Yu. V. Shabatura, Cand. Sc. (Eng); I. V. Shtelmakh; M. Yu. Shabatura

NEW METHOD FOR IMPROVEMENT SOFTWARE PERFORMANCE IN COMPUTING SYSTEMS BY AUTOMATICAL SEARCHING OF «BOTTLENECKS»

Task of improving the software efficiency of computing systems integrated into the computer network by automatic determination of the «bottlenecks» is described. Task of improvement the effectiveness of software according to standard ISO9126 is formalized, model of the process of its operation on the basis of graph is constructed, the scheme of the system improving the effectiveness of automated software computing systems by reducing the influence of «bottlenecks» is elaborated, principles of its operation are suggested.

Keywords: *Efficiency, software, bottlenecks, profiling.*

Introduction

Computer systems have been penetrated almost in all spheres of human activity. Computer systems integrated into the global computer networks, characterized by a large number of users and large load are widely used. An essential condition for successful operation of such systems is their high quality and efficiency.

The computing system (CS) is data processing system, configured to solve problems in various spheres of applications. CS include hardware and software. Standard ISO 8126 [1] defines the performance characteristics of software quality for CS. Structure of software quality model is shown in Fig. 1.

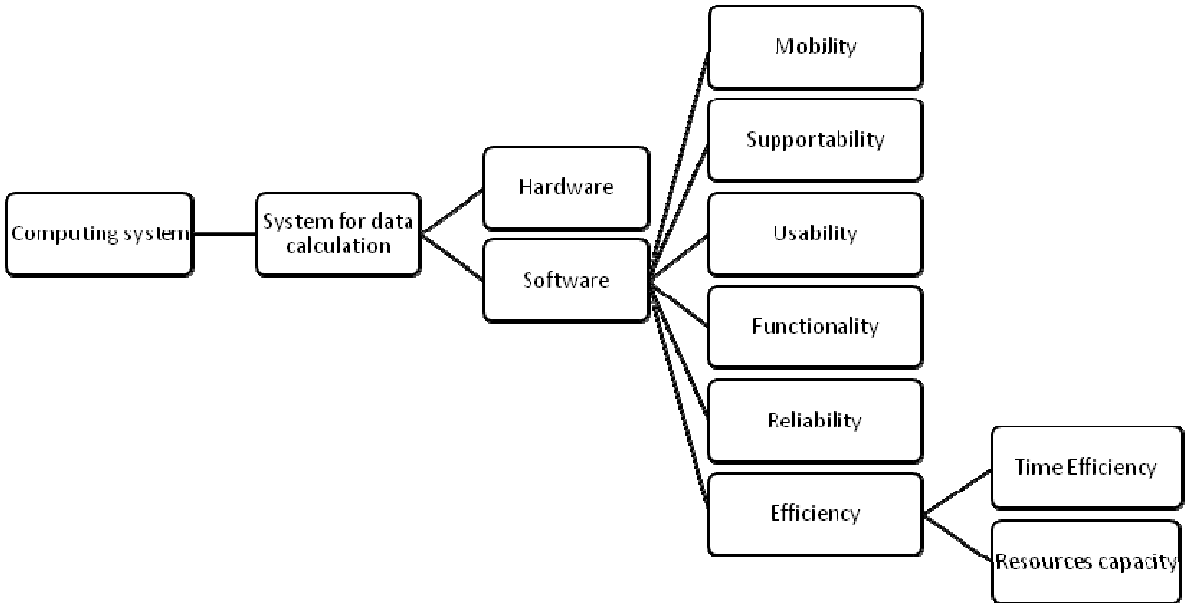

Fig. 1. Structure of software quality model according to ISO9126

One of the key characteristics of software is its efficiency. According to [1], the efficiency is defined as a set of attributes which are determined as the ratio of the quality of the software and resources used.

The index of time effectiveness shows the ability to perform the steps in the time interval that meets specified conditions. Resource capacity shows minimally necessary computing resources and the number of service personnel to operate the software.

Problem description

Time efficiency can be defined as

$$T_{ef} = \frac{F}{T_{run}}, \qquad (1)$$

where F –needed functionality amount, T_{run} - time of the programs run.

Thus, the objective of enhancing the effectiveness of the software can be defined as:

$$t'_{ef} = \max\left(\frac{F}{T_{run}}\right). \qquad (2)$$

It should be noted that the functionality of the software, is usually specified in the technical project, and is defined for a specific computer system, so it can be assumed as a constant: $F = F_{const}$.

Then the task of improving the efficiency of software is reduced to minimize the execution time of the program:

$$t'_{ef} = \max_{T_{run}\to\min}\left(\frac{F_{const}}{T_{run}}\right). \qquad (3)$$

Model of the software operation process

According to [2], the implementation of the program can be represented as a graph

$$G = (V, E), \qquad (4)$$

where the calling routine, or execution of operations, create a node set V, call processes of such operations or sub-programs are the edges E.

An example of such graph is shown in Fig. 2.

Let us introduce weight for each edge t_i, which characterizes the duration of the operation or sub-program execution . Thus, the total time of execution of the program on the basis of this model can be defined as

$$T_{run} = \sum_{i=0}^{n} t_i, \qquad (5)$$

and the duration of all graph nodes execution represents a set of

$$T = \{t_1, t_2, ..., t_i\}, \qquad (6)$$

where n – the number of calls, transactions and subprogrammes in the process of program execution as a whole.

Thus, increase of overall efficiency of software is reduced to decrease of the duration of its individual elements execution.

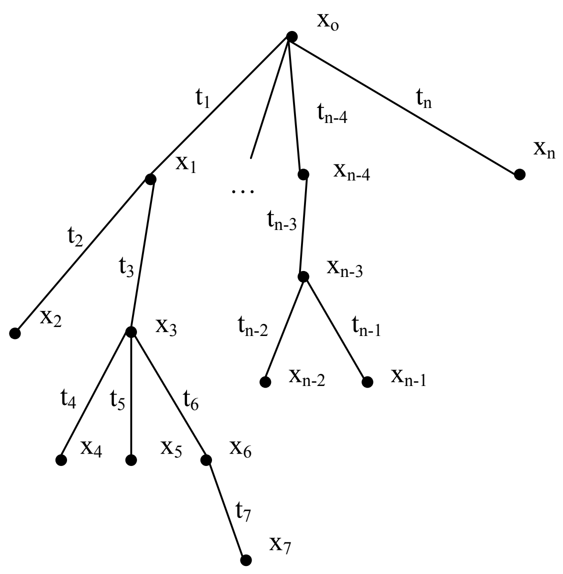

Fig. 2. Presentation of code run in the form of calls graph

In engineering phenomenon known as «bottleneck», which, according to [3] is the limitation the capacity or the productivity of the system caused by its individual elements. In programming search «bottleneck» is called performance analysis. The procedure of such elements of the code search is called profiling.

Thus, having defined the set of software elements

$$B = \{x_{k1}, x_{k2}, ..., x_{kb}\}, \tag{7}$$

which constitute the « bottlenecks», and having optimized them, we get run-time reduction of the program:

$$T_{run} = T_{run} - T'_{run}, \tag{8}$$

where T'_{run} – optimized for program execution time by optimizing the «bottlenecks».

On the basis of (1) we can express the rate of the software efficiency increase as a result of optimization:

$$k_{ef} = \frac{t'_{ef}}{t_{ef}} = \frac{\dfrac{F}{T'_{run}}}{\dfrac{F}{T_{run}}} = \frac{T_{run}}{T'_{run}}. \tag{9}$$

Thus, the objective of improving the efficiency of software is reduced to two sub-tasks: the definition of «gaps» in the code and their optimization. The scheme of the automatic increase of efficiency for computing systems based on this approach, is shown in Fig. 3.

In normal mode, of computing system operation , users perform queries and receive result Y.

Additional unit of optimization is introduced in the system, which allows the administrator to switch the system into optimization mode.

In optimization mode carried out some test queries X' are realised, on the basis of which the result Y' is obtained. In parallel, using a special server-side libraries Xdebug [4], we obtain data regarding the sequence and time execution of the program. Such information is obtained in the form of data array F, each element of which contains information on the duration of the subprogramme, its name and location in the file. This array is easily transformed into a graph of calls G, on the basis of which the search for multiple sub-programmes«bottlenecks» (7) and their further optimization is performed.

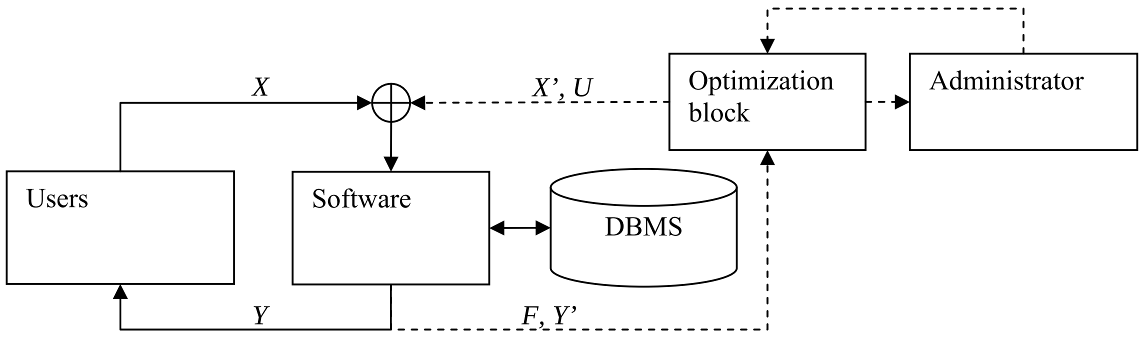

Fig. 3. Scheme of the automated system for improving the efficiency of computing systems by reducing the influence of «bottlenecks»

The task of automatic determination of the «bottlenecks»

Using the definition of «bottlenecks» in the software, as shown in [3], we will use this term to denote the set of vertex B of the call graph G, for which execution duration of each makes a significant impact on the total duration of the execution of the program T_{run}.

The extent of this impact for each vertex can be defined as:

$$b_i = \frac{t_i}{T_{run}}. \tag{10}$$

It should be taken into account that if the vertex contains calls of other peaks sub-programmes , the duration of the call will include the duration of all subsequent calls. Therefore, if the condition

$$b_i \approx \sum_{j=1}^{d} \frac{c_j}{T_{run}}, \tag{11}$$

where $\{c_1, c_2, ..., c_j\}$ – many peaks -subprogrammes, the call of which is performed from the given vertex, then this vertex can not be considered as a bottleneck, since considerable duration of this subprogramme realization is due to the large number of calls of affiliated sub-programmes. Therefore, these peaks are excluded from the set of «bottlenecks» B.

In addition, there can be vertices in the graph G, which describe the sub-programs of programming libraries used by the software. If these libraries are standardized and are characterized by high quality, their calls can not be considered as a bottleneck, but rather they are caused by improperly designed software. That is why we introduce the set

$$L = \{l_1, l_2, ..., l_{1b}\}, \tag{12}$$

which describes the files of the libraries. Subprogrammes found in these files will be excluded from the search process.

Having compiled the algorithm described in [5], of search in depth and constraint (11) and (12), we obtain as a result the algorithm for finding the set B of all vertices sub-programmes of the-system for which the optimization is possible. Each element of this set is characterized by the degree of influence (10) on total duration of the program execution , and consequently on the effectiveness (1).

Having selected a subset of vertices sub-programmes having the greatest degree of impact on efficiency, we obtain the required set of «bottlenecks» of the system B', over the elements of which optimization operation is performed.

The task of «bottlenecks» optimization

The task of «bottlenecks» optimization is to minimize the duration of sub-routines B execution . Set of «bottlenecks» B, can be divided into several subsets, depending delays nature

$$\{B\} = \{\{B_1\}, \{B_2\}, ..., \{B_{bc}\}\}, \tag{13}$$

where bc – amount of delays nature classes.

At this stage of research we have allocated 2 main classes of delays reasons: delay in realization of SQL-queries to the database, and delays resulting from the use of incomplete algorithms. For each class of delay various approaches to optimization are used. Regarding the first class of delay schemes of genetic optimization of SQL-queries is elaborated , but further work to increase the number of delay classes and approaches to their optimization is required.

Practical implementation and results of experimental studies

Based on the results of theoretical studies, shown in the work, the system of time delays search in the computer systems of mass use of PHP / MySQL architecture is realized. The system constructs a graph of sub-programmes calls in the software in the form of a hierarchical drop-down list (accordion) (Fig. 4).

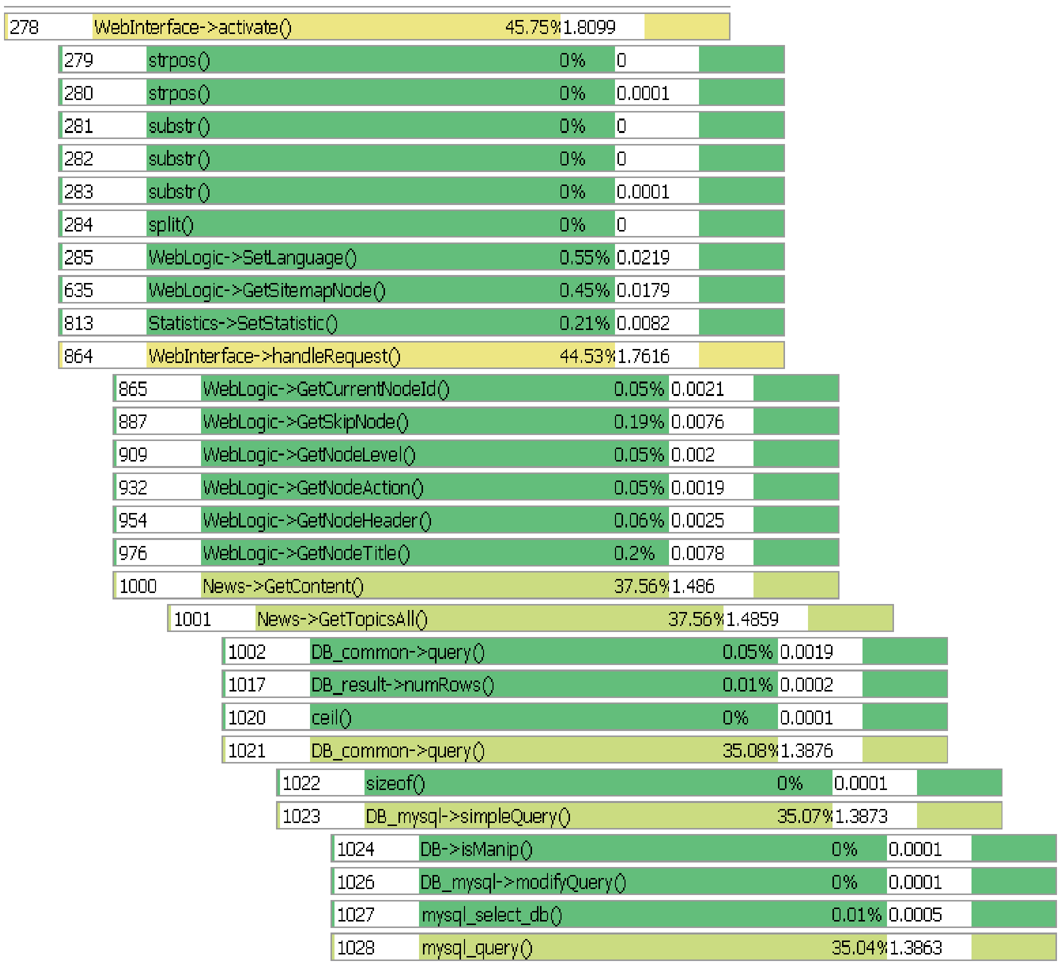
```
278   WebInterface->activate()            45.75% 1.8099
    279   strpos()                            0%     0
    280   strpos()                            0%     0.0001
    281   substr()                            0%     0
    282   substr()                            0%     0
    283   substr()                            0%     0.0001
    284   split()                             0%     0
    285   WebLogic->SetLanguage()             0.55%  0.0219
    635   WebLogic->GetSitemapNode()          0.45%  0.0179
    813   Statistics->SetStatistic()          0.21%  0.0082
    864   WebInterface->handleRequest()       44.53% 1.7616
        865   WebLogic->GetCurrentNodeId()        0.05%  0.0021
        887   WebLogic->GetSkipNode()             0.19%  0.0076
        909   WebLogic->GetNodeLevel()            0.05%  0.002
        932   WebLogic->GetNodeAction()           0.05%  0.0019
        954   WebLogic->GetNodeHeader()           0.06%  0.0025
        976   WebLogic->GetNodeTitle()            0.2%   0.0078
        1000  News->GetContent()                  37.56% 1.486
            1001  News->GetTopicsAll()                37.56% 1.4859
                1002  DB_common->query()                  0.05%  0.0019
                1017  DB_result->numRows()                0.01%  0.0002
                1020  ceil()                              0%     0.0001
                1021  DB_common->query()                  35.08% 1.3876
                    1022  sizeof()                            0%     0.0001
                    1023  DB_mysql->simpleQuery()             35.07% 1.3873
                        1024  DB->isManip()                       0%     0.0001
                        1026  DB_mysql->modifyQuery()             0%     0.0001
                        1027  mysql_select_db()                   0.01%  0.0005
                        1028  mysql_query()                       35.04% 1.3863
```

Fig. 4. Visualization of the code implementation

The color of each node of the list is established according to its influence (10) on the duration of program execution. As a result of experimental studies, the system showed the ability to visualize the process of program execution of computer systems and optimize their performance. In the future this system will be expanded by the function of automatic search of « bottlenecks», and their optimization.

Conclusions

1. Mathematical model for the implementation of automatic searching for «bottlenecks» in software is developed.
2. The structure and algorithmic support for construction of the automated system intended for improvement the computer systems efficiency by reducing the influence of «bottlenecks» is developed.
3. Experimental study was carried out , which showed the efficiency of developed technique and the possibility of practical implementation to reduce software time lags of different nature.

REFERENCES

1. Scalet R. ISO/IEC 9126 and 14598 integration aspects: A Brazilian viewpoint / Scalet R. // The Second World Congress on Software Quality. – Yokohama: 2000 – 350 p.
2. Novikov F.A. Discrete mathematics for programmers: For students / Novikov F.A. – Spb.: Piter, 2004 – 256 p.
3. Bottleneck (engineering). Wikipedia, the free encyclopedia. Режим доступу: http://en.wikipedia.org/wiki/Bottleneck_(engineering).
4. Derick Rethans. Documentation for: Xdebug 2. Function Traces. Режим доступу: http://www.xdebug.org/docs/execution_trace.
5. Ananiy V.O., Levitin G.V. Alhorithms: Introduction to development and analisys / Ananiy V.O., Levitin G.V. – M.: Wilyams, 2006. – p. 212 – 215.

Shabatura Yuri – Cand. Sc.(Eng), Professor. Phone.: (0432) 59-85-71, e-mail: shabatura@vstu.vinnica.ua

Shtelmakh Igor – Graduate student. phone.: (0432) 50-58-55, e-mail: ceo@sbsgroup.com.ua

Shabatura Maxim – Student. e-mail: smartmax@i.ua

Vinnitsa National Technical University