SIMULATION AND SELECTION OF EFFICIENT DECISION RULES IN BANK'S MANUAL UNDERWRITING PROCESS

Mikhail Konovalov
Institute of Informatics Problems
of the FRC CSC RAS, Moscow, Russia,
Email: mkonovalov@ipiran.ru

Rostislav Razumchik
Institute of Informatics Problems
of the FRC CSC RAS, Moscow, Russia,
Peoples' Friendship University of Russia,
Moscow, Russia
Email: rrazumchik@ipiran.ru

KEYWORDS

bank's underwriting, optimal rule, scheduling, dispatching, simulation

ABSTRACT

Bank's manual underwriting involves a group of underwriting inspectors, which perform a known set of procedures with the loan applications submitted by the borrowers, in order to determine the risk of providing a loan and eventually approve or disapprove it. Due to the fact that the evaluation process of applications must satisfy quality of service requirements usually set at legislator level and the bank resources are limited, one has to define such dispatching rules, that specify which application must be sent to which inspector and when in such a way that the requirements are met. This paper presents a case study of the application of "computer-aided scheduling" to the new problem of optimal management of applications, which is seen in the bank manual underwriting process. Here it is shown that the problem of optimal distribution of applications between the inspectors in the bank's manual underwriting can be represented as an optimal dispatching problem, commonly encountered in the distributed processing environment. We build the simulation model of the corresponding dispatching system and find best decision rule with the help of computational simulations. The realization of best decision-making is done by finding in a given set of dispatching rules the best one (either static or adaptive) for a given criterion and by estimating its parameters (if needed). By virtue of numerical examples it is shown how the quality of service requirements are met using different dispatching rules.

INTRODUCTION

The problem which is being considered in this paper frequently arises in the bank manual underwriting process and concerns algorithms, which are implemented in the bank information support systems and are used to manage incoming loan applications[1]. In banking, underwriting is the process of approving or denying of a loan, based on the financial information and credit history of the borrower. An appropriate information support system is usually used for the the management of the underwriting process. Up to now there are two types of underwriting systems: automatic and manual. In an automatic underwriting system (credit scoring system), the information from the application is entered by a bank worker into a computer program which determines whether the borrower financially fits the loan conditions. Usually it takes from 5 minutes up to 5 hours to make a decision. If the loan is approved through an automatic system, the bank will proceed with it. If it is not, then the bank either rejects the application or hands it further for the manual underwriting.

In the manual underwriting[2] a bank worker (underwriting inspector) performs a set of procedures instead of a computer program and eventually determines the risk of providing a loan. In cases of high risks he can propose new conditions under which the loan can be provided. These procedures are specific for each bank and can be very different from the procedures run inside the automatic system. For example, they can include sending official requests to the federal state services, checking of the persistent arrears problems, evaluation of financial information, contacting internal bank services (such as security service). It is being reported that in case of manual underwriting it takes from 1 to 10 days to make a decision but this period heavily depends on the bank, application type and size of a loan. Despite the fact that in this case underwriting is done almost manually, a bank needs a supporting information system which will manage applications and track their status. Another goal of the system is to distribute applications among the available underwriting inspectors in order to keep them equally or unequally loaded[3]. But due to the fact that applications must be evaluated within the given time limits (deadlines), which are usually specified at a legislator level, the supporting information system must be able to make such dispatching decisions (i.e. specify which application must be sent to which inspector and when), that al-

[1]Documents that provide the essential financial and other kind of information about the borrower on which the bank bases the decision to lend.

[2]In general the manual underwriting is used for commercial (or business) loans.

[3]The objectives may be very different and usually depend on the bank's goals.

Proceedings 30th European Conference on Modelling and Simulation ©ECMS Thorsten Claus, Frank Herrmann, Michael Manitz, Oliver Rose (Editors)
ISBN: 978-0-9932440-2-5 / ISBN: 978-0-9932440-3-2 (CD)

low the bank to meet the required deadlines. From the given rough description of the manual underwriting process, it is clear that the evaluation process of an application requires a random amount time even if the bank has well-established internal underwriting procedures. This fact in combination with the unpredictability of the application submission times and the requirement to meet the target deadlines makes the dispatching decisions complicated. The situation becomes even rougher as soon as one tries to take into consideration more and more details of the manual underwriting such as skills of the underwriting inspectors, working schedule, priorities of the applications, load balancing etc. It can also turn out that the deadlines and other quality of service (QoS) requirements, that may be specified at a legislator level (or within the bank) in order to increase its competitiveness, cannot be met by the bank. In the latter case the reason may not be in the bank's internal underwriting procedures, but in the deficit of the number of inspectors or in the lack of experienced stuff. The true reason is not that easy to figure out because, for example, the given number of employees cannot cope with the flow of applications because the incoming applications are distributed among them in a wrong way, which can be optimized. It can be seen in practice, that when the bank realizes that it cannot cope with the current flow of applications, the dispatching decisions are taken under manual control of the executive manager, who decides when and who must evaluate which application. Eventually this leads to the accumulation of expired applications and inevitable consequences like penalties. Now most of the banks have an appropriate information support system for the manual underwriting and the bank's main goal is to tune it in a proper way so that the QoS requirements are met. In order to tune the system one has to answer the question: given that the bank understands its internal manual underwriting procedures and has relevant statistical information on how well (quantitatively and qualitatively) these procedures were performed for a certain period of time in the past, what is the best strategy (algorithm) to distribute the incoming applications between underwriting inspectors, which allows one to meet the QoS requirements? Despite a great generality of the question, from our point of view, the answer to it can be given by using computer-aided scheduling techniques and by seeing an analogy between the described information support system and the task/resource allocation problems typically met in the field of the distributed computations. The idea is that firstly one represents the problem of distribution of incoming applications among inspectors under given constraints (inspector skills, application priorities etc.) as a resource allocation problem (or a dispatching problem). Secondly one identifies QoS requirements that need to be met (for example, not more than 10% of overdue applications). Thirdly, one builds the simulation model of the system, which (at least to some extent) can reproduce the values of the chosen performance characteristics provided by the information support system currently operated by the bank. Fourthly, one finds the best possible solution by simulation and generates the improved dispatching rule. Finally, the bank, having implemented the new rule in the information support system, keeps tracking of the QoS requirements. As soon as it detects severe violations, the simulation model is re-built (if there were any changes in the underwriting procedures or staff) and run again with the updated historical data eventually generating the new rule, that is implemented in place of the previous one and used until the next violation.

Even though the idea is simple[4] it has a number of drawbacks. The first step looks to be the most difficult one because of the high level of ambiguity: there are many ways in which manual underwriting procedures can be formalized. Some procedures can be left out of consideration, others cannot and it is hard to determine what the right granularity level is. Another aspect is the performance evaluation of the underwriting procedures. Inter alia this includes the understanding of performance characteristics of underwriting inspectors (how fast one copes with different stages of application evaluation process), estimation of the true times needed to fulfil the internal procedures and external official requests. The latter is not possible without enough historical data. Moreover the estimations of time frames can be done only on the probability basis. Finally, making the simulation results conform to the results achieved in real-life is another challenge which can be done only through trials and errors. Additionally, the need to re-build model each time the underwriting processes are changed, requires such a simulation framework in which models are built in an algorithmic manner. Given that simulation experiments are cheap compared to real-life implementation, trials and errors may have the price that can be paid. Our experience shows that in close cooperation with the bank these difficulties can be overcome at some abstraction level. Even if the abstraction level is high, which means that only basic procedures are being formalized, simulation can be advantageous, because the optimal decision rules may depend exactly on these basic procedures[5]. By building even a rough though consistent simulation model one obtains a basis to judge if the QoS requirements can be met without any changes in the underwriting procedures or staff number.

The case study presented in this paper demonstrates that the proposed idea indeed can be fulfilled. To our knowledge there are no studies on this topic in the literature. Here we demonstrate that the management (including assignment) of applications by inspectors during the manual underwriting process can be seen as a service process in a dispatching system. We give a short description of our algorithmic simulation framework and present the results of the numerical experiments based on syn-

[4]And not new. It is reported to be quite common approach in production scheduling in the field of industrial manufacturing. See, for example, Harmonosky and Robohn (1987).

[5]Of course, this is not always the case. But in the situation when one needs a more or less argumentative improved decision rule, this looks to be a feasible approach.

thetic data, which show how the QoS requirements of the manual underwriting process are met using different dispatching rules. Even though the comparison of the simulation results with the results from real-life experiments is not given here, one can see that in considered environment complex rules may be advantageous to simple ones. However the great increase in the rule's complexity does not lead to the prospective increase in the performance. Thus the appropriate rule is a matter of trade-off.

MANUAL UNDERWRITING PROCESS AS A DISPATCHING PROCESS

On a certain abstraction level one can see that the manual underwriting processes flow within one system, composed of typical objects: applications and inspectors. Typically there are several types of applications, that the bank works with and a pool of inspectors with different qualifications, which indicate the types of applications the inspector is allowed to evaluate. The manual underwriting process itself is just a sequence of actions that an inspector performs with each application. The number and sequence of actions depend on the type of the application and are usually fixed and are rarely changed within the bank. Due to the fact that the evaluation of the application requires contacting internal and external services, inspector is not busy with the evaluation of a single application during all his working time. It can be seen in practice that one inspector is evaluating several applications in parallel. The number of applications that an inspector can evaluate at the same time depends on his qualification level and rules of the bank. Each inspector is busy with the evaluation of the assigned applications strictly according to his working schedule: thus there are periods of unavailability when the applications are postponed until an inspector becomes available. The bank launches the underwriting process for each new application. Thus each application has a start time (when the application is submitted) and a finish time (when the application is approved or disapproved). In order to meet the quality of service requirements the bank has to appropriately control the objects of the system. But there is not too much in the system that can be controlled. For example, the bank cannot control the submission times of new applications; they arrive in stochastic manner. Time frames during which applications are evaluated are not deterministic and can vary significantly due to different reasons such as sudden illness of the inspector or unexpected delay at external service. Most of the unknown values can be estimated only on the probabilistic basis and the bank possesses only one major control option: set the rule according to which arriving applications are distributed between the inspectors.

The analogy between the described system and the dispatching system is apparent. In a dispatching system, each server (underwriting inspector) has its own queue and the task is to assign the arriving jobs (applications) to servers either immediately upon arrival or later, in order to meet the objectives. Each job consists of several tasks (actions which inspector perform on the application), which have to be served in a prescribed sequence. The next subsection contains the detailed description of the dispatching system, which models key aspects of a manual underwriting process.

DESCRIPTION OF THE DISPATCHING SYSTEM

The dispatching system consists of N of servers without a dedicated queue each. There are M types of job flows that arrive at the system. Flows are independent and times between successive arrivals in each flow are i.i.d. random variables. A job within each flow has a deadline, consists of a number of tasks, which have to be served sequentially, one by one and a job is considered to be completed when all tasks it is comprised of are completed. Each task within a job is served in two steps. The first step is the preparation phase[6], which does not require the processing time of the server. The second step is the service phase, when the server processes the task. Preparation and service times are considered to be i.i.d. random variables with cumulative distribution functions (CDF) D^{I} and D^{II}, respectively, which are considered to be known[7]. Thus each task $\mathbb{T}$ can be described by a pair $(\mathrm{D}^{I}, \mathrm{D}^{II})$, which defines how long a task is prepared and then served. Thus in order to introduce different task types one only needs to change CDF D^{I} and/or D^{II}. Jobs within one flow are homogeneous: a job $\mathbb{J}$ in a flow can be described by a set $(d, \mathbb{T}_1, \ldots, \mathbb{T}_k)$, where d is a deadline of a job, k is the total number of tasks within a job and $\mathbb{T}_i$ is the task type[8]. Finally, each flow $\mathbb{F}$ can be defined by a pair $(\mathrm{D}, \mathbb{J})$, where D is the CDF of the inter-arrival times of job $\mathbb{J}$. The joint arrival flow in the system can be described by a set $\mathbb{F} = (\mathbb{F}_1, \ldots, \mathbb{F}_M)$.

Each server can handle a certain number of jobs in parallel and this number is called the server's capacity, which we denote by c. Servers' capacities can be different and are assumed to be known[9]. The server is considered to be busy when it has at least one job in service. Due to the fact that jobs consist of several tasks and each task is performed in two stages we have to specify how the service process goes on. When a job arrives at the server, its first task immediately starts getting prepared. This (preparation) time does not require server's processing power and though the server is considered to be busy it is in fact idle and ready to process other, already prepared tasks (if any are present). When one has finished

[6]The preparation phase is introduced in order to model times when the application is served by internal and external services. During these times the server (i.e. inspector) is not performing any work and is only considered to the busy. Thus it can process other tasks which are already prepared.

[7]We assume that these times can be estimated using historical data.

[8]Here one can observe clear resemblance with practice. If one considers the flow of a certain type of applications, when it is clear that applications are homogeneous, require the same number and type of actions from the inspector.

[9]This corresponds to the fact that an inspector can handle several applications at a time.

the preparation of the task and the server is idle, it starts processing the task and the service time is determined by server's speed r (which can be different for different servers) and task's service time v, and is equal to v/r. In the meanwhile the next task of the job starts getting prepared (which again does not require server's processing power). The server can process only one task at a time and if at the moment when the server becomes idle there are prepared tasks it starts to process a next task[10]. The speeds of the servers are assumed to be known. We also assume that server's availability is periodic[11] and denote by t_0 and t_1 the availability and unavailability periods correspondingly. Note that the preparation of each task does not depend on the server's availability (except for the fact that it can be started only when the server is available). But the server can become unavailable, even if it currently serves the task. The pre-emption of tasks is not allowed. Thus the type of a server can be described by the set $\mathbb{R} = (c, r, t_0, t_1)$ and the specification of the system's resources is given by $\mathbb{R} = (\mathbb{R}_1, \ldots, \mathbb{R}_N)$. We again note that the set of parameters $(\mathbb{F}, \mathbb{R})$ is considered to be known a priori.

CONTROL IN THE SYSTEM AND THE COST FUNCTION

The dynamics of the described dispatching model depends on the dispatching rule for the incoming jobs[12]. As each job is admitted into the system and each server can handle several jobs at a time, then the dispatching rule must specify how the server is assigned for the newly arriving job and how the server (upon service completion) chooses the next task to be served. Here a variety of options exists. For example, the assignment of the job to a server can be based on server's current utilization[13] and the choice of the next task can depend on the total number of prepared tasks and on jobs' deadlines. In order to limit the number of options we assume that the system has a two-level hierarchical structure. All servers are grouped into K disjoint sets (clusters). Number of servers in the i-th cluster is n_i, $i = 1, \ldots, K$, and servers within a cluster may be non-homogeneous[14]. Clearly, $\sum_{i=1}^{K} n_i = N$. These K clusters form the bottom level. The upper level of the system consists of a single entity called dispatcher. Each newly arriving job firstly goes to the dispatcher, which routes it immediately to one of the clusters. When a job arrives at the cluster and there are available servers in the cluster for this job, it is immediately assigned to one of them. Otherwise it is kept in a virtual queue of the cluster until one of the appropriate servers becomes available. Note that once the job has entered a cluster it cannot leave it.

For such a two-level hierarchical structure one can introduce an agent based structure of a dispatching rule. Dispatcher agent A_0 makes decisions whereto route newly arriving jobs. Decisions are made at once. Agent A_i, $i = 1, \ldots, K$, are responsible for assigning jobs within i-th cluster. Specifically the agent A_i consists of two agents $(A'_i; A''_i)$. Agent A'_i decides whereto route the job when it arrives at the i-th cluster. Due to the fact that at that moment all appropriate for the job servers may be busy, the agent may decide to put a job in a queue. Agent A''_i is responsible for assigning jobs, which are waiting in a queue, to servers each time when a server in the i-th cluster becomes available. Because a server may be busy with multiple jobs at a time, then upon service completion of a task there may be several other tasks ready for service. Agents A_{ki} are responsible for choosing the next task when server k in the i-th cluster finishes service of the previous task.

Thus a dispatching rule A consists of $(K+N+1)$ agents and symbolically can be written as

$$A = (A_0; A'; A''; \tilde{A}),$$

where $A' = (A'_1, \ldots, A'_K)$, $A'' = (A''_1, \ldots, A''_K)$. $\tilde{A} = (A_{ki}, i = 1, \ldots, n_K; k = 1, \ldots, K)$.

For the considered dispatching problem it is hardly possible to find analytically an optimal dispatching rule A. Following the common practice, we will use heuristic rules in conjunction with simulation in order to find the best rule in the given set of rules. Here we will present the results for the following common heuristics[15]: uniform random, least loaded first, first-in-first-out. Each dispatching rule A is assumed to be constructed from one or several of these heuristics[16]. In the next section we show the efficiency of different dispatching rules with respect to the cost function which is equal to the mean number of on-time jobs (i.e. mean number of jobs that were served within their deadlines). It is important to note that this cost function is not fair. Indeed it implies that some jobs may not be served at all (in case of high load) or may starve (i.e. may be served with severe deadline violations), which does not influence the average value of the cost function. A number of ways exist to make it fair (for example, one can introduce progressive penalties which grow with the growth of deadline violation times) but we don't consider them here.

OBTAINING THE BEST POSSIBLE SOLUTION

The optimization is based on the statistical simulation techniques. The problem considered in this paper lies in

[10]It means, that we consider only work-conserving disciplines.

[11]This corresponds to working schedules of the inspectors.

[12]This directly corresponds to our assumption that the bank can control the underwriting process only by assigning newly submitting applications to inspectors.

[13]But in the considered problem it is unclear how to uniquely determine server's utilization. This is due to the fact that tasks which comprise a job are served in two stages: preparation phase and service phase. During the preparation phase the server either may process another task (if any) or may be free (though is still considered to be busy) if there are no other already prepared tasks.

[14]That is of different capacities and speeds.

[15]A review of dispatching policies up to 2011 can be found, for example, in Semchedine et al. (2011).

[16]For example, the agent A_0 may route newly arriving jobs either to a random cluster, or least loaded one.

the area of distributed computing and evolutionary computation. Not need to mention that in this area there are numerous research papers devoted to solving dynamic optimization problems using simulation which resulted in a variety of methods. Among the latest ones which include reviews on the topic one can refer, for example, to Kolodziej (2012); Doroudi et al. (2014); Broberg et al. (2006); Harchol-Balter et al. (1999). Below we describe in short our approach and highlight its peculiar features (for the details one can refer to Konovalov (2007)).

The solution of the optimization problem is found by building the simulation model and using adaptive optimization algorithms on simulated trajectories. For the first step we use our flexible simulation framework for job allocation problems in distributed processing systems. It allows one to build logical processes governing jobs' handling, is algorithmic and, to some extent, allows assembly of complex models from simple ones. Its formal description is based on the concept of communicating sequential processes introduced by C.A.R. Hoare (see Konovalov and Razumchik (2014); Konovalov (2014, 2007)).

The simulation model allows one to obtain the value of the cost function for any dispatching rule A. By carrying out series of experiments with different dispatching rules one can find the one which leads to the best value of the cost function. But there are reasons which may make this exhaustive search inapplicable. One reason is the time needed to obtain the stable value of the cost function. This time depends on the rate of convergence in the strong law of large numbers and may take too long. Exhaustive search becomes also difficult whenever the number of dispatching rules under test is infinite. We try to overcome these kind of difficulties by using adaptive search algorithms for partially observable Markov decision processes. Such algorithms use a single trajectory for each dispatching rule and tune its parameters in such a way that the probability of better values of the cost function is increased. For the detailed description of the algorithms used one can refer to Sragovich (2007); Konovalov (2007).

NUMERICAL EXAMPLE

Consider a bank which has 5 different business lines and thus the bank's manual underwriting process must handle 5 different types of loan applications. The specification of each application flow is given in Table 3. One can see that each flow is considered to be Poisson and on average the bank has 26 applications per day. We assume that the bank identifies 5 different actions (i.e. task types) that can be performed by an inspector when evaluating an application. The assumptions concerning the task types, including preparation and processing times, are stated in Table 1. We assume that each application type (i.e. job type) consists of 3 tasks and jobs differ from one another only by types of tasks they consist of (see Table 2). A total of 55 underwriting inspectors work in the bank and have different qualifications[17]. Bank ranges inspectors by granting each one a certain category. We assume that there are 5 different categories (from 1 to 5), with # 1 denoting the highest one. In practice the category indicates which types of applications are available to the inspector. Usually the higher category of the inspector, the more important applications is allowed to evaluate. Within one category all inspectors are assumed to behave identically. The specification of each of the inspector's category is given in Table 4. One can see that, for example, inspectors belonging to the 1st category are able to handle 4 applications at a time, have 720 working minutes (12 hours) which are succeeded by 720 off-work minutes.

We also introduce an assumption that all inspectors are grouped into clusters (or teams). As mentioned in the previous section this artificial hierarchical view is introduced purely in order to make a decision process a little simpler. Table 5 shows the specification of each cluster.

Table 1: Specification of the task types

Task type	Time to prepare, D^I $D^I \sim Pareto(x_{min}, a)$	Service time, D^{II} $D^{II} \sim N(m, \sigma)$
$\mathbb{T}_1$	$x_{min} = 800, a = 6$ mean=960	$m = 960, \sigma = 20$
$\mathbb{T}_2$	$x_{min} = 480, a = 3$ mean=720	$m = 480, \sigma = 20$
$\mathbb{T}_3$	$x_{min} = 320, a = 3$ mean=480	$m = 240, \sigma = 10$
$\mathbb{T}_4$	$x_{min} = 160, a = 3$ mean=240	$m = 480, \sigma = 1$
$\mathbb{T}_5$	$x_{min} = 80, a = 3$ mean=120	$m = 120, \sigma = 1$

Table 2: Specification of the job (application) types

Job type	Types of task	deadline, d in min.
$\mathbb{J}_1$	$\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3$	8640
$\mathbb{J}_2$	$\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_5$	7200
$\mathbb{J}_3$	$\mathbb{T}_1, \mathbb{T}_3, \mathbb{T}_4$	5760
$\mathbb{J}_4$	$\mathbb{T}_2, \mathbb{T}_3, \mathbb{T}_4$	5760
$\mathbb{J}_5$	$\mathbb{T}_2, \mathbb{T}_3, \mathbb{T}_5$	4320

The cost function under consideration is the mean number of jobs served within their deadlines. We define several dispatching rules (see Table 6) and look for the one, which allows us to make the value of the cost function as high as possible. In Fig. 1 and Fig. 2. one can see the values of the cost function and corresponding values of the mean sojourn time (i.e. mean evaluation time for an arbitrary application) for each of the dispatching rules.

One can see from the figures that the best result one managed to achieve is little more than 86% of jobs served on time (dispatching rule A_5). But the dispatching rule A_1, which is much easier to implement, is worse only by less than 1%. Moreover the rule A_4, which does not take

[17]For example, security classification, experience with stock market clients, municipalities.

Table 3: Specification of the flow (application) types. Times between successive arrivals for each flow are exponential

Flow (application) type	Rate	Job (application) type	Suited servers (inspectors)
$\mathbb{F}_1$	$0,006$ ≈ 9 per day	$\mathbb{J}_1$	$\mathbb{R}_2, \mathbb{R}_4$
$\mathbb{F}_2$	$0,002$ ≈ 3 per day	$\mathbb{J}_2$	$\mathbb{R}_1$,
$\mathbb{F}_3$	$0,003$ ≈ 5 per day	$\mathbb{J}_3$	$\mathbb{R}_1, \mathbb{R}_3$
$\mathbb{F}_4$	$0,001$ ≈ 2 per day	$\mathbb{J}_4$	$\mathbb{R}_4$
$\mathbb{F}_5$	$0,005$ ≈ 7 per day	$\mathbb{J}_5$	$\mathbb{R}_2, \mathbb{R}_5$

Table 4: Specification of the servers (inspectors)

Server (inspector) type	Capacity, c	Speed, r	Schedule (on/off)	Total number
$\mathbb{R}_1$	4	3	720/720	5
$\mathbb{R}_2$	3	2.5	600/840	10
$\mathbb{R}_3$	3	2	600/840	10
$\mathbb{R}_4$	2	1.5	600/840	10
$\mathbb{R}_5$	2	1	480/960	20

Table 5: Specification of the clusters (teams)

	Number of servers (inspectors) of type				
	$\mathbb{R}_1$	$\mathbb{R}_2$	$\mathbb{R}_3$	$\mathbb{R}_4$	$\mathbb{R}_5$
Cluster 1	1	2	2	2	4
Cluster 2	1	3	4	0	0
Cluster 3	1	3	2	3	4
Cluster 4	1	1	2	3	2
Cluster 5	1	1	0	2	10

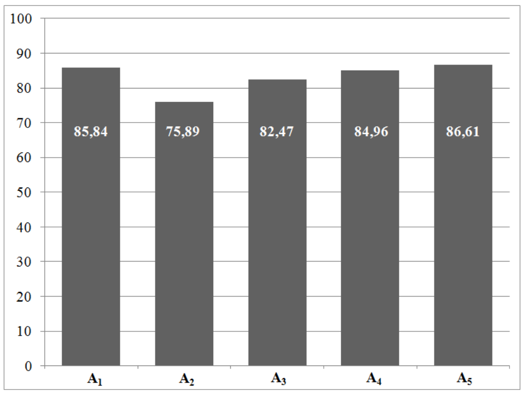

Figure 1: Percent of the jobs served within a deadline for each dispatching rule

into account type of incoming job, is worse by less than 2% than the best rule A_5. Finally the simplest rule with random choice is worse by only 10% (approx. 10 hours). Here one can see that the complication of the dispatching rule does not lead to significant increase in the value of the cost function but may require too much for implementation (see, for example, Konovalov and Razumchik

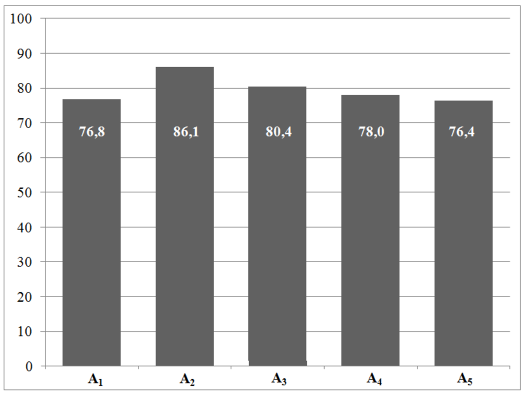

Figure 2: Mean evaluation time of an application (in hours) for each dispatching rule

(2014)). A simple heuristic rule (for example, decision based only on the number of applications) may lead to very good performance. As our experiments indicate the gain of the dispatching rule is very sensitive to the structure of the system (i.e. number of inspectors, number and types of applications etc.) and sometimes a change of a dispatching rule may lead to 20-30% performance increase.

SUMMARY

Here one has demonstrated that the bank's manual underwriting process can be played back in a large-scale dispatching system, which is a common mathematical model for various distributed processing systems. The goal of such modelling is to select a decision rule under which the bank's manual underwriting process will perform at a given level of quality of service. The idea, which was utilized to select an efficient decision rule, is well-known: perform search in a given set of heuristic rules using simulation. Due to the complexity of the system it is analytically intractable and this idea is apparently the only one which allows one to obtain an adequate solution. The special simulation framework and algorithms that we have used[18] allowed us to carry simulation experiments with fairly large-scale systems (500 servers/inspectors) on a stand-alone PC. It is worth noticing that even a simple model of the manual underwriting process may be of an advantage for a bank because it allows to select an appropriate decision rules based on something more that intuition and expert's opinion. Generally speaking it is possible to build an adequate model of the bank's manual underwriting process, which will take into account not only those basic properties mentioned in this paper, but also many specific ones: skills of the inspectors, restricted access to applications, occasional events (illnesses), load balancing. Such model will not require much more computational resources and thus will not incur huge extra cost from the bank. Though in this paper the control is restricted only to the most

[18]Reader can refer to Konovalov and Razumchik (2014) and Konovalov (2007) for more details.

Table 6: Specification of the decision rules used

Decision rule	Rule for A_0	Rule for A'_k	Rule for A''_k	Rule for A_{ki}
A_1	minimum load per flow type[a]	minimum load per flow	FIFO	FIFO
A_2	random	random	random	random
A_3	minimum number of jobs	minimum number of jobs	FIFO	FIFO
A_4	minimum mean backlog[b]	minimum mean backlog	FIFO	FIFO
A_5	minimum load per flow type	minimum mean backlog with thresholds[c]	FIFO	FIFO

[a]This value is calculated as the mean total service time of all jobs in a cluster of the same type as the incoming one.

[b]This value is calculated as the mean total service time of all jobs in a cluster.

[c]This value is calculated as the product (mean total service time of all jobs in a cluster of the same type as the incoming one)× (threshold value). This is the analogue of the threshold policy used, for example, in Hyytia (2013).

common case – routing of applications, – one can also speculate whether the performance of the manual underwriting process depends on the staff structure (i.e. the number and type of inspectors and teams). Our experiments show that if one has control over this component as well, the performance of the underwriting processes can be improved even more.

Acknowledgements This work was supported by the Russian Foundation for Basic Research (grant 15-07-03406).

REFERENCES

Haruhiko, S., Hiroaki, Sa. 2013. Online Scheduling in Manufacturing. A Cumulative Delay Approach. Springer-Verlag London.

Min Hee Kim, Yeong-Dae Kim. 1994. Simulation-based real-time scheduling in a flexible manufacturing system. Journal of Manufacturing Systems. Vol. 13. Issue 2. Pp.85–93.

Harmonosky, C.M., Robohn, S. F. 1991. Real-time scheduling in computer integrated manufacturing: a review of recent research. International Journal of Computer Integrated Manufacturing. Vol. 4. No. 6. Pp. 331–340.

Konovalov, M., Razumchik, R. 2014. Simulation Of Task Distribution In Parallel Processing Systems. Proceedings of the 6th International Congress on Ultra Modern Telecommunications and Control Systems. Pp. 657–663.

Konovalov, M. G. 2014. Building a simulation model for solving scheduling problems of computing resources. Systems and Means of Informatics. Vol. 24. No. 4. Pp. 45–62. (in Russian)

Konovalov, M. G. 2007. Methods of Acaptive Information Processing and Their Applications. Moscow: IPI RAN. (in Russian)

Sragovich V.G. 2005. Mathematical Theory Of Adaptive Control. Singapore: World Scientific.

Hyytia, E. 2013. Optimal Routing of Fixed Size Jobs to Two Parallel Servers. INFOR: Information Systems and Operational Research. Vol. 51. No. 4. Pp. 215–224.

Konovalov M. G., Razumchik R. V. 2015. Approximate optimization of resource allocation strategy: the case of bank underwriting system. Systems and Means of Informatics. Vol. 25. No. 4. Pp. 31–51. (in Russian)

Semchedine, F., Bouallouche-Medjkoune, L., Aissani, D. 2011. Review: Task assignment policies in distributed server systems: A survey. J. Netw. Comput. Appl. Vol. 34. No. 4. Pp. 1123–1130.

Kolodziej, J. 2012. Evolutionary Hierarchical Multi-Criteria Metaheuristics for Schedulingin Large-Scale Grid Systems. Studies in Computational Intelligence Series. Vol. 419. Berlin-Heidelberg: Springer.

Doroudi, S., Hyyti0Ł1, E., Harchol-Balter, M. 2014. Value Driven Load Balancing . Performance Evaluation. Vol. 79. Pp. 306–327.

Broberg, J., Tari, Z., Zeephongsekul, P. 2006. Task assignment with work-conserving migration. Journal of Parallel Computing. Vol. 32. Pp. 808–830.

Harchol-Balter, M., Crovella, M., Murta, C. 1999. On Choosing a Task Assignment Policy for a Distributed Server System. Journal of Parallel and Distributed Computing. Vol. 59. Issue 2. Pp. 204–228.

AUTHOR BIOGRAPHIES

MIKHAIL KONOVALOV is a Doctor of Sciences in Technics and holds position of the principal scientist at Information Technologies Department at Institute of Informatics Problems of the Federal Research Center ”Computer Science and Control” of the Russian Academy of Sciences. His research activities are focused on adaptive control of random sequences, modelling and simulation of complex systems. His email address is `mkonovalov@ipiran.ru`.

ROSTISLAV RAZUMCHIK received his Ph.D. degree in Physics and Mathematics in 2011. Since then, he has worked as a senior research fellow at Institute of Informatics Problems of the Federal Research Center ”Computer Science and Control” of the Russian Academy of Sciences (FRC CSC RAS). Currently he holds the position of Head of the Information and Telecommunication System Modelling section at the FRC CSC RAS and associate professor position at Peoples’ Friendship University of Russia. His current research activities are focused on queueing theory and its applications for performance evaluation of stochastic systems. His email address is `rrazumchik@ipiran.ru`