ORIGINAL PAPER

Learning-Agent-Based Simulation for Queue Network Systems

Daniel Barry Fuller, Edilson Fernandes de Arruda[a] and Virgílio José Martins Ferreira Filho[a]

[a]Industrial Engineering Program, Alberto Luiz Coimbra Institute - Graduate School and Research in Engineering, Federal University of Rio de Janeiro

ARTICLE HISTORY
Compiled September 9, 2019

ABSTRACT
Established simulation methods generally require from the modeler a broad and detailed knowledge of the system under study. This paper proposes the application of Reinforcement Learning in an Agent-Based Simulation model to enable agents to define the necessary interaction rules. The model is applied to Queue Network Systems, which are a proxy for broader applications, in order to be validated. Simulation tests compare results obtained from learning agents and results obtained from known good rules. The comparison shows that the learning model is able to learn efficient policies on the go, providing an interesting framework for simulation.

KEYWORDS
Simulation; Agent; Machine Learning; Queue Network

1. Introduction

Established simulation methods generally require from the modeler a broad and detailed knowledge of the system under study. Such a comprehension should encompass the system elements and their interactions and translating that knowledge into a model may require a considerable amount of time. Agent-based Simulation (ABS) represents systems through agents and their rule-based interaction with the environment (Macal & North, 2010) in an attempt to separate the systemic knowledge into smaller parts. Agent-based representation aims to be more useful than other simulation paradigms by making the real world representation more faithful through elements which are more or less in a one-to-one correspondence with the real systems (Macal, 2016). This helps making technical explanations easier and more convincing (van Dam, Adhitya, Srinivasan, & Lukszo, 2009) and curtails the need for complete knowledge of the system's behaviour as a whole.

In developing ABS models, it is necessary to guarantee that the behavior and interaction rules, which form the agents' policies, are sufficient and valid for a wide range

CONTACT Daniel Barry Fuller. Email: fullerdb@ufrj.br

of expected states in all cases. This paper proposes the application of Reinforcement Learning to let the agents define adequate rules "by themselves" during simulation runs, thus reducing the model development time and producing useful simulation results quickly.

In order to cover a broad application domain, the concept of dynamically defined policies in ABS shall be presented through its application in queue network systems (QNSs). This is posited as an adequate choice for three reasons:

(1) Queue network models may be used to represent many real-world systems, such as manufacturing (Azaron, Katagiri, Kato, & Sakawa, 2006; Bitran & Morabito, 1996), supply chain (Kerbache & MacGregor Smith, 2004), computing systems (Osman, Awan, & Woodward, 2009; Xu, Li, Hu, & Li, 2014), communications (Choi & Silvester, 1999; Pourmohammad, Fekih, & Perkins, 2015) and healthcare (C & Appa Iyer, 2013),
(2) Many queue network systems require control or management akin to ABS rules to attain good performance, therefore research is devoted to defining effective policies, as exemplified by Azaron et al. (2006), who use a multi-objective model, Ghazel and Saïdane (2015), who define a control method that regulates arrival, and Pourmohammad et al. (2015), who propose a control method and compare it to other approaches.
(3) Analogously to general simulation, much of the concern with QNSs involves the performance evaluation of the systems under different conditions and policies (e.g. Bitran and Morabito (1996); Horváth (2015); Morabito, de Souza, and Vazquez (2014); Osman et al. (2009)).

In this paper, the concept of ABS with agents capable of determining their own rules is called *Learning-agent*-based Simulation (LABS). There are other ways to train ABS agents, but these are often domain based (García-Magariño & Palacios-Navarro, 2016). The original idea presented here is the application of R-Learning, which is a Reinforcement Learning (RL) technique (Sutton & Barto, 1998), as a means for ABS agents to identify effective, near-optimal policies *as the simulation runs* regardless of the system under study.

Reinforcement learning is already applied to many domains, but, usually, the goal is to define policies for a fixed system. For instance: Chen, Dong, and Zhou (2014) apply RL to a negotiation environment, Dogan and Güner (2015) apply it to a competitive ordering and pricing problem, Jiang and Sheng (2009) and Yu and Wong (2015) study the application of RL in supply chain inventory control and supplier selection, and Sun and Zhao (2012) and Kara and Dogan (2018) look into suppy chain ordering policies. What these papers have in common is that they concern themselves with evaluating the learning methods themselves. This paper aims to present a simulation method in which the necessary policies are defined concurrently with the simulation of the system, which outputs system performance measures: the policies are necessary, but the main goal of the simulations is to evaluate the systems' structure and generate understanding of their behavior.

The test instances were developed based on those presented by Kelly and Laws (1993), who apply reflected Brownian motion to estimate the performance of a set of QNSs under different routing and sequencing policies. They assume heavy traffic conditions and consider the case when a route is determined upon arrival and cannot change later. In the proposed simulation method, neither of these assumptions is necessary, therefore broadening the possible application domain. Tests are made with system expected occupation as low as 50% and additional test instances, called service models, allow routes to be dynamically defined in order to demonstrate how the proposed technique remains valid without these assumptions.

The performance of the QNSs was evaluated under different learning configurations and predefined rules based on Kelly and Laws (1993) to demonstrate that the learning method can consistently find good policies for diverse scenarios. A modification to the standard R-Learning is also proposed that improves the results obtained for the test instances.

System performance was measured mainly by customers' time in the system (a.k.a. system time or sojourn time) and the tests were run through a simulation tool detailed in Section 5. The main goal is to validate the general method in order to motivate its application in diverse domains, where the reference rules presented in that paper do not apply. For one such example, we refer to Fuller, Ferreira Filho, and de Arruda (2018).

Sections 2 and 3 present the relevant definition of QNS and a LABS model that can represent these systems. Section 4 explains alternative definitions of *Actions*, *States* and *Rewards* necessary for the learning method. Section 5 describes the simulation tool that was implemented to test the proposition, while test instances are detailed in Section 6. Simulation experiments and results are discussed in Section 7.

2. Queue network systems

A description of queueing networks and an associated control problem is provided by Laws (1992). In that paper, a network is composed of single-server stations. Customers of multiple types arrive according to type-specific renewal processes and are allocated a route. Routes are sequences of stations a customer must pass through. If a route includes the same station more than once, each pass is considered a separate stage. Each station is assigned one queue for each stage of each route that passes through it. When a station is assigned to a customer, the station's server takes some (possibly random) time to complete the service.

The application of dynamic control to queue networks and analysis of several system topologies under heavy traffic conditions is presented by Kelly and Laws (1993) by means of a systematic sequence of emblematic examples. Two types of decision are proposed: routing and sequencing. *Routing* is the decision about the sequence of queues and servers a customer should visit in the system; once chosen, the route cannot be changed. *Sequencing* refers to the choice by a server of which customer to serve when there are multiple concurrent requests (which, when defined by the network's structure, is all the time under heavy traffic conditions).

Kelly and Laws (1993) assume the system to be under heavy traffic conditions, which enables the Brownian representations demonstrated there. In this paper, this premise is not necessary, which makes the application possibilities broader. Servers have the option to postpone a service if another, with higher priority potential, is probabilistically expected. The priority is potential because a customer's priority is under the dynamic policy, i.e. it may change over time or depend on the state of the system.

The value of allowing a route to be dynamically decided instead of assigned to a customer upon arrival will be explored. This allows single queues for multiple servers. In order to allow routing decisions to be made as the customer moves through the network, it is necessary to redefine routes as sequences of *services* sought by the customers. In this new case, servers have a list of offered services added to their definition and the server for the next service that a customer requires is defined only when a customer is about to leave the service queue. In Section 6.1, it will be shown that, under these new conditions, routing decisions can be converted to sequencing decisions.

3. Simulation model

Agent-based simulation models are defined by *agents*, *environment* and their *interaction rules* (Macal & North, 2015). Agents and parts of the environment are collectively dubbed *elements*. This section will define an agent-based queue network model in which agents have to make choices akin to the Multi-armed bandit problem (Auer, Cesa-Bianchi, & Fischer, 2002). Section 4 shows the method proposed to make these choices.

3.1. Simulation elements

In queue network systems (QNS), as defined in Section 2, the elements are customers, servers (also called service stations), queues, routes, and customer generators (arrival processes).

Furthermore, queues and routes form the model's environment, because they interact with agents, but not with each other. Customers, servers and customer generators (CGs) are the agents; interacting with the environment and each other. A further distinction can be made between *Decision Agents* (also known as *Proactive Agents*) and *Passive Agents* (also known as *Reactive Agents*). Servers and CGs actually have to make decisions regarding what actions they should perform. Customers, on the other hand, are passive and may only follow the decisions made by the other agents when notified.

Figure 1 shows a basic class diagram for the simulation elements and a description of the agents follows.

CGs are defined by a list of routes to allocate to the customers it creates and the customer arrival processes. During the simulation, they generate customers, decide which route to allocate to them and introduce them into the system.

Customers are defined by an individual label and a route to follow. During the simulation, they join the appropriate queue for the first server (or service) on their route when they enter the system. After each service, they move to the next queue according to their route or leave the system if their route is completed. They make no decisions, but interact with other elements as agents.

Servers are defined by a service time probability distribution. During the simulation, they are either serving a customer or idle. All queues are always candidates for the decision to select which one is to be served, even if there is no current demand.

3.2. Decision rules

CGs and Servers have to make the routing and sequencing decisions respectively. This means choosing an action from a list of possibilities. For CGs, this is a list of the routes to assign and assigning each route is an action. For Servers, the list contains the queues it may serve and serving a customer from these queues is an action. This selection is

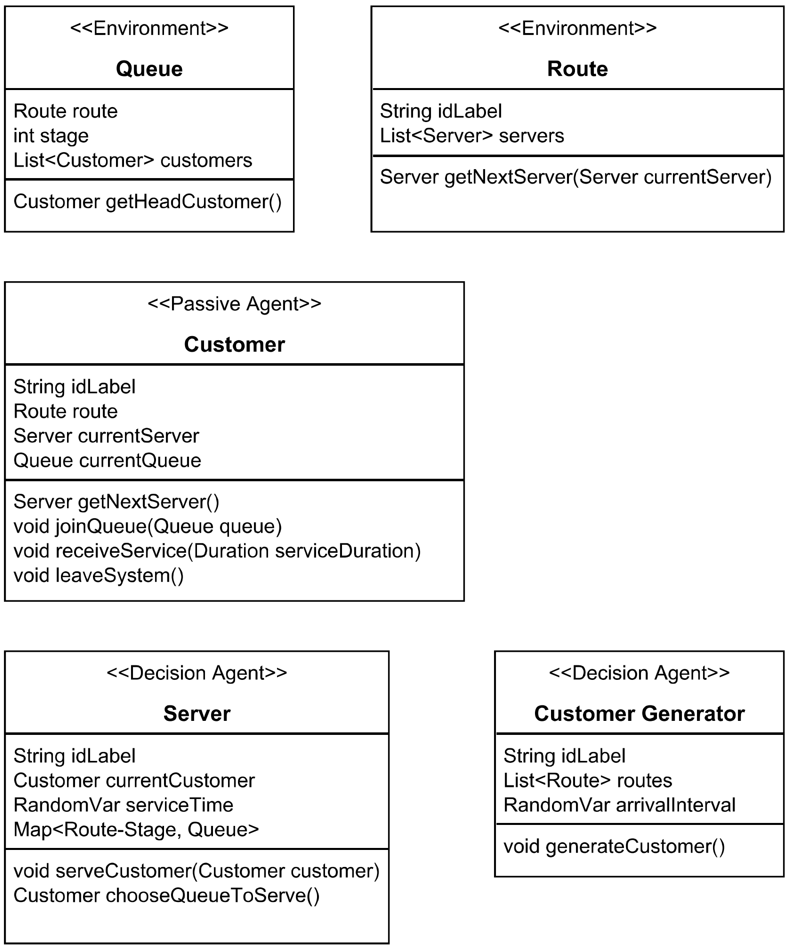

Figure 1. Basic class diagram for the simulation elements.

akin to the Multi-armed bandit problem (Auer et al., 2002) and is implemented here in two modes: fixed policy and learning.

In fixed policy mode, the decisions are always made according to a predefined policy. For testing and comparison, three possible fixed policies defined by Kelly and Laws (1993) are used:

Random: the action (see Section 4.1) is chosen randomly.

Shortest/Longest: for routing, the route with the shortest first queue is chosen, for sequencing, the server's longest queue is chosen to be served.

By turn: routes to assign and queues to serve are chosen according to a looped list (option A, option B, option C, option A, option B, and so on).

These fixed policies, like many others mentioned by García-Magariño and Palacios-Navarro (2016), reflect a prior knowledge that they are good for each specific system instance. It should be remarked that defining policies of future systems based on previous experience (as the case with the approach of García-Magariño and Palacios-Navarro (2016) was) is not always straightforward and the application of the Reinforcement Learning technique may be helpful, because servers and CGs can find good policies by themselves. This removes the need to previously define and test policies, therefore reducing the model development time and possibly improving its accuracy, or, at least, mitigating its reliance on experts' opinions.

The fixed policies are also useful to validate the learning models through the *Comparison to other models* technique described by Sargent (2013).

The learning technique that was applied was *R-Learning*, as defined by Sutton and Barto (1998). R-Learning is a Reinforcement Learning technique similar to Q-Learning (Watkins, 1989), but it allows for continuous, undiscounted activities. This is the case with the analyzed queue network systems, because the expected values of the states do not vary over time (undiscounted) and there are no episodes or terminal states (continuous), since the considered queue systems operate uninterruptedly.

R-Learning is a reinforcement learning technique, which is a family of unsupervised machine learning methods that do not rely on fitting behavior to recorded data, but relies on a mechanism which rewards "good behavior".

The standard off-policy routine that was used is presented in Algorithm 3.1. Each learning agent (CGs and servers) runs the algorithm independently.

In Algorithm 3.1, s and s' are the pre- and post-action states, a^* is a chosen activity, ρ is the current average reward and $r(s)$ is a reward or penalty for achieving state s. $R(s, a)$ is the expected reward for activity a taken at state s. α and β are, respectively, the learning and average reward updating rates.

The set of activities A could be, more generally, dependent on the states and, as such, defined for each state s in the state space S as $A(s)$. However, this is not necessary for the defined agents, because all activities are available at all states.

In Algorithm 3.1, activity a^* is chosen through Function chooseAction. This function, where the symbol $\neg$ negates the following boolean value, implements an original, quasi-ϵ-greedy strategy with decaying ϵ.

A standard ϵ-greedy strategy, such as described by Sutton and Barto (1998), may choose an activity randomly with probability $\epsilon \in [0, 1]$ or greedily with probability $1 - \epsilon$. In this decaying ϵ strategy, the value of ϵ is initialized as 1 and decreases with each decision according to a decay parameter ϵ_{decay}. This makes the first choices mostly random, but, as $\epsilon \to 0$, the policy becomes fixed and purely greedy.

The greedy selection does have one problem: if an agent remains in the same state

Algorithm 3.1: Off-policy, temporal-difference R-Learning algorithm adapted from Sutton and Barto (1998).

```
   // All Data is specific for each agent
   Data: $\rho$                                          // expected global reward
   Data: $R(s,a) \forall s \in S, a \in A$  // expected reward for action $a$ at state $s$
   Data: $S$                                        // set of all possible states
   Data: $A$                                       // set of all available actions
   Data: $r(s)$                                 // reward paid to the agent $\forall s \in S$
   Data: $\alpha$                                     // update step size for $R(s,a)$
   Data: $\beta$                                          // update step size for $\rho$
   Data: $s^{\dagger}$                                              // last observed state
   Data: $\Delta^{\dagger}$                          // last calculated reward update value
1  Initialize $\rho$ and $R(s,a) \forall s \in S, a \in A$ arbitrarily  // Usually, with 0
2  Loop
3      $s^* \leftarrow$ current state                 // observe current state
4      $n \leftarrow$ a random number $\in [0,1]$         // generate a random number
5      $a^* \leftarrow$ chooseAction $(s^*, n, s^{\dagger}, \Delta^{\dagger})$  // call the chooseAction function
6      Take action $a^*$      // apply the effects of action $a^*$, including
        updating the agent's internal clock's time
7      $s' \leftarrow$ new state  // observe state immediately after all effects of
        $a^*$ have been applied
       // Update learning knowledge
8      $\Delta \leftarrow r(s') - \rho + max_a R(s',a) - R(s^*,a^*)$         // update value
9      $R(s^*,a^*) \leftarrow R(s^*,a^*) + \alpha\Delta$                 // reward update
10     if $R(s^*,a^*) = max_a R(s^*,a)$ then update $\rho$
11         $\rho \leftarrow \rho + \beta\Delta$
12     $\Delta^{\dagger} \leftarrow \Delta$
13     $s^{\dagger} \leftarrow s^*$
14     wait            // wait for a signal to repeat the loop
```

```
Function chooseAction($s, n, s^\dagger, \Delta^\dagger$)
  Input: $s$                                  // a state of the agent
  Input: $n \in [0, 1]$                       // a threshold value
  Input: $s^\dagger$                          // previously observed state
  Input: $\Delta^\dagger$                     // a previous value of a reward update
  // All Data is specific for each agent
  Data: $\epsilon$                            // current value of $\epsilon$
  Data: $\epsilon_{\text{decay}}$             // value of the update for $\epsilon$
  Data: $A$                                   // set of available actions
  Data: $R(s, a) \forall a \in A$             // expected reward for action $a$ at state $s$
  // This function chooses an action $a$ from $A$
  Result: $a$
1 if $\epsilon < n \land \neg(s = s^\dagger \land \Delta^\dagger < 0)$ then is greedy and is not stuck in a bad state
2   $a \leftarrow \text{argmax}_{a \in A} R(s, a)$
3 else is random
4   $k \leftarrow k \in [1, |A|]$             // $k$ is chosen randomly
5   $a \leftarrow a_k$                        // $a_k$ is the $k$th element of $A$
6 $\epsilon \leftarrow \epsilon \times \epsilon_{\text{decay}}$
7 return a
```

after a poor action, it will repeat the poor action until the rewards are updated to reflect the unwanted consequences. This is not a large issue if the learning algorithm is allowed to run for a large number of iterations and is only evaluated at the end of the run. However, the learning algorithm is run during the simulation and its bad decisions may affect the simulation output. This leads to the proposition of a modified strategy, called *quasi*-ϵ-greedy. In this strategy, shown in the chooseAction function, the agent chooses the next activity randomly if it is still in the same state it was at the time of the last activity choice ($s = s^\dagger$) and the perceived reward decreased the value of the state-activity pair ($\Delta^\dagger < 0$). Without this, the agent could greedily repeat the reward-reducing action until either the reward became lower than that of another action or the state changed due to actions of other agents. Either way, this modification helps to avoid a series of poor choices that would have a detrimental impact in the system's simulated performance which would find no correspondence in the real system.

It should be noted that the learning algorithm is part of the simulation and is in effect throughout the simulation run, although it is expected (but not enforced) to be mostly stable (i.e. greedy) after the warm-up period.

4. Learning method

As can be seen in Algorithm 3.1, the learning method requires that actions, states and rewards be defined. This section will present alternatives to these definitions. Multiple alternatives are presented in order to be evaluated in Section 7.

4.1. Actions

The available actions to CGs correspond to the routes they can assign to customers. Every time a customer is generated to enter the system, its generator will choose a route for it. Once a customer is in the system, its route cannot change. In the alternate version, where a route is not set upon arrival, routing decisions as defined are unnecessary and CGs become passive (see Sections 2 and 6).

For servers, each action corresponds to a queue attached to it. Every time a service ends, the server chooses a new action to perform. Note that a server may always choose any queue to serve, even if it is empty. In this case, the server idles for a predefined duration of simulation time and then chooses a new action based on the (possibly new) current state.

4.2. States

There are many ways to define states for learning agents and each manner may bring different results. Multiple ways were proposed and tested for CGs and Servers. In the remainder of this paper, roman numerals are used to identify each case.

4.2.1. Customer Generator States

Two ways to define CG states are proposed:

I The first proposal was to define a state by the length of the first queue in each route. For instance, if a CG has two possible routes, A and B, and there are, say, three customer in the first queue of route A and two customers in the first queue of route B, the state is A-3/B-2 (but see Section 4.2.3). The order of the routes (A/B or B/A) is irrelevant.

II An alternative definition of states was to simply sort the routes by their first queues' occupation regardless of their actual length. In this way, if the first queue of route A is longer than that queue of route B, the state is simply BA. If route B has the longer queue, the state is AB. Ascending or descending sorting orders are equivalent.

4.2.2. Server States

An additional way to define server states are proposed, with options I and II analogous to those for CGs in Section 4.2.1.

I The queues the Server may serve have their length used to define the states in the same way as the states for CGs.

II As for CGs, it is also possible to define the states by sorting the queues the Server may serve.

III This is similar to alternative I, but the set of queues the Server may serve is replaced by the set of queues the customers may join immediately after leaving the server.

4.2.3. State dimensionality

The number of states in option II is the number of permutations of the relevant queues, but, in options I and III, the states are defined by the combination of the queue lengths, which may lead to a very large number of different states, as every additional customer on each queue defines a new state.

Dealing with large numbers os states is an inherent problem of algorithms derived from Dynamic Programming, such as R-Learning and other Reinforcement Learning methods (Barto & Mahadevan, 2003). In this specific application of Reinforcement Learning, which is also based on the Monte-Carlo method, there is an additional issue in that some states may not happen often enough during the simulation for good policy estimates to be calculated. To circumvent this, a way to reduce the state space is proposed.

The premise assumed is that when queues are short, the difference between the states is more relevant to the decisions; when queues are long, the most useful actions may not differ. In other words, the difference between having, say, 19 or 20 customers in a queue is arguably less relevant than the difference between 1 and 2 customers. A logarithmic scale, defined by Equation (1), was employed to reduce state domain size by replacing the raw number of customers in the queue ($length(queue)$) with an occupation level for the queue. $\lfloor \cdot \rfloor$ is the floor function.

$$level(queue) := \begin{cases} 0 & \text{if } length(queue) = 0 \\ \lfloor log_{10}(length(queue)) \times 5 + 4.9 \rfloor & \text{otherwise} \end{cases} \quad (1)$$

Figure 2 shows how the application of Equation (1) aggregates many queue lengths in a small number of states. Only the lower states are presented as an example; there is no upper limit for the queue lengths. It should also be noted that more queue lengths are considered the same state as the queues get longer. The constants were experimented with, and those presented are used throughout this paper.

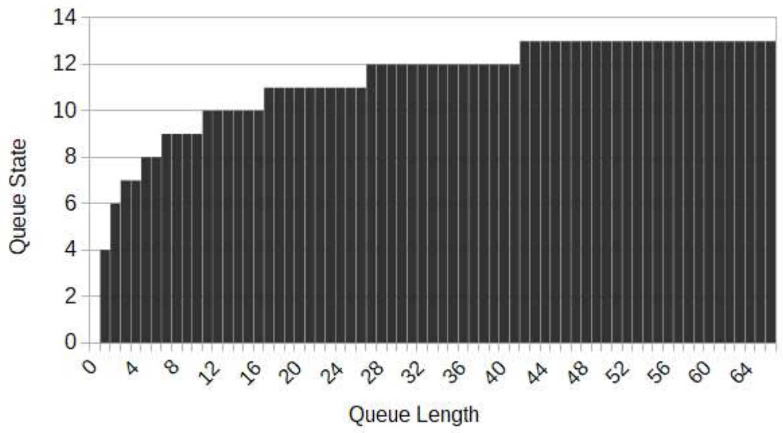

Figure 2. Lower queue states defined from queue lengths with Equation (1).

When the states are defined through this reduced space, they are indicated **Ib** and **IIIb**.

4.3. Rewards and penalties

After choosing an action, the learning agent must be informed about the quality of the ensuing results. This is done by assigning rewards or penalties r to the action results.

There is no real difference between rewards and penalties. Algorithm 3.1 works just as well with $r > 0$ as with $r < 0$, as long as reward r_2 is always better than r_1 if $r_2 > r_1 \forall r_1, r_2 \in \mathbb{R}$.

To define the rewards, it is first necessary to define some system metrics:

L is the length of a queue a customer joins; the queue is either the first on a route or the next on the customer's route after a service. If the customer leaves the system and, therefore, joins no queue, $L := 0$.

L_c is the current length of the first queue in a route chosen by a CG.

W_c is the current average wait time in the first queue in a route chosen by a CG; this time is updated during the simulation every time a customer leaves the queue.

C_w is the time a customer has waited in the current queue when a server selects it.

I is the set of the first queues of the routes the CG can assign. L_i and W_i are, respectively, the current length and current average wait time for the queues $i \in I$.

T is the sum of the expected service times of the unvisited servers on a customer's route.

NC is a penalty for serving no customer. $NC \in [0, \infty)$. The value of NC should be such that unduly leaving customers waiting is discouraged. And to prevent situations where servers unduly make the customers wait, $-NC$ should also be the worst penalty a served customer may report.

Four reward modes were tested (roman numeral labels in parenthesis indicate how the modes are referenced in the results):

Queue length (I) provides a reward based on Equations 2, for CGs, and 3, for servers. In the case of servers, it is necessary to define a penalty if no customer is served. In order to make this the worst possible result and prevent situations where servers unduly make the customers wait, $-NC$ is also the worst reward a served customer may report.

$$r = -L \tag{2}$$

$$r = \begin{cases} -NC & \text{if no customer was served} \\ \max(-NC, -L) & \text{otherwise} \end{cases} \tag{3}$$

Wait time (II) provides a reward based on Equations (4), for CGs, and 5, for servers. If no customer is selected by a server, a default penalty is applied to discourage idling.

$$r = -L_c \times W_c \tag{4}$$

$$r = \begin{cases} -NC & \text{if no customer was served} \\ \max(-NC, -C_w) & \text{otherwise} \end{cases} \tag{5}$$

Relative wait time (III) provides a reward based on Equations (6), for CGs, and 7, for servers.

$$r = \frac{\sum_{i \in I} L_i \times W_i}{|I|} - L_c \times W_c \tag{6}$$

$$r = \begin{cases} -NC & \text{if no customer was served} \\ \max(-NC, T - C_w) & \text{otherwise} \end{cases} \tag{7}$$

Expected time (IV) provides a reward based on Equations (8), for CGs, and 9, for servers.

$$r = T \tag{8}$$

$$r = \begin{cases} 0 & \text{if no customer was served} \\ T & \text{otherwise} \end{cases} \tag{9}$$

5. Simulation tool

The model was implemented as an event-based, agent-oriented simulator. This tool keeps a list of agents sorted by the time when the action each agent is currently performing will end. When an action is chosen to be performed, it generates an activity which will be performed by the agent during a time window and may affect the states of agents and the environment. At every simulation step, the first agent of the list decides its next action. If the activity ends before the end of the simulation, the agent is returned to the list at the appropriate position, given by the end time of the latest activity; it is removed from the list otherwise. Algorithm 5.1 is the basic instruction set of the simulator.

How a decision agent chooses its next action depends on the rules currently in effect (see Section 3.2). The effect of an activity depends on whether the agent that performs it is a CG or a Server.

CGs' activity is to assign a route to the next Customer and its effect is to place the Customer at the respective queue for the first server in the route. Servers' choice is from which queue to take a Customer to serve. When there are no Customers at the chosen queue, the server's activity is to wait for a prescribed time and choose again. This has no other effect. In contrast, when there are customers waiting, the activity is to define a service time and the effect is that the served Customer is moved to the appropriate queue for the next server in its route (or leaves the system, if the route

Algorithm 5.1: Simulator's main loop algorithm.

```
Data: $A$                                           // List of agents
Data: $h$                                    // The simulation horizon
1 while $A \neq \emptyset$ do
2     sort($A$)     // sort agents according to their internal clocks' times
3     $agent \leftarrow a_0$       // $a_0$ is the first (earliest) element of $A$
4     $agent$ chooses and executes next activity
5     $time \leftarrow$ currentTime($agent$)   // The current time according to the agent's internal clock
6     if $time > h$ then remove $agent$ from $A$
7         $A \setminus \{agent\}$
```

is over). The Customer can only join the next queue (or leave the system) after the service time has elapsed.

Customers, as Passive Agents, make no decisions and are only moved from queue to queue by CGs and Servers until their route ends and they leave the system. Although Customers produce no effect in the system in the same sense that Decision Agents do, Customers do report their experience to their generator and the servers on their route when they are leaving the system. This report includes waiting times and total time in the system. This data is essential for producing the simulation output, but may also be used in the definition of states and rewards (see Wait time (II) in Section 4.3, for instance).

It should be remarked that there are situations where the rule in effect (see Section 3.2) is overridden for some agents: when a CG can only assign one route to the Customers it generates or when a Server can only select Customers from one queue. Other agents in the same scenario apply the selected rule regardless. See Figures 3 and 4 for examples where some agents apply the rules and others have no decision to make.

The simulator's main output is the number of customers that went through the system and the average time they spent in it. These values are the main performance evaluators. Other variables, such as quantity of customers assigned to each route and number of customers that went through each server and each server's utilization level are calculated and checked to detect spurious cases.

6. Test instances

Test instances were taken from Kelly and Laws (1993). They start simple, but add complexity by steps. The figures in this section represent the test cases. They show the Customer Generators (hexagons), Queues (rectangles) and Servers (circles). Routes are indicated by different line types and named next to the CG that may assign them.

The first system (Figure 3) only has a routing decision to be made by Customer Generator C (CG C). There are two routes: one through Server S1 and another through Server S2. CG A and CG B can only assign one route each; therefore they make no decisions. Each server can only serve one queue and they make no decisions either. In the second system (Figure 4), S1 must make sequencing decisions while the CGs and S2 make no decisions.

The system in Figure 5 combines routing decision in CG B, which can assign two routes, and sequencing in servers S1 and S2, which can serve two queues each. CG A makes no decisions. Figure 6 shows a system with multiple routing and sequencing decisions. Both CGs can assign two routes each and all four Servers must choose from two queues each to serve.

The system in Figure 7 expands that on Figure 6. In this case, CG B can assign three routes and there are six servers, all of which can make sequencing decisions concerning two queues each.

6.1. Dynamic routes

As mentioned in Section 2, allowing the customers' routes to be defined as they move through the system will be evaluated. In order to do this, the concept of *Service* needs to be introduced. When a Server selects a Customer, it will provide a Service for a duration of time. Each server has a set of Services it may provide.

Customer routes shall be redefined in terms of service, i.e. customers will have a list of services they wish to receive instead of a list of servers to visit. Instead of queueing for servers, customers queue for services. It should be noted that multiple servers may now select customers in this arrangement, whereas before a customer was linked to a single server by its route.

The conversion from the test case in Figure 6 to a new model, which is service-oriented, is presented in Figure 8. This new system has two types of customers: A, which seeks services α and β, and B, which seeks services γ and δ. It also has four servers: S1, S2, S3 and S4, which can provide, respectively, services α and γ, β and γ, α and δ, and β and δ. When queues are defined for each service, as shown in the bottom part of Figure 8, each CG has only one possible route to assign. This means no routing decision is necessary, but the servers still make sequencing decisions. The service-oriented model obtained by this example is shown in Figure 12.

The new models for the test cases are defined in Figures 9, 10, 11, 12 and 13. Figure 9's version of the system of Figure 3 has no decision making elements and is only included for completeness; it was not simulated.

7. Simulation experiments

Simulation scenarios were defined by QNS instance, expected system utilization level, and decision rules (including alternative State and Reward definitions). The main output is the customer time in the system, but the customer count was checked to

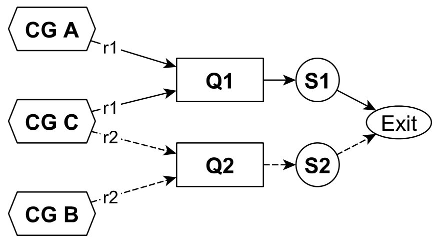

Figure 3. Parallel queues (routing).

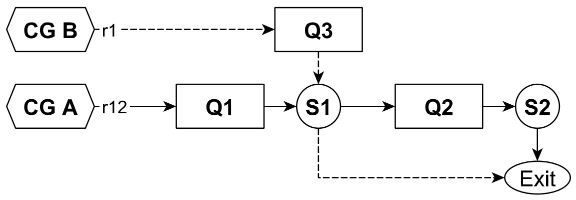

Figure 4. Network with sequencing.

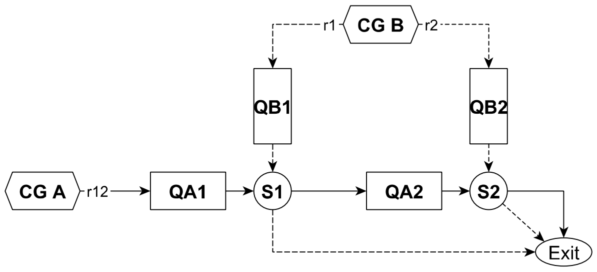

Figure 5. Routing and sequencing on two stations.

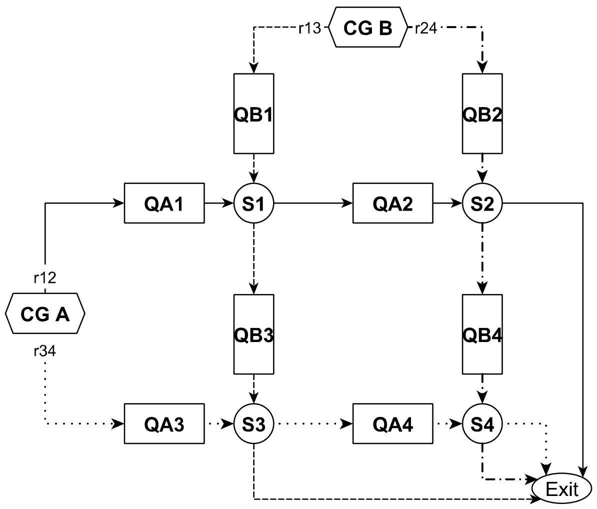

Figure 6. Four-station network.

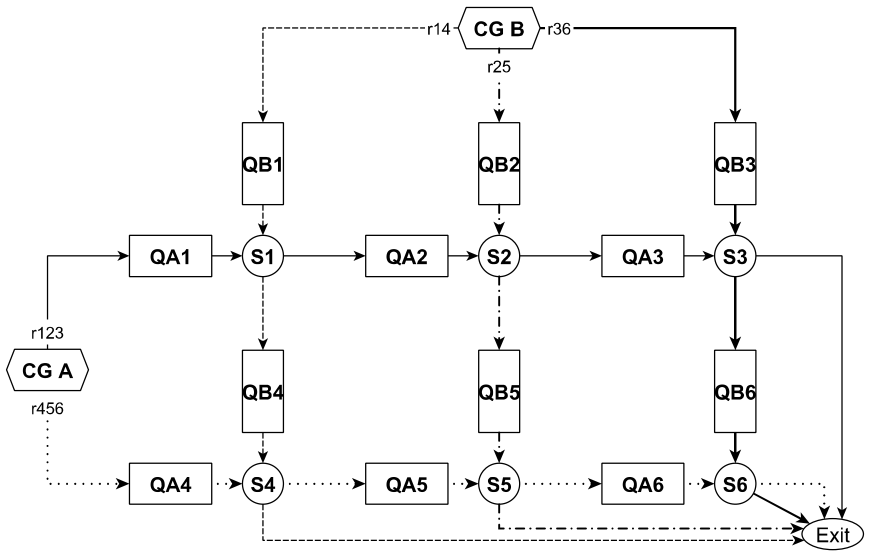

Figure 7. 2 × 3 network.

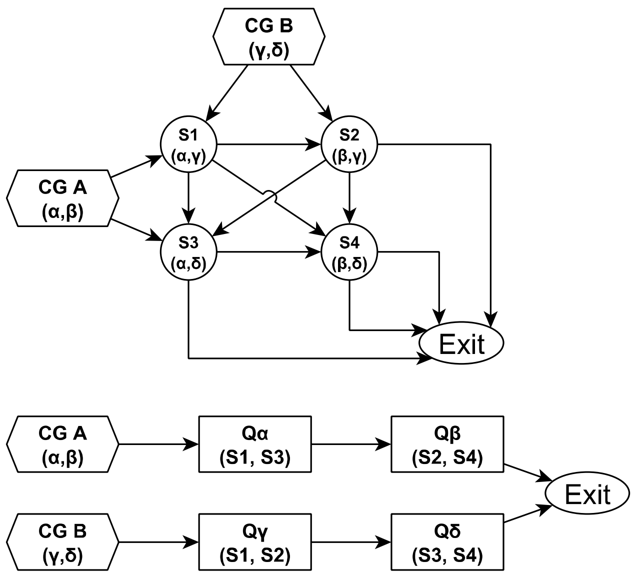

Figure 8. Transition to a service-based model of the system from Figure 6. The final model is in Figure 12.

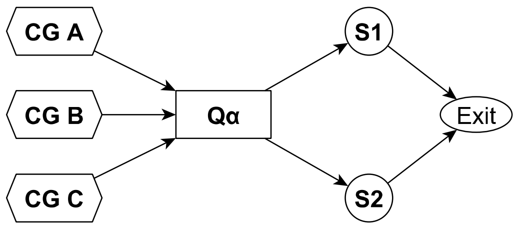

Figure 9. Service model corresponding to Figure 3.

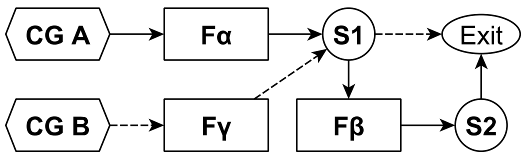

Figure 10. Service model corresponding to Figure 4.

identify spurious results. The QNS instances were the five presented in Section 6 and their service-based versions in Section 6.1.

7.1. Input parameters

Utilization level (ratio of busy time by available time) for the system is defined as the utilization level of the busiest server. The amount of expected customers to be generated is calculated from this with Equation (10) and the total is distributed evenly among the available routes of all CGs (except for the system of Figure 3, where CG C generates half the expected customers). All tested arrival processes were independent and exponentially distributed.

$$\text{Expected customers} = \frac{\text{Expected utilization level} \times \text{Simulation horizon}}{\text{Expected service time}} \tag{10}$$

The decision rules are those from Section 3.2. For the learning rules, a few sets of parameters where tested, but eventually they where fixed as $\alpha = \beta = 1$ with the initial value of $\epsilon = 1$ and $\epsilon_{\text{decay}} = 0.5$ for all presented results. The same was done to define the NC penalties as 100, 5,000 and 1,000 for Queue length (I), Wait time (II) and

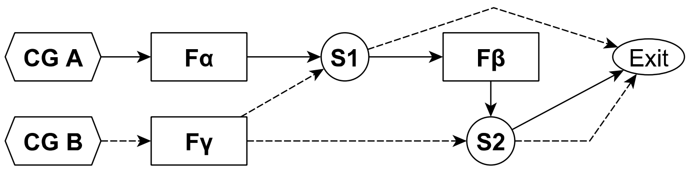

Figure 11. Service model corresponding to Figure 5.

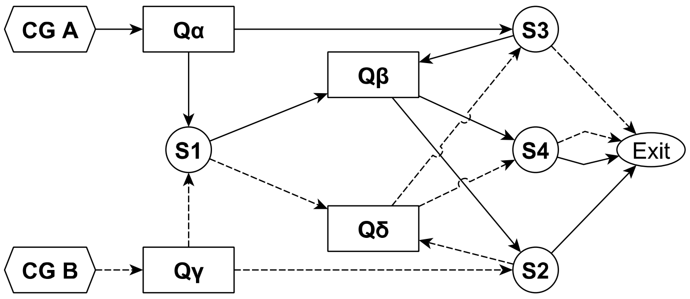

Figure 12. Service model corresponding to Figure 6.

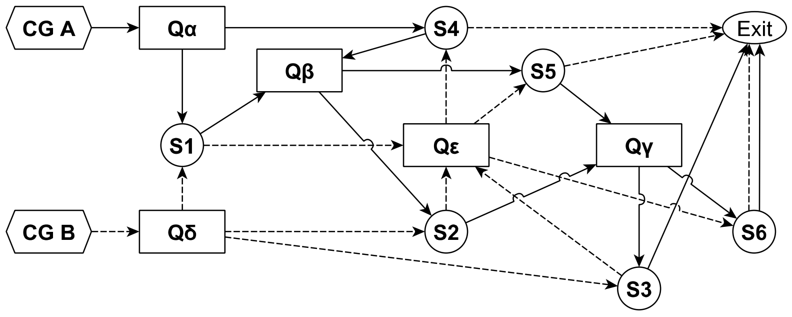

Figure 13. Service model corresponding to Figure 7.

Relative wait time (III), respectively. Service times for all servers were exponentially distributed with mean 100 seconds.

Simulation horizon was 604,800 seconds (7 days) and 100 Monte-Carlo replications were run for each scenario. The scenarios were defined by a combination of the following:

- A case from Figures 3 to 7 or 10 to 13.
- Expected utilization level: 50%, 85% or 95%.
- A predefined rule or learning with modes.

  One combination of CG state (I, Ib or II) and reward (I, II, III, IV) modes for Figures 3 and 5 to 7.

  One combination of Server state (I, Ib, II, III or IIIb) and reward (I, II, III or IV) modes for Figures 4 to 7 and 10 to 13
- Normal or modified learning.

The total valid combinations amount to 5,154.

7.2. Randomness

Random numbers were generated with the MRG31k3p method (L'Ecuyer & Touzin, 2000) from the SSJ library (L'Ecuyer, 2012). Each replication was assigned its own seed.

7.3. Simulation results

García-Magariño and Palacios-Navarro (2016) state that agent-based simulation frequently relies on domain-specific behavior rules, which requires a broad and detailed knowledge of the simulated system. Frameworks help to implement these rules faster, but fail to define the rules themselves. The simulation results show that the learning model can achieve results which approximate those obtained based on the predefined rules specified by Kelly and Laws (1993) without assuming that such knowledge is available. In most cases, results are statistically equivalent.

It should be noted that the policies defined are secondary to the simulated system results. The main goal of the simulations is to evaluate the structure of QNSs and the policies are here only means to attain this goal, as opposed to the cases of Ghazel and Saïdane (2015) and Pourmohammad et al. (2015), that aimed at defining optimal policies.

Some highlights and insights regarding the simulation results follow.

7.3.1. Expected results under heavy traffic conditions

Table 1 shows the results for the QNS from Figure 3, which only has routing decisions. It is noteworthy that, according to Kelly and Laws (1993), the waiting time under the Shortest rule in heavy traffic conditions is half of that under the Random rule (under certain conditions). The simulated scenarios comply with most of the necessary conditions for this result to hold, but do not enforce heavy traffic conditions. Nevertheless, results interestingly seem to converge to that ratio as the utilization level rises, which loosely emulates heavy traffic conditions.

7.3.2. Effect of the R-Learning modification

The application of machine learning and simulation simultaneously aims to provide adequate policies when optimal policies are unknown or too costly to determine. It may generate results such as those seen in Figure 14, which depicts confidence intervals for the Time in the System for some scenarios obtained with the standard learning method. As the fixed policies that were tested were known to be good policies (Kelly & Laws, 1993), it is to be expected that not all of the learning scenarios match their performance. Many of them, however, are worse than the Random policy, which is unacceptable.

Results for the same scenarios obtained with the modified R-Learning method are depicted in Figure 15. Most results are much closer to the fixed policies and all beat the Random policy by a significant margin.

The average reduction in the simulated Customer Time in the System obtained through the modified method considering all learning scenarios was 45% in relation to the simulation results without the modification.

7.3.3. Best agent configurations

Not all configurations of CGs and Servers provide good results. In order to evaluate the quality of the results, a relative result is defined as the ratio between Customer Average Time in the System for learned policies and for the best fixed policy under the same conditions. For instance: in Table 2, the best fixed policy result for 85% utilization is 521s; for configuration Server[I/I], the result 562s gives a relative result of 1.079. Relative results greater than 1 indicate that the fixed policy's Time in the

Table 1. Results for the simulation of the system from Figure 3 with standard R-Learning.

Rule	**Ut.**	**Cust. Gen.**		**System time**		**Cust. Count**
	Level	**State**	**Reward**	**Avg.**	**St.Dev.**	**Average**
Random	50%			198	7	6,069
By turn	50%			183	7	6,078
Shortest	50%			171	4	6,071
Learning	50%	I	I	173	5	6,063
Learning	50%	I	II	173	5	6,074
Learning	50%	I	III	175	6	6,065
Learning	50%	I	IV	331	33	6,079
Learning	50%	Ib	I	172	5	6,069
Learning	50%	Ib	II	172	5	6,070
Learning	50%	Ib	III	174	5	6,069
Learning	50%	Ib	IV	329	32	6,079
Learning	50%	II	I	173	5	6,060
Learning	50%	II	II	172	5	6,070
Learning	50%	II	III	171	5	6,072
Learning	50%	II	IV	332	34	6,077
Random	85%			663	88	10,295
By turn	85%			584	68	10,305
Shortest	85%			419	46	10,307
Learning	85%	I	I	447	59	10,285
Learning	85%	I	II	451	59	10,289
Learning	85%	I	III	468	57	10,307
Learning	85%	I	IV	45,172	4,964	8,649
Learning	85%	Ib	I	437	43	10,278
Learning	85%	Ib	II	439	48	10,292
Learning	85%	Ib	III	439	42	10,293
Learning	85%	Ib	IV	45,570	3,279	8,638
Learning	85%	II	I	427	39	10,306
Learning	85%	II	II	416	36	10,274
Learning	85%	II	III	425	34	10,299
Learning	85%	II	IV	45,497	3,244	8,640
Random	95%			1,858	523	11,477
By turn	95%			1,638	492	11,484
Shortest	95%			1,047	309	11,495
Learning	95%	I	I	1,443	703	11,479
Learning	95%	I	II	1,478	694	11,487
Learning	95%	I	III	1,567	756	11,480
Learning	95%	I	IV	60,474	7,858	8,966
Learning	95%	Ib	I	1,094	343	11,480
Learning	95%	Ib	II	1,091	358	11,490
Learning	95%	Ib	III	1,084	328	11,480
Learning	95%	Ib	IV	61,463	3,602	8,918
Learning	95%	II	I	1,098	397	11,481
Learning	95%	II	II	1,032	345	11,466
Learning	95%	II	III	1,055	349	11,488
Learning	95%	II	IV	61,429	3,626	8,918

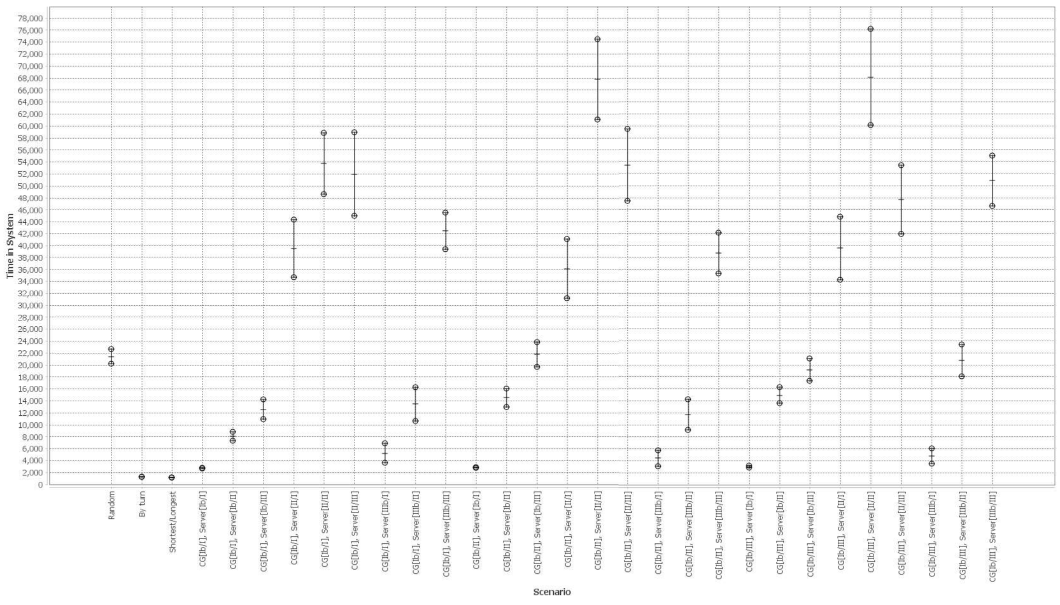

Figure 14. Confidence interval for some scenarios (Figure 7, 85% occupation, standard R-Learning).

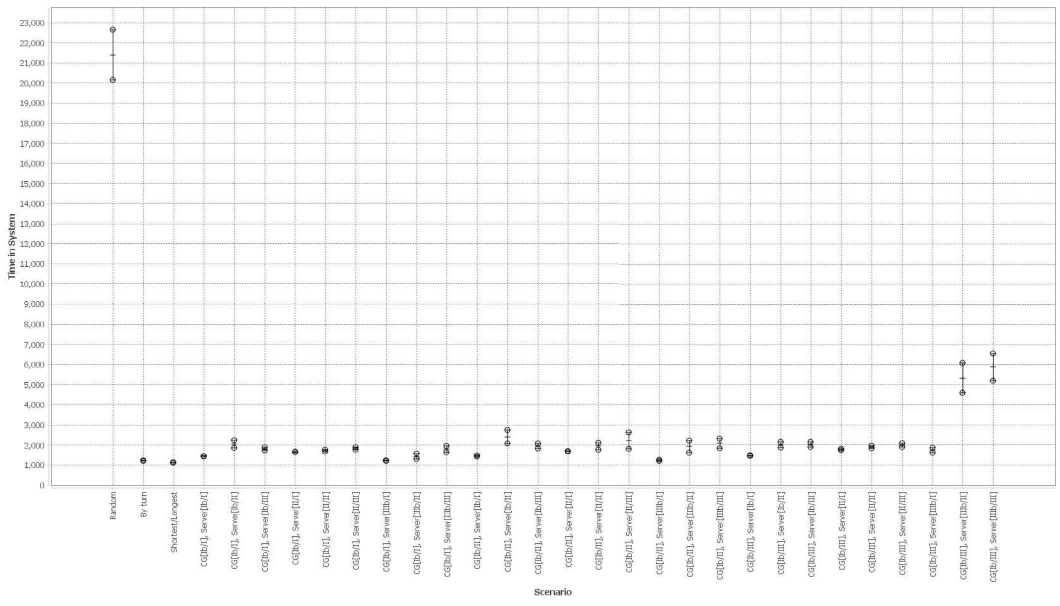

Figure 15. Confidence interval for some scenarios (Figure 7, 85% occupation, modified R-Learning).

System is lower than the learning policy's.

All learning results in the table are significantly different from the best fixed policy with 95% statistical confidence, unless indicated by † in the relative result.

Table 2.: Results for the simulation of the system from Figure 13 with modified R-Learning.

Rule	Ut.	Server		System time		Cust. Count	Relative
	Level	State	Reward	Avg.	St.Dev.	Average	Result
Random	50%			6,026	3,456	7,452	
By turn	50%			281	4	7,569	← reference
Longest	50%			298	4	7,566	
Learning	50%	I	I	310	5	7,556	1.103
Learning	50%	I	II	307	4	7,561	1.093
Learning	50%	I	III	309	5	7,568	1.100
Learning	50%	I	IV	326	5	7,567	1.160
Learning	50%	Ib	I	314	5	7,548	1.117
Learning	50%	Ib	II	309	5	7,554	1.100
Learning	50%	Ib	III	312	5	7,542	1.110
Learning	50%	Ib	IV	330	5	7,562	1.174
Learning	50%	II	I	334	8	7,553	1.189
Learning	50%	II	II	331	9	7,567	1.178
Learning	50%	II	III	337	9	7,566	1.199
Learning	50%	II	IV	366	9	7,566	1.302
Learning	50%	III	I	284	4	7,561	1.011
Learning	50%	III	II	284	4	7,577	1.011
Learning	50%	III	III	284	4	7,555	1.011
Learning	50%	III	IV	285	4	7,556	1.014
Learning	50%	IIIb	I	284	3	7,564	1.011
Learning	50%	IIIb	II	283	3	7,563	1.007
Learning	50%	IIIb	III	285	4	7,570	1.014
Learning	50%	IIIb	IV	285	3	7,574	1.014
Random	85%			5,804	2,140	12,677	
By turn	85%			521	24	12,844	← reference
Longest	85%			552	27	12,853	
Learning	85%	I	I	562	27	12,859	1.079
Learning	85%	I	II	572	35	12,851	1.098
Learning	85%	I	III	579	45	12,845	1.111
Learning	85%	I	IV	571	36	12,833	1.096
Learning	85%	Ib	I	581	30	12,835	1.115
Learning	85%	Ib	II	590	38	12,847	1.132
Learning	85%	Ib	III	600	42	12,842	1.152
Learning	85%	Ib	IV	599	32	12,857	1.150
Learning	85%	II	I	546	24	12,827	1.048
Learning	85%	II	II	553	28	12,848	1.061
Learning	85%	II	III	556	31	12,848	1.067
Learning	85%	II	IV	557	32	12,848	1.069
Learning	85%	III	I	542	35	12,854	1.040
Learning	85%	III	II	548	31	12,857	1.052
Learning	85%	III	III	561	39	12,847	1.077
Learning	85%	III	IV	538	28	12,844	1.033
Learning	85%	IIIb	I	540	28	12,836	1.036
Learning	85%	IIIb	II	546	33	12,857	1.048
Learning	85%	IIIb	III	567	46	12,843	1.088
Learning	85%	IIIb	IV	549	34	12,846	1.054
Random	95%			8,006	2,454	14,132	
By turn	95%			1,138	257	14,358	
Longest	95%			1,065	173	14,330	← reference
Learning	95%	I	I	1,267	343	14,342	1.190
Learning	95%	I	II	2,160	1,321	14,304	2.028
Learning	95%	I	III	2,090	818	14,311	1.962
Learning	95%	I	IV	1,676	599	14,333	1.574
Learning	95%	Ib	I	1,132	178	14,320	†1.063
Learning	95%	Ib	II	1,728	841	14,330	1.623
Learning	95%	Ib	III	1,935	791	14,304	1.817

Table 2.: Continued...

Rule	Ut. Level	Server State	Server Reward	System time Avg.	System time St.Dev.	Cust. Count Average	Relative Result
Learning	95%	Ib	IV	1,361	355	14,343	1.278
Learning	95%	II	I	1,029	185	14,339	†0.966
Learning	95%	II	II	1,241	588	14,346	1.165
Learning	95%	II	III	1,283	322	14,342	1.205
Learning	95%	II	IV	1,228	260	14,347	1.153
Learning	95%	III	I	1,146	201	14,351	1.076
Learning	95%	III	II	4,133	3,375	14,188	3.881
Learning	95%	III	III	4,259	2,613	14,184	3.999
Learning	95%	III	IV	1,211	222	14,319	1.137
Learning	95%	IIIb	I	1,107	186	14,335	†1.039
Learning	95%	IIIb	II	3,462	2,625	14,220	3.251
Learning	95%	IIIb	III	3,143	1,516	14,244	2.951
Learning	95%	IIIb	IV	1,348	373	14,334	1.266

Figures 16 to 21 aggregate relative results from all the presented QNS scenarios to which the shown configurations apply.

7.3.3.1. Customer Generator Configurations. Figure 16 shows the average relative result for all CG configurations. Configurations with reward mode IV (Expected time) clearly generate worse results. Even if the results are filtered to remove configurations which presented congestion resulting in a loss of more than 10% of arrivals in relation to the best fixed policy, Figure 17 shows that reward mode IV is still the worst case. Even when only the best relative result is observed (Figure 18), reward mode IV is still clearly worse than the others, which produce similar results.

7.3.3.2. Server Configurations. There is no clear pattern of the best Server configurations (Figure 19). It is valid to point out that the relative results of the service versions of the models (Figure 20) are better than those from the original versions (Figure 21) and of the CG configurations. Taking the relative results of the system from Figure 4, which only has Servers as learning agents, the results are similar to those from Figure 20. This seems to indicate that the learning method was more successful in defining good policies for Servers than for CGs, which points an advantage of defining the service oriented versions of the models whenever possible.

7.3.4. Model comparison

Models with fixed policies and learning models are essentially different models for the same system. By following the *Comparison to other models* technique described by Sargent (2013), it is, therefore, possible to validate the learning models by comparing their results with those obtained through fixed policies. Fixed-policy models (except Random, of course) are considered valid because they are based on methods known to be effective.

Results from Table 2 show that most of those produced by learning models closely match those produced by the fixed policies, indicating that the learning models are valid.

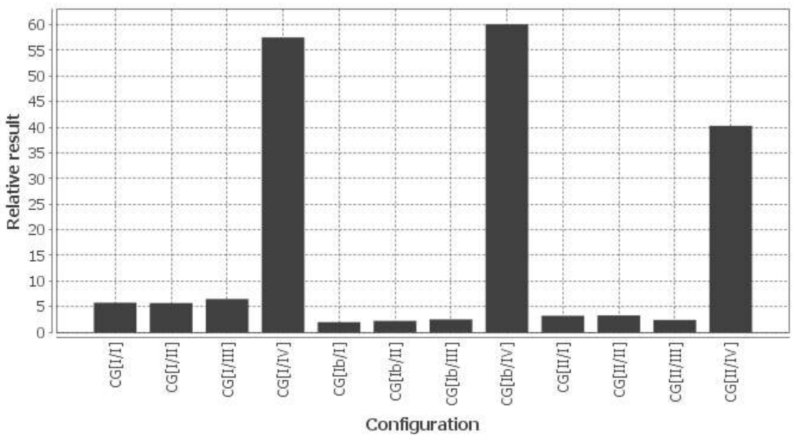

Figure 16. Average relative result for CG configurations.

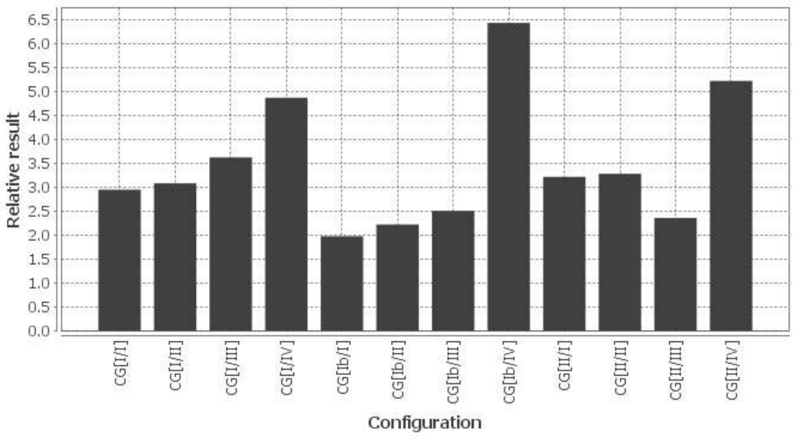

Figure 17. Average relative result for CG configurations disregarding results with customer count less than 90% of reference fixed policy.

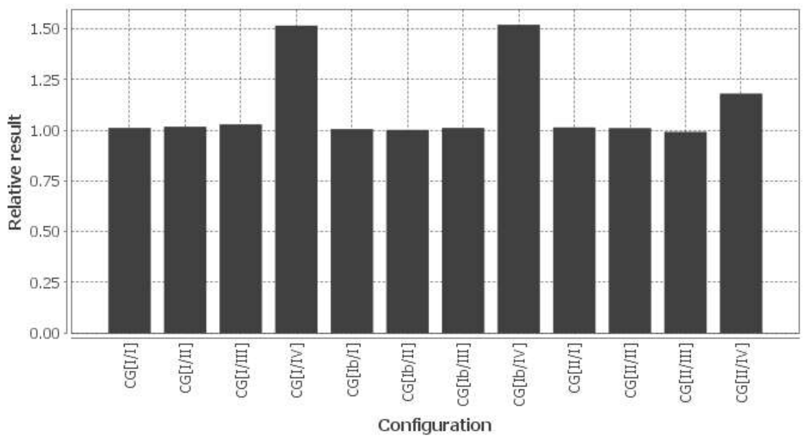

Figure 18. Best relative result for CG configurations disregarding results with customer count less than 90% of reference fixed policy.

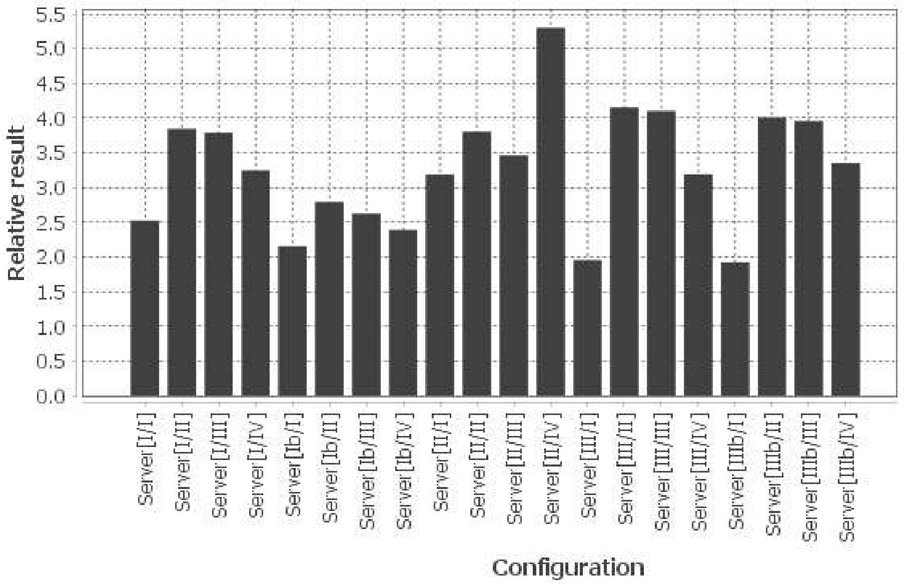

Figure 19. Average relative result for Server configurations.

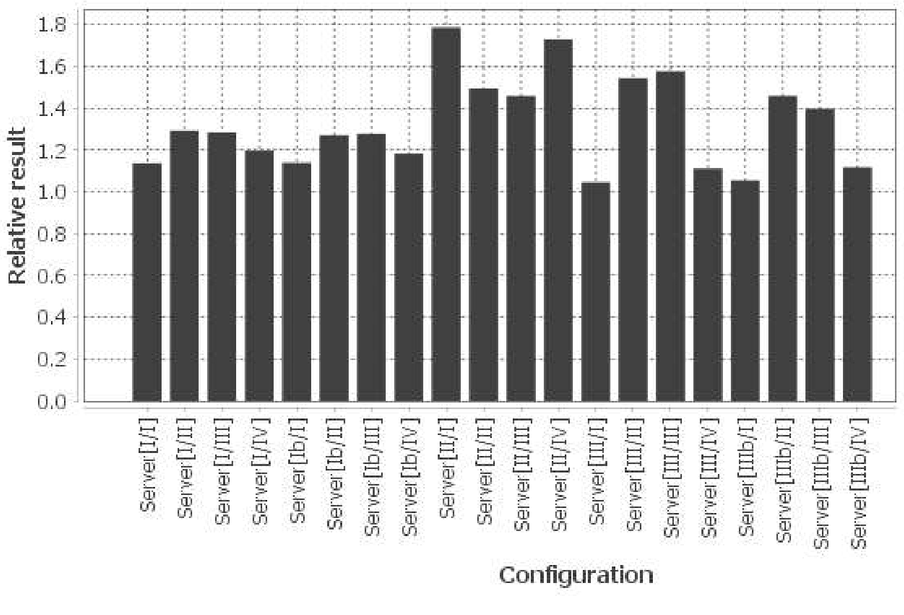

Figure 20. Average relative result for Server configurations in service-based models.

8. Concluding remarks

The presented simulation method, LABS, requires that its agents apply policies to select routes for customers and sequence multiple service requests. In order to achieve that, concurrent machine learning is applied during the simulation. Examples suggest that the performance of the simulated system closely follows that of known good policies for the proposed application. Even though the method cannot be seen as a replacement for analytical tools which find the optimal policy, it is an efficient substitute when optimal policies are unknown or too expensive (if at all possible) to obtain.

The successful application of LABS to the study of queue network systems indicates that it should also be useful in other domains, as many systems can be modeled as QNSs. This is particularly useful in applications where the rules are unknown or very hard to determine and calibrate. One such example is presented by Fuller et al. (2018).

Funding

The authors would like to thank the Carlos Chagas Filho Foundation for Research Support of the State of Rio de Janeiro, FAPERJ, for supporting the research by means of Grant E-26/202.789/2015. This work was partially supported by the National Council for Scientific and Technological Development – CNPq, under Grants 303543/2015-9 and 312725/2013-2.

References

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time Analysis of the Multi-armed Bandit Problem. *Machine Learning*, *47*(2/3), 235–256. Retrieved from `http://link.springer.com/10.1023/A:1013689704352`

Azaron, A., Katagiri, H., Kato, K., & Sakawa, M. (2006, oct). Modelling complex assemblies as a queueing network for lead time control. *European Journal of Operational Research*, *174*(1), 150–168. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0377221705001347`

Barto, A. G., & Mahadevan, S. (2003). Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems: Theory and Applications*, *13*(12), 343–379. Retrieved from `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.9749{&}rep=rep1{&}type=pdf`

Bitran, G. R., & Morabito, R. (1996). *Open queueing networks: Optimization and performance evaluation models for discrete manufacturing systems* (Vol. 5) (No. 2). Retrieved from `http://www.scopus.com/inward/record.url?eid=2-s2.0-0000309631{&}partnerID=tZOtx3y1`

C, L., & Appa Iyer, S. (2013, mar). Application of queueing theory in health care: A literature review. *Operations Research for Health Care*, *2*(1-2), 25–39. Retrieved from `http://www.sciencedirect.com/science/article/pii/S2211692313000039`

Chen, L., Dong, H., & Zhou, Y. (2014). A reinforcement learning optimized negotiation method based on mediator agent. *Expert Systems with Applications*, *41*(16), 7630–7640. Retrieved from `http://dx.doi.org/10.1016/j.eswa.2014.06.003`

Choi, J., & Silvester, J. A. (1999, feb). Simulation of controlled queuing systems and its application to optimal resource management in multiservice cellular networks. *Journal of the Brazilian Computer Society*, *5*(3), 00–00. Retrieved from `http://www.scielo.br/scielo.php?script=sci{_}arttext{&}pid=S0104-65001999000100005{&}lng=en{&}nrm=iso{&}tlng=en`

Dogan, I., & Güner, A. R. (2015). A reinforcement learning approach to competitive ordering and pricing problem. *Expert Systems*, *32*(1), 39–48.

Fuller, D. B., Ferreira Filho, V. J. M., & de Arruda, E. F. (2018). Oil industry value chain simulation with learning agents. *Computers & Chemical Engineering*, *111*(4), 199–209.

García-Magariño, I., & Palacios-Navarro, G. (2016). ATABS: A technique for automatically training agent-based simulators. *Simulation Modelling Practice and Theory*, *66*, 174–192. Retrieved from `http://linkinghub.elsevier.com/retrieve/pii/S1569190X15301647`

Ghazel, C., & Saïdane, L. (2015). Satisfying QoS Requirements in NGN Networks Using a Dynamic Adaptive Queuing Delay Control Method. *Procedia Computer Science*, *56*, 225–232. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1877050915016841`

Horváth, G. (2015, oct). Efficient analysis of the MMAP[K]/PH[K]/1 priority queue. *European Journal of Operational Research*, *246*(1), 128–139. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0377221715001976`

Jiang, C., & Sheng, Z. (2009). Case-based reinforcement learning for dynamic inventory control in a multi-agent supply-chain system. *Expert Systems with Applications*, *36*(3 PART 2), 6520–6526. Retrieved from `http://dx.doi.org/10.1016/j.eswa.2008.07.036`

Kara, A., & Dogan, I. (2018). Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. *Expert Systems with Applications*, *91*, 150–158. Retrieved from `http://dx.doi.org/10.1016/j.eswa.2017.08.046`

Kelly, F. P., & Laws, C. N. (1993). Dynamic routing in open queueing networks: Brownian models, cut constraints and resource pooling. *Queueing Systems*, *13*(1-3), 47–86.

Kerbache, L., & MacGregor Smith, J. (2004, oct). Queueing networks and the topological design of supply chain systems. *International Journal of Production Economics*, *91*(3), 251–272. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0925527303002883`

Laws, C. N. (1992). Resource Pooling in Queueing Networks With Dynamic Routing. *Advances in Applied Probability*, *24*(3), 699–726.

L'Ecuyer, P. (2012). Random Number Generation. In *Handbook of computational statistics* (pp. 35–71). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from `http://link.springer.com/10.1007/978-3-642-21551-3{_}3`

L'Ecuyer, P., & Touzin, R. (2000). Fast combined multiple recursive generators with multipliers of the form $a = \pm 2^q \pm 2^r$. In J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick (Eds.), *Proceedings of the 2000 winter simulation conference* (pp. 683–689). San Diego, USA. Retrieved from `http://informs-sim.org/wsc00papers/090.PDF`

Macal, C. M. (2016, may). Everything you need to know about agent-based modelling and simulation. *Journal of Simulation*, *10*(2), 144–156. Retrieved from `http://link.springer.com/10.1057/jos.2016.7`

Macal, C. M., & North, M. J. (2010, sep). Tutorial on agent-based modelling and simulation. *Journal of Simulation*, *4*(3), 151–162. Retrieved from `http://www.palgrave-journals.com/doifinder/10.1057/jos.2010.3`

Macal, C. M., & North, M. J. (2015, dec). Introductory tutorial: Agent-based modeling and simulation. In *Proceedings - winter simulation conference* (pp. 6–20). Huntington Beach, CA, USA.

Morabito, R., de Souza, M. C., & Vazquez, M. (2014, feb). Approximate decomposition methods for the analysis of multicommodity flow routing in generalized queuing networks. *European Journal of Operational Research*, *232*(3), 618–629. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0377221713006796`

Osman, R., Awan, I., & Woodward, M. E. (2009, mar). Application of Queueing Network Models in the Performance Evaluation of Database Designs. *Electronic Notes in Theoretical Computer Science*, *232*, 101–124. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1571066109000589`

Pourmohammad, S., Fekih, A., & Perkins, D. (2015, apr). Stable Queue Management in communication networks. *Control Engineering Practice*, *37*, 67–79. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0967066115000039`

Sargent, R. G. (2013). Verification and validation of simulation models. *Journal of Simulation*, *720*(1), 12–24. Retrieved from `https://pdfs.semanticscholar.org/9917/5990a45f646a8c5e2a47137beb0837bb5a50.pdf`

Sun, R., & Zhao, G. (2012). Analyses about efficiency of reinforcement learning to supply chain ordering management. *IEEE International Conference on Industrial Informatics (INDIN)*, 124–127.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* Cambridge, Massachusetts, USA: Bradford. Retrieved from `http://webdocs.cs.ualberta.ca/{~}sutton/book/ebook/the-book.html`

van Dam, K. H., Adhitya, A., Srinivasan, R., & Lukszo, Z. (2009, oct). Critical evaluation of paradigms for modelling integrated supply chains. *Computers & Chemical Engineering*, *33*(10), 1711–1726. Retrieved from `http://dx.doi.org/10.1016/j.compchemeng.2009.01.023`

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards* (Unpublished doctoral dissertation). King's College.

Xu, Y., Li, K., Hu, J., & Li, K. (2014, jun). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences*, *270*, 255–287. Retrieved from `http://www.sciencedirect.com/science/article/pii/S002002551400228X`

Yu, C., & Wong, T. N. (2015). An agent-based negotiation model for supplier selection of multiple products with synergy effect. *Expert Systems with Applications*, *42*(1), 223–237. Retrieved from `http://dx.doi.org/10.1016/j.eswa.2014.07.057`

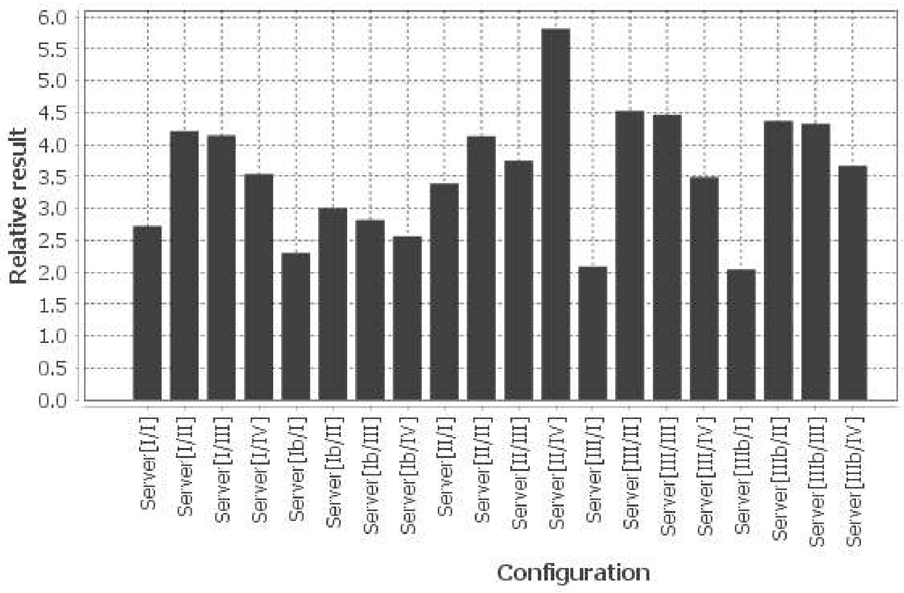

Figure 21. Average relative result for Server configurations.