \leq s_i$, for all i).

Proof. Case i is based on the fact that Join the Shortest Queue (JSQ) is optimal when the system is symmetric [13] and on the fact that the index policy coincides with JSQ in such a case. This last point is a direct consequence of the fact that the index function is increasing with the queue size (as seen in the proof of Theorem 2.1). As for Case ii, this is a direct consequence of the fact that when the buffer size is smaller than the number of servers the system can be seen as a collection of $\sum_i B_i$ servers with no waiting room which are either active or idle to which customers are routed. This is a Multi-armed Bandit Problem for which the index policy introduced above is optimal as shown in [14]. □

In the rest of the paper, an algorithm is designed for computing the optimal threshold for index based computational grid routing very fast. This will be one of the main contribution of this paper. This problem is solved in two parts:

- First we have to compute the optimal threshold problem for a queue with reject: this will give us a function $\Theta : \mathbb{R}^+ \to \{0, \ldots, B\}$, which gives, for a cost of reject R, the optimal threshold θ. This is done by dynamic programming techniques.
- Then, we have to inverse Θ, in order to obtain the index $\mathtt{I} : \{0, \ldots, B\} \to \mathbb{R}^+$. More precisely, this index $\mathtt{I}$ is defined as follows: $\mathtt{I}(x) = \{\sup R | \Theta(R) \leq x\}$. This is done by a generalized dichotomy.

2.4 Computing the Optimal Threshold

The objective of the section is to design an algorithm to find the optimal threshold policy in the system described in Figure 2, *i.e.* a $M/M/s/B$ queue (where s is the number of servers, and B is the queue size). A policy u_θ with *threshold* θ is defined as follows. When a job arrives while x jobs are currently present in the system (x is the *queue state*),

- if $x < \theta$, the new job is accepted, and will cost c per step (time between two events) spent in the system,
- otherwise ($x \geq \theta$), the job is rejected, and costs R (only once).

Bellman's equation (1) for the α-discounted cost of one queue with rejection, can be adapted for threshold policy u_θ. By switching the order of the actions and the events, one gets:

$$\begin{aligned} J(x) = \quad & \alpha\Big(\quad \lambda' J(x+\delta_{x<\theta}) \\ & \qquad + \quad \mu' \min\{x,s\} J(x-1) \\ & \qquad + \quad (1-\lambda'-\mu'\min\{x,s\}) J(x) \Big) \\ & + c(x + \lambda'\delta_{x<\theta} - \mu'\min\{x,s\}) \\ & + R\lambda'\delta_{x\geq\theta}, \end{aligned} \tag{2}$$

where $J(x)$ is the infinite horizon α-discounted cost starting with x jobs in the queue.

Equation (2) can easily be rewritten in a matrix form. Let F_θ be a matrix $B \times B$ and S_θ a matrix $B \times 1$, defined as follows:

$$F_\theta(x,y) = \begin{cases} \mu' \min\{x,s\} & \text{if } x = y-1 \\ 1 - \lambda'\delta_{x<\theta} - \mu'\min\{x,s\} & \text{if } x = y \\ \lambda'\delta_{x<\theta} & \text{if } x = y+1 \\ 0 & \text{otherwise.} \end{cases}$$

$$S_\theta(x) = c(x + \lambda'\delta_{x<\theta} - \mu'\min\{x,s\}) + R\lambda'\delta_{x\geq\theta}.$$

Equation (2) can now be written as:

$$J = \alpha F_\theta J + S_\theta. \tag{3}$$

F_θ can be seen as being the impact of future on the cost, S_θ as the cost of the current step.

Finding the threshold θ which gives the smallest cost corresponds to looking for a θ solution of the following system, which is the same as equation (2) with a min on the right hand side:

$$J = \min_{\theta \in \{0,\dots,B\}} (\alpha F_\theta J + S_\theta). \tag{4}$$

We will call $\Theta(R)$ the function which gives the optimal threshold for a rejection cost R (assuming that other parameters are constant).

If all parameters are known (R and θ included), computing $J(x)$ $\forall x \in \{0,\dots,B\}$ (or solving equation (3)) is rather easy. We just need to solve the following tridiagonal linear system, which can be done in $O(B)$:

$$(\alpha F_\theta - I)J = -S_\theta. \tag{5}$$

Let J_θ be the solution of the system from equation (3), found by solving equation (5), for a chosen θ. Then, J_θ satisfies

$$J_\theta = \alpha F_\theta J_\theta + S_\theta. \tag{6}$$

Once J_θ is computed, one can define the cost $J_{\theta,\theta'}$, by using values found for J_θ under policy $u_{\theta'}$:

$$J_{\theta,\theta'} = \alpha F_{\theta'} J_\theta + S_{\theta'}. \tag{7}$$

Once J_θ is known, $J_{\theta,\theta'}(x)$ can be obtained for all x and θ' without solving any other system, it just requires to replace values into the formula.

Lemma 1. *if θ' is such that*

$$J_{\theta,\theta'}(x) \leq J_\theta(x) \ \forall x \in \{0,\ldots,B\}$$

then,

$$J_{\theta'}(x) \leq J_\theta(x) \ \forall x \in \{0,\ldots,B\}.$$

Proof. This is a rather direct corollary of the policy iteration principle to compute the optimal policy. Here, there exists an optimal policy of threshold type. Therefore, one can use the policy iteration by restricting the choices of the new policy to threshold policy in policy iteration scheme. The inequality $J_{\theta,\theta'}(x) \leq J_\theta(x)$ means that the policy with threshold θ' has a smaller cost than the current policy with threshold θ, used to compute the cost. This is one possible iteration of the policy iteration for thresholds only. □

This result is the basis of our first algorithm:

1. choose θ
2. solve J_θ
3. find θ^* such as $J_{\theta,\theta^*}(x) \leq J_\theta(x) \ \forall x \in \{0,\ldots,B\}$
4. restart in 2. with θ^* as long as $\theta \neq \theta^*$.

In this version, we choose $\theta^* = argmin_{\theta'} \sum_{x=0}^{B} J_\theta(x) - J_{\theta,\theta'}(x)$. The worse case complexity of this algorithm is $O(B^3)$ (because bounds are known for $\Theta(R)$).

This allows us to compute the function $\Theta(R)$ from $\mathbb{R}^+$ into $\{0,\ldots,B\}$. This function will be the base of our index $\mathtt{I}(x)$, which goes from $\{0,\ldots,B\}$ into $\mathbb{R}^+$. $\mathtt{I}(x)$ is then an array. $\mathtt{I}$ is defined as the inverse of Θ, or, more exactly, $\mathtt{I}(x)$ gives the larger rejection cost R such as $\Theta(R) \leq x$. Or,

$$\mathtt{I}(x) = \sup\{R|\Theta(R) \leq x\}.$$

Functions $\mathtt{I}$ and Θ are presented on Figure 3.

Lemma 2. *Under the foregoing notations, the functions $\Theta(R)$ and $\mathtt{I}(x)$ have the following properties.*

i $\mathtt{I}(0) = \mathtt{I}(1) = \cdots = \mathtt{I}(s-1)$.

ii $\Theta(R)$ *and* $\mathtt{I}(x)$ *are non-decreasing.*

iii *For all* $R \geq \frac{c}{1-\alpha}$, $\Theta(R) = B$.

Proof. Point *i* is a direct consequence of the structure of the cost function. As long as at least one server is idle, the optimal action only depends on the comparison between the discounted service time and the rejection cost and is independent on the actual number of active servers. Therefore, $\Theta(R) = 0$ when R is small enough and $\Theta(R) \geq s$ as soon as R is larger than the discounted service time. This implies $\mathtt{I}(0) = \mathtt{I}(1) = \cdots = \mathtt{I}(s-1)$.

Point *ii* is a direct consequence of the Bellman's Equation (1).

As for point *iii*, let us first prove that for a queue with an infinite capacity, no packet is never rejected if $R > \frac{c}{1-\alpha}$.

For that, let us consider $J^*(x+1) - J^*(x)$, the difference of the cost when starting with $x+1$ instead of starting at x. Since the optimal policy is of threshold type, on every trajectory of the system, whenever a packet is rejected when starting with x, it is also rejected when starting with $x+1$. Therefore, the difference in the number of packets present in both systems under the optimal policy remains non-negative and is always bounded by one. Therefore, on average,

$$J^*(x+1) - J^*(x) \leq \sum_{k=0}^{\infty} c\,\alpha^k = \frac{c}{1-\alpha}.$$

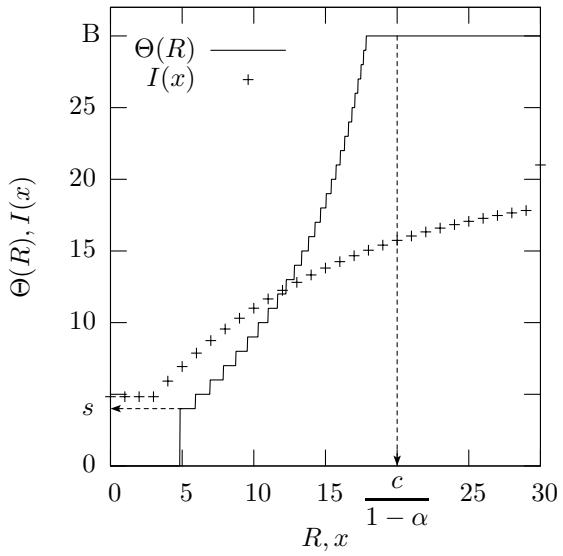

Figure 3: The "best threshold" function $\Theta(R)$ (continuous line) and its "inverse" $\mathtt{I}(x)$ (dots) which we use as Index function. We point out some characteristics, such as the ceiling of $\Theta(R)$ in B reached before $\frac{c}{1-\alpha}$, or the first step at s (cf. Lemma 2). The parameters are $\mu = 1$, $\lambda = 0.8$, $s = 4$, $c = 1$, $B = 30$, $\alpha = 0.95$.

Therefore, if $R \geq \frac{c}{1-\alpha}$, then $J^*(x+1) \leq R + J^*(x)$. Now, looking at Bellman's equation (1) for $J^*(x)$, this means that $\min\{J^*(x+1), R + J^*(x)\}$ is always reached by $J^*(x+1)$, whatever x, so that no packet is ever rejected under the optimal policy and $\Theta(R) = +\infty$.

The rest of the proof is a rather direct consequence of Lemma 4. Indeed, applying the lemma with $\theta^* = +\infty$, any B satisfies $B \leq \theta^*$ so that $\Theta(R) = B$. $\Theta(R)$ has then a vertical asymptote in $\dfrac{c}{1-\alpha}$. □

Lemma 3. *If $\theta \geq \theta' \geq \Theta(R)$ (or if $\theta \leq \theta' \leq \Theta(R)$), then $J_\theta(x) \geq J_{\theta'}(x)$, for all x.*

Proof. We only consider the case $\theta \geq \theta' \geq \Theta(R)$ (the other case being similar). The policy $u_{\theta'}$ coincides with u_θ over all states x such that $x \geq \theta$ (reject) and $x < \theta'$ (accept). For states $\theta' \leq x < \theta$, u_θ coincides with the optimal policy (reject) while u_θ accepts. The end of proof is a straightforward consequence of the policy iteration principle and of the convex increasing shape of the cost function, to make sure that policy $u_{\theta'}$ is better than u_θ. □

3 Algorithmic improvements and complexity

In this section, several improvements of the algorithm used to compute Θ (and thus $\mathtt{I}$) are explained. They are mostly based on Lemmas 2, 3 and 4.

3.1 First improvement: algebraic simplifications

We first explain how to speed up the loop by checking if the current θ' is admissible and by computing the gain of θ' faster.

Let $g(x, \theta, \theta') = J_\theta(x) - J_{\theta,\theta'}(x)$. From equations (6) and (7), we have

$$\begin{aligned} g(x,\theta,\theta') &= \alpha\lambda'(J_\theta(x+\delta_{x<\theta}) - J_\theta(x+\delta_{x<\theta'})) \\ &+ \lambda' c(\delta_{x<\theta} - \delta_{x<\theta'}) \\ &+ \lambda' R(\delta_{x\geq\theta} - \delta_{x\geq\theta'}). \end{aligned}$$

We will now check the relation between x, θ and θ'.

$$g(x,\theta,\theta') = \begin{cases} 0 & \text{if } x < \min\{\theta,\theta'\} \\ 0 & \text{if } x \geq \max\{\theta,\theta'\} \\ \gamma & \text{if } \theta \leq x < \theta' \\ -\gamma & \text{if } \theta' \leq x < \theta, \end{cases}$$

where $\gamma = \lambda'\big(\alpha(J_\theta(x) - J_\theta(x+1)) + R - c\big)$.

The admissibility condition for θ' is now rather easy; a threshold θ' is admissible (ad.) iff $J_{\theta,\theta'}(x) \leq J_\theta(x)\ \forall x \in \{0,\ldots,B\}$ (see Lemma 1), or $g(x,\theta,\theta') \geq 0\ \forall x \in \{0,\ldots,B\}$. Then,

$$\text{If } \theta < \theta', \left[\exists x \in [\theta,\theta'[: J_\theta(x) - J_\theta(x+1) < \frac{c-R}{\alpha}\right] \Leftrightarrow \neg\text{ad.} \tag{8}$$

$$\text{If } \theta' < \theta, \left[\exists x \in [\theta',\theta[: J_\theta(x) - J_\theta(x+1) > \frac{c-R}{\alpha}\right] \Leftrightarrow \neg\text{ad.} \tag{9}$$

The condition has only to be checked for $x \in \{\min\{\theta,\theta'\},\ldots,\max\{\theta,\theta'\}\}$, and not for $x \in \{0,\ldots,B\}$ as in the first algorithm, and the values to compare are really simpler to compute.

Among all admissible θ', we choose the one which maximizes $\sum_{x=0}^{B} g(x,\theta,\theta')$. This allows to have a trivial selection test. If $\theta < \theta'$:

$$\sum_{x=0}^{B} g(x,\theta,\theta') = \lambda'\big(\alpha(J_\theta(\theta) - J_\theta(\theta')) + (\theta' - \theta)(R-c)\big).$$

We have the same result for $\theta' < \theta$. The θ' which maximizes this last equation also maximizes:

$$G(\theta,\theta') = -\alpha J_\theta(\theta') + \theta'(R-c). \tag{10}$$

Then for each admissible θ', we choose the one maximizing $G(\theta,\theta')$. Notice that $G(\theta,\theta')$ is not the gain between θ and θ', the gain is: $\lambda\big(\alpha J_\theta(\theta) - \theta(R-c) + G(\theta,\theta')\big)$. The stop condition has to be adapted.

The algorithm is now the following:

1. choose θ
2. solve J_θ
3. For each admissible θ' (check if there exists a x making θ' not admissible using equations (8) and (9))
   - $G \leftarrow -\alpha J_\theta(\theta') + \theta'(R-c)$ (see equation (10))
   - if G is better than the current best one (G^*), remember $\theta^* \leftarrow \theta'$, and $G^* \leftarrow G$
4. restart in 2. while $\alpha J_\theta(\theta) - \theta(R-c) + G^* > 0$ with $\theta \leftarrow \theta^*$

3.2 Second improvement: admissibility check improvements

Let `Adm(`θ'`)` be a function checking the admissibility of θ' using equations (8) and (9). We remark that if θ' goes from $\theta+1$ up to B, and if `Adm(`θ'`)` is false, then `Adm(`$\theta'+1$`)` will be false as well. Then if $\theta' \in \{\theta+1,\ldots,B\}$ is not admissible, it is not worthy to check another $\theta'' > \theta'$. The same if true if θ' goes from $\theta-1$ down to 0, it is not worthy to check $\theta'' < \theta'$ if θ' is not admissible.

Furthermore, if θ' goes from $\theta+1$ up to B, we only need to check that $J_\theta(\theta') - J_\theta(\theta'+1) > \dfrac{R-c}{\alpha}$, because every $x \in \{\theta+1,\ldots,\theta'\}$ has already been checked when the analyzed θ' was x.

If θ' goes from $\theta-1$ down to 0, we only need to check that $J_\theta(\theta'+1) - J_\theta(\theta') < \dfrac{R-c}{\alpha}$.

This improvement is shown in Algorithm 1.

Algorithm 1: Gives the best threshold $\Theta(R)$ for a rejection cost R (second improvement)

Data: Rejection cost R, buffer size B
Result: $\Theta(R)$
$\theta^* =$ first estimation;
repeat
  $\theta \leftarrow \theta^*$;
  $G^* \leftarrow -\alpha J_\theta(\theta) + \theta(R-c)$;
  Find J_θ, solution of $(\alpha F_\theta - I)J = -S_\theta$;
  $\theta' \leftarrow \theta + 1$;
  while $\theta' \leq B$ **and** $\big(J_\theta(\theta') - J_\theta(\theta'-1)\big) \leq \dfrac{R-c}{\alpha}$ **do**
    $G \leftarrow -\alpha J_\theta(\theta') + \theta'(R-c)$;
    if $G > G^*$ **then**
      $G^* \leftarrow G$; $\theta^* \leftarrow \theta'$;
    $++\theta'$;
  $\theta' \leftarrow \theta - 1$;
  while $\theta' \geq 0$ **and** $\big(J_\theta(\theta'+1) - J_\theta(\theta')\big) \geq \dfrac{R-c}{\alpha}$ **do**
    $G \leftarrow -\alpha J_\theta(\theta') + \theta'(R-c)$;
    if $G > G^*$ **then**
      $G^* \leftarrow G$; $\theta^* \leftarrow \theta'$;
    $--\theta'$;
until $\alpha J_\theta(\theta) - \theta(R-c) + G^* = 0$;
return θ^*;

3.3 Third improvement: reducing the problem size

Lemma 4. *Let θ^* the optimal threshold for a system with a queue size of $B > \theta^*$. Then,*

1. *θ^* is the optimal threshold for any identical system with queue size $B' > \theta^*$.*
2. *B' is the optimal threshold for any identical system with queue size $B' \leq \theta^*$.*

Proof. This is a rather direct corollary of the form of the J function used in the proof of Theorem 2.1. The optimal policy with buffer capacity B satisfies Bellman's Equation; If $0 \leq x < B$,

$$\begin{aligned} J_B^*(x) = \quad & \alpha\Big(\quad \lambda' \min\{R + J_B^*(x), J_B^*(x+1)\} \\ & \quad + \quad \mu' \min\{s,x\} J_B^*(x-1) \\ & \quad + \quad (1-\lambda'-\mu'\min\{s,x\}) J_B^*(x)\Big) \\ & + cx, \end{aligned}$$

and if $x = B$,

$$\begin{aligned} J_B^*(B) = \quad & \alpha\Big(\quad \lambda'(R + J_B^*(B)) \\ & \quad + \quad \mu' \min\{s,B\} J_B^*(B-1) \\ & \quad + \quad (1-\lambda'-\mu'\min\{s,B\}) J_B^*(B)\Big) \\ & + cB. \end{aligned}$$

Since the optimal policy has threshold θ^*, then $R + J_B^*(x) < J_B^*(x+1)$ as soon as $\theta^* \leq x \leq B$.

Now, let us construct a function $H(x)$ defined by $H(x) = J_B^*(x)$ for all $x \leq B$ and for all $x > B$,

$$\begin{aligned} H(x) = \quad & \alpha\Big(\quad \lambda'(R + H(x)) \\ & \quad + \quad \mu' \min\{s,x\} H(x-1) \\ & \quad + \quad (1-\lambda'-\mu'\min\{s,x\}) H(x)\Big) \\ & + cx. \end{aligned}$$

The function H is convex (the proof is the same as for J^n). Furthermore, $R+H(x) < H(x+1)$ is true when $\theta^* \leq x < B$ because H and J^*_B coincide up to $x = B$. Convexity of H implies that $R + H(x) < H(x + 1)$ remains true for all $x > B$.

This means that H is a solution of the fixed point equation verified by the optimal cost in the queue with infinite capacity. Since the solution is unique, $H(x) = J^*(x)$. As for any queue with capacity $B' > \theta^*$, H truncated at B' is a solution of the fixed point equation verified by the optimal cost and uniqueness implies $H(x) = J^*_{B'}(x)$ for all $x \leq B'$.

As for the second point in the lemma, assume that there exists $B' < \theta^*$ with an optimal threshold $\theta' < B'$. Using the first point of the theorem at capacity B', this would imply that the optimal threshold for $B > B'$ is also θ'. Since $\theta' < \theta^*$, this is a contradiction. □

This result will greatly help us to improve the complexity of linear solving calls (solving of J_θ), which represents the main cost of our algorithm. Let us imagine that we have previously obtained $\Theta(R)$ the best threshold of R, and that we have now to compute $\Theta(R')$ with $R' < R$. As we know that Θ is an increasing function of R, we know that $\Theta(R') \leq \Theta(R)$. We can then invoke $\Theta(R')$ as if the problem size were, for instance, $B' = \Theta(R)$. Let $\Theta_{B'}(R')$ be this function: from Lemma 4, we know that $\Theta_{B'}(R') = \Theta_{B>B'}(R') = \Theta(R')$.

This method strongly reduces the time spent in solving $J_\theta(x)$. Indeed, solving $J_\theta(x)$ goes from a problem in $O(B)$ to $O(B')$.

In the dichotomy algorithm we will present below, we can show that in a large majority of cases, the result of $\Theta(R)$ can only take 2 consecutive values, we can then use $B' =$ the largest one.

In the following, $\Theta_{B'}(R, \theta')$ will denote the function giving the best threshold, in which we know that the result will be lower or equal to B', and where θ' will be use as a first estimation for the best threshold.

3.4 Computing the index function

In order to compute the index vector $\mathtt{I}(x)$ with precision ϵ, one need to compute $\Theta(R)$ for every R such that $\Theta(R) = \Theta(R + \epsilon) - 1$. Each such R will give us the corresponding value of the index ($\mathtt{I}(\Theta(R)) = R$). The algorithm will be composed of two parts:

1. First we need to find (at least) one point on each step of $\Theta(R)$, meaning that $\forall x \in \{0, ..., B\}$, we have a value R_x such as $\Theta(R_x) = x$. This is done by dichotomy. While we have R_1 and R_2 with $R_1 < R_2$ and $\Theta(R_2) - \Theta(R_1) > 2$, for which we do not have any point between them, we use Θ for some value between R_1 and R_2, for instance $\frac{R_1+R_2}{2}$.

2. Then a second dichotomy is used to get the values of the jumps of Θ. For every couple R_1, R_2 found at the previous step such as $\Theta(R_1) + 1 = \Theta(R_2)$, we search for a value $R_1 < R_m < R_2$ such that $\Theta(R_m) + 1 = \Theta(R_m + \epsilon)$.

A few comments have to be made.

Before starting the algorithm, we need to start with a point before the first jump of $\Theta(R)$, and another one after the last jump. For the first one, $(0, \Theta(0))$ works. For the last one, any $R \geq \frac{c}{1-\alpha}$ is such as $\Theta(R) = B$ (cf. Lemma 2).

During the first phase, when we compute $\Theta(\frac{R_1+R_2}{2})$, we know that the result will be below $\Theta(R_2)$, previously computed. We can then use $\Theta_{B'}(R_m, \theta')$, where $R_m = \frac{R_1+R_2}{2}$, $B' = \Theta(R_2)$, and θ' is any value in $\{\Theta(R_1), \ldots, \Theta(R_2)\}$.

During the second phase, a similar comment can be made: when looking for the jump between R_1 and R_2 (with $\Theta(R_1) + 1 = \Theta(R_2)$), we can invoque $\Theta_{\Theta(R_2)}(R_m, \theta')$, where $\theta' = \Theta(R_1)$ or $\Theta(R_2)$.

It can occur than Θ increases too fast, and that we cannot find some steps. We have then R_1 and R_2 such as $\Theta(R_2) - \Theta(R_1) > 2$ and $R_2 - R_1 < \epsilon$. In that case, we assign every $\mathtt{I}(i)$ for $i \in \{\Theta(R_1), \ldots, \Theta(R_2) - 1\}$ to R_2 (or R_1). This situation should be avoided, because it causes the index vector to be less discriminating. This problem can be reduced by increasing the discount factor α.

3.5 Complexity

Theorem 3.1. *Computing the index vector* $\mathtt{I}(0), \cdots, \mathtt{I}(B)$ *with precision* ϵ *can be done in* $O(eB^2 + B^2 \log B)$ *in the worst case, where* $e = -log(\epsilon)$.

Proof. (*sketch*) The first phase of the algorithm needs to identify $B+1$ values of R, one for each possible threshold.

For a given R, computing $\Theta(R)$ can be done by using a dichotomy over the values of θ in the set $\{0, \cdots, B\}$ by using Lemma 3. The cost for each θ corresponds to solving a tridiagonal system of size smaller than B and has a known solution in $O(B)$ so that the complexity to get $\Theta(R)$ is $O(B \log B)$. Since B values have to be computed, the overall complexity of phase one is $O(B^2 \log B)$.

As for phase 2, the dichotomy imposes to compute less than eB values $\Theta(R)$. Each of them can be done in time $O(B)$ because only two values for $\Theta(R)$ are possible at this point (see the previous comments).

Hence, the overall complexity is $O(eB^2 + B^2 \log B)$. □

Actually, the first phase benefits on average from using the policy iteration with the next "best" θ (as shown in Algorithm 1) instead of using a dichotomy search. While the worse case jumps from $O(\log B)$ to $O(B)$, in practice, using policy iteration with the next best θ converges in less than 3 steps (number of resolutions of the tridiagonal system) in more than 99.9% of our runs.

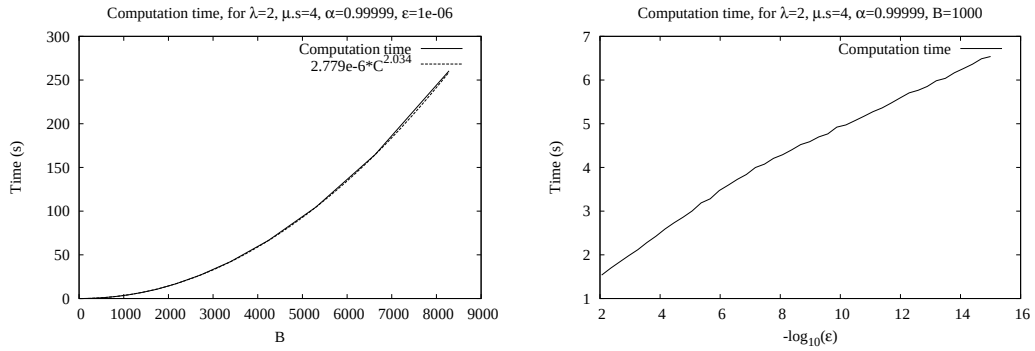

Figure 4: Computation times for the B values of the index of one queue. On the left plot, we show this computation times for various problem sizes (B between 0 and 8000). The quadratic dependence in B is obvious. On the right plot, we vary the precision of the dichotomy between 10^{-2} and 10^{-15}.

We performed some benchmarks allowing to have a better idea about the complexity of the index computation.

If the load if not to high, we observed, as expected, that the computation time is approximately in $O(B^2)$, and in $O(-\log \epsilon)$.

Figure 4 shows the time needed for computing an index for a problem size varying from 0 up to 8000. We put on this plot the function $a \cdot B^b$ fitting the best the curve. For not too high loads, the exponent is, as expected, close to 2. We made these benchmarks on a PC with Intel Celeron 2GHz CPU, with 256 MB of RAM.

Notice that we obtained here times such as 4 minutes for obtaining an index for a problem size of 8000. But in most real situations, the problem size is more like 1000, which requires around 3 seconds for obtaining an index.

4 Numerical experiments

In this section, we will present some simulation results where we compare several routing strategies. We compare our index strategy, some classical LCR strategies, and, when it is possible, the optimal (stochastic) strategy.

The input rate generally varies between 0 and 100% of the total service capacity. The plots display the ratio between the *average sojourn time* in the stationary regime for a strategy and the same metric in the case of optimal routing. Notice that when the discounting factor is close to one and $c = 1$, this is very close to the average discount cost. When the system is too big for computing the optimal routing, we present the ratio with our index policy. The strategy used for comparison is specified in the legend (we put '*' around the name of the strategy of reference).

For every strategies but the optimal and random ones, we use perfect simulation techniques [15] for obtaining our results. This technique happens to be fast for simulation of monotone queueing networks [12], which is the case here.

The optimal and random strategies have been simulated by "classical simulation technique".

4.1 Strategies

As stated in the previous section, several strategies have been compared. Most of them are part of LCR family routing strategies, and are already available in EGEE. Here are the names we use in the plot legends.

`optim`: optimal strategy,

`random`: Bernoulli routing (with weight: $\mu_i s_i$),

`JSQ` (Join the Shortest Queue): $\mathtt{I}_i(x) = x \ \forall x \in [0, \dots, B_i]$,

`JSQ-mu`: $\mathtt{I}_i(x) = \dfrac{x}{\mu_i s_i} \ \forall x \in [0, \dots, B_i]$,

`JSQ-mu2`: $\mathtt{I}_i(x) = \dfrac{x+1}{\mu_i s_i} \ \forall x \in [0, \dots, B_i]$,

`JSW` (Join the Shortest Waiting time):

$$\mathtt{I}_i(x) = \begin{cases} \dfrac{1}{\mu_i} & \forall x \in [0, \dots, s_i[\\ \dfrac{x+1}{\mu_i s_i} & \forall x \geq s_i, \end{cases}$$

`Index`: "our" strategy.

Notice that in the case of monoprocessor systems, `JSQ-mu2` and `JSW` are equivalent.

Each LCR curve required approximately between 8 and 10 hours of computations. The time needed for obtaining and simulate the optimal routing was longer: up to 5 days of CPU for a curve. Each plot below required then approximately one week of computation at 3.4 GHz. This was however highly parallelizable, which allowed us to obtain a plot in a few hours on a cluster.

4.2 Monoprocessor systems

In Figure 5, we show two experiments. In the first one, we have three queues, two with a fast CPU ($\mu = 8$), the two others with a slow CPU ($\mu = 1$). In the second plot, the speed difference is reduced.

We observe the following facts for the first plot:

- Except at high loads, `random` is a bit more than two times slower than the optimal routing.
- `JSQ` behaves better, but is still 50% worse than `optim`, and performs less efficiently that any strategy but `random`.

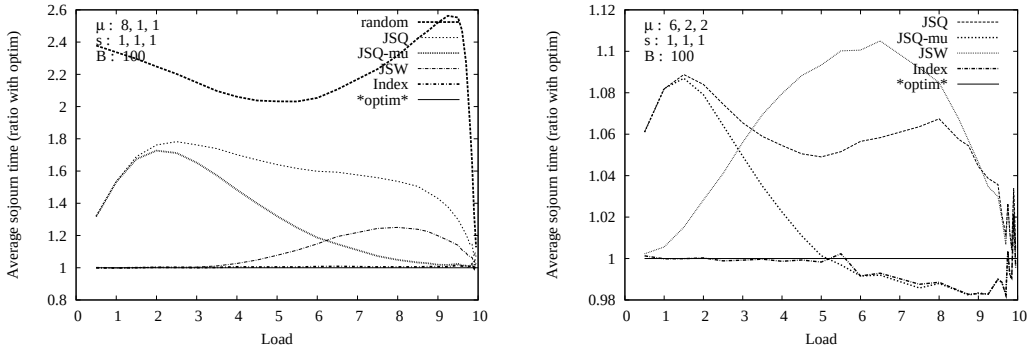

Figure 5: Ratio between the average sojourn time of several routing strategies and the optimal routing, on monoprocessor systems. On the left plot, we observe that, while even the best classical strategies stay within 20% from the optimal, our Index strategy is almost indistinguishable from the optimal. On the right plot, the difference between strategies is less visible, but our strategy is still highly better than any non-optimal others.

- `JSQ-mu` is not efficient at low load, but performs rather well at high load ($\pm$ 10% from `optim` when the load is higher than 80%).
- `JSQ-mu2` (equivalent to `JSW`) has a symmetric behavior: it performs pretty well at low load, but "gets beat" by `JSQ-mu` for loads higher than $\pm$ 60%. But this strategy is so far the best one, with a worst performance around 80% of load, with a distance from `optim` around 25%.
- `Index` is obviously better that any of these strategies, whatever the load. `Index` is never further that 2% from `optim`, while the best of "JSQ-like" strategies reaches 25% at some loads.

In the second plot of Figure 5, our comments can still be done, but the difference between strategies is greatly reduced, except for `random` which we didn't plot here.

We observe on the right plot of Figure 5 a phenomena which could seem to be astonishing: `Index` and `JSQ-mu` perform better than `optim`. It is due to two reasons. Firstly, LCR strategies have been simulated by perfect simulation tools, while optimal routing has been simulated through classical techniques. Secondly, optimal is obtained by a (slow) converging process. This plot requiring approximately 5 days-CPU of computation (at 3.4 GHz), we have chosen to accept this level of precision.

4.3 Multiprocessors systems

We show on Figure 6 two examples of multiprocessor systems. Similar comments can be done in this situation: `Index` behaves drasticely better than any other strategies (except `optim`), and is really close to the optimal one. On the left plot (the right one being too complex for computing the optimal strategy), `Index` is never worst than 2% from `optim`, while even the best JSQ-like strategy is for some load almost at 10% from `optim`.

4.4 Robustness

The main drawback of our index strategy is that indices are computed by taking into account the input rate, assuming it to be constant. But in real systems, the input rate can vary according to the time, which would require, firstly to obtain an estimation of this load, secondly to recompute our indices.

This leads us to the following question: how robust is index routing according to the input rate (or the system load) ? Does an index computed for a load of 50% perform efficiently if the load is 20% or 80% ?

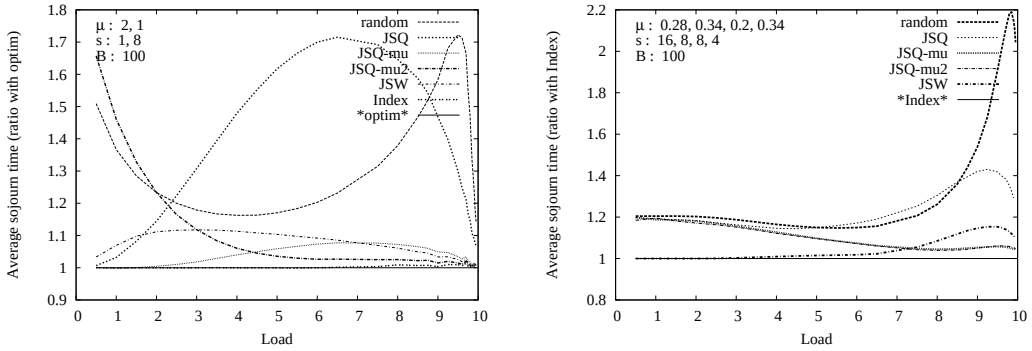

Figure 6: As is Figure 5, we show ratios with optimal sojourn time (left) or `Index` (right), due to the size of the system. It still shows that, even on multiprocessors platforms, `Index` behaves drasticely faster than other strategies.

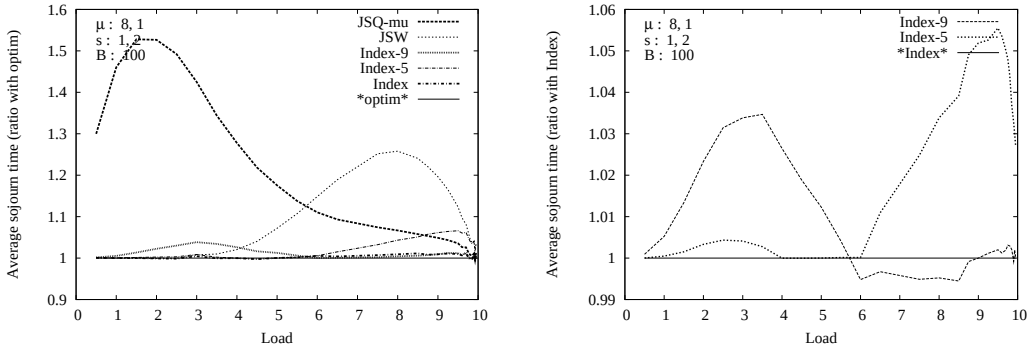

Figure 7: We show how behaves an index computed for a wrong input rate. We observe that even if we have a rough approximation of the load, `Index` obtains really good performances.

We show on Figure 7 the behavior of a system where three different indices are used. The first index (`Index`) is the classical one, which means that this index is computed according to the actual load. The second index (`Index-5`) is an index computed for a load of 50%, and used for every load between 0 and 100%. The first one (`Index-9`) is similar, for a load of 90%. On the right plot of Figure 7, we isolate `Index`, `Index-5` and `Index-9`.

It appears that the 50%-index (resp. 90%-index) is still better or equivalent than any other classical strategies for a load varying between 0 and 85% (resp. 40% and 100%). This shows that our index are really robust, and that they do not require to known the exact input rate. The following strategy would be very efficient:

- Compute two sets of indices $\mathcal{I}_1 = \{\mathtt{I}_1, \ldots, \mathtt{I}_N\}$ and $\mathcal{I}_2 = \{\mathtt{I}'_1, \ldots, \mathtt{I}'_N\}$, the first one corresponding to a system with a load of 50%, the second one to a load of 90%
- Choose two thresholds T_1 and T_2, for instance $T_1 = 70\%$ and $T_2 = 50\%$
- Estimate continuously the input rate λ.
- If the current set of index is $\mathcal{I}_1$ and that $\lambda > T_1$, use the set of index $\mathcal{I}_2$.
- If the current set of index is $\mathcal{I}_2$ and that $\lambda < T_2$, use the set of index $\mathcal{I}_1$.

This strategy can easily be generalized to a system with more than 2 index sets.
From the right plot of Figure 7, we observe two interesting facts:

- In this configuration, even if we use `Index-5` for any load, we are never worse than 4% from the normal index;
- `Index-9` performs more efficiently than `Index` between 60% and 90%, without taking into account the system load. This probably means that we can still improve the computation of our index. However, the gain is here really tiny ($\pm 0.05\%$).

5 Conclusions

In this paper, we have shown that index policies are very efficient in practice for task allocation in grids. They yield very good performances, almost indistinguishable from optimal and can be computed quickly even for very large systems. Our future work includes extending this work to batch arrivals (typical for grids) and implementing our proposal in actual scheduling tools for grids.

References

[1] EGEE: Grids for E-sciencE. Http://www.eu-egee.org.

[2] GridPP. Http://www.gridpp.ac.uk.

[3] Grid 5000, a large scale nation wide infrastructure for grid research. Https://www.grid5000.fr/.

[4] P. Whittle, *A celebration of applied probability*, j. gani ed. J. Appl. Probab. Spec., 1988, vol. 25A, ch. Restless bandits: activity allocation in a changing world, pp. 287–298.

[5] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of optimal queueing network control," *Math. Oper. Res.*, vol. 24, pp. 293–305, 1999.

[6] C. Coucourbetis, G. Kesidis, A. Ridder, J. Walrand, and R. Weber, "Admission control and routing in ATM networks using inferences from measured buffered occupancy," *IEEE Transactions in Communications*, vol. 43, pp. 1778–1784, 1995.

[7] J. Palmer and I. Mitrani, "Optimal and heuristic policies for dynamic server allocation," *Journal of Parallel and Distributed Computing*, vol. 65, no. 10, pp. 1204–1211, 2005, special issue: Design and Performance of Networks for Super-, Cluster-, and Grid-Computing (Part I).

[8] J. Niño-Mora, "Dynamic allocation indices for restless projects and queueing admission control: a polyhedral approach," *Mathematical Programming, Series A*, vol. 93, no. 3, pp. 361–413, 2002.

[9] R. R. Weber and G. Weiss, "On an index policy for restless bandits," *Journal of Applied Probability*, vol. 27, pp. 637–648, 1990.

[10] P. Ansell, K. D. Glazebrook, J. Nino-Mora, and M. O'Keeffe, "Whittle's index policy for a multi-class queueing system with convex holding costs," *Mathematical Methods of Operational Research*, vol. 57, pp. 21–39, 2003.

[11] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley, 1994.

[12] J.-M. Vincent, "Perfect simulation of queueing networks with blocking and rejection," in *Saint IEEE conference*, Trento, 2005, pp. 268–271.

[13] A. Hordijk and G. Koole, "On the optimality of the generalized shortest queue policy," *Probability in Engineering and Information Sciences*, vol. 4, pp. 477–487, 1990.

[14] J. C. Gittins, "Bandit processes and dynamic allocation indices. with discussion," *J. R. Stat. Soc, Series B, Stat. Methodol.*, vol. 41, pp. 148–177, 1979.

[15] J. G. Propp and D. B. Wilson, "Exact sampling with coupled Markov chains and applications to statistical mechanics," *Random Structures and Algorithms*, vol. 9, no. 1&2, pp. 223–252, 1996.

A Convexity

In this appendix, we show that $J^{n+1}(x) + J^{n+1}(x+2) - 2 * J^{n+1}(x+1) \geq 0$ as claimed in the proof of Theorem 2.1.

The result is true for $n = 0$. Let us now check the induction step for the infinite case

$$J^{n+1}(x) + J^{n+1}(x+2) - 2 * J^{n+1}(x+1) = cx + c(x+2) - 2c(x+1) \tag{11}$$

$$+ \frac{\alpha\lambda}{\Lambda} \min\{R + J^n(x), J^n(x+1)\} + \frac{\alpha\lambda}{\Lambda} \min\{R + J^n(x+2), J^n(x+3)\} \tag{12}$$

$$- 2\frac{\alpha\lambda}{\Lambda} \min\{R + J^n(x+1), J^n(x+2)\} \tag{13}$$

$$+ \frac{\min\{s,x\}\mu\alpha}{\Lambda} J^n(x-1) + \frac{\min\{s,x\}\mu\alpha}{\Lambda} J^n(x+1) - 2\frac{\min\{s,x\}\mu\alpha}{\Lambda} J^n(x) \tag{14}$$

$$+ \frac{\alpha(\Lambda - \lambda - \mu(\min\{s,x\}))}{\Lambda} J^n(x) + \frac{\alpha(\Lambda - \lambda - \mu(\min\{s,x\}))}{\Lambda} J^n(x+2) \tag{15}$$

$$- 2\frac{\alpha(\Lambda - \lambda - \mu(\min\{s,x\}))}{\Lambda} J^n(x+2). \tag{16}$$

First, the term (11) is obviously 0. As for the term (12) and (13), four cases need to be checked, depending on the values of the two minima. If the minimum is reached for the first two terms, then

$$R + J^n(x) + R + J^n(x+2) \geq 2(R + J^n(x+1)) \geq 2\min\{R + J^n(x+1), J^n(x+2)\}.$$

If the minimum is reached for the last two terms, then

$$J^n(x+1) + J^n(x+3) \geq 2J^n(x+2) \geq 2\min\{R + J^n(x+1), J^n(x+2)\}.$$

If the minimums is reached for the first and second terms respectively,

$$R + J^n(x) + J^n(x+3) \geq J^n(x+1) + R + J^n(x+2) \geq 2\min\{R + J^n(x+1), J^n(x+2)\}.$$

Finally, if the minimum is reached for the second and first terms respectively,

$$J^n(x+1) + R + J^n(x+2) = J^n(x+1) + R + J^n(x+2) \geq 2\min\{R + J^n(x+1), J^n(x+2)\}.$$

This finishes the case with terms (12)-(13)

As for the terms (14)-(15)-(16) , one needs to distinguish two cases. If $s \leq x$ then one gets

$$\alpha\frac{s}{\Lambda}(J^n(x-1) + J^n(x+1)) - 2\alpha\frac{s}{\Lambda}J^n(x)) \geq 0$$

by induction, and terms (15)-(16) are zero.

If $x \leq s - 2$ then for terms (14) to (16) together, one gets

$$\alpha\frac{x}{\Lambda}J^n(x-1) + \alpha\frac{x+2}{\Lambda}J^n(x+1)) + \alpha\frac{s-x}{\Lambda}J^n(x) + \alpha\frac{s-x-2}{\Lambda}J^n(x+2) \tag{17}$$

$$- 2\alpha\frac{x+1}{\Lambda}J^n(x) - 2\alpha\frac{s-x-1}{\Lambda}J^n(x+1). \tag{18}$$

This is positive by induction.

The remaining case is $x = s - 1$ (or $s = x + 1$). For terms (14) to (16) together, one gets,

$$\alpha\frac{x}{\Lambda}J^n(x-1) + \alpha\frac{x+1}{\Lambda}J^n(x+1)) + \alpha\frac{1}{\Lambda}J^n(x) - 2\alpha\frac{x+1}{\Lambda}J^n(x).$$

This is again positive by induction and non-decrease of J^n is x.

As for the finite capacity case, one additional equation must be checked to make sure the cost function is convex.

$$J^{n+1}(B-2) + J^{n+1}(B) = \tag{19}$$
$$cB + c(B-2) \tag{20}$$
$$+ \frac{\alpha\lambda}{\Lambda}\min\{R + J^n(B-2), J^n(B-1)\} + \frac{\alpha\lambda}{\Lambda}(R + J^n(B)) \tag{21}$$
$$+ \frac{\min\{s,x\}\mu\alpha}{\Lambda}J^n(B-3) + \frac{\min\{s,x\}\mu\alpha}{\Lambda}J^n(B-1) \tag{22}$$
$$+ \frac{\alpha(\Lambda - \lambda - \mu(\min\{s,x\}))}{\Lambda}J^n(B-2) + \frac{\alpha(\Lambda - \lambda - \mu(\min\{s,x\}))}{\Lambda}J^n(B). \tag{23}$$

The only new inequality to check is that

$$\min(R + J^n(B-2), J^n(B-1)) + (R + J^n(B)) \geq 2\min(R + J^n(B-1), J^n(B)).$$

This is true because on one hand

$$R + J^n(B-2) + R + J^n(B) \geq 2(R + J^n(B-1)) \geq 2\min(R + J^n(B-1), J^n(B)),$$

and on the other hand

$$J^n(N-1) + R + J^n(B) = R + J^n(B-1) + J^n(B)) \geq 2\min(R + J^n(B-1), J^n(B)).$$

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399