Bulletin of Electrical Engineering and Informatics
Vol. 9, No. 3, June 2020, pp. 1106~1120
ISSN: 2302-9285, DOI: 10.11591/eei.v9i3.1714

Using queuing theory to describe adaptive mathematical models of computing systems with resource virtualization and its verification using a virtual server with a configuration similar to the configuration of a given model

Alexey I. Martyshkin[1], Dmitry V. Pashchenko[2], Dmitry A. Trokoz [3], Mihail P. Sinev[4], Boris L. Svistunov[5]
[1,3]Department of Computational Machines and Systems, Penza State Technological University, Rusia
[2]Penza State Technological University, Rusia
[4]Department of Computer Science, Penza State University, Rusia
[5]Department of Mathematics and Physics, Penza State Technological University, Rusia

Article Info

Article history:

Received Aug 21, 2019
Revised Nov 12, 2019
Accepted Feb 19, 2020

Keywords:

Adaptive model
Closed queueing network
Computational system
Mathematical modeling
Natural resource virtualization
Verification
Virtual server

ABSTRACT

The article describes the issues of preparation and verification of mathematical models of computing systems with resource virtualization. The object of this study is to verify of mathematical models of computer systems with virtualization experimentally by creating a virtual server on the host platform and monitoring its characteristics under load. Known models cannot be applied to the aircraft with virtualization, because they do not allow a comprehensive analysis to determine the most effective option for the implementation of the initial allocation of resources and its optimization for a specific sphere and task of use. The article for the study used a closed queueing network. Simple models for the analysis of various structures of computer systems are experimentally obtained. To implement the properties of adaptability in the models, triggers are used that monitor and adjust the power of the processing channel in individual Queuing systems, depending on the specified conditions. Experiments prove the obtained results reliable and usable as a flexible tool for studying the virtualization properties when structuring computing systems. This knowledge could be of use for businesses interested in optimizing the server configuration for their IT infrastructure.

Corresponding Author:

Alexey I. Martyshkin,
Department of Computational Machines and Systems,
Penza State Technological University,
440039, Russia, Penza, 1/11 Baydukova proyezd/Gagarina ul, 1/11, Russia.
Email: Alexey314@yandex.ru

1. INTRODUCTION

Data storage and processing infrastructure is one of the crucial components of corporate IT systems; its effectiveness is fundamental to the business performance in a dynamic and competitive market, which is why computing systems (CS) and data storage systems of today shall meet stringent requirements. As such, they must be able to adapt to rapidly changing tasks and objectives; to guarantee the required application performance; to be have necessary scalability with an option to increase resources in-service; to minimize downtime due to failures or maintenance; to be easy to use and maintain. The most efficient way to meet such requirements is to use virtualized CS; at the operating system (OS) level, this technology uses up

to 20% of the server CPU capacity. However, this research aims at describing the creation of efficient CS models, which is why such models shall be based on natural virtualization that uses more efficient software/hardware-level mechanisms. Based on this, the following problems are relevant today: creating models of virtualized CS; model implementation and application feasibility testing, e.g. using virtual servers to test and develop software, to set up a remote office, or to rent out as a basis for outsourcing in computing, etc.

2. METHODOLOGY

This is generally an exploratory paper. While studying the subject matter, the authors hereof have analyzed a bulk of literature [1-10] to find uncovered or unresolved issues, such as using virtual servers to study and verify models of virtualized systems. Some issues relating to the possibility of creating and verifying a mathematical virtualized-CS model are not properly covered in papers; however, [11-14] address the most problematic issues in part. The goal hereof is to analyze the existing CS that use natural virtualization to describe how models or their implementations (virtual servers) could be used, to study them, and to obtain the results of using a virtual server to verify a virtualized-CS model; the characteristics and models of such server are detailed in [15]. Another goal is to develop a method for making virtualization-based models of adaptive CS. These issues are relevant today in view of global computerization and nearly universal use of big (and various) data and virtual servers. To attain this goal, the following must be addressed: describe, and prepare the source data of, virtualized-CS models for different classes of tasks; adapt the mathematics behind the queueing theory to computing virtualized-CS models, i.e. to verifying such models by means of virtual servers; develop a method for constructing and evaluating adaptive-system models. The research methods used herein are based on the queueing theory as well as on mathematical statistics and experimental model verification.

Virtualization means a variety of methods for abstracting from various physical computational resources (CR). Virtualization tools can represent a single physical resource as a set of separate logically independent resources (logical servers) to isolate applications from each other; conversely, virtualization can combine separate physical resources within a heterogeneous structure, be it servers or drives, into a single logical resource. CPU virtualization is possible in theory as substantiated by the Church-Turing thesis [16-19]. The thesis is essentially about computer simulation of a Turing machine (an abstract computing machine), which is assumed possible; the assumption means that as tools for handling algorithmic problems, all computers are equivalent regardless of their implementation. The thesis is not a proven theorem; nevertheless, it suggests that any computing environment can be simulated by another such environment. Important theoretical research into CPU virtualization was carried out by Gerald Popek and Robert Goldberg in the form of three virtualization requirements [1]: equivalence; resource management; efficiency.

Server efficiency is very low, especially in the case of x86 servers; its commonly recognized level is about 5% to 15%. Such efficiency largely depends on the coherence of CPU and server architecture with the operating system. If that coherence is good, as is the case of the RISC/Unix combination, server efficiency may reach 25-30% or above [19-25]. Virtualization can raise this figure to above 85% while improving the reliability, scalability, and other characteristics critical for data centers; besides, it helps save the costs of hardware, support, and administration. The existing models are not applicable to virtualized CS as they cannot run comprehensive analysis to find the most efficient way of initial resource distribution and to optimize such distribution for a particular application while CS is running. Simulation models are common; they simulate the behavior of a real system by introducing special conditions and lags that configure the sequence, in which the system components transition from one state to another. One important advantage simulation model has over analytical models is that a simulation model could potentially be made even closer to the simulated object by injecting additional complications. However, it should be borne in mind that complex simulation models require substantial CR to run, which means that such models are only advisable if analytical methods are not suitable.

Literature review shows that Russian and international researchers mostly use less intensive analytical methods suitable for parametric analysis and optimization. The time characteristics of systems can be assessed by the queueing theory. To produce the estimated ratios that comprise mathematical models, analytical methods require constrains and assumptions that limit their applicability. Thus, the models proposed by L. Kleinrock and M. Schwartz [21, 22] consider a message-switching communication network consisting of M channels and N switching nodes. The mathematical model uses the following assumptions: all channels and all switching nodes are noiseless and absolutely reliable; switching-node processing time is zero; the transmitting end of a channel can queue messages in an unlimited memory; the traffic the communication network receives from external sources (e.g. from host machines) forms a Poisson process; for many analytical relations, the exclusive path is known for each transmitter-receiver pair; for some problems, the probability p(j,k)

of transition from the jth node to the kth node is introduced; message lengths are independent and distributed by an exponential law. These constrains and assumptions can be used to find the time ti a message stays in the network; the communication channel load factors r(r,v), and the queue lengths li. They also help address the issues of efficient design. In [21], L. Kleinrock focuses on three problems: configuring the channel throughput; configuring the distribution of streams in channels; and selecting the network topology. These are single-attribute problems that minimize the mean messaging latency in the communication network while keeping the costs within the required limits. Methods developed and summarized in [23] consider packet-switching networks that are studied as bipolar multiphase queueing systems. These methods use the following assumptions and constraints: the distribution of any random variable is assumed exponential except the third queueing phase, where the service time distribution is deemed regular; the specific subscriber load at subscriber terminals and computers is deemed uniformly distributed network-side; the message queue discipline is FIFO; the time to establish a logical connection is included in the switching time; the queueing system is non-priority; while transmitted over the network, messages age at the specified rate. The basic criteria of evaluating a data transmission network are usually the probability a message will be delivered in time; and the mean delivery time.

Despite the well-elaborated nature of the existing approaches, some of which have evolved into engineering methods, these models have one significant disadvantage: they cannot comprehensively consider both the intra-model information flows and changes in the components of the model itself due to random factors such as hardware failures or CS reconfigurations, which are typical for naturally virtualized systems. Consider the mathematical basis, i.e. the closed queueing networks (CQN)-based calculation method [6-9, 24, 25] chosen because such networks are used to represent processes occurring in CS with limited number of requests; the limitation is due to an inherent limitation, in this case, the limited number of CPUs available to the CS. For instance, multiprocessor systems (MPS) can only connect a limited number of CPUs to a shared bus. In case of virtualization, we have a pool of CPUs, RAM, and input-output adapters.

On the other hand, one example of a QN is the simplest multi-program computer, where the finite number N of programs corresponding to a multiprogramming level will turn to one of the M CPUs, i.e. be processed by M CPUs at the probability $P_i(i=\overline{1,M})$. The mathematics behind the CQN is analyzed and summarized below. When running, each of the M CPUs requests the hypervisor to grant access to a resource, i.e. RAM, an external storage (ES), or an I/O adapter. While the hypervisor grants such access to a CPU, others process data from their caches or local RAM, or wait for a similar access permission; as such, they do not generate new queries to the hypervisor until the running CPU frees the resource requested by another CPU; in the case of time-sharing, such suspension lasts until the running CPU's allocated cycle is over. Figure 1 shows the general information-flow model of a virtual server with a limited number of requests. In a closed model, requests come from the system S_0 that contains M channels $T_1,...,T_M$ and displaying the CPUs as they function in an MPS. The parameter ϑ_0 of the model equals the mean time the CPU spends to analyze the results of processing its preceding request [12-14].

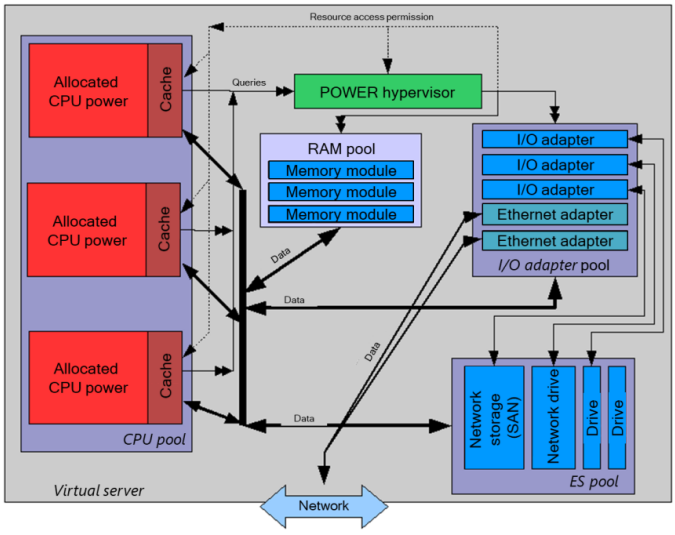

Figure 1. Information-flow model of a virtualization server

In this model, the number of circulating requests equals the number of QN channels S_0, hence no queue. The rate λ_0 at which requests come from the system S_0 to other systems $S_1,...,S_n$ depends on the number of requests in the system S_0.

$$\lambda_0 = \left(M - \sum_{j=1}^{n} M_j \right) \Big/ \vartheta_0 \quad (1)$$

where M_j is the number of requests in the j*th* queueing system(QS) of the network.

Given equal intensity of the input/output request streams S_0, the rate λ_0 will determine the performance of the simulated MPS, i.e. the mean number of CPU queries processed by the hypervisor per unit of time. Consider a CQN with exponentially distributed request processing time in each of the systems $S_j \quad (j=1,...,n)$. For each of the network's systems, define the parameters: K_j is the number of channels; ϑ_j is the mean channel-specific request processing time; α_j is the transfer factor. Another known variable is the number M of circulating requests. The parameters K_j, ϑ_j, α_j, and M are source data for calculating the network's steady state, in particular the probabilities of its states, in terms of which all other characteristics are given [5].

Find the expression for the loads ρ_j of the systems S_j. For a single-channel QS, a load is a difference between 1 and the probability that this QN is idle. The probability that the system S_j has exactly r requests while the requests of other systems are distributed in any possible combinations is written as $\Pr(M_j = r) = \sum_{M_j = r} \Pr(M_1,...,M_n)$. Then the probability that a single-channel system does not have any current requests is $\Pr(M_j = 0) = \sum_{M_j = 0} \Pr(M_1,...,M_n)$. Therefore,

$$\rho_j = 1 - \Pr(M_j = 0) = 1 - \sum_{M_j = 0} \Pr(M_1,...,M_n) \quad (2)$$

To find the load of a channel in a multichannel system, first find the mean number of idle channels,

$$K_j - k_j = \sum_{r=0}^{K_j - 1} (K_j - r) \Pr(M_j = r) \quad (3)$$

where K_j is the number of channels in the system; k_j is the mean number of busy channels; $\Pr(M_j = r)$ is the total probability of all states from the set $A(M,n)$, for which $M_j = r$. The load of each channel in a multichannel system S_j is defined as the difference between 1 and the mean number of idle channels from the total number of channels

$$\rho_j = 1 - \frac{K_j - k_j}{K_j} = 1 - \sum_{r=0}^{K_j - 1} \frac{K_j - r}{K_j} \Pr(M_j = r) \quad (4)$$

From (3), find the mean number of busy channels in the system S_j.

$$k_j = K_j - \sum_{r=0}^{K_j - 1} (K_j - r) \Pr(M_j = r) \quad (5)$$

Apparently, $\rho_j = k_j / K_j$. Given that for a multichannel QS $k_j = \lambda_j \vartheta_j$, obtain the expression for the incoming stream rate.

$$\lambda_j = k_j / \vartheta_j \tag{6}$$

To calculate the mean number of channels m_j and the mean number of channels l_j (incoming and waiting in the system S_j), use the expressions,

$$m_j = \sum_{r=0}^{M} r \Pr\left(M_j = r\right) \tag{7}$$

$$l_j = \sum_{r=K_j+1}^{M} \left(r - K_j\right) \Pr\left(M_j = r\right) \tag{8}$$

The mean stay time u_j and the mean waiting time ω_j in the systems $S_j \left(j = 1,...,n\right)$ equal $u_j = m_j / \lambda_j$ and $\omega_j = l_j / \lambda_j$, respectively, where λ_j, m_j, and l_j are found from (6) to (8). Refer to the mean time interval between two consecutive exits of a request from the system S_j as the system cycle time. The mean stay time of a request in the system S_i over its single stay in the system S_j equals $\left(\lambda_i / \lambda_j\right) u_j$. The mean cycle time U can be found by summing these values for all the systems in the network $U_j = \sum_{j=1}^{n} \left(\lambda_i / \lambda_j\right) u_j$. Given that the mean stay time of a request in the system S_i $u_i = m_i / \lambda_i$, get

$$U_j = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_j} \cdot \frac{m_i}{\lambda_i} = \frac{1}{\lambda_j} \sum_{i=1}^{n} m_i = \frac{M}{\lambda_j} \tag{9}$$

Given that we are expected to model virtualized CS, it is safe to say that the obtained models will depend on, and self-adapt to, the load on a rule-based principle that will make use of the data received while computing a model. It is also worth noting that the mean queue length and performance will depend on the QS service rate. Mean queue length is a monotonically decreasing (increasing) function, while the performance is a monotonically increasing (decreasing) function of the service rate.

3. INPUT DATA FOR MODELS

The initial number of QS channels is set forth in the assumed number of dedicated or shared virtual devices for each specific virtual server. The concept of QS channel number is replaced with the concept of allocated processing power, given as a percentage or proportion of a whole processing channel/CPU. For instance, two channels can be assigned to a CPU, which will mean the virtual server has two virtual CPUs; if the model is assigned two QS CPUs, it means that the virtual server has two CPUs, each of which can be further divided into virtual CPUs as channels. On the other hand, a CPU QS can be allocated specific processing power starting from 0.1 and incrementing at 0.01, which corresponds to similar virtualization properties.

The meaning devices are set in the model in a similar way, i.e. if an input-output device is not dedicated to a particular virtual server, or if a virtual server uses a shared device provided by the virtual I/O server. Such a device could be a network adapter, the I/O adapter to access ES of different types. Before models could be built for studies, consider the structural diagram of a host platform used for creating virtual servers as shown in Figure 2. It shows all the components of future virtual servers; unused resources are pooled together to form the CPU pool, the RAM pool, the ES pool, etc.; the diagram also shows the primary component of resource virtualization, a POWER hypervisor, and zoomed-in diagrams of model virtual servers.

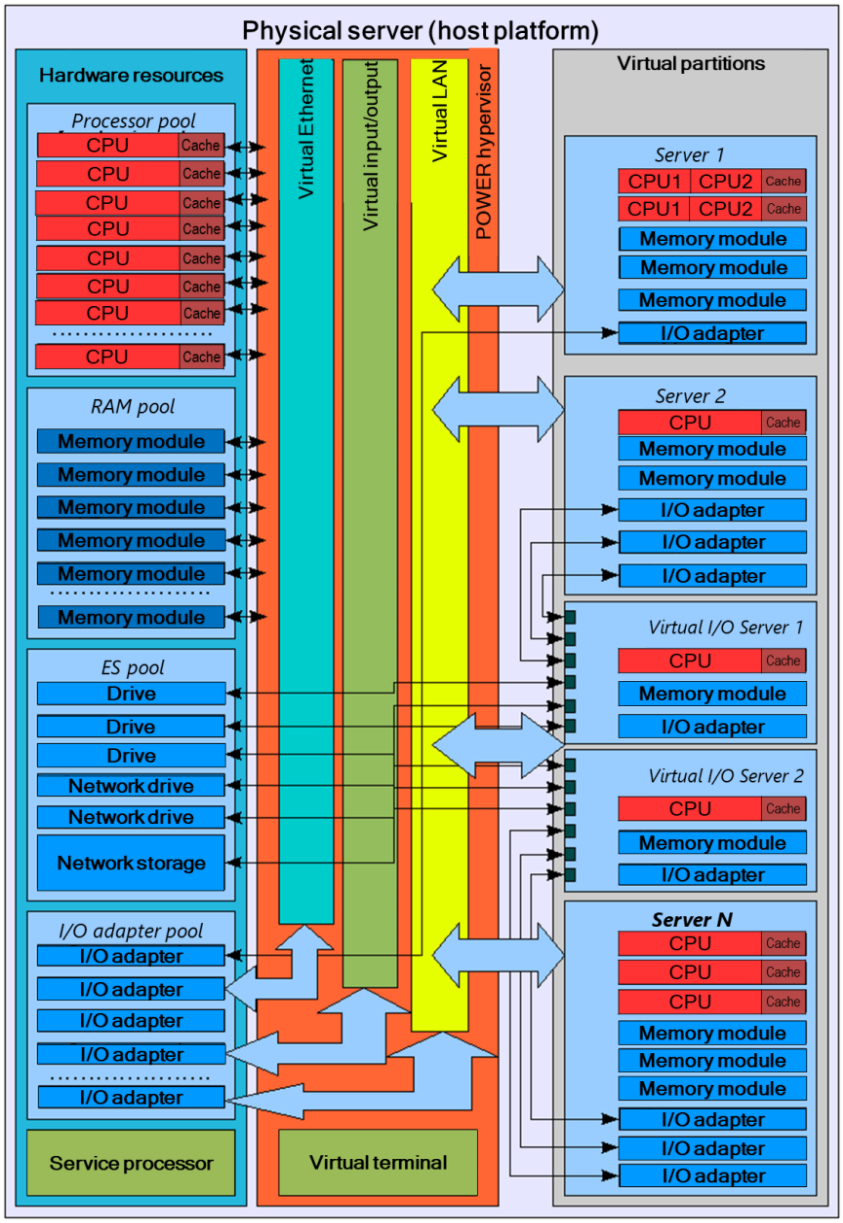

Figure 2. Structural diagram of a host platform

Figure 3 shows a part of Figure 2 (virtual servers) in detail; it also demonstrates virtual input-output servers with CP distribution and logical links between the system components. The diagram of a non-virtualized MPS would be functionally similar except that it would use a fixed amount of hardware resources instead of pools. Re-configuring a server with a fixed amount of hardware resources will at least require a server shutdown; besides, it might require reconfiguring the runtime environment, the OS, or the app server. When using virtualization, resources can be added to or removed from the configuration while the system is running. To model such operating conditions using the selected model computing method, the method can be adjusted or use in two ways:

a. Either use multichannel QS where the number of channels can be adjusted during simulation. This might be inappropriate for percentage distribution of CPU power when the number of channels Kj and the number of busy channels kj in a multichannel QS could be a fractional number
b. Or simulate the process by adding or removing QS dynamically, which will entail a full rebuild of the transmission probability matrix and recalculating all of the earlier collected statistics, which is not acceptable

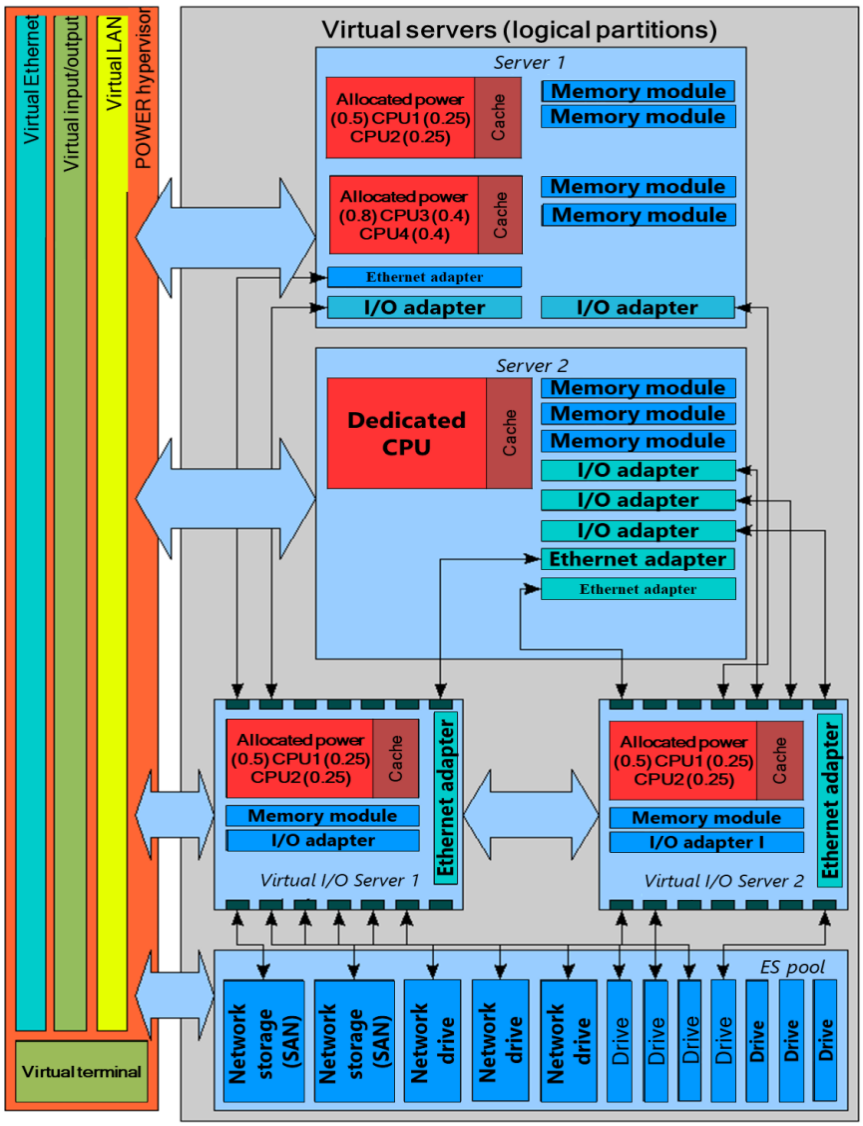

Figure 3. Structural diagram of logical partitions

The first option is a more optimal choice; however, some of the formulas have to be modified, e.g. the formula for finding the mean number of busy channels in the QS, as in this case, the CPU power can be distributed starting at 0.1 x the CPU-total power.

$$\beta_j = \lambda_j \vartheta_j = \begin{cases} \rho_j & npu \quad K_j = 1 \\ k_j & npu \quad K_j > 1 \end{cases} \tag{10}$$

where the load of the multichannel system is defined as $\rho_j = k_j / K_j = \lambda_j \vartheta_j / K_j$. Another formula that uses factorial that is generally only applicable to integers

$$R_j\left(M_j\right) = \begin{cases} 1/M_j! & npu \quad M_j \le K_j \\ 1/\left(K_j! K_j^{M_j - K_j}\right) & npu \quad M_j > K_j \end{cases} \tag{11}$$

The problem of calculating the factorial of a fractional number can be solved by using the asymptotic factorial formula (the Stirling formula) that can calculate the approximate factorial (12).

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1+\frac{1}{12n}+\frac{1}{288n^2}-\frac{139}{51840n^3}+O\left(n^{-4}\right)\right) \quad (12)$$

where *O* capital is the mathematical notation for comparing the asymptotic behavior of functions, which means the way the function is altered when approaching a certain point. The essence of the term *O* capital depends on the application; however, it never $O(f)$ grows faster than f .In many cases, approximate calculation of a factorial requires only the dominant term of the Stirling formula

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \quad (13)$$

It can be argued that;

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n < n! < \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/(12n)} \quad (14)$$

One of the absolute musts is the ability to create random events so as to simulate random failures in the CS model. The probability of a failure is set as a trigger of pseudo-random failures to simulate the failure of this or that device in the CS model. The failure trigger generates events of the model runtime. While the model is running, such events manifest as a drop in a channel-specific power in a QS to the minimum of 0.1. Full exclusion of the failing QS from the model is not an option as that would entail resizing the entire transmission matrix and could deteriorate the statistics collected before the failure. Therefore, post-failure restoration of the processing power is possible provided there is a pool of available virtual resources. The technology of aggregating the network channels and input-output channels backup can be simulated by several intermediate QS. Based on the collected model computation results, one can prove the efficiency of using the CR and the better balance of the simulated virtualized CS as compared to ordinary CS that cannot dynamically allocate resources. Therefore, given the above-described triggers, the models become adaptive; in other words, one can create adaptive models, specifically mathematical models used in combination with the operator-assigned dynamic characteristics, i.e. machine decision-making procedures applicable to adjustments in the model resources.

4. RESULTS

4.1. Model verification

In general, verification means confirming that the CS model description fully matches the specification or the analyzed system. To check whether the experimental system works as planned, it is necessary to trace the system response to an input and compare it with the simulated response or to the response of another model [26]. Model verification is apparently a very important process that can be done in several ways:

a. Experimentally
b. By using simulation models in case of testing analytical models, or vice-a-versa
c. By using a third method to build a similar model and compare the results

Bearing in mind that papers [12-14] describe virtualized-CS models, the most rational way is to create virtual servers and measure their dynamic characteristics; in other words, to carry out a computational experiment. Verification tests have been run using a virtual server, see Table 1 for specifications.

Table 1. Specifications of the virtual server used for model verification

Resource name	Quantity and characteristics
Processor	Two Power 5 virtual processors, 1.6 GHz (up to 1 physical processor allocated); also a double dedicated processor.
Cache	L1 combined cache: 64 kb instructions and 32 kb data; 1.9 Mb L2; 36 Mb L3
Memory	2 Gb (range: 1 to 4 Gb)
Drive	20 Gb, connected via a virtual I/O server, virtual SCSI adapter
Network	100 Mbps or 1 Gbps, also a virtual Ethernet adapter, via a virtual I/O server.
OS	AIX 6.1, 64bit

Each virtual processor has SMT on, i.e. comprises two logical processors. Figure 4 shows an image from the hardware management console (HMC) connected to the physical server (System P) hosting the virtual server.

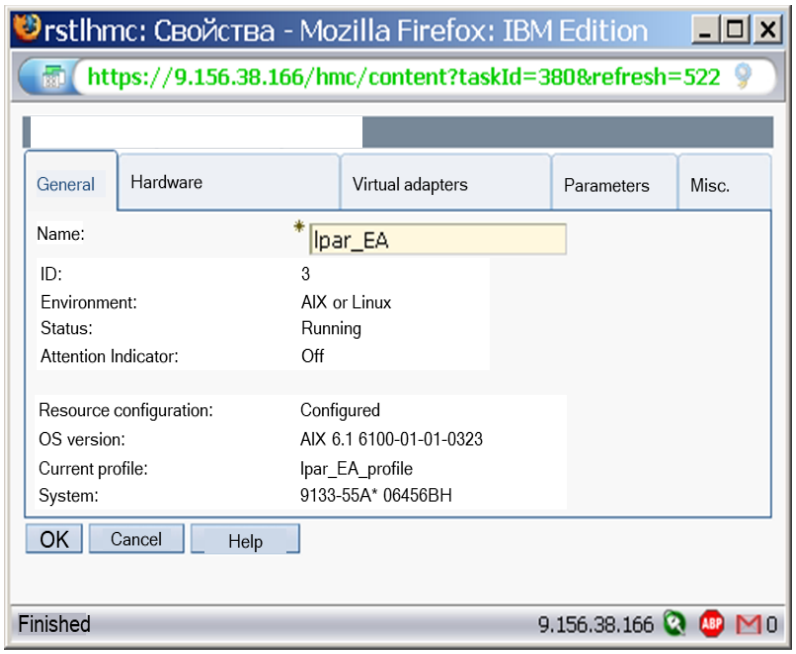

Figure 4. General partition properties

To measure the functional indicators of the virtual server, this research uses the Nmon utility [27]; this utility can collect and log statistics on the virtual server operations. Data can further be visualized as graphs or shown in the console window in a symbolic form. Nmonanalyser is used to convert this statistic into a convenient representation [28]. Stress is used to generate server load, i.e. to simulate request processing in a way similar to the models [29]. This C utility is extended to generate loads not by time but by the set number of simplest cycles (counter decrement and sqrt () function are computed instead of timeout). This effectively simulates running "requests" of a specific computational intensity so as to link model calculation results to verification results, as well as to draw findings on the performance. Besides, this program uses child processes, i.e. makes effective use of parallel processing.

The experiment is a two-part test: it is to monitor a virtual server with allocated processing power, and a virtual server with a dedicated processor, using the load generation utility and dynamic reconfiguration to adjust to the load. While verifying a model, it should be borne in mind that any operating system (OS) runs various system processes that load the CPU(s) at 5% to 10% on average, which is comparable to the modeling error and represents the verification error, which is acceptable for engineering studies. This conclusion is confirmed by analyzing the Nmon-collected statistics on the idle load, as the virtual server is only running the OS itself, see Figure 5.

Experiment Steps.

a. Start Nmon to collect and log statistics every second until Stress completes a run
b. Start Stress at the required computational intensity (number of requests)
c. Collect statistics on the load of the virtual server or its separate subsystems while running in a static mode, i.e. without reconfiguring
d. Collect statistics on the load of the virtual server or its separate subsystems while running in a dynamic mode
e. Use Nmonanalyser to process the statistics
f. Approximate the results and compare it to the modeled ones

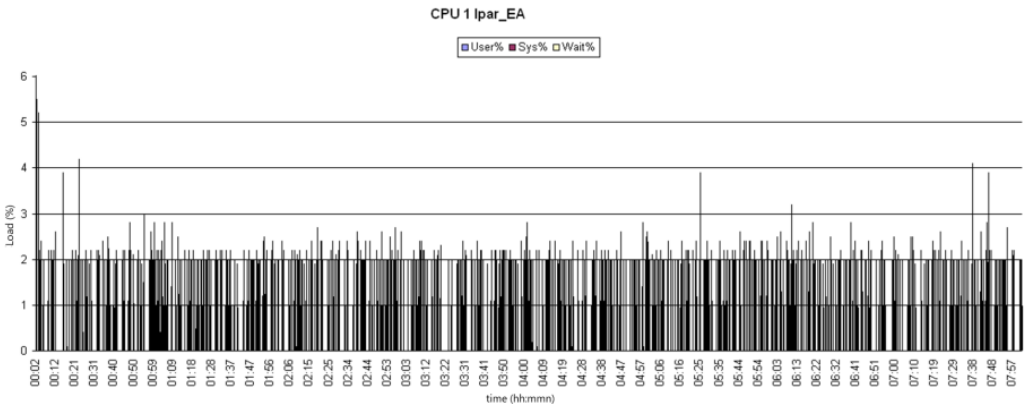

Figure 5. Idle system load

Assume that a request equals 10,000,000 cycles of an ordinary counter. First collect data on the static request processing time: the virtual server is subject to no reconfiguration, and only the number of processed request will change, see Figure 6. Stress is configured as follows:

a. cpu 2: two threads are intensively loading the CPU by calculating sqrt () for a random number
b. io 2: two threads are intensively loading the I/O system, namely the buffers
c. hdd 1 --hdd-bytes 256M: 1 thread is intensively writing onto the disk in 256-Mb blocks
d. vm 2 --vm-bytes 32M: 2 threads are intensively using the RAM in 32-Mb blocks
e. loops: the number of counter cycles or the number of requests, from 0.5 to 16

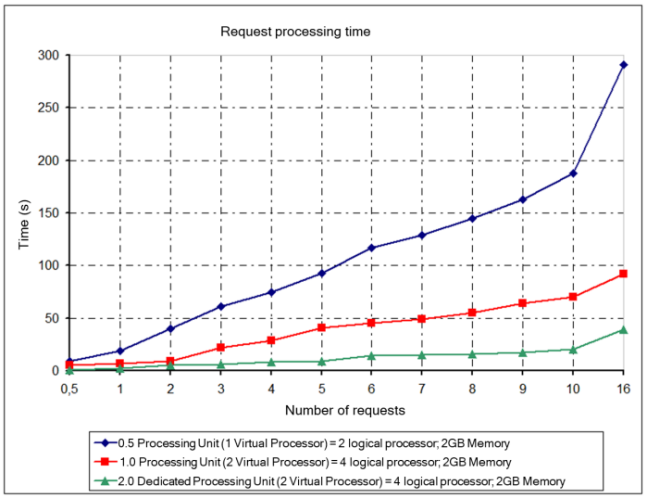

Figure 6. Request processing time

Data has been collected from 3 virtual server configurations that only differ in allocated processing power, RAM=2 Gb:

a. 0.5 processor units (1 virtual processor=2 logical processors)
b. 1 processor unit (2 virtual processors=4 logical processors)
c. 2 dedicated processor blocks (2 virtual processors=4 logical processors)

Repeated measurements identify a reduction in the processing time as more iterations are run, which is explainable by greater amounts of data stored in the cache (up to 100%). Each CPU-loading thread is processed by a separate virtual processor, which makes clear the efficiency of using multithreading in software. For example, if there are only two threads, two of four virtual processors will be idle; in this case, virtualization enables flexible adaptation to the load. A similar feature was demonstrated in the models in [12-14].

Maximum request processing speed is attained when the number of virtual processors equals the number of threads in a program. Mean request processing time of the three configurations equals 0.053, 0.141, and 0.483 (requests/sec). Comparison of the results shows that the model behaves similarly to the real system, see Figure 7. Find below the results of virtual server monitoring as collected by Nmon and Nmonanalyser with PLM (partition load manager) enabled and a specified resource management policy. The batch job contains 20 requests. The batch is processed twice. The graphs show the "stepped" dynamic resource buildup, particularly in the case of CPU (Figure 8) and RAM (Figure 9).

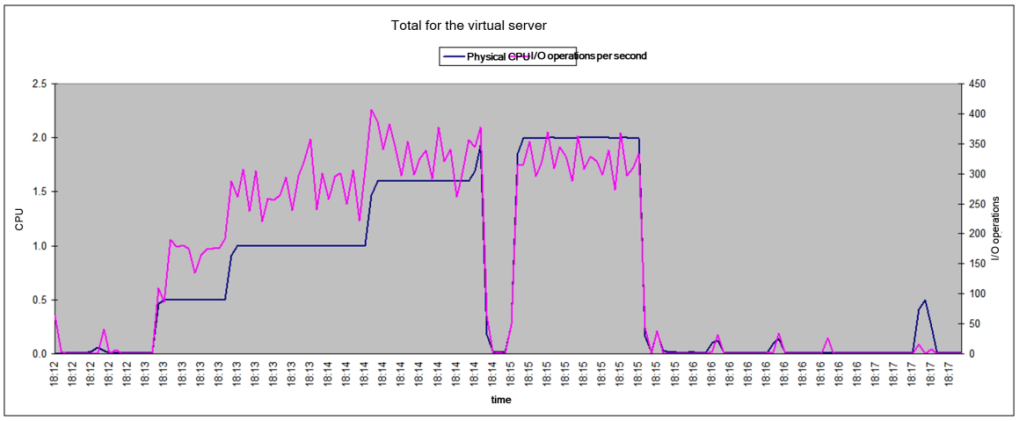

Figure 7. Summary virtual server statistics

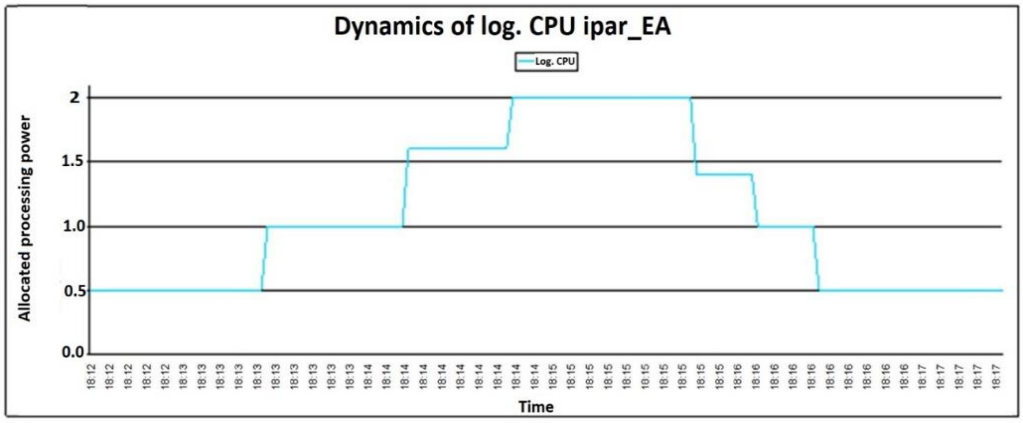

Figure 8. Dynamics of logical CPUs

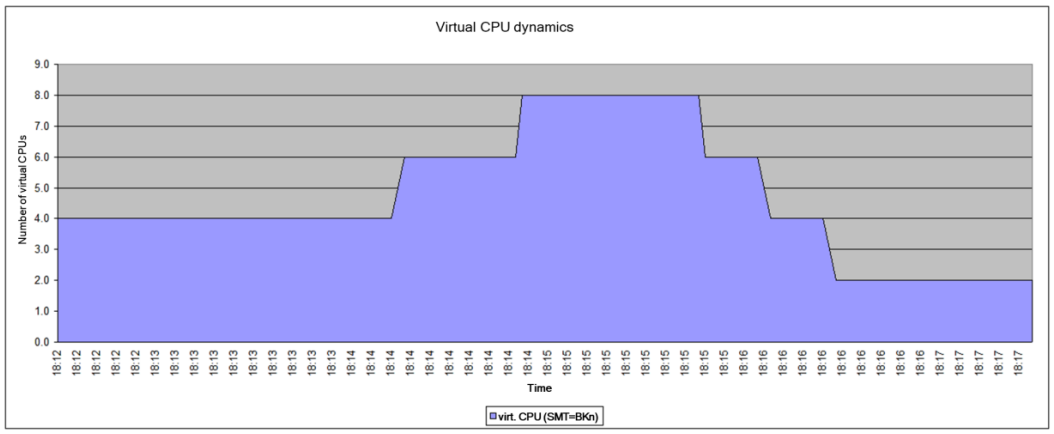

Figure 9. Dynamics of virtual CPUs

The CPU utilization (load) will depend on how well a running program is parallelized in comparison to Stress. Approximating the graph above makes it clear that CPU are loaded at 100% right from receiving the first request; request multithreading maximizes the CPU utilization. Figure 10 shows the dynamics of load across all CPUs.

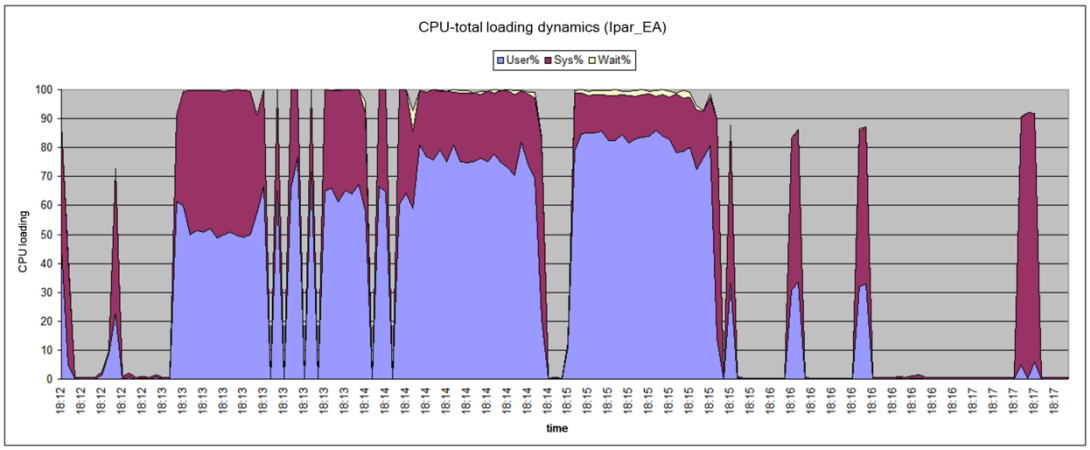

Figure 10. Dynamics of load across all CPUs

Starting at 18:15, the graphs show the processing of the second request batch; apparently, it takes far less time to process since the maximum processing power is available right away, unlike in the case of the first batch, which the server used to adapt itself to the load. Efficient caching enhances the performance, too. As can be seen applicable to CPU7 and CPU8, increasing the number of processors beyond that number of the test threads will cause "unneeded" CPUs to idle.

Thus, verification using a virtual server configured similarly to the model reveals similar dynamics of the server and the model, both in terms of the load and in the resource utilization; it also makes clear the effect of adaptability, which means that the models developed and described in [26-29] are appropriate for studying virtualized CS. However, there are some differences, as the input stream of the model and the real-world task batch are different.

4.2. Adaptive model building and evaluation

As mentioned earlier in [5], using a parallel algorithm for p processors as compared to sequential computing will solve the problem on P CPUs P times faster than on a single CPU and/or multiply the amount of processed data by P; however, such acceleration is rarely attainable, as most executables are not optimized and feature a considerable portion of non-parallel code. Given that most state-of-the-art heavy-load software systems use parallelization, hardware utilization efficiency can be maximized by coupling parallel computing with virtualization for dynamic allocation of processing power and RAM.

Judging from the above, building an efficient virtualized-CS model is key; such virtualization shall best suit the needs of an app planned to run on the future virtual server based on the model. On the other hand, given that virtualized CS adapt well to loads, the easiest approach would be using a minimum configuration and allow the CS to optimize its configuration while processing a task batch so as to adapt itself to the load. It is also possible to run each app on a separate virtual server with a separate OS, which will isolate the processes in terms of security and fail-safety. Let us define the basic criteria of modeling efficient virtualized CS:

a. Adequate source data for the model, adjusted to the parameters of the future hardware host platform, as the accuracy of the inputs will directly affect the accuracy of simulation
b. Flexibility of model adaptability to load, which is attained by using multiple criteria and conditions of adaptability triggering
c. Cost optimization to the researcher's requirements as part of the modeling effort
d. Request processing speed and quality optimization to the researcher's requirements as part of the modeling effort. Such optimization shall provide minimum request processing time, minimum set of resources, etc.
e. It is assumed that workload tasks are optimized for multithreading

Powerful and efficient CS and visualization tools will considerably reduce the task processing, analysis, and prediction time applicable to electronic workflow, real-time transaction processing, and creation

of data storages for decision-making systems, climate and global warming modeling, etc. However, a fast CPU is not everything. The architecture must be balanced to fully utilize the power of a modern CPU. Efficient computing platforms shall provide balanced performance in many aspects, including memory access, system switch, input-output, graphics accelerator, network operations, and CPU computing.

As performance and scalability bar is being set ever higher, conventional workstations (even multiprocessor ones) become too expensive and impractical, making it more effective to use virtual servers with natural virtualization, since OS virtualization tools will use 10% to 30% of the CPU power while natural virtualization is provided as firmware level or by specialized hardware, which is way more efficient.

Multiprocessor computing can be made more efficient by parallelization, which accelerates database query processing, provides efficient access to remote file systems, speeds up resource-intensive applications. Indeed, the POWER architecture provides such flexibility that additional virtual CPUs can simply be added if necessary, or pre-installed ones can be activated to handle peak loads. Besides, the OS and the related software and technologies do support and make active use of the hardware advantages this platform offers.

Thus, for the above criteria, the method of modeling an adaptive system will comprise the following steps: 1) build a model in a software system (e.g., in [30]); 2) calculate and analyze the results; 3) define the significant cost, quality, and processing speed criteria; 4) vary the model parameters to optimize by the previously defined criteria; 5) compare to the initial version and make the necessary adjustments; 6) create a virtual server to verify the model, as this method produces the most reliable data; 7) define the virtual server resource management policy specifying the pool of available resources or the donor group of virtual servers; 8) configure and start the server tasks; 9) monitor; 10) analyze the resource manager functioning and adjust the original virtual server profile to finalize the virtual server configuration.

5. CONCLUSION

This research has experimentally produced simple models for analyzing various systems designed for tasks of varying responsibility and requiring various resource groups to efficiently handle whatever they are tasked with. The proposed models can analyze various CS options that use virtualized resources with the above constraints. Speaking of the real-world application of virtual servers, outsourcing IT infrastructures is an increasingly popular solution, as it eliminates the need to purchase expensive servers that will also require hardware and software support.

Therefore, there exist two separate products for resource and load management, which causes inconvenience and makes it difficult to configure an integrated system that would take into account both the resource load and the responsibility of each application. It is therefore optimal and convenient to use a single integrated resource management mechanism based on the resource loads and on the responsibility of tasks assigned to each partition. The researchers have verified the mathematical virtualized-CS models by using a similarly configured virtual server.

Comparing the verification results and the calculations shows that the model and its virtual server implementation are identical in dynamics. The differences in the calculations and the experimental results are due to the difficulty of simulating the model-generated input stream of requests, which is quite abstract compared to real-world tasks; for maximum similarity, the research team has used a multithreaded load generator that clearly shows the specifics of multiprocessor CS with respect to the thread distribution between processors and thread parallelization.

CS cost and service quality optimizations are case-specific; however, what can be concluded for use is that maximum performance is non-attainable within cost restrictions. The developed model for building virtualized-resource CS models and optimizing them by various criteria reveals an interesting effect: it is possible to build systems capable of self-adaptation to load while being autonomous, which reduces the maintenance costs. The effect is observed when flexible system resource management policies are configured. In conclusion, it should be noted that the considered adaptive systems feature using both model-generated and expert data for decision-making; the expert data are idiosyncratic decision charts provided by the researcher.

ACKNOWLEDGEMENTS

Research was supported by the RFBR Grant for Best Projects of Basic Research, Grant No. 19-07-00516 A.

REFERENCES

[1] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412-421, 1974.
[2] L. Chernyak, "Renaissance of virtualization: chasing a steam locomotive (Renessans virtualizatsii–vdogonku za parovozom)," *Open Systems*, no. 2, pp. 26-35, 2007.
[3] W. J. Armstrong. *et al.*. "Advanced virtualization capabilities of POWER5 systems," *IBM Journal of Research and Development*, vol. 49, no. 4/5, pp. 523-532, 2005.
[4] M. Tulloch, "Understanding microsoft virtualization solutions," *Microsoft Press*, p. 464, 2010.
[5] O. I. Shelukhin, "Modeling of information systems (Modelirovaniye informatsionnykh system)," *Moscow: Goryachaya liniya-Telekom Publ.*, p. 516, 2012.
[6] A. P. Kirpichnikov, "Applied queueing theory (Prikladnaya teoriya massovogo obsluzhivaniya)," *Kazan: KSU Publ.*, p. 118, 2008.
[7] T. I. Aliyev, "Fundamentals of discrete systems modeling (Osnovy modelirovaniya diskretnykh sistem)," *Saint-Petersburg, St. Petersburg State University for information technologies, mechanics and optics Publ.*, p. 363, 2009.
[8] M. A. Matalytsky, et al., "Queueing systems and networks: analysis and applications (Sistemy i seti massovogo obsluzhivaniya: analiz i primeneniya)," *Grodno, Grodno State University Publ.*, p. 816, 2011.
[9] A. G. Lozhkovsky "Queueing theory in telecommunications (Teoriya massovogo obsluzhivaniya v telekommunikatsiyakh)," *Odessa, Odessa National Popov Academy of Telecommunications, Publ.*, p. 112, 2012.
[10] W. C. Poon, A. K. Mok "Improving the latency of VMExit forwarding in recursive virtualization for the x86 architecture," *Proceedings of the 45th Hawaii International Conference on System Science*, pp. 5604-5612, 2012.
[11] Rockwell team, "Virtualization for process automation systems," *Rockwell Automation Publication: PROCES-WP007A-EN-P*, 2013.
[12] O. O.Valova, A. I. Martyshkinn, "Development, research into, and use of virtualized computational system models (Razrabotka, issledovaniye i primeneniye modeley vychislitelnykh sistem s virtualizatsiyey)," *Sovremennye informatsionnye tekhnologii*, no. 20, pp. 50-57, 2014.
[13] A. I. Martyshkin, "Application of the apparatus of queuing theory in the description of adaptive models of computing systems with resources virtualization, " *XXI Century: Resumes of the Past and Challenges of the Present plus*, vol. 7, no. 2 (42), pp. 15-21, 2018.
[14] A. I. Martyshkin, et al., "Using elements of queuing theory in describing and managing resources and workload of adaptive mathematical models of computing systems with technology of virtualization of resources," *XXI Century: Resumes of the Past and Challenges of the Present plus*, vol. 7, no. 4 (44), pp. 71-78, 2018.
[15] A. I. Martyshkin, "Basic directions and paths in the advancement of modern embedded operating systems (Osnovnye napravleniya i puti razvitiya sovremennykh vstraivayemykh operatsionnykh sistem)," *Sovremennye informatsionnye tekhnologi*, no. 27, pp. 63-69, 2018.
[16] A. Church, "An unsolvable problem of elementary number theory," *American journal of mathematics*, vol. 58, no. 2, pp. 345-363, 1936.
[17] A. Church, "A note on the Entscheidungs problem," *The journal of Symbolic Logic*, vol. 1, no. 1, pp. 40-41, Jun 1936.
[18] A. M. Turing, "On computable numbers, with an application to the Entscheidungs problem," *Proceedings of the London Mathematical Society*, vol. 2, no. 1, pp. 230-265, 1937.
[19] A. M. Turing, "On computable numbers, with an application to the Entscheidungs problem. A Correction", *Proceedings of the London Mathematical Society*, vol. s2-43, no. 6, pp. 544-546, 1938.
[20] G. Sinha, et al., "A comparative strategy using PI & fuzzy controller for optimization of power quality control," *Indonesian Journal of Electrical Engineering and Informatics*, vol. 6, no. 1, pp. 118-124, 2018.
[21] L. Kleinrock, "Queueing systems," *Moscow: Mir*, p. 600, 1979.
[22] M. Schwartz, "Telecommunication networks: Protocols, modeling and analysis: in two parts," *Moscow: Nauka*, p. 336, 1992.
[23] J. Ikäheimo, "Method, apparatus and computer program product for monitoring data transmission connections," *United States patent US*, vol. 8, p. 168, 2011.
[24] G. I. Ivchenko, et al., "Queueing theory (Teoriya massovogo obsluzhivaniya)," *Moscow: Vysschaya shkola*, p. 256, 1982.
[25] P. Bocharov and A. V. Pechinkin, "Queueing theory (Teoriya massovogo obsluzhivaniya)," *RUDN Publ*, p. 529, 1995.
[26] A.I. Grushin, "Verification in computing (Verifikatsiya v vychislitelnoy tekhnike)," II Potential, no. 4, 2007.
[27] J. Zhao and S. Zdancewic, "Mechanized verification of computing dominators for formalizing compilers," *International Conference on Certified Programs and Proofs*, pp. 27- 42, Dec 2012.
[28] G. Sakellari and G. Loukas, "A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing," *Simulation Modelling Practice and Theory*, vol. 39, pp. 92-103, 2013.
[29] E. Kalyvianaki, et al., "Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters," *in Proceedings of the 6th international conference on Autonomic computing*, pp. 117-126, Jun 2009.
[30] J. Ikäheimo, "Method, apparatus and computer program product for monitoring data transmission connections," *United States patent US*, vol. 8, p. 168, 2011.

BIOGRAPHIES OF AUTHORS

Alexey Ivanovich Martyshkin in 2010 received a diploma in Computing machines, complexes, systems and networks", in 2013 a Ph.D. in technical sciences in the field of mathematical modeling, numerical methods and software complexes in Russia, at Penza State Technological University (PSTU). Since September 2011, he worked as a teacher at PSTU as an assistant and since 2014 he has been working as an assistant professor at the Department of Computing Systems. The main areas of activity are: modeling of models of multi-threaded and multiprocessors computing systems using the mathematical apparatus of systems and Queuing networks; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems. E-mail: alexey314@yandex.ru

Dmitry Vladimirovich Pashchenko in 1998 received a diploma in "Computing machines, complexes, systems and networks". In 2003, he defended his PhD thesis on specialties "System analysis management and information processing" and "Mathematical and software of computers, complexes and computer networks". In 2013-defended his doctoral dissertation on specialty "System analysis management and information processing". Since September 2019-rector of the Penza state technological University.The main areas of activity are: modeling and formalization of models of multi-threaded computing systems using the mathematical apparatus of Petri nets; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems. E-mail: dmitry.pashcenko@gmail.com

Dmitry Anatolyevich Trokoz in 2011 received a master's degree, in 2013 a Ph.D. in technical sciences in the field of theoretical foundations of computer science in Russia, at Penza State University. Since January 2014, he worked as a teacher at PSU as an assistant professor at the Department of Computing Engineering, and since September 2019, he has been working as a teacher at Penza State Technological University. The main areas of activity are: modeling and formalization of models of multi-threaded computing systems using the mathematical apparatus of Petri nets; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems. E-mail: dmitriy.trokoz@gmail.com

Sinev Mikhail Petrovich in 2011 received a master's degree, in 2013 received a Ph.D. in technical sciences in Russia, from Penza State University in the field of theoretical foundations of computer science. Since 2014, he has been working as a teacher at PSU as an assistant professor at the Department of Computer Engineering. The main field of activity are: development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems. E-mail: mix.sinev@gmail.com

Boris Lvovich Svistunov in 1971 received a diploma in "Automation and telemechanics", in 2004 received a doctoral thesis in the field of devices and measurement methods in Russia, at the Penza state University. Works as a Professor of the Department of Mathematics and physics. The main field of activity are: the use of structural and algorithmic redundancy in devices for measuring and monitoring parameters of electric circuits and output values of parametric sensors E-mail: sbl@penzgtu.ru