Dynamic call blending in the presence of time-varying arrivals

Andras Balint
E-mail: abalint@few.vu.nl

Supervisor: Sandjai Bhulai
E-mail: sbhulai@few.vu.nl

Vrije Universiteit Amsterdam, Faculty of Sciences
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

August 15, 2005

Abstract

Call blending (dynamically mixing inbound and outbound traffic) is an efficient method in call centers for obtaining high productivity while satisfying a constraint on service levels. We investigate, using computer simulation, how call blending should be implemented efficiently in the presence of time-varying arrivals. For homogeneous arrival processes, the optimal policy is of threshold type, where the threshold represents the maximum number of agents working on outgoing calls. The optimal threshold can be obtained by using a formula for the expected average waiting time. We apply this idea for the case of time-varying arrivals and give easily implementable methods, resulting in a considerable improvement compared to the traditional policy, i.e., to using a fixed threshold. We also show that even better results can be obtained by using stochastic approximation.

1 Introduction

Call centers provide an important link between companies and their customers. They have become the preferred way of communication – where customer contact has to be arranged, it is very likely that a call center is used. The economic role of telephone call centers is significant and growing. But in spite of their importance, most call centers cannot achieve simultaneously high service levels and efficiency (i.e., high productivity). One can find interesting facts about this topic in [3].

The aim of this paper is to find ideas worth pursuing in reality – it is focused much more on the applicability of the techniques than on their theoretical aspects. Thus, the results here are generally not supported by strict proofs – they are obtained by computer simulation. Therefore, although being more or less informal, they could be very useful: firstly, because the best methods can be implemented, and secondly, because the results can show what is worthy of further investigation.

First of all, we give a short description of the problem. Imagine the following situation: there is a certain number of employees (called 'agents' from now on) working in a call center. Their primary task is to handle the incoming calls. Since the call center wants to deliver some quality of service in terms of delay, there is a constraint on the average waiting time of these calls. But there are also outgoing calls to handle – we assume that they are available in an infinite quantity, because there is always something to work on. The problem is to decide which job to assign to an idle agent such that service level constraints are satisfied for the incoming calls and the throughput is maximized for the outgoing calls simultaneously.

In this paper we examine methods to cope with all these requirements. It is strongly based on the results achieved by Bhulai and Koole, who have already given a solution for the case when the parameter of the Poisson process of the incoming calls is constant (see [1] for details). Unfortunately, the assumption of the process being stationary is unrealistic: from observing call arrival data of call centers it can be seen that there are busy and more or less idle periods during one day. This can be easily understood as we think about the daily schedule of the people: for example, a peak period should be expected before the end of their worktime, because it is a common habit to call the call center before going home.

The most primitive solution of the problem is to separate the agents in two groups by assigning always the same agents to incoming and to outgoing calls (group 1 and 2). This policy is still quite commonly used due to its simplicity, despite the obvious disadvantages: during the busy periods there could be a need for more people working on incoming calls to satisfy the constraint

on the average waiting time, while during other periods the productivity of group 1 can be too low: the fraction of time that agents spend working is too small.

A better method, that tries to balance waiting times and productivity, is the so-called *call blending*: dynamically assigning agents either to incoming or to outgoing traffic. Of course, it should be done properly, according to the properties of the system, because otherwise it could produce even worse results than the traditional policy. The optimal way of call blending in the case of constant parameters can be found in [1]. It has been shown that agents should be assigned to outbound calls only if the number of available agents exceeds a certain threshold. But in reality, the parameter of the arrival process changes over time.

The second problem is that the function of the parameter that changes over time is unknown. It can certainly be estimated, before doing anything else, using data from the past (provided that we have access to a sufficient amount of information on the previous years), but an estimate of this kind can be difficult to compute and not representative with respect to the changes of the system over the years. Hence, it could be more convenient to find a method that gives an estimate of the parameter on the spot.

First, we will try to apply the usual principles, the common way of obtaining results regardless of the model: find a reliable estimate of the parameter and substitute it in a proper formula. Different methods will be tried to get the estimate: a moving average, exponential smoothing, and linear prediction. Then we use the formula for the expected average waiting time (published in [1], Section 3) to set a threshold. We always choose a threshold that would be optimal if the process of arriving calls was homogeneous with the actual (estimated) rate.

However, surprisingly enough, this way of thinking is not optimal in this case. As it is revealed later, the approach using stochastic approximation, adjusting the threshold directly, is significantly more efficient here. Not only because it turns out to work better than any of the methods using estimates, but it outperforms even the local optimum, where the estimate is replaced by the real parameter.

In Section 4 we will observe generalizations of the cases considered in the first three sections, for example, when the service parameters are not equal or when the rate function of the arrival process is very peaked. We will examine how well our algorithms perform when applied to these more general processes.

2 Model and important results

In the first part of this section we describe the mathematical formulation of the system that models the call center we want to work with. It is almost the same model that is described in [1]; the only difference is that the parameter of the arrival process is time-varying here. But we still need the most important results for the constant parameter case, therefore we give a short summary of those in the second part.

2.1 The model

Consider a system in which there are s identical servers (playing the role of the agents). There are two types of traffic: type 1 for incoming calls and type 2 for outgoing calls. Their service times are independent, exponentially distributed random variables, with rates μ_1 and μ_2, respectively. Type 1 jobs arrive according to a non-homogeneous Poisson process; the rate at time t is denoted by λ_t. There is an infinite waiting queue for type 1 jobs that cannot be served yet. There is an infinite supply of type 2 jobs: it is always possible to start serving a job of this type, provided that there is at least one idle server. The long-term average waiting time of type 1 jobs should be below a constant α. (Waiting excludes the service time; if the response time is to be considered, then the average service time, $\frac{1}{\mu_1}$, should be added to the average waiting time.)

The objective is to maximize the throughput of type 2 jobs, i.e., to serve on average per unit of time as many type 2 jobs as possible, of course, at the same time obeying the constraint on the average waiting time of type 1 jobs.

Controlling the system can be done through the following actions: whenever a server is idle, it can
– start serving a type 1 job (of course, only if there is at least one waiting in the queue for service);
– start serving a type 2 job;
– remain idle.

Note that it can be optimal here to schedule jobs between completions and arrival instants, not like in the case of a homogeneous arrival process. For instance, if λ_t tends to 0 at a certain moment u, and stays 0 after that, then it is evidently optimal to assign all the idle servers to type 2 jobs at u, regardless of the fact whether or not there has been a completion or arrival at u.

It is very important that in our model preemption of jobs in service is not allowed. The case when preemption is allowed would be trivial. First of all, note that in that case idling is obviously suboptimal, so we need to consider

only scheduling a type 1 job or a type 2 job as action. Here it is optimal to start the service of a type 1 call as soon as possible.

The following coupling argument shows that this is indeed true.

Theorem 1: Suppose that a server is working on type 2 jobs while there are type 1 jobs waiting in the queue. Then, under preemption, the action that interrupts the service of the type 2 job and schedules a type 1 job instead is among the set of optimal actions when service is not lost.

Proof: Let π be an arbitrary policy that respects the waiting time constraint on the type 1 jobs. Take a realization ω under this policy. Suppose that there is a time instant, when a type 1 job (denoted by j_1) arrives and is not scheduled immediately, although there is a server, say s_1, working on type 2 jobs. Let us mention that there must be a later time instant when j_1 is scheduled, since π respects the waiting time constraint on type 1 jobs.

Formally: suppose that there exist t_1, t_2, t_3, and t_4 such that
– j_1 arrives at t_1;
– s_1 is working on type 2 jobs between t_1 and t_2 (let us denote this set of type 2 jobs by J_2), $t_1 < t_2$;
– the service of j_1 starts at t_3 on a server called s_2, and ends at t_4 (by finishing or interrupting its service), $t_1 < t_3 < t_4$.

Note that all the quadruples t_1, t_2', t_3, t_4, where $t_1 < t_2' < t_2$ satisfy all these requirements. Choosing a t_2' small enough, we can get $t_1 < t_2' < t_3 < t_4$ with the same properties. Therefore, we can assume that $t_1 < t_2 < t_3 < t_4$.

We can also assume without loss of generality that $s_1 = s_2$ in this case. (The servers are identical, hence using s_1 to handle j_1 between t_3 and t_4 while letting s_2 serve the jobs that s_1 would have worked on gives the same waiting time and throughput.) Therefore, from now on, we will talk about one server only.

Furthermore, t_2 can be chosen so small that $t_2 - t_1 < t_4 - t_3$. The importance of this property will be revealed later.

So, let us suppose, that there exist $t_1 < t_2 < t_3 < t_4$, with $t_2 - t_1 < t_4 - t_3$, such that
– a type 1 job, j_1, arrives at t_1;
– there is a server working on type 2 jobs (denoted by J_2) between t_1 and t_2;
– j_1 stays in the waiting queue until t_3, when it is scheduled to the server mentioned above and its service there ends at t_4.

Now consider the policy π' with the following actions:
– it follows all actions of π until t_1;
– schedules j_1 at t_1 and lets the server work on it until t_2;
– between t_2 and t_3, the actions of π' are the same as those of π;
– at t_3, lets the server continue the work on j_1 until $t_3+((t_4-t_3)-(t_2-t_1)) = t_4-(t_2-t_1)$;
– then assigns the server to the work on J_2 until t_4;
– after t_4, it follows the actions of π again.

These time intervals are well-defined because of the inequality $t_2 - t_1 < t_4 - t_3$.

The total number of type 2 customers served after t_4 is equal under both policies, thus also the throughput. However, the average waiting time under policy π' is the same as or lower than under π, because the type 1 job is served earlier. Since π and ω were arbitrary, the result follows.

Certainly, the preemptive case is at least as good as the non-preemptive. In practice, the jobs that can be preempted in call centers are e-mail messages. Therefore, it is beneficial for call centers to encourage customers to send their requests by e-mail.

Remark: We used the assumption in the proof that the work already done on the type 2 job that is preempted is not lost, so it can be continued later. This is also true for e-mail messages.

2.2 Results for constant λ

In [1] there are many results that can be used here. We will mostly deal with the case of equal service requirements, i.e., $\mu_1 = \mu_2$, hence we give here only the theorems about that case.

The reason for doing this is that the algorithm of computing the threshold when $\mu_1 \neq \mu_2$ is quite complicated, even in the case of a constant λ, and it obviously gets much worse when the parameter of the arrival process is time-varying. So, even in Section 4, where we will try to handle the case of unequal service times, we use a much more practical approach: simulation instead of applying theorems.

Let us consider the case of equal service requirements first. Define the common service parameter $\mu := \mu_1 = \mu_2$.

First of all, there is an almost obvious, but still very important statement:

Theorem 2: Suppose that a server becomes idle while there are type 1 jobs waiting in the queue. Then the action that schedules a type 1 job is among the set of optimal actions.

We do not give the proof here (all the proofs of the theorems mentioned in this part can be found in [1]), but the following informal argument might help to give insight into this.

Consider the event that a server becomes idle, while there are one or more type 1 jobs waiting. Then the controller has to choose between scheduling a type 1 or a type 2 job (or idling, but this is evidently suboptimal). Giving priority to a type 2 job and delaying type 1 jobs obviously leads to higher waiting times. Delaying the processing of a type 2 job does not change the performance for this type, as we are interested in the *long-term* throughput.

This intuitive argument implies that when a server becomes idle and a type 1 job is waiting, it is optimal to assign this type 1 job to the server.

Let us model the system as a (constrained) Markov decision process as follows:

The state space is $\chi := \mathbb{N}_0$; a state x represents the number of jobs in service plus the number of type 1 jobs in the queue.

We denote the transition rate of going from $x \in \chi$ to $y \in \chi$ (before taking any action) by $p(x, y)$. Then we have $p(x, x-1) = \min\{x, s\}\mu$ (for $x > 0$) and $p(x, x+1) = \lambda$.

The possible actions in state $x \in \chi$, $x < s$ are $a = 0, \ldots, s - x$, corresponding to scheduling a number of a jobs of type 2.

In the states $x \in \chi$, $x \geq s$, a type 1 job is automatically scheduled, because, according to Theorem 2, it is an optimal action.

Next, we uniformize the system (see [7], Section 11.5). For simplicity we assume that $s\mu + \lambda \leq 1$. (We can always get this by scaling.) Uniformizing is equivalent to adding dummy transitions (from a state to itself) such that the rate out of each state is equal to 1; then we can consider the rates to be transition probabilities.

The objectives are modelled as follows. If action a is chosen, then a reward of a is received (1 for each type 2 job that enters the service). This models the throughput. Due to Poisson arrivals and uniformization, the average waiting time is obtained by taking the cost rates equal to the expected waiting time of an arriving customer. Thus, to obtain the average waiting costs, we can take the cost rates equal to $[x - s + 1]^+ / s\mu$, i.e., 0 if $x < s$, and $\frac{x-s+1}{s\mu}$ if $x \geq s$. Note that the cost rates in this case are equivalent to lump costs at each epoch.

We will next see that the optimal policy is of threshold type:

Theorem 3: There is a level c, called the *threshold*, such that if $x < c$, then the optimal action is $c - x$, otherwise an optimal action is 0 (with the intention of capacity reservation for a posterior need of serving type 1 jobs).

A policy obeying these rules is called a *threshold policy with threshold level* c. Note that under such a policy there are always at least c agents working.

Examining the proof in [1], it can be observed that there can be a case when it is both optimal to schedule a type 2 job and not to schedule one (consider scheduling a group of type 2 jobs as scheduling them one by one; it is obvious that both forms are equivalent as is stated in the proof). In this case the two threshold policies, with threshold level c and $c + 1$, are both optimal, and so are all the policies that randomize between these two.

Let the expected average waiting time be denoted by $\mathbb{E}W^q$. In general, to find a threshold policy that satisfies $\mathbb{E}W^q = \alpha$, we need to randomize. Randomization between c and $c+1$ (where $c+1 \leq s$) can be done as follows: If a transition from $c+1$ to c occurs, then we stay in state c with probability δ or we go back to state $c + 1$ with probability $1 - \delta$, i.e., with probability $1 - \delta$ a type 2 job is immediately scheduled.

Finally, we give a formula for the waiting time and throughput under a threshold policy that randomizes between c and $c + 1$ using randomization parameter δ.

Unfortunately, the original expression in [1] is inaccurate. Therefore, we give a proof for these new formulae here. The only difference is that now we will express all the stationary probabilities q_x of the states $x \in \chi$ in q_{c+1} instead of q_c, because state $c + 1$ always exists independently of the value of δ, unlike state c.

We need the following important fact from queuing theory for the proof (see, e.g., Cooper [8], Expression 4.17):

Theorem 4: Define $C(s, \rho)$ as the stationary *delay probability* for s servers and a load of $\rho = \frac{\lambda}{\mu}$ Erlang, i.e., the probability that an incoming request has to wait before the start of its service. Let the stationary probability of a state x be denoted by q_x. Then $\mathbb{E}W^q$ is given by

$$\mathbb{E}W^q = \frac{C(s, \rho)}{\mu(s - \rho)}, \tag{2.1}$$

with

$$C(s, \rho) = \sum_{x \geq s} q_x. \tag{2.2}$$

So, the theorem about the threshold policies is given as follows:

Theorem 5: Let $\rho := \frac{\lambda}{\mu}$. Then the expected average waiting time as a function of the threshold c (where $c \in \mathbb{N}_0$, $c < s$) and the randomization parameter $0 \leq \delta \leq 1$ is given by

$$\mathbb{E}W^q_{(c,\delta)} = \frac{\rho^{s-(c+1)}(c+1)!}{\mu(s-1)!(s-\rho)^2}\, q_{c+1}, \tag{2.3}$$

with

$$q_{c+1} = \left[\frac{\delta(c+1)}{\rho} + \Big(\sum_{x=c+1}^{s-1} \frac{\rho^{x-(c+1)}(c+1)!}{x!}\Big) + \frac{\rho^{s-(c+1)}(c+1)!}{(s-1)!(s-\rho)}\right]^{-1}. \tag{2.4}$$

The expected throughput of type 2 jobs is

$$\mathbb{E}\xi^q_{(c,\delta)} = \mu q_{c+1}\left[\frac{\delta c(c+1)}{\rho} + \Big(\sum_{x=c+1}^{s-1} \frac{\rho^{x-(c+1)}(c+1)!}{(x-1)!}\Big) + \frac{s\rho^{s-(c+1)}(c+1)!}{(s-1)!(s-\rho)}\right] - \lambda. \tag{2.5}$$

When $c = s$, there is no randomization, which is equivalent to choosing $\delta = 1$ as a randomization parameter. The expected average waiting time and throughput of type 2 jobs can be computed as follows:

$$\mathbb{E}W^q_{(s,1)} = \frac{1}{\mu\,(s-\rho)}, \tag{2.6}$$

and

$$\mathbb{E}\xi^q_{(s,1)} = \mu\,(s-\rho)\,. \tag{2.7}$$

Proof: First, consider the case where $c < s$.

Randomization results in a change of the transition rates in the Markov process model from $c+1$ to c of $p(c+1,c) = \delta(c+1)\mu$ and from $c+1$ to itself of $p(c+1,c+1) = (1-\delta)(c+1)\mu$. The lowest possible state is c, as the state moves immediately up to c as soon as $c-1$ is reached. The other positive transition rates are $p(x,x+1) = \lambda$ for all $x \geq c$ and $p(x,x-1) = \min\{x,s\}$ for all $x > c+1$ (see Figure 1).

The standard balance equations are given as follows.

$$\begin{aligned}
\lambda q_c &= \delta(c+1)\mu q_{c+1} && (2.8)\\
\big(\delta(c+1)\mu + \lambda\big)q_{c+1} &= \lambda q_c + (c+2)\mu q_{c+2} && (2.9)\\
(x\mu+\lambda)q_x &= \lambda q_{x-1} + (x+1)\mu q_{x+1}, \quad (c+1 < x < s) && (2.10)\\
(s\mu+\lambda)q_x &= \lambda q_{x-1} + s\mu q_{x+1}, \quad (s \leq x) && (2.11)
\end{aligned}$$

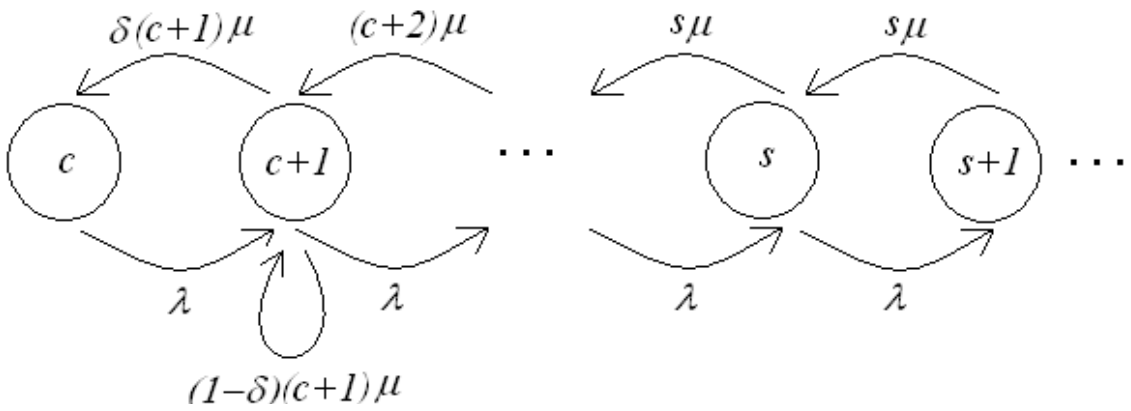

Figure 1: The Markov process for threshold level $c < s$

We can express everything in q_{c+1}:

$$q_c = \frac{\delta(c+1)}{\rho}\, q_{c+1},$$
$$q_x = \frac{\rho^{x-(c+1)}(c+1)!}{x!}\, q_{c+1}, \qquad (c+1 \le x < s),$$
$$q_x = \frac{\rho^{x-(c+1)}(c+1)!}{s^{x-s}s!}\, q_{c+1}, \qquad (x \ge s).$$

These stationary probabilities must sum up to one:

$$1 = q_c + \sum_{x=c+1}^{s-1} q_x + \sum_{x \ge s} q_x.$$

We obtain the expression for q_{c+1} stated in the theorem from this, because

$$C_{(c,\delta)}(s,\rho) := \sum_{x \ge s} q_x = \sum_{x=s}^{\infty} \frac{\rho^{x-(c+1)}(c+1)!}{s^{x-s}s!} q_{c+1} = \frac{\rho^{s-(c+1)}(c+1)!}{(s-1)!(s-\rho)} q_{c+1}.$$

The notation defined above can be justified, for this sum is exactly the probability of delay under the given threshold policy.

The waiting time here is completely equivalent to the one without type 2 jobs. Therefore, we can compute the expected average waiting time with the equivalent form of (2.1):

$$\mathbb{E}W^q_{(c,\delta)} = \frac{C_{(c,\delta)}(s,\rho)}{\mu(s-\rho)} = \frac{\rho^{s-(c+1)}(c+1)!}{\mu(s-1)!(s-\rho)^2} q_{c+1}.$$

This is the formula for the type 1 waiting times. Next, we derive the expression for the type 2 throughput, which we will denote by ξ^q.

The throughput of type 2 is the total throughput minus the type 1 throughput. This equation stays true also in expectation. Hence,

$$\mathbb{E}\xi^q_{(c,\delta)} = \left[\mu c q_c + \sum_{x=c+1}^{s-1} \mu x q_x + \sum_{x\geq s} \mu s q_x\right] - \lambda.$$

The desired result follows by substitution of the expressions of the stationary probabilities in q_{c+1}.

The case $c = s$ is very simple. There are no states below s, so we can give the delay probability without further calculations:

$$C_{(s,1)}(s,\rho) = \sum_{x\geq s} q_x = 1.$$

Thus,

$$\mathbb{E}W^q_{(s,1)} = \frac{C_{(s,1)}(s,\rho)}{\mu\,(s-\rho)} = \frac{1}{\mu\,(s-\rho)}.$$

Computing the expected throughput is even easier. There are s servers working at every time instant, so the expected total throughput is $s\mu$. By subtracting the expected type 1 throughput λ we obtain

$$\mathbb{E}\xi^q_{(s,1)} = s\mu - \lambda = \mu(s-\rho),$$

exactly what we wanted to prove.

Remark: One has to be very careful, because the case $c = s$ is really different: (2.3) with $c = s, \delta = 1$ does not give the same result as (2.6), and nor does (2.5) as (2.7). The reason is that Equation (2.8) is not true here: there should be an s instead of $s + 1$ on the right hand side.

3 Techniques and experiments (numerical results)

We have used a computer simulation of the mathematical model to be able to compare the efficiency of the techniques described later. The algorithm that we used to generate a non-homogeneous Poisson process (by thinning) can be found in [2], Section 4.

Note that a policy that randomizes between c and $c+1$ should obviously perform between the policies using fixed thresholds c and $c+1$. Certainly, if the value of δ is close to 1, the performance will be more similar to the results of threshold policy c, and if δ is very small, then we get approximately the same as with threshold policy $c+1$. Therefore, it is convenient to introduce a notation that allows us to use non-integer thresholds: the integer part of a generalized threshold would denote the original threshold, and the remainder would be $1-\delta$. For instance, a (generalized) threshold policy with threshold level $c = 3.3$ denotes a policy that randomizes between the thresholds 3 and 4, using $\delta = 0.7$ as randomization parameter. This notation will be used quite frequently from now on.

The function of the rate of the arrival process over time will be a linear approximation of the data of the Charlotte Call Center, according to the data in [3] (see Figure 2). By using real data we can expect the shape of the function to be more or less realistic and representative.

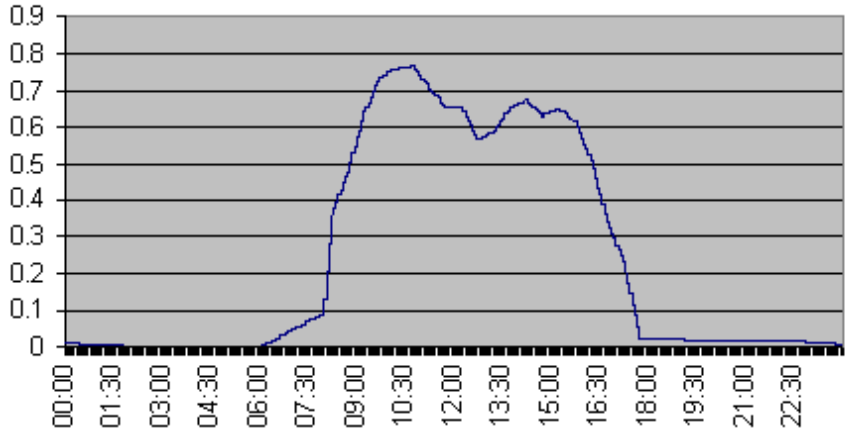

Figure 2: Parameter of the arrival process

We always take the same number of servers: $s = 5$, mostly the same service parameter: $\mu = \mu_1 = \mu_2 = 0.34$, and (when needed) the same constraint: $\alpha = 0.2$, in order to be able to compare the results obtained by using the different techniques. But according to the results of many numerical experiments, these concrete values of the parameters do not matter that much. The conclusions remain the same even when we use different values.

3.1 The traditional solution (fixed thresholds)

The traditional way of assigning work to the agents is simply by dividing them in two groups: the first group (of size n_1) handles the incoming calls, and the second group (of size $s - n_1$) is in charge of handling the outgoing calls.

A clearly better method can be given very easily by taking the threshold policy with $c := s - n_1$ as threshold. (The expression 'clearly better' is supported by Theorem 2.) Thus, we have examined what the simulation gives for all the possible values of the threshold: from 0 to s (see Table 1).

Threshold	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
0	0.04932	0
1	0.05856	907
2	0.08312	1950
3	0.11429	3071
4	0.34443	4205
5	0.93713	5251

Table 1: Fixed thresholds ($s = 5, \mu = 0.34$)

Note that if we had a constraint of, for instance, $\alpha = 0.2$, we could use only $c \leq 3$. This is evidently not optimal.

If we allow using (fixed) randomization parameters, then we can get slightly better results. The experienced average waiting time under the fixed threshold policy with a generalized threshold level of $c = 3.44$ happens to be approximately 0.2 (actually it was 0.19948). The throughput of type 2 jobs for this case was 3171 jobs/hour.

This was the best fixed policy obeying the constraint $\alpha = 0.2$, hence the primary requirement of more complex methods is to at least produce this throughput with an average waiting time of approximately this level.

3.2 Local optimum

In this section, we will apply Formula (2.3) (valid only for constant λ), using the actual real parameter λ_t at time t. Of course, it is not known in real life, but we need to observe this case to be able to compare the results of the implementable techniques to it.

So, we would like to get the locally optimal threshold and randomization parameter. We call it *local optimum*, because it would be optimal if the parameter of the arrival process was constant. An algorithm for finding the optimal threshold in the constant case (applying Equation (2.3) and Equation (2.6)) is as follows:

1. $c := 0$;
2. while $(c+1 \leq s)$ AND $(\mathbb{E}W^q_{(c+1,1)} \leq \alpha)$ do
3. begin
4. $c := c+1$;
5. end;

This algorithm gives a c such that $\mathbb{E}W^q_{(c,1)} \leq \alpha$, and $\mathbb{E}W^q_{(c+1,1)} > \alpha$ (or $c = 0$ or $c = s$ in extremal cases; one of the equalities might not be true then, but there is nothing to do about it as 0 and s are natural bounds for c).

Note that this method is based on an implicit assumption, namely on the monotonicity of $\mathbb{E}W^q_{(c,1)}$ in c. This is evidently true, but we give a formal proof, because this algorithm plays a very important role in the methods we want to use in this paper.

Theorem 5: $\mathbb{E}W^q_{(c,1)}$ is increasing in c.

Proof: Let c_1 and c_2 be different thresholds: $c_1, c_2 \in \mathbb{N}_0$, $c_1 < c_2 \leq s$. Let us denote the stationary distribution of the Markov reward processes corresponding to the threshold policies with threshold level c_1 and c_2 by ν_1 and ν_2, respectively. The expected average waiting time can be computed with Formula (2.1) again:

$$\mathbb{E}W^q_{(c_i,1)} = \frac{C_{(c_i,1)}(s,\rho)}{\mu\,(s-\rho)},$$

with

$$C_{(c_i,1)}(s,\rho) = \sum_{x \geq s} \nu_i(x), \qquad (i = 1, 2).$$

We will show that $\nu_1(x) < \nu_2(x)$ for all $x \geq c_2$ (thus, for all $x \geq s$ too). In the states $x \in \chi, x \geq c_2$, the behaviour of the system is the same in both cases. Hence, the fraction of the stationary probabilities must be the same:

$$\frac{\nu_1(x)}{\nu_1(y)} = \frac{\nu_2(x)}{\nu_2(y)}, \qquad (x, y \in \chi; x, y \geq c_2).$$

Let $x \in \chi$, $x \geq c_2$ be fixed. We can express all the stationary probabilities with $\nu_1(x)$ and $\nu_2(x)$:

$$\nu_2(z) = \nu_1(z)\frac{\nu_2(x)}{\nu_1(x)}, \qquad (\forall z \in \chi, z \geq c_2).$$

Summing this over the states $z \geq c_2$ we obtain

$$\sum_{z \geq c_2} \nu_2(z) = \left(\sum_{z \geq c_2} \nu_1(z) \right) \frac{\nu_2(x)}{\nu_1(x)}.$$

But in the second system, the lowest possible state is c_2, so

$$\sum_{z \geq c_2} \nu_2(z) = 1,$$

while in the first system, the state can be below c_2 with a probability $p > 0$, hence

$$\sum_{z \geq c_2} \nu_1(z) = 1 - p < 1.$$

The result follows.

After using this algorithm, we know that we need to randomize between c and $c+1$ to achieve $\mathbb{E}W^q_{(c,\delta)} = \alpha$. Such a δ must exist, with a possible exception in the extremal cases. If $\mathbb{E}W^q_{(0,1)} \geq \alpha$, then we cannot do anything but to choose $\delta = 1$, and if $c = s$, then we cannot randomize as choosing $s+1$ as a threshold is not possible (no randomization in this case is equivalent to choosing $\delta = 1$). Otherwise, the existence of a good δ is ensured by the theorem of Bolzano, due to the continuity of $\mathbb{E}W^q_{(c,\delta)}$ in δ between this c and $c+1$ (which can be seen clearly by looking at Expression (2.3)).

But the proper randomization parameter still needs to be found. This is not difficult at all, taking in account that $\mathbb{E}W^q_{(c,\delta)}$ is decreasing in δ under a fixed c. This property follows directly from the definition of the randomization parameter, keeping in mind that $\mathbb{E}W^q_{(c,1)} < \mathbb{E}W^q_{(c,0)}$, because the second expression equals to $\mathbb{E}W^q_{(c+1,1)}$, and $\mathbb{E}W^q_{(c,1)}$ is increasing in c.

Therefore the problem can be solved as follows (for $c \leq s-1$):

1. $stepsize := 0.01$; (the accuracy of the method)
2. $\delta := 1$;
3. while $(\delta - stepsize > 0)$ AND $(\mathbb{E}W^q_{(c,\delta-stepsize)} \leq \alpha)$ do
4. begin
5. $\delta := \delta - stepsize$;
6. end;

By adjusting the step size (choosing it small enough) we can get arbitrarily close to the optimal δ.

So now we know how to compute the optimal threshold and the randomization parameter, provided that the values of s, μ, and λ are known (they are really needed, because we use Expression (2.3)).

The only difference in the time-varying parameter case is that we need to use λ_t at time t instead of using always the same λ when computing $\mathbb{E}W^q_{(c,\delta)}$ with Expression (2.3). This way we can obtain the locally optimal threshold: c_t, and the corresponding locally optimal randomization parameter: δ_t.

However, it needs to be stressed, that these thresholds do not have to give the best results possible – being everywhere locally optimal in this sense does not necessarily mean being globally optimal. But at least we can hope that it will be good enough – at least better than the policies with a fixed threshold. Let us see whether this is true (see Table 2).

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.19519	4112
2	0.21129	4100
3	0.20011	4121

Table 2: Results using locally optimal thresholds ($s = 5, \mu = 0.34, \alpha = 0.2$)

Remember: the fixed thresholds that could have satisfied the constraint were $c = 0, 1, 2, 3$. But the throughput they can produce is much less than the throughput of our new method. Even the best constant policy obeying the constraint ($c = 3$, $\delta = 0.56$) is by about 30% behind with respect to the productivity. From another point of view: we would need a fixed threshold of at least 4 to get the throughput obtained here. But the waiting time (which represents the quality of the service!) is more than one and half times as much as the one experienced here.

Note that the average waiting time has stayed quite close to the constraint. According to many experiments, the difference is almost always within 10% (usually even less than 5%). Hence, if we really want to stay below α, then this can be done with high probability by taking $\alpha' := 0.9\alpha$ as constraint. Nevertheless, we can certainly claim that this method is reliable: we could never have satisfied the constraint for sure due to the randomness of the process.

So it can be seen that we can outperform the traditional method by far if we know the real parameter. Thus, it might be really worth trying to obtain good estimates for λ_t, hoping that a small difference would not cause too much trouble and we could still get better performance than in the case of fixed thresholds.

3.3 Methods for estimating λ_t

The superiority of the results using the locally optimal thresholds and randomization parameters over the method of fixed thresholds seems to be impressive. But actually, although being promising, in reality it could be totally useless, because naturally we do not know the function of the arrival rates over time. We can never even measure this parameter directly, we can only estimate it; the estimate of λ_t has to be based on the number of arrivals before t, because that is observable. There are two different attitudes to consider.

The first possibility is to estimate λ_t in advance, according to the data of the previous days/months/years. This can be pretty problematic. First of all, there might not be enough data at new companies to get a reliable estimate – unless using data of other firms, which is always a dubious thing to do concerning the differences likely to appear between the two systems. Secondly, even if our company is old enough to produce a sufficient amount of data, circumstances might change over time, and changes are sometimes not so easy to be traced. Finally, even if we assume everything to be approximately constant, estimating the parameter is still quite hard, due to the complicated cyclic behaviour that can be observed in a call center (for example, see [3], Section 3.2).

Therefore, we prefer the other possibility: calculating the estimate of λ_t on the spot, based only on the number of incoming calls in the interval $[0, t)$, where 0 is the very first moment of the day. Note that an estimate of λ_t that uses only the number of arrivals is not model-dependent. Thus, the methods described in this section can be used to estimate the rates of a general (non-homogeneous) Poisson process.

Our goal is to find a proper, implementable method with which we could

obtain results close to the local optimum or even outperform it – but most importantly it should work better than the fixed threshold policies. Hoping that something better than the local optimum could be found is not unrealistic: a difference between the estimate and the real parameter is not necessarily bad, it might even help to get closer to the global optimum. The locally optimal policies do not anticipate on the future, whereas the estimates might.

So, the first task is to estimate λ_t as precise as possible. The next step could be a method that can even predict λ_t producing a curve of the function not reacting to the changes of the real one, but predicting the changes. We call a curve of this type *pro-active*, because it changes before the real one would have done the same. Using estimates corresponding to a pro-active curve could be a possible way of outperforming the local optimum, because capacity is provisioned before the actual surge in the call rate.

Let us see the techniques and their performances.

3.3.1 Average-type estimate

A first thought could be to take the average number of arrivals until t as an estimate at time t. This would be a good estimate in the case of a stationary process. A short argument corroborating this is the following.

Suppose that we have a homogeneous Poisson process with rate λ. Let the number of events between s and t be denoted by $N(s,t)$ (we will use the same notation in the non-homogeneous case as well). We know that the distribution of $N(s,t)$ is Poisson with parameter $\lambda(t-s)$. Therefore, its expected value equals the parameter: $\mathbb{E}N(s,t) = \lambda(t-s)$. So, an unbiased estimate for the parameter is given by $\hat{\lambda} := \frac{N(s,t)}{t-s}$.

Thus, if the function of the arrival parameter over time was "not too far from being constant" (e.g., very slowly varying), then we could expect quite good results by applying the following formula:

$$\hat{\lambda}_t := \frac{N(0,t)}{t}. \tag{3.1}$$

Let us try it out (see Table 3).

This performance is quite disappointing: none of the good properties of the locally optimal case seem to appear here. Although the throughput is good enough, the waiting times are much too high: they exceed the constraint by far. Therefore, this method is useless. But this is not surprising at all, if we take a look at Figure 3 comparing the real function and the estimate.

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.58055	4382
2	0.55525	4350
3	0.56568	4402

Table 3: Results using average-type estimates ($s = 5, \mu = 0.34, \alpha = 0.2$)

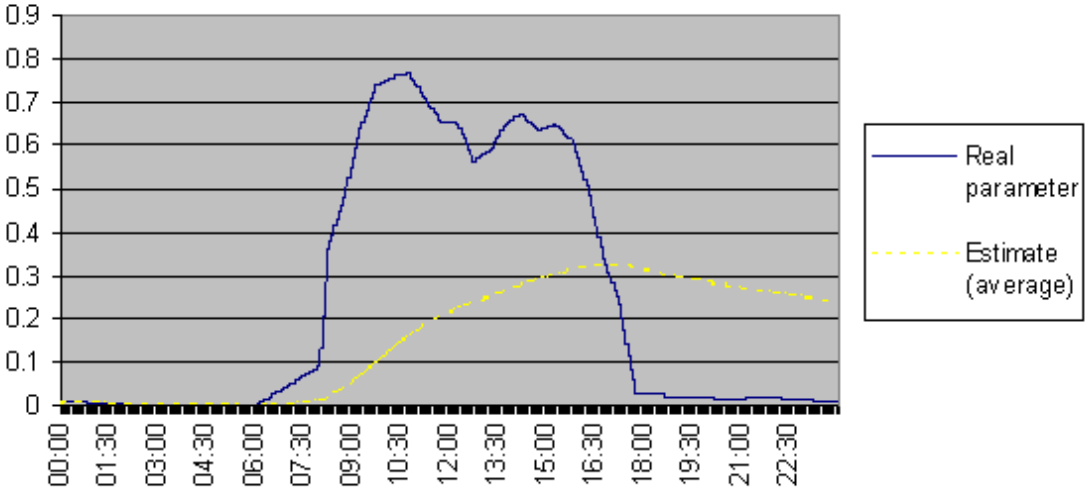

Figure 3: Average-type estimate of λ_t

The estimates are very far from the real function – this type of estimate might have been good in the constant parameter case, but it miserably fails here. This is certainly a sufficient reason explaining the poor performance.

It can be concluded that taking this average is too primitive to work well; we need more sophisticated methods.

3.3.2 Moving average

We saw in Section 3.3.1 that the call rate function is very far from being constant. But in small time intervals of length l, the difference might not be so big. Hence, we use only the arrivals in the last l seconds.

The method can be formulated as follows:

$$\hat{\lambda}_t := \frac{N(t-l,t)}{l}. \tag{3.2}$$

This formula seems to be much more promising than Formula (3.1), because it reacts faster to the changes.

Unfortunately, the choice of the parameter is crucial here: l should not be too small, because a sample of a sufficient size is needed to estimate the parameter of a process. On the other hand, l should not be too large either, because then the average-type estimate would take effect, which was not satisfying.

Nonetheless, this method gave much better estimates of λ_t than Formula (3.1), almost regardless of the value of l (see Figure 4).

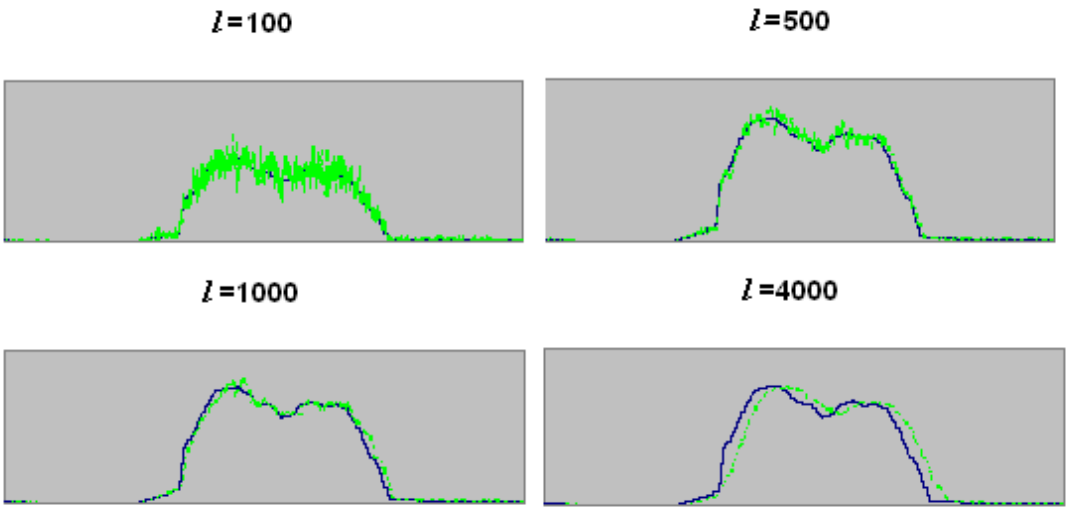

Figure 4: Moving average with different l interval lengths

It seems that $l = 1000$ gives the best estimate. For smaller interval lengths, the fluctuation around the real function is too strong; for $l = 4000$, the lag of the estimate with regard to λ_t is too large. In the case of $l = 1000$, there is almost no lag, and the fluctuation is not that big either, although a smoother function would be preferable.

The results of the computer simulation for different interval lengths can be found in Table 4.

The performance more or less validates our preconceptions about the curves. The throughput is not that high in the case of $l = 100$, and there are problems with the average waiting time if $l = 4000$.

According to the data of many experiments with different parameters s and μ, the average waiting time stays always around the constraint. Even if it exceeds α, the difference is very small. Therefore, this method can be considered reliable enough. Its productivity is high – almost as high as in

Number of experiment	Length of the interval (l)	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	100	0.19841	4056
2	100	0.20821	4052
3	500	0.19968	4102
4	500	0.19800	4106
5	1000	0.19832	4113
6	1000	0.20081	4074
7	4000	0.20543	4128
8	4000	0.21533	4095

Table 4: Results with the moving average ($s = 5, \mu = 0.34, \alpha = 0.2$)

the case of using the real parameter. Hence, we can say, that this method has unambiguously outperformed the policies using fixed thresholds. The implementation is very easy: only the arrival times of the incoming calls in the last 1000 seconds (i.e., approximately 17 minutes) need to be registered.

Regarding to these good properties, we can say that we have achieved our first goal: using this easily implementable method instead of fixed threshold policies would result in an improvement of around 30% in the productivity. But this method only approaches the local optimum, it does not give *better* results than that.

Hence, we try a third technique, that hopefully unifies the good properties of these estimates with different interval lengths.

3.3.3 Exponential smoothing

The key idea of exponential smoothing is the (exponentially) decreasing importance of the data of the past compared to the recent observations. Actually it is a weighted average where the weights of the new events are big, but old observations count as well, albeit with much smaller weights.

We used 1000 seconds as a time unit and 2 as smoothing parameter. This means that the importance of the number of arrivals in $[t - 1000, t)$ is twice as much as those in $[t - 2000, t - 1000)$, four times as much as the same number in $[t - 3000, t - 2000)$, etc. For simplicity, we take into account only the events that occurred in the last 7000 seconds (which is about a 2 hours' period). With this choice, we expect that the fluctuations of the estimate disappear and that the lag gets smaller.

The mathematical formulation of this idea is given by

$$\hat{\lambda}_t := \frac{64N_1 + 32N_2 + 16N_3 + 8N_4 + 4N_5 + 2N_6 + N_7}{127000}, \qquad (3.3)$$

with the notation $N_k := N\big(t - 1000k, t - 1000(k-1)\big)$.

Note the number in the denominator: 127=64+32+16+8+4+2+1.

Unfortunately, the results are not as good as we expected (see Table 5).

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.20598	4089
2	0.20369	4096
3	0.20346	4121

Table 5: Results with exponential smoothing ($s = 5, \mu = 0.34, \alpha = 0.2$)

The throughput of type 2 jobs is very high, but not significantly more than it was in the case of using a moving average. The average waiting time is slightly above the constraint. Therefore, this method is at most of the same quality as taking the moving average in this particular application. Maybe the relationship between the estimate and the real function can explain this.

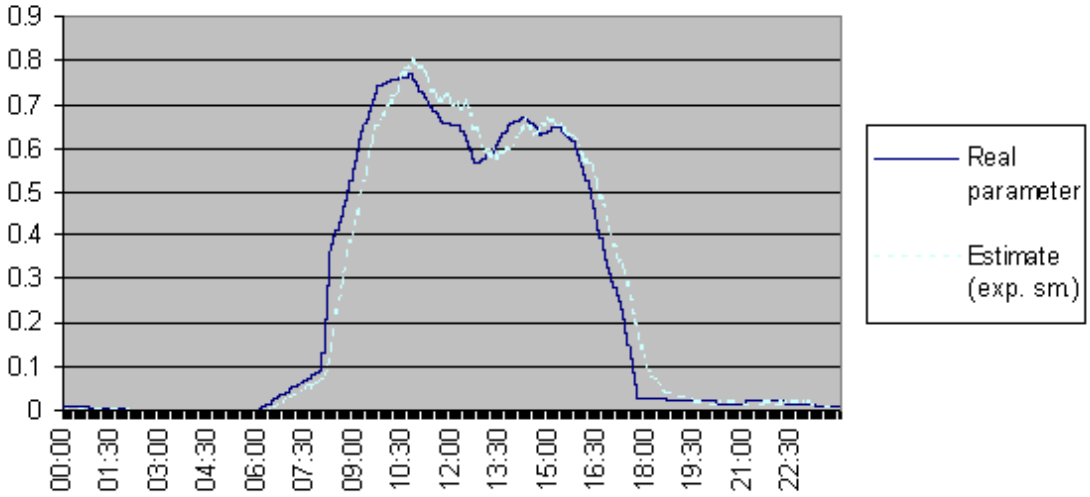

Figure 5: Estimate of λ_t using exponential smoothing

We can see in Figure 5, that this curve is smooth enough – the lag is

smaller, but it still exists. Therefore, our next target is to find a method that can produce a pro-active curve.

3.3.4 Linear extrapolation

Linearly projecting the past to get a forecast for the future is probably one of the simplest way of predicting. But, as can be read in [5]: '... when applied to web-server page-request traffic, even elementary prediction techniques can have a surprising forecasting power'. The excellent results described in that paper inspired us to try out the method here.

The general problem is the following:

Given points in a plain with coordinates $(x_i, y_i)_{i=1}^n$, what are the parameters of the line $y = ax + c$, that fits these points best?

A solution can be given as follows:

The problem can be reformulated as searching for a and c that satisfy the matrix-equation

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ & \\ 1 & x_n \end{pmatrix} \begin{pmatrix} c \\ a \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ \\ y_n \end{pmatrix} \tag{3.4}$$

the best, in the sense that the norm of the error is minimal.

The next well-known theorem gives the solution:

Theorem 4: For a given matrix $A \in \mathbf{C}^{n\times m}$ and a vector $b \in \mathbf{C}^n$, the vector $x^- := A^- b \in \mathbf{C}^m$ satisfies $||Ax^- - b|| \leq ||Ax - b|| \; \forall x \in \mathbf{C}^m$, where A^- denotes the Moore-Penrose generalized inverse of A.

The columns of our matrix A are independent (except in the case of $x_1 = x_2 = \cdots = x_n$, where the line we are looking for can be given as $y = x_1$), thus the generalized inverse can be given easily by $A^- = (A^T A)^{-1} A^T$.

Using all these results we get

$$\begin{pmatrix} c \\ a \end{pmatrix} = \begin{pmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{pmatrix}. \tag{3.5}$$

We just have to apply this method for forecasting. As we cannot observe the arrival parameter directly, we have to use estimates of the arrival parameter from the past. Of course, it needs to be decided which estimation method we want to use. Taking the moving average is very simple and still

very effective. In the case of $l = 1000$, the estimate was always very close to the real parameter. So, we decided to use it here too. The other question to decide on is the number of points to which the line should be fitted. An extrapolation using only two points is evidently not reliable enough. So we will try $n = 3$ and $n = 4$.

Nevertheless, the following formulation of the problem includes all the cases. Let l be the length of the interval we want to use.

The estimates of the old parameters are the following:

$$\hat{\lambda}_j := \frac{N(t - jl, t - (j-1)l)}{l}, \qquad (j = 1, \ldots, n).$$

Thus, the points to fit a line on are given by

$$\begin{array}{cc} x_1 := t - \frac{l}{2}, & y_1 := \hat{\lambda}_1, \\ x_2 := t - \frac{3l}{2}, & y_2 := \hat{\lambda}_2, \\ \vdots & \vdots \end{array}$$

$$x_n := t - \frac{(2n-1)l}{2}, \quad y_n := \hat{\lambda}_n.$$

So, the estimate of the parameter of the arrival process at time u, using c and a from Expression (3.5) with these points is given by

$$\hat{\lambda}_u := au + c, \quad (u \geq t). \tag{3.6}$$

It needs to be mentioned that this number can happen to be negative. In this case, one should take $\hat{\lambda}_u := 0$.

Certainly, the old parameters need to be updated regularly. We did it whenever an event (a new arrival or the end of a service) had occurred.

Finding the most appropriate length of the interval is a very difficult task in itself. Fortunately, the variance of the results is small in this case too, so at least we can have an idea about it by comparing the numerical experiments. Eventually, $l = 1000$ turned out to give quite good results (see Table 6).

Number of experiment	Value of n	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	3	0.20087	4103
2	3	0.19597	4096
3	4	0.19761	4078
4	4	0.18963	4101

Table 6: Results using linear extrapolation ($s = 5, \mu = 0.34, \alpha = 0.2$)

So, we have found yet another method of approximately the same quality as the moving average and exponential smoothing. All three methods have similar advantages and disadvantages: while having an average waiting time very close to the constraint, their productivity with regard to type 2 jobs is high, but not better than the local optimum.

Let us see the relationship between the real function and the estimates:

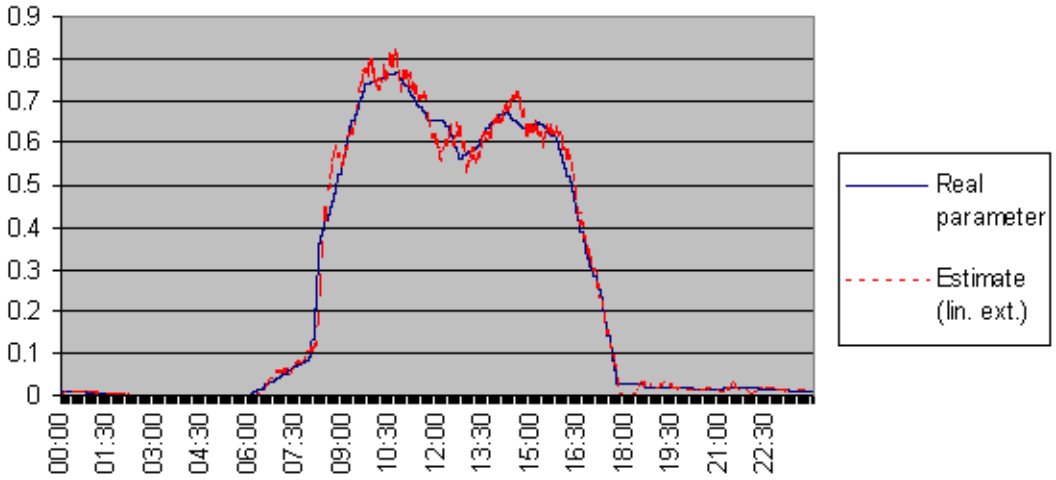

Figure 6: Linear extrapolation-type estimate of λ_t using $n = 3$ points

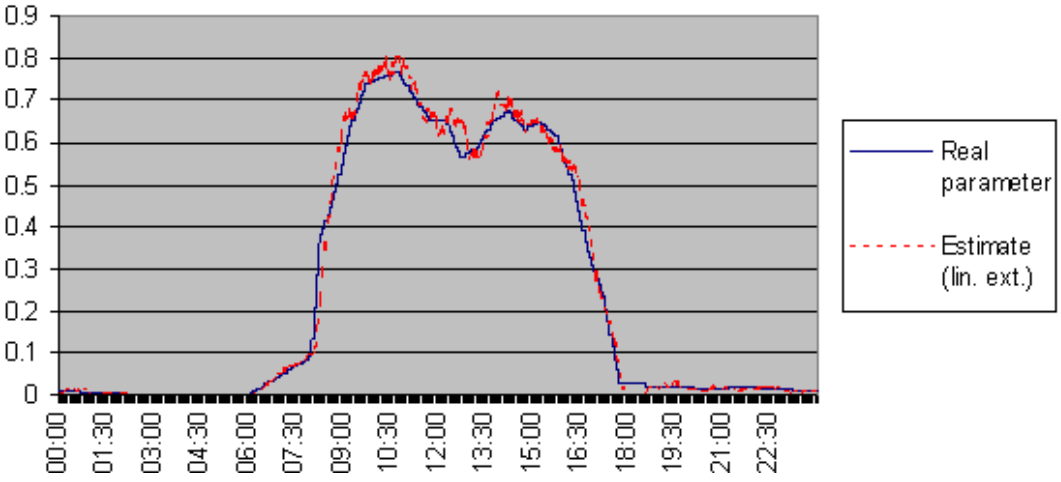

Figure 7: Linear extrapolation-type estimate of λ_t using $n = 4$ points

These curves are still not pro-active enough, neither in the case of $n = 3$, nor when $n = 4$. Therefore, in the next section we try something more drastic.

3.4 Known function

Here we assume that the whole function of the arrival parameter is known. This is obviously an unrealistic assumption. This allows us to check whether we could really outperform the local optimum in our model by using a pro-active curve.

3.4.1 Future average

Taking simply λ_{t+u} as the "estimate" of λ_t would certainly predict the changes of the curve (precisely u seconds in advance), but should a sudden jump occur, it could cause very large differences between the used and the real parameter. We prefer a method that takes also the actual parameter into account.

Therefore, our first idea was to take the average:

$$\hat{\lambda}_t := \frac{\lambda_t + \lambda_{t+u}}{2}, \qquad (u > 0). \tag{3.7}$$

Results for different values of u are in Table 7.

Number of experiment	Value of u (seconds)	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	600	0.20153	4121
2	600	0.19860	4109
3	1200	0.19884	4140
4	1200	0.19701	4132
5	1800	0.20510	4117
6	1800	0.20912	4127

Table 7: Future average ($s = 5, \mu = 0.34, \alpha = 0.2$)

We can see that this method works well, indeed. It produces high throughput with average waiting times close to the constraint.

Undoubtedly $u = 1200$ gives the best results: the throughput of type 2 jobs is the highest here, while the experienced average waiting times are below the ones when using other values of u. However, these results are not significantly better than the ones using the real λ_t, in spite of the curves being pro-active (see Figure 8).

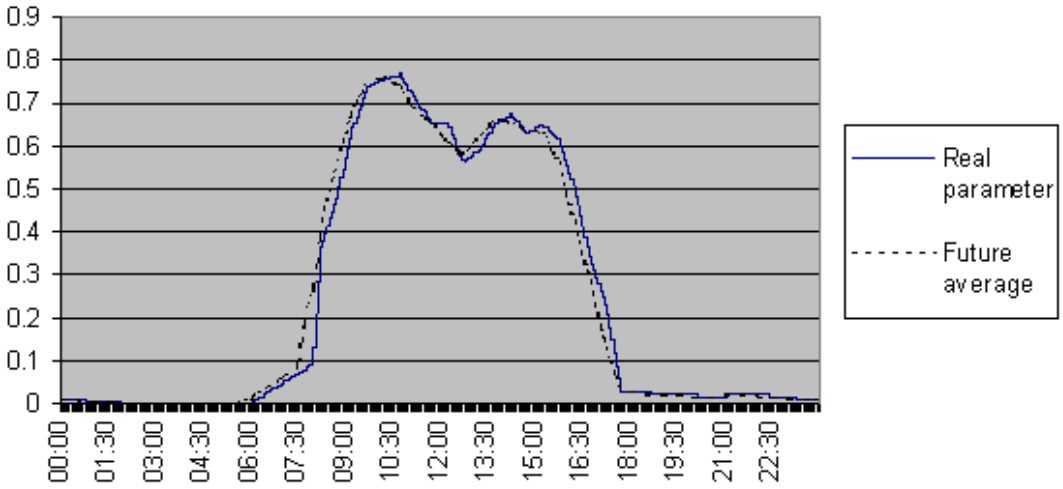

Figure 8: Pro-active estimate 1 (Future average, $u = 1800$)

3.4.2 Forward exponential smoothing

The results achieved by using exponential smoothing had been close to the local optimum, so it seemed interesting to try it the other way: using data from the future. Furthermore, it might have been a big advantage in this case that we were allowed to use the real parameters here, there was no need for estimating them. The estimated function can be seen in Figure 9.

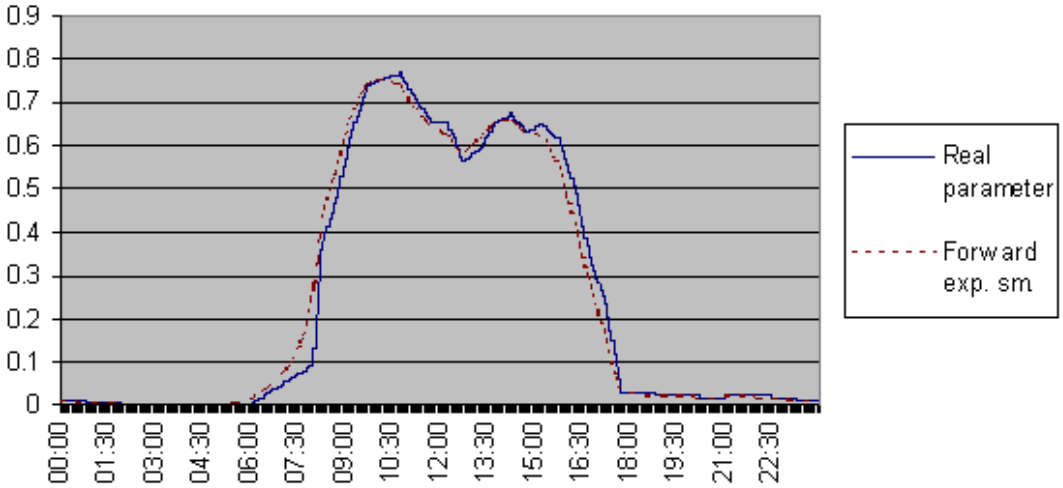

Figure 9: Pro-active estimate 2 (Forward exponential smoothing)

This is virtually the same as Figure 8, only this curve is slightly smoother.

The results do not differ too much either, though both the throughput and the average waiting times are lower (see Table 8).

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.19787	4035
2	0.19169	4041
3	0.19509	4031

Table 8: Results of forward exponential smoothing ($s = 5, \mu = 0.34, \alpha = 0.2$)

We can see that although having all the information possible, the results are not significantly better than the local optimum. Hence, it seems that we need to find a different way to gain better results.

Until now, we made decisions independently of the effects of our actions beforehand. The next step will be a method that has a learning effect.

3.5 Stochastic approximation

When we evaluate a method, the primary requirement is that it has to satisfy the constraint on the type 1 waiting times (with high probability). Hence, it seems to be reasonable to take actions depending on the relationship between the constraint and the actual experienced average waiting time. If the waiting time exceeds the constraint, then using a smaller threshold may be preferable, while if the waiting time is far below the constraint, then we might gain a higher throughput on type 2 jobs at the expense of having a slightly higher average waiting time. The Robbins-Monro algorithm, published in [10] – a seminal work in the area of stochastic approximation – is based on exactly this idea. We will use something similar to that, so we give the short review of the algorithm and the corresponding main theorem, following the context of [9].

3.5.1 Robbins-Monro algorithm

Robbins and Monro consider a problem where an experiment is performed at different levels (inputs), and the response to the experiment can be measured. Formally, to each input value x , the measurement made is drawn from a random variable Y with distribution $p(Y|x)$ conditioned on x. Assume that measurements are independent, and are identically distributed conditioned on the input x. Let $M(x)$ denote the expected response to the experiment when the input is x, formally: $M(x) := \mathbb{E}(Y|x)$. Let α be a given number.

The problem is to determine the input that produces an expected response of α, i.e., the root θ of the equation $M(x) = \alpha$.

Robbins and Monro wanted to design a recursive procedure that converges to the root, according to the scheme as follows.

1. Start with some input x_1, and measure the response y_1.
2. The input to the system in the second step is given as some function of these values, formally $x_2 := f(x_1, y_1)$.
3. Repeat this procedure, choosing the input to the measurement mechanism in step n as a function of the inputs and the measured responses in the previous $(n-1)$ steps.

The authors considered the update scheme described by Equation (3.8):

$$x_{n+1} = x_n + a_n(\alpha - y_n). \tag{3.8}$$

They showed that in the presence of certain regularity assumptions, the sequence of the inputs converges to the desired root in probability.

The main result in [10] is the following theorem.

Theorem 6: Assume that:
– $\forall x \ \exists C \in \mathbb{R}$ such that $P(|Y(x)| \leq C) = 1$ $\big($assumption on $p(Y|x)\big)$;
– $M(x)$ is non-decreasing in the input; the equation $M(x) = \alpha$ has a unique root in θ $\big($assumptions on $M(x)\big)$;
– The step sizes a_n are nonnegative, but not too big, and they are of type $1/n$. Formally: $a_n \geq 0$, $\sum a_n^2 < \infty$, and $\exists m', m''$ such that $\frac{m'}{n} \leq a_n \leq \frac{m''}{n}$ $\big($assumptions on the step sizes$\big)$.

Then, $x_n \to \theta$ in mean square (and therefore, in probability).

The proof and generalizations of this theorem can be found for example in [9] or in [10].

3.5.2 Call blending with SA

We will use a method similar to the Robbins-Monro algorithm described above. First of all, we choose a (short) time interval of length l. We will set the (general) threshold at time instants nl where $n \in \mathbb{N}$, with some initial threshold. It is convenient to take $l = 1$. We want to adjust the threshold with regard to the relationship between the constraint and the waiting time. In our case, the parameter of the arrival process is time-varying, therefore the step size should not go to zero. About choosing the proper step sizes,

one can find applicable methods in [4] and in [6]. We chose to use a constant step size instead of taking a sequence of different step sizes.

Let us denote the experienced average waiting time until time t by α_t (this can always be measured). With this notation, the formula we used is as follows:

$$c_{n+1} = c_n + \varepsilon(\alpha - \alpha_n). \tag{3.9}$$

The results obtained by using this update scheme can be seen in Table 9.

Number of experiment	Value of ε	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.01	0.91239	4967
2	0.01	0.92287	4952
3	1	0.50701	3434
4	1	0.48634	2959
5	3	0.28429	2155
6	3	0.27633	2672
7	5	0.20176	2696
8	5	0.22046	2443
9	10	0.23651	2624
10	10	0.22937	2803

Table 9: Results with update scheme (3.9) ($s = 5, \mu = 0.34, \alpha = 0.2$)

We can see that at least $\varepsilon = 5$ is needed to have a chance of satisfying the constraint. But even in those cases, the throughput of the outgoing calls is low compared to the ones considered earlier. Therefore, this method is not good enough.

But we did not use all the information we had. We could perhaps construct a method that produces consistently high performance by using our formulae (Expressions (2.3) and (2.6)) for $\mathbb{E}W^q$.

The only problem with this is that the knowledge of all the parameters (including the rate of the arrival process) is needed. Thus, we need to estimate λ_t again, for example with exponential smoothing.

The new formula for updating the threshold is given by

$$c_{n+1} = c_n + \varepsilon \left[c_n - (\mathbb{E}W^q_{\hat{\lambda}_n})^{-1}(\alpha) \right], \tag{3.10}$$

where $(\mathbb{E}W^q_{\hat{\lambda}_n})^{-1}(\alpha)$ denotes the generalized threshold that gives an expected average waiting time of α if the rate of the arrival process is $\hat{\lambda}_n$, and can be obtained for example by the algorithms in Section 3.2.

The results in Table 10 show that this method works much better than the previous one.

Number of experiment	Value of ε	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.5	0.23249	4067
2	0.5	0.21142	4088
3	1	0.21598	4093
4	1	0.21340	4100
5	1.5	0.19613	4140
6	1.5	0.20654	4111
7	2	0.18770	4061
8	2	0.20766	4091
9	3	0.20637	3798
10	3	0.21137	3777

Table 10: Results with update scheme (3.10) ($s = 5, \mu = 0.34, \alpha = 0.2$)

It seems that the value of ε should be between 1 and 2. In those cases, the average waiting time is close to α, while the throughput of the outgoing calls is high – sometimes even higher than the local optimum. But the difference is very small.

We also tried a third, hybrid update scheme given by

$$c_{n+1} = c_n + \varepsilon_1 \left[c_n - (\mathbb{E}W^q_{\hat{\lambda}_n})^{-1}(\alpha) \right] + \varepsilon_2 \, sgn(\alpha - \alpha_n). \tag{3.11}$$

This method uses the information on the average waiting time, and has a learning effect as well, so we can expect good performance. Indeed, it gives very good results with the choice of $\varepsilon_1 = 1.4$ and $\varepsilon_2 = 0.1$ (see Table 11):

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.19992	4190
2	0.19985	4218
3	0.19997	4114
4	0.20001	4358

Table 11: Results of computer simulation using SA, update scheme (3.11) ($s = 5, \mu = 0.34, \alpha = 0.2, \varepsilon_1 = 1.4, \varepsilon_2 = 0.1$)

The results show that this is clearly the best policy among the ones tried here. Albeit only slightly, it has outperformed the local optimum, repeatedly, while obeying the constraint. Note that the experienced average waiting time is very close to α every time. Unfortunately, the variance of the throughput of the type 2 jobs is larger than in the other cases. Nevertheless, even the worst one here is as good as the local optimum.

It might be possible to obtain even better results with a proper choice of the parameters. But the optimal parameters depend on the function of the arrival rate and the other parameters of the model as well, hence they need to be determined somehow in every particular case. The most important thing is the fact that the local optimum can be outperformed by using stochastic approximation.

4 Generalizations

We examined different methods for call blending in Section 3 according to the model described in Section 2. But the cases considered there were special: we always used the same function of the arrival rate and assumed that the service parameters were equal. It needs to be examined how the algorithms perform in more general cases.

4.1 Unequal service requirements

The assumption that the service parameters are equal is a bit strange – it is a very opportunistic case when the service times of type 1 jobs and type 2 jobs are exactly of the same distribution. But if it is not true, then we cannot use formulae (2.3) and (2.6) for computing $\mathbb{E}W^q$ in the algorithms of Section 3.2 to obtain the optimal threshold and randomization parameter. Hence, there is a problem with all the methods based on estimating the arrival parameter and calculating the locally optimal threshold with the algorithms by substituting the estimate in the formulae.

As we can see from this argument, the only thing needed to be able to use the methods in Section 3 is to compute the optimal threshold and randomization parameter somehow. It could be done by using the recursive formula for calculating $\mathbb{E}W^q$ in the case of $\mu_1 \neq \mu_2$ described in [1], but this is technically not so easy. Therefore, we tried to obtain a more simple algorithm.

4.1.1 Averaging μ_1 and μ_2

Our first idea was to reduce this case to the one with equal service parameters, i.e., to make somehow one common parameter from the two different parameters. After doing this, we can compute $\mathbb{E}W^q$ with Expressions (2.3) and (2.6), and everything goes like before. Our task is to find a method that unifies μ_1 and μ_2 so that the system with the new common parameter μ behaves similarly to the one with the different parameters. Of course, we do not even know whether or not such a μ exists.

The simplest way of unifying two numbers is to take the average:

$$\mu := \frac{\mu_1 + \mu_2}{2}. \tag{4.1}$$

We computed the local optimum with this μ, i.e., we always used the threshold that would have been optimal in the equal parameter case with the real actual arrival parameter and this service parameter. Unfortunately, this procedure does not work well (see Table 12).

Number of experiment	μ_1	μ_2	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.20	0.50	1.20336	5205
2	0.20	0.50	0.97920	5214
3	0.30	0.40	0.24594	4595
4	0.30	0.40	0.23833	4604
5	0.40	0.30	0.17675	3653
6	0.40	0.30	0.17001	3672
7	0.50	0.20	0.20651	2554
8	0.50	0.20	0.21278	2558
9	0.75	0.05	1.12203	732
10	0.75	0.05	1.12562	730

Table 12: Local optimum with Formula (4.1) ($s = 5, \alpha = 0.2$)

If the difference of the two parameters is small, then it is not that bad, but otherwise this method is useless, because the average waiting time exceeds α by far.

Despite these results being very disappointing, we tried an other unifying method too, the geometric average:

$$\mu := \sqrt{\mu_1 \mu_2}. \tag{4.2}$$

The results can be seen in Table 13.

Number of experiment	μ_1	μ_2	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.20	0.50	1.06300	5057
2	0.20	0.50	0.98324	5056
3	0.30	0.40	0.23411	4587
4	0.30	0.40	0.23339	4590
5	0.40	0.30	0.17389	3654
6	0.40	0.30	0.16990	3648
7	0.50	0.20	0.12988	2454
8	0.50	0.20	0.13175	2459
9	0.75	0.05	0.05767	495
10	0.75	0.05	0.05767	493

Table 13: Local optimum with Formula (4.2) ($s = 5, \alpha = 0.2$)

The average waiting time is still too big compared to α in the first four experiments, so it does not really matter that the throughput of type 2 jobs is high. After the fifth experiment, the average waiting times are too small, so one can suspect that better results could be obtained with other thresholds. It will be revealed in the next subsection that this conjecture is indeed true.

It seems that these unifying methods are not appropriate. We need to find the optimal thresholds in a different way.

4.1.2 Optimal threshold by simulation

We know from the law of large numbers that under a fixed threshold policy, the average waiting time converges to its expectation, i.e., to $\mathbb{E}W^q$, almost surely. Hence, if we let a simulation of the system with fixed parameters s, μ_1, μ_2, λ run long enough, then we can hope to experience an average waiting time close to the corresponding $\mathbb{E}W^q_{(c,\delta)}$. With this information in hand, we can compute the optimal threshold for these parameters via the algorithms in Section 3.2.

The only question to decide on is when to run the simulation. We have two possibilities to consider.

The first one can be described as follows.

1. Compute a pro-active estimate for λ_t, i.e., let $\hat{\lambda}_u$ be a forecast of λ_t with $u < t$.

2. Run a simulation of the system with $\hat{\lambda}_u$ as arrival rate from u until t with different thresholds and randomization parameters to get the largest generalized threshold for which the experienced average waiting time is under α.

3. Apply the threshold obtained in step 2 at time t.

The advantage of this procedure is that it can be always used, regardless of the values of the parameters. The disadvantages are that the time available for the simulation might not be sufficient to obtain a proper threshold that is close to the optimum, and anyway, it is not that easy to produce pro-active estimates.

It needs to be mentioned that it is possible to use a dynamic programming algorithm (e.g., value iteration) with $\hat{\lambda}_u$ as an arrival rate, instead of the simulation. That method might converge faster to the expected waiting time than the observed average waiting time in the simulation, but the same problems may still exist even in that case.

If we have some information on the possible values of the arrival rate, e.g., that the values are in an interval $[a, b]$, then a much more promising procedure is possible as follows.

1. $stepsize := 0.01$; (the accuracy of the method)
2. $\lambda := a$;
3. while $\lambda \leq b$ do
4. begin
5. Get the optimal threshold for this λ via simulation.
6. Store the result.
7. $\lambda := \lambda + stepsize$;
8. end;

A table of an arbitrary accuracy can be constructed easily by choosing a small step size properly, and providing enough time for the simulation so that the observed average waiting time can be close enough to the expected average waiting time. An example of such a table can be seen in Table 14.

λ	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
c_{opt}	5	4.67	4.36	4.04	3.89	3.73	3.54	3.34	2.95	2.79	2.52

Table 14: Some optimal thresholds for different values of the arrival rate ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2$)

A very big advantage of this method is that a table constructed once can be used as long as the parameters of the system remain the same, so it is really worth to build a very accurate one. It can also be extended without problems if we experience a λ that is far from the ones included in the table.

After having created a table, we can apply all the methods described in Section 3. We give the results of three methods, namely the local optimum in Table 16, using estimates with the moving average in Table 17, and the results using stochastic approximation with the third update scheme in Table 18.

But first of all, we need to give the results with fixed thresholds in Table 15, because that is the traditional method, and our primary goal is to outperform that if possible.

Threshold	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
0	0.00584	0
1	0.01167	626
2	0.03559	1289
3	0.11747	1958
4	0.38298	2646
5	1.24532	3252

Table 15: Fixed thresholds ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2$)

Choosing $\alpha = 0.2$ as a constraint again, we see that at most $c = 3$ agents should be assigned to outgoing calls. If we allow to use a fixed randomization parameter, then the best fixed threshold policy that obeys the constraint on the average waiting time of incoming calls is the one of threshold level 3.49 (i.e., $c = 3, \delta = 0.51$), producing a throughput of 2049 served type 2 jobs per hour.

Next, let us investigate the local optimum, i.e., the results of a procedure that uses always the optimal threshold corresponding to the actual real parameter. More precisely, it uses the optimal threshold corresponding to the closest value to the real parameter in the table constructed earlier.

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.20717	2551
2	0.19836	2553
3	0.19896	2555

Table 16: Results of computer simulation using locally optimal thresholds ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2$)

These results are much better than the ones that the best fixed threshold policy satisfying the constraint can provide. The throughput of this method on type 2 jobs is about 25% higher than the one of the best allowed fixed threshold policy, while there are no problems with obeying the constraint.

The only problem is that here we used the real parameters of the arrival process that are unknown in reality. Therefore, the results of the following method using the moving average for estimating the parameters might be more interesting, because that is implementable.

Let us see the results in Table 17.

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.19684	2554
2	0.20380	2541
3	0.20296	2557

Table 17: Results using the moving average for estimating the arrival rate ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2$)

The results show that this method works approximately as good as the local optimum. The throughput is high, and this method is reliable enough: the average waiting time stays close to α. Note that the throughput is indeed better than the ones in Table 13. Implementing this method is exactly as easy as in the case of equal service parameters.

However, these results are not better than the local optimum, so it is worth trying to apply the stochastic approximation algorithm that worked so good in the case of $\mu_1 = \mu_2$. The results can be found in Table 18.

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.20006	2661
2	0.20030	2591
3	0.20002	2614

Table 18: Results using stochastic approximation, with update scheme (3.11) ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2, \varepsilon_1 = 1.5, \varepsilon_2 = 0.1$)

It seems that this last method outperforms all the others (including the local optimum) again. It can be observed how little the difference is between the constraint and the experienced average waiting time. The throughput of type 2 jobs is undoubtedly higher in this case than the local optimum. It needs to be stressed that the properties of this method were equally good for other values of μ_1 and μ_2 as well.

Hence, we can conclude that under the given circumstances (i.e., using the arrival rate function of the Charlotte Call Center) a stochastic approximation method seems to be the most appropriate, even in the case of unequal parameters.

4.2 Peaked arrival rate function

The examination of the system with the data of the Charlotte Call Center that we used until now is complete with the case considered in Section 4.1, because that is the most general case that fits in our model.

It might be interesting to see how the estimation methods work if the function of the parameters of the arrival process is of a different shape. The second thing to observe is whether we can obtain the best results with stochastic approximation methods like before.

We will consider a very peaked function, because handling slowly varying, smooth functions is much easier; even the average-type estimate is not that bad in that case. The plot of the function that we used can be seen in Figure 10.

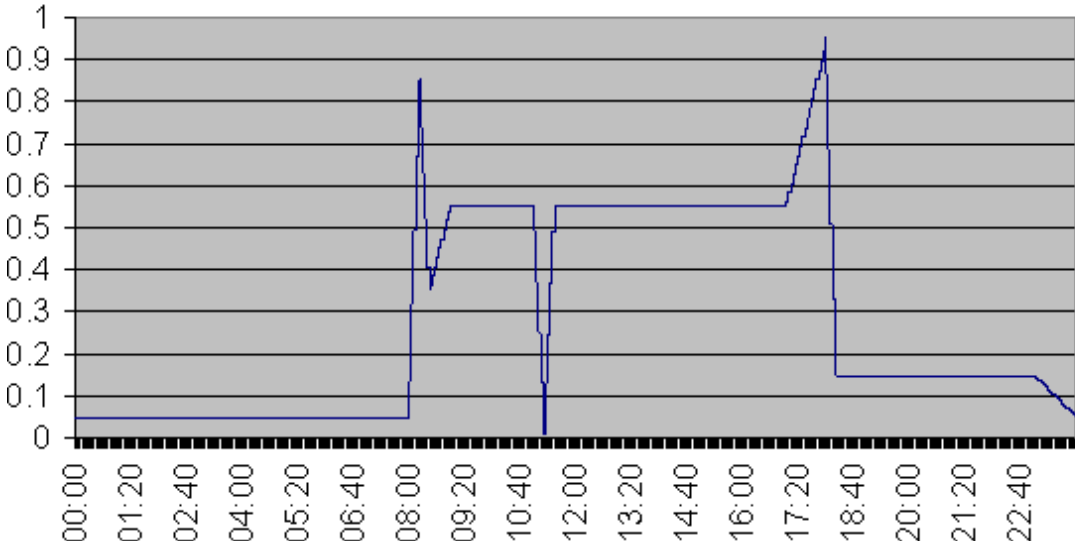

Figure 10: Parameter of the arrival process

First of all, it needs to be examined whether the estimation methods considered before can approximate this function as well. There should be no problems with estimating the parts where the function is constant. But there are sudden changes as well, and handling those may make a distinction between appropriate and non-appropriate methods here.

Because of the peaks, the evaluation of the methods here will be primarily determined by the lags they produce. For example, if the estimate is 30 minutes behind the real function, then it stays constant between 8:00 and 8:30, while there is a high peak in the real function. So, the number of agents assigned to incoming calls will be much lower than needed, hence there will be a serious backlog with regard to the average waiting time.

Not surprisingly, the average-type estimate that uses Formula (3.1) gives just as poor results as in Section 3.3.1 (see Figure 11).

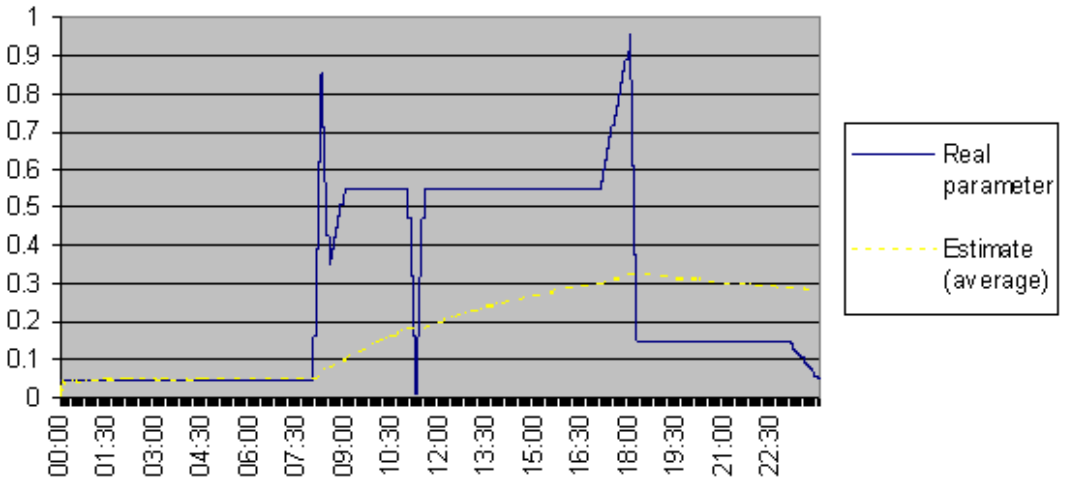

Figure 11: Average-type estimate of λ_t

This estimate is virtually useless, except that it follows the real function until its first change very well.

Let us see the moving average-estimate next. According to the argument about the importance of not producing a lag, we chose a rather small interval length, $l = 400$.

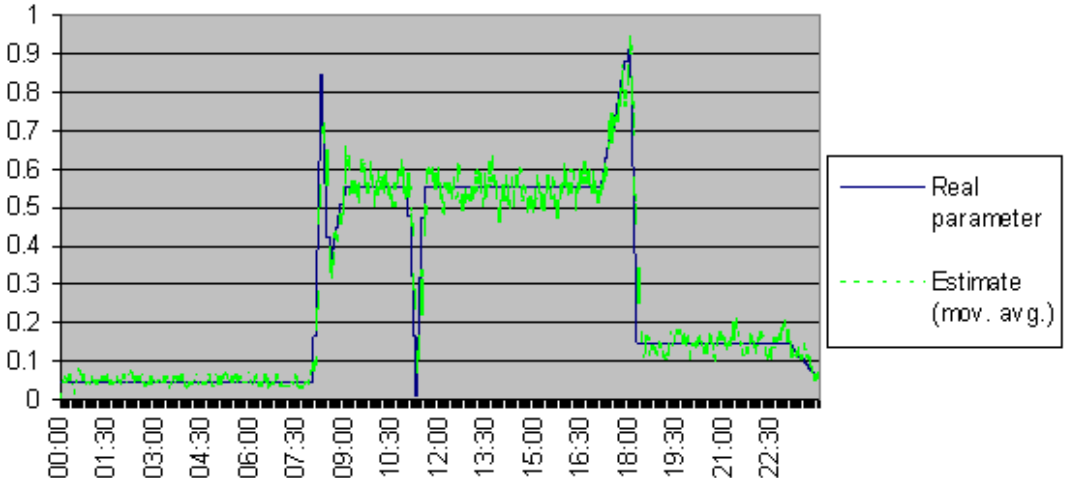

Figure 12: Moving average estimate with interval length $l = 400$

We can see that there is almost no lag indeed, but choosing a small interval length has as a side effect that the fluctuation is quite big.

The exponential smoothing method might help to get an estimate that is still not behind the real function, but has a smaller variance. The estimate with $l = 400$ as a time unit and 2 as smoothing parameter can be seen in Figure 13.

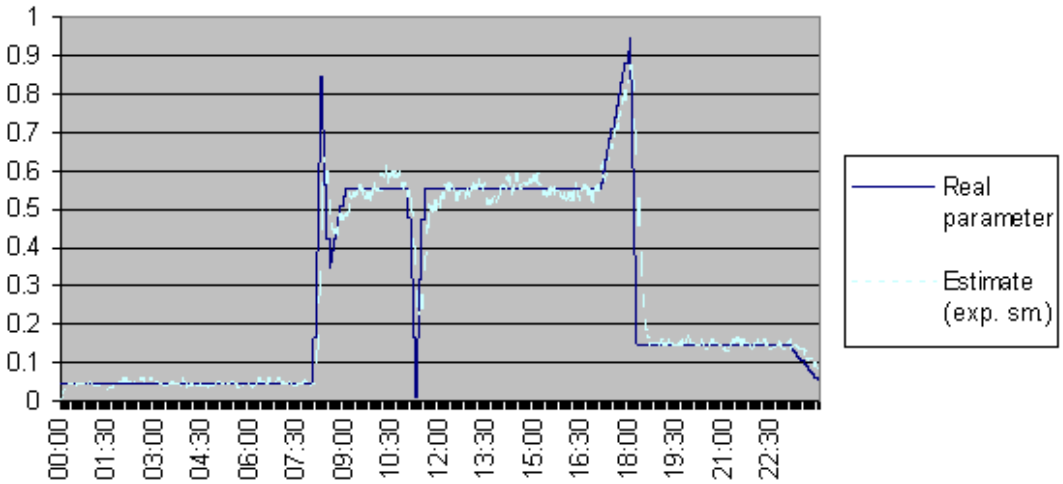

Figure 13: Estimate of λ_t using exponential smoothing

It seems that this curve is worse than the previous one. Although the fluctuation is really not that big, it is still significant, and there is also a not-negligible lag present.

The next method that we consider is linear extrapolation (see Figure 14).

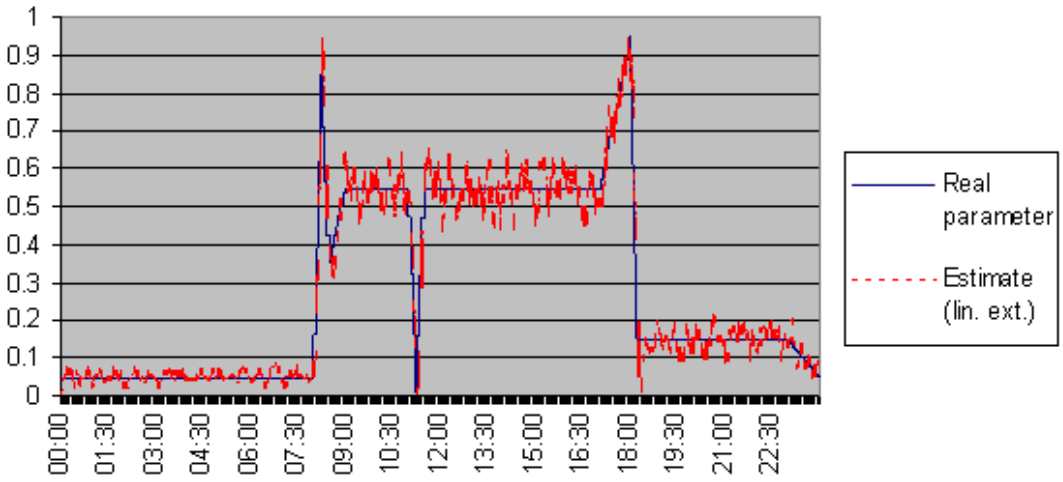

Figure 14: Linear extrapolation-type estimate of λ_t using $n = 4$ points, $l = 250$

This curve is a bit like the one when we used a moving average to estimate the function. We can see that linear extrapolation displays the changes as soon as they have occurred, but the estimate alternates around the real function very frequently.

We have tried all the estimation algorithms that we used in Section 3. Now, we just have to apply them. Naturally, we will consider the unequal parameter case, because that is more general. The parameters for which we will give the results of the computer simulation are always the same: $s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2$.

Let us see first what the traditional method with a fixed threshold gives for all the possible values of the threshold (see Table 19).

Threshold	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
0	0.00344	0
1	0.00889	609
2	0.02568	1261
3	0.08563	1935
4	0.31351	2601
5	1.18851	3193

Table 19: Fixed thresholds ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2$) (peaked arrival rate function)

We can see that if we want the average waiting time to be below 0.2, then we can choose the threshold to be at most $c = 3$. The best allowed generalized threshold would be 3.71, with a throughput of 2149 served type 2 jobs per hour.

The results in Table 20 show that the local optimum is by far better than this.

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.19745	2528
2	0.20027	2515
3	0.20277	2522

Table 20: Results of computer simulation using locally optimal thresholds ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2$) (peaked arrival rate function)

It is important that the experienced average waiting times of the incoming

calls here are almost the same as α, so this method is reliable enough – it respects the constraint at a reasonable extent. The difference between the throughput of the type 2 jobs of the best allowed fixed threshold policy and the local optimum is about 30% again, similar to what is was in the case when we used the arrival rate function of the Charlotte Call Center.

We have just seen that the local optimum gives indeed very good results again. But the most important thing is how the implementable methods work here. Let us start with the moving average (see Table 21).

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.20225	2526
2	0.19899	2528
3	0.19976	2521

Table 21: Results using the moving average for estimating the arrival rate ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2, l = 400$) (peaked arrival rate function)

We can conclude that taking the moving average is again an easily implementable method that provides a high throughput and an average waiting time that is close to the constraint.

The next method is the one that uses exponential smoothing for estimating the parameter (see Table 22).

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.21229	2527
2	0.21245	2522
3	0.20664	2524

Table 22: Results using exponential smoothing for estimating the arrival rate ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2$) (peaked arrival rate function)

These results are unequivocally worse than the ones in Table 21. The throughput is virtually the same, but the average waiting times are higher here, and the difference with which they exceed α is barely acceptable. It seems that the argument at the beginning of this subsection about the lag having a worse effect here than the fluctuations is indeed true. Therefore, we can expect good results from the next method again, as one can remember that the linear extrapolation estimate had almost no lag.

The results of the computer simulation are given in Table 23.

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.20392	2536
2	0.20272	2528
3	0.20266	2536

Table 23: Results using linear extrapolation for estimating the arrival rate ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2$) (peaked arrival rate function)

It is clear by looking at these results, that this estimating algorithm works very good in this case. It gives the highest throughput of type 2 jobs experienced until now. It is even better than the local optimum, albeit the difference between them is not really significant. The average waiting time exceeds α by 1-2%, but such a small difference might be acceptable.

The last procedure that we try here is the stochastic approximation method with update scheme (3.11). We hope that it would work as good as in the case of the other arrival function. Unfortunately, this is not true (see Table 24).

Number of experiment	Average waiting time (seconds)	Throughput (served type 2 jobs per hour)
1	0.19997	2525
2	0.20004	2515
3	0.20015	2529

Table 24: Results using stochastic approximation, with update scheme (3.11) ($s = 5, \mu_1 = 0.5, \mu_2 = 0.2, \alpha = 0.2, \varepsilon_1 = 1.3, \varepsilon_2 = 0.11$) (peaked arrival rate function)

We can be really satisfied with the average waiting times that this method provides – the difference between those and the constraint is very small. The throughput of type 2 jobs is not that bad either. It is almost the same as the local optimum, but this time it is not better than that.

There might exist some parameter values of ε_1 and ε_2 with which the local optimum could be outperformed even in this case, but we have not found any despite trying many different values. But this is still a very good method, and let us mention that there are no problems with implementing it.

5 Conclusions

The problem that we wanted to investigate was how call blending in a call center should be done if the incoming calls arrive according to a non-homogeneous Poisson process. The arrival rate function that we used is constructed according to the data of the Charlotte Call Center.

We considered the equal service parameter case first, and saw that the traditional method used in call centers can be outperformed by using threshold policies with the locally optimal thresholds. These thresholds depend on the rate of the arrival process, but that is unknown. So, it needs to be estimated in applications.

We examined procedures for estimating the time-varying parameter of a non-homogeneous Poisson process. All the methods are based only on the number of arrivals in the process, hence they are model-free, they can be used for different systems as well.

With the proper choice of the parameters, estimates of different types (e.g., pro-active, smooth) can be produced, according to what is needed in the particular case. In this model, it depends primarily on the shape of the function of the arrival process. If there are sudden big changes, then having no lag is of primary importance, while if the function is smooth and slowly varying, then we might allow the estimate to have a little lag to avoid fluctuations.

Then, we tried stochastic approximation methods with a learning effect in order to obtain better results than the local optimum.

The most important observations in the equal service parameter case are the following:

1. The local optimum is much better than using fixed thresholds.

2. Taking the moving average as an estimate is an easily implementable method that outperforms the traditional method by far, but is not better than the local optimum.

3. Using stochastic approximation works even better than the local optimum.

In the general case when $\mu_1 \neq \mu_2$, we can draw the same conclusions.

The first two statements remain true in the case of the peaked function that we examined in Section 4.2, but it seems that the stochastic approximation method with upgrade scheme (3.11) is only as good as the local optimum, not better than that in that case.

There are many parameters to choose in the different estimation algorithms. The optimal choice of those might improve the performance significantly in each particular case. The possibilities in stochastic approximation are also not fully exploited, other update schemes or these with other step sizes might work better in other cases.

We would like to emphasize that all the evaluations of the methods are based on results obtained by computer simulation, therefore they are not necessarily completely accurate. However, it seems to be certain that using one of these procedures instead of the traditional method would result in a remarkable improvement in productivity.

References

[1] Sandjai Bhulai, and Ger Koole:
A queuing model for call blending in call centers.
IEEE Transactions on Automatic Control, 48:1434–1438, 2003.

[2] Peter A.W. Lewis, and Gerald S. Shedler:
Simulation methods for Poisson processes in non-stationary systems.

[3] Noah Gans, Ger Koole, and Avishai Mandelbaum:
Telephone Call Centers: Tutorial, Review and Research Prospects.
Manufacturing & Service Operations Management, 5:79–141, 2003.

[4] Harold J. Kushner, and J. Yang:
Analysis of Adaptive Step Size SA Algorithms for Parameter Tracking, 1994.

[5] Yuliy Baryshnikov, Ed Coffman, Guillaume Pierre, Dan Rubenstein, Mark Squillante, and Teddy Yimwadsana:
Predictability of Web-Server Traffic Congestion, 2005.

[6] Harold J. Kushner, and G. George Yin:
Stochastic Approximation and Recursive Algorithms and Applications (second edition), 2003.

[7] M.L. Puterman:
Markov Decision Processes, 1994.

[8] R.B. Cooper:
Introduction to Queuing Theory, 1981.

[9] Vivek Raghunathan:
Stochastic Approximation.

[10] Herbert Robbins, and Sutton Monro:
A Stochastic Approximation Method.
Annals of Mathematical Statistics, 22:3:400–407, 1951.