A performance model of a multiprocessor computer appliance of a real-time control system

P V Galushin¹, O Yu Serikova², V A Terskov³ and K V Yarkov²

¹ Siberian Law Institute of Ministry of Internal Affairs of the Russian Federation, Krasnoyarsk, Russia
² Krasnoyarsk Institute of Railway Transport, branch of the Irkutsk State University of Communications, Krasnoyarsk, Russia
³ Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russia

E-mail: galushin@gmail.com, serikova_oy@krsk.irgups.ru, terskovva@mail.ru, yarkov_kv@krsk.irgups.ru

Abstract. The article proposes a performance model of a multiprocessor computer appliance in a real-time control system, built using the mathematical apparatus of queuing theory, and different from existing similar models in that they take into account the possibility of failure and recovery of the hardware components of computer appliances. This article also explores the possibility of optimizing technical and economic characteristics using the proposed model. Due to optimization problem characteristics we should conclude that such optimization may be performed by evolutionary algorithms.

1. Introduction

Computers have become an integral part of industry and everyday life. Computer technology has turned from a means of performing computations into a hardware basis for the comprehensive automation of the processes of collecting and processing information, decision making and automatic control. Many modern control systems are real-time and therefore their performance is a critical parameter: the control action must be generated in the required time, otherwise it becomes useless. This class of control systems includes, for example, the control systems of air traffic and technological processes.

The performance requirements for computing systems used in real-time control systems are constantly increasing due to the complexity of control objects.

There have traditionally been two ways of increasing the performance of computers: by increasing the clock speed of processors and by developing multiprocessor systems. Today it can be stated that the possibilities of increasing the clock frequency have been exhausted, and this is due to physical limitations [1]. This means that real-time control systems will inevitably be created on the basis of multiprocessor computing systems.

It is important to understand that the requirements for the hardware performance of a real-time control system are determined by the software that is used to generate the control action. Such software also has special requirements associated with the need to guarantee the generation of the
correct control action in a strictly defined time. Therefore, the performance of multiprocessor computing systems should be studied in connection with the functioning of the software.

There are no methods for developing non-trivial defect free software. Creating a reliable system from unreliable components is possible only through the introduction of redundancy. But since program errors are internal and copies of the program are completely identical, a simple duplication of components will not increase reliability. In order to ensure that failures in different modules duplicating each other do not occur on the same input data, these modules must be independently developed, this approach is called multi-version programming [2]. Obviously, the performance requirements increase with an increase in the number of different versions of software and an increase.

The traditional method for evaluating the performance of multiprocessor systems is to run typical tasks on manufactured samples. However, at the design stage, this method causes great difficulties associated with a significant investment of time and resources.

For the more efficient design of multiprocessor hardware and software systems, a model of their productivity is needed that would allow the performance of architecture options to be determined without experimentation, which can be extremely lengthy and require significant costs.

The existing performance models of multiprocessor computing systems do not take into account the possibility of hardware failures and its restoration. In practice, when designing hardware-software complexes of real-time control systems, this aspect cannot be ignored since a decrease in productivity due to the failure of one of the processors can render it impossible to generate a control action in the required time, which is unacceptable for real-time systems.

2. The proposed performance model of a multiprocessor computer appliance

Let us consider a more general model of performance, including additional states in which not all processors and buses are operational as well as transitions between states corresponding to the failures and recovery of processors and buses.

The considered multiprocessor computer appliance consists of \( N \) types of processors that contain \( M_i \) (\( i = 1, 2, \ldots, N \)) processors of each type, and the average execution of one instruction is equal to \( T_{0i} \). The processors are combined with random access memory of the device via \( N_i \) buses. The service time of the request from the processor of the \( i \)-th type is equal to \( i \). It is assumed that the time interval between any two related applications is subject to the Poisson distribution law with parameter \( \lambda_i \). The total flow of failures from processors of all types and interface buses also obeys the Poisson distribution law with parameter \( \lambda_i \). In addition, when evaluating the performance of the multiprocessor computer appliance, it is most often assumed that the time interval between two adjacent services obeys the exponential distribution law with parameter \( \mu_i \) and the recovery time of buses and processors of the \( i \)-th type obeys the exponential law with parameter \( \xi_i \).

A newly received request for service is sent with equal probability to any of the available buses and accepted for service. If all the buses are busy, then the service request is queued and waits for its service. The discipline of service is a random uniform choice from the queue.

The considered queue system (QS) may be in the following states:

\[ a_{0 \ldots 0 \ldots 0 \ldots 0 \ldots 0} \] - all buses and all processors are faulty and are being restored. The computing process is stopped.

\[ a_{1 \ldots 0 \ldots 0 \ldots 0 \ldots 0} \] - one bus is operational, and the other buses are faulty and are being restored. All processors are faulty and are being restored. The computing process is stopped.

\[ a_{0 \ldots 0 \ldots 1 \ldots 0 \ldots 0 \ldots 0} \] - all buses are faulty and restored, and all processors, with the exception of one of the first type, are faulty and being restored. The computing process is stopped.

\[ a_{0 \ldots 0 \ldots 0 \ldots 1 \ldots 0 \ldots 0} \] - all buses are faulty and being restored, and all processors, with the exception of one of the second type, are faulty and being restored. The computing process is stopped.
Figure 2 demonstrates transitions related to the faults and repairs of the transitions given in this figure. The system can transit from one state to another if some faulty devices of that type) are faulty and being restored. The probability of impossible states is 0. The intensities of the corresponding transitions. Figure 1 demonstrates transitions related to the faults and recoveries of hardware components. Figure 2 demonstrates transitions related to the processor units accessing the memory.

Some neighbour states may not exist due to physical limitations (for example, if all buses are faulty, it is not possible for more buses to become faulty). Such states and corresponding transition rates must be ignored since the probability of impossible states is zero.

Let us calculate the intensities of the transitions given in this figure. This system can transit from one state to another if some processor units or bus becomes faulty, if some faulty processor unit or bus recovery occurs, if some processor unit tries to get access to the data in the memory, or if some processor unit frees the bus that was locked before.

The total intensity of processor unit or bus failures is proportional to the number of well-functioning processor units or buses respectively (total number of devices of some type minus number of faulty devices of that type). The total intensity of processor unit or bus recoveries is proportional to the number of faulty processor units or buses respectively.

The total intensity of accesses to the memory is proportional to the number of well-functioning processor units. This access flow is divided into two branches. The first branch includes accesses to the locked buses. The second branch includes accesses to the free buses. If accesses to blocks of memory are totally random and uniform, then the probability of access to a locked bus is equal to the number of locked buses divided by the number of all well-functioning buses (n+1). The probability of accessing a free bus is equal to the number of free buses divided by the total number of well-functioning buses.
Figure 1. Fragment of the state chart of the considered queuing system depicting transitions related to the faults and recoveries of hardware components.

Figure 2. Fragment of the state chart of the considered queuing system depicting transitions related to the processor units accessing the memory.
The total intensity of events when a processor unit frees a bus it locked before is proportional to the number of locked buses. When a processor unit gets access to the memory, reads data and frees a bus, it can decrease the number of locked buses or not change it. After a processor unit has freed a bus it locked before, it is possible that this bus will be immediately locked by other processor units waiting to access this bus, or this bus may stay free if no more processor units are waiting to access it in the waiting queue. In the latter case, the length of the waiting will not change, whereas in the former case, this length will decrease by one. Let us calculate probabilities of transitions of each kind.

We denote the probability that the queue length \( l \) does not change after one of the \( k \) processor units locking memory buses has freed a bus as \( \pi_{k,l} \). The total number of possible queues of length \( l \) to the \( k \) buses is equal to \( k^l \). If after one processor unit has freed the memory, the length of the waiting queue does not change, then all \( l \) processor units in that waiting queue were waiting for a bus other than the one that was freed. The total number of possible queues of length \( l \) to the \( (k-1) \) buses is equal to \( (k-1)^l \). According to the classical definition of probabilities, the probability in question is equal to the ratio of favourable outcomes to the total number of outcomes:

\[
\pi_{k,l} = \left( \frac{k-1}{k} \right)^l.
\]

The probability of transition when the length of the waiting queue has decreased by one can be obtained as the probability of the complimentary event:

\[
1 - \pi_{k,l} = 1 - \left( \frac{k-1}{k} \right)^l.
\]

These considerations allow us to calculate all the coefficients of the Chapman-Kolmogorov equations of the considered queuing system. Chapman-Kolmogorov equations are a system of linear ordinary differential equations. These equations describe the dynamics of the queuing system in question, but the greatest interest is the steady state in which the probabilities of all possible states no longer change because it is assumed that real-time control systems must have been operating for a very long time and all transient processes have time to be completed. Direct way to obtain these steady state probabilities is to solve Chapman-Kolmogorov equations to obtain functional dependencies of the system’s state probabilities over time and calculate limits of these functions of time as time approaches infinity. But there exists a much simpler way. Since in the steady state probabilities will no longer change, their derivatives will be equal to zero.

Formally equating the derivatives to zero in the Chapman-Kolmogorov equations, we will obtain a system of linear algebraic equations for state probabilities in the stationary mode.

Solving this system with one of the numerical linear algebra methods, we obtain state probabilities. Since the number of states is very high, these probabilities themselves are inconvenient to analyze.

But they can be used to determine any performance characteristics of the analysed system. In particular, it is possible to compute the probability that designed computer appliance with given architecture has performance high enough to software was able to generate control action within the time set by terms of reference for the designed real-time control system.

The proposed performance model of a multiprocessor computer appliance is quite complex and should be implemented as a software system for effective use in practice. Such implementation was used by the authors.

3. Optimizing the performance of a multiprocessor computer appliance in real-time control systems

While existing performance models allow the minimum hardware configuration of a multiprocessor computer appliance to be determined, the performance model proposed in this article allows configurations to be found that have hardware redundancy (compared to the minimum configuration). However, due to this, there is a high probability of them being in states providing sufficient performance to achieve the functioning goals of the designed real-time control system. The proposed
approach is more flexible than the simple duplication of all hardware components of the minimum configuration, which can be used to reduce the cost of creating and operating the designed control system.

The proposed model can be used to optimize the performance of the multiprocessor computer appliance of real-time control systems. The resources allocated for the creation and operation of the computer appliance are always limited. Therefore, it is advisable to consider the problem of performance optimization as multi-objective: one objective will be productivity, and the other will be the cost of creating a computer appliance.

As a solution to the multi-objective optimization problem, the Pareto set should be considered, meaning that the set of possible configurations surpass the rest by at least one criterion. The choice of a specific configuration of computer appliance from the Pareto set is made by the decision maker based on his expert knowledge since from the point of view of optimization theory, the decisions included in the Pareto sets are incomparable.

Another possible statement of the problem of optimizing the architecture of a multiprocessor computer appliance is a constrained optimization problem, in which the objective function is the creation cost, and the constraint is the condition that the probability of working with a capacity sufficient for the functioning of the software is not less than the value specified by the customer of the control system. If we consider the cost of creation and operating costs of control system separately, we get the multi-criteria optimization problem with significant constraint.

Thus, the problem of optimizing the performance of a multiprocessor computer appliance is multi-criteria and/or constrained. Let us consider which of the features that affect the choice of optimization methods have objective functions.

First of all, it should be noted that the space of possible solutions is discrete since the configuration of the computer appliance is determined by the number of processors of various types and random access memory buses, which can only be integers. At the same time, the size of the search space grows rapidly with an increase in the number of processor types.

Traditional optimization methods cannot be used to solve such problems. Gradient methods are not applicable instead of the discreteness of variables, an exhaustive search cannot be performed in a reasonable time due to the search space being too large, and there is no a priori information on the properties of objective functions that can decrease (for example, convexity).

In solving such optimization problems, evolutionary optimization algorithms have proven themselves, for example, the genetic algorithm [4], probabilistic genetic algorithm [5] or asymptotic probabilistic genetic algorithm [6].

These optimization methods have a large number of customizable parameters. Their effective application requires careful tuning, which can only be done by a specialist in the field of evolutionary optimization methods.

To simplify the configuration of the parameters of the genetic algorithm and make it accessible to specialists in applied fields, a self-adjusted genetic algorithm was developed [7, 8]. The proposed technique instead of using single variant of each genetic operators or single value of each parameter is based on the evaluation of application probabilities for all variants of operators. At the start of optimization all variants of every genetic operator have equal probabilities. After each generation the probabilities of using each genetic operator are recalculated taking into account quality of individuals (with respect to objective) which were generated using the operator in question. Operators that generate better solutions will get more opportunities to generate new solutions at the expense of the operators that generate relatively bad solutions. But there is a certain threshold below which the probability of using the operator cannot fall. This threshold allows operators which performed poorly in the early stages of the optimization process to get the opportunity to reach their potential in the late stages of optimization, when it may be necessary to change the search strategy.

Similar self-adjustment can be applied to other evolutionary optimization methods mentioned above to reduce the qualification requirements of specialists using them to solve complex optimization
problems, including optimization of the architecture of the computer appliance for real-time control systems.

4. Conclusion and future research

The article proposes a performance model of a multiprocessor computer appliance in a real-time control system, built using the mathematical apparatus of queuing theory, and different from the existing similar models in that it takes into account the possibility of failure and recovery of the hardware components (processor units and random access memory blocks) of the computer appliance in question.

The constructed model was implemented as a computer program that allows the performance and the reliability characteristics of the computer appliance in the designed real-time control system to be calculated. These characteristics can be optimized by self-adjusted evolutionary optimization techniques to minimize the cost of creating and maintaining the designed real-time control system while providing the level of reliability and performance characteristics required by the terms of references given by customer.

Therefore, the study of the effectiveness of self-adjusted evolutionary algorithms in optimizing the reliability of the multiprocessor computer appliances in the real-time control systems can be considered as a possible direction for the further research.

Another possible area of improvement could be an attempt to find an analytical solution of the system of linear algebraic equations for state probabilities in stationary mode. If an analytical solution will be obtained in a closed form, then the computational complexity and accuracy (in particular – numeric error accumulation rate) of calculating the probabilities using the formulas found and using the numerical solution of the original system of linear algebraic equations should be compared.

As mentioned above, a realistic approach to creating a reliable real-time control system based on multiprocessor computer appliance implies the use of N-version software and that certain versions of this software may not work properly.

It would be reasonable to take this fact into account in the performance model of the multiprocessor computer appliance. The modified in this way model would include additional states corresponding to a different number of software versions that are functioning incorrectly. In this case, the required performance would depend on the specific state in which the control system belongs. The more software versions function correctly, the smaller will be the performance reserve of the multiprocessor computer appliance of the designed control system. Such a model would make it possible to find more flexible compromises between the probability of obtaining the correct control action and the probability of obtaining the control effect in the time specified in the terms of references.

This model could become the basis for a decision support system in the area of the design of highly reliable multiprocessor computer appliances for real-time control systems. Such a decision-making system would make it possible to determine both possible options of the hardware architecture of a computer appliance and the number and desired characteristics of the software versions for such systems.

References

[1] Sutter H 2005 The free lunch is over: A fundamental turn toward concurrency in software Dr. Dobb's Journal 30 (3)

[2] Avizienis A 1985 The N-Version Approach to Fault-Tolerant Software. IEEE Trans. Soft. Eng. SE-11 (12) 1511-7

[3] Ross S M 2014 Introduction to Probability Models (11th ed.) p 187

[4] Goldberg D 1989 Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley Professional

[5] Sopov E and Sopov S 2011 The convergence prediction method for genetic and PBIL-like algorithms with binary representation IEEE International Siberian Conference on Control and Communications (SIBCON 2011) pp 203–6
[6] Galushin P, Semenkina O and Shabalov A 2012 Comparative analysis of two distribution building optimization algorithm 9th International Symposium on Distributed Computing and Artificial Intelligence. Salamance, Spain pp 759-66

[7] Semenkin E and Semenkina M 2012 Self-Configuring Genetic Algorithm with Modified Uniform Crossover Operator. Tan Y., Shi Y., Ji Z. (Editors) Advances in Swarm Intelligence, ICSI, Part 1, LNCS 7331, Springer, Heidelberg pp 414–21

[8] Semenkin E and Semenkina M 2012 Spacecrafts’ Control Systems Effective Variants Choice with SelfConfiguring Genetic Algorithm. Ferrier, J.-L., Bernard, A., Gusikhin, O. and Madani, K. (editors), Proceedings of the 9th International Conference on Informatics in Control, Automation and Robotics 1 pp 84–93