Methodology for Performance Analysis of Distributed Knowledge-Based Systems

Nataliia Kulykovska[a], Artur Timenko[a], Svitlana Hrushko[a] and Stepan Skrupsky[a]

[a] *"Zaporizhzhia Polytechnic" National University, Zhukovsky str., 64, Zaporizhzhia, 69063, Ukraine*

Abstract

In this paper, the performance of distributed knowledge-based systems is the time spent by an application to execute messaging functions. To analyze the performance of distributed knowledge-based systems, direct performance measurement with tracing of web service function calls is used. The main advantage of this approach is the preservation of complete information about the interaction of processes in the trace, which makes it possible to analyze the performance of web services in detail. The identification of performance problems based on the analysis of the collected data is determined by the selected criterion. The paper proposes an expert methodology for solving the performance problem. The scientific novelty of the work lies in the presentation of the model of the problem of the performance of web services, as a set of actions of processes, certain combinations of which, under certain conditions, can lead to an identifiable performance problem.

Keywords 1

Performance, analysis, web service, knowledge engineering, distributed knowledge-based systems, methodology

1. Introduction

The development of distributed knowledge-based systems requires research in the field of knowledge representation models, methods for describing them, and improving the performance of web services. Knowledge engineering in distributed knowledge-based systems implies the use of an ontological representation of knowledge. Thus, an object is formed that has a logical programmatic descriptive environment for further automatic machine processing [1].

The performance of distributed systems is a rather complex and multifaceted concept [2]. In [3] it is noted that under the metric of measuring performance, such diverse indicators as total operating time, parallel efficiency, scalability, memory requirements, bandwidth, latency, development cost, etc. can be considered.

In this paper, the performance of distributed knowledge-based systems refers to the time spent by an application to perform messaging functions.

Performance analysis is understood as the study of the values of service performance metrics [4]. In [5] the following goals of performance analysis are listed:

- determination of system performance under certain conditions;
- study of the system performance when varying parameters;
- comparison of performance of different systems;
- performance debugging - identifying the reasons why the system's performance does not meet expectations.

CMIS-2021: The Fourth International Workshop on Computer Modeling and Intelligent Systems, April 27, 2021, Zaporizhzhia, Ukraine
EMAIL: natalya.gontar@gmail.com (N. Kulykovska); timenko.artur@gmail.com (A. Timenko); grushko_ss@i.ua (S. Hrushko); sskrupsky@gmail.com (S. Skrupsky)
ORCID: 0000-0003-4691-5102 (N. Kulykovska); 0000-0002-7871-4543 (A. Timenko); 0000-0002-0064-408X (S. Hrushko); 0000-0002-9437-9095 (S. Skrupsky)

- performance tuning – i.e. determining the values of system parameters that provide the best overall performance.

2. Methods for analyzing and measuring the performance of distributed systems

Currently, there are three main methods for analyzing the performance of distributed systems [5]:

1. Analytical modeling [6]. This method provides a mathematical description of the system under consideration and, in comparison with other methods, it often provides the least detailed information about the system. But a simple analytic model can provide insight into the general behavior of a system or its components and what details need to be learned by other methods.
2. Simulation [7]. This method involves writing a simulator to simulate important characteristics of the original system. The advantage of this method is that the simulator can be easily modified to study certain aspects of the original system. The disadvantages include the inability to imitate every detail of the original system. As a result, simplifying assumptions must be made in order to have a reasonable execution time for the simulator, but this limits the accuracy of this method.
3. Measuring the performance of an existing system [8, 9]. In most cases, this method is preferred because no simplifying assumptions need to be made - the real system is being measured. On the other hand, this method is not very flexible, since provides information only about the system that it is measuring and for a specific set of its parameters. While it is possible to investigate performance for a few specific data parameters, it is generally difficult to do so.

When analyzing performance based on the method of measuring the performance of an existing system, there are three main methods of measuring performance [5, 10]:

- Event-driven method. This method collects performance indicators when a specific event occurs. In the simplest case, a global counter is used to count the number of events that have occurred. This method is best suited for events where the frequency of occurrence is not very high. otherwise, the behavior of the program changes.
- Tracing. This method is similar to the previous one, but in addition to recording the event that has occurred, some information about the state of the system is also used. The disadvantages of this method include increased memory requirements for storing the collected information, which can lead to an even greater change in the behavior of the program.
- Sampling. In contrast to the event-driven method, this method stores system state and performance metrics at fixed time intervals. As a result, the measurement overhead does not depend on the timing of the events. However, using this method, information is not collected about all the events that have occurred. The measurement result is the statistics of the program operation.

To analyze the performance of distributed knowledge-based systems, direct performance measurement with tracing of web service function calls is most often used. The main advantage of this approach is the preservation of complete information about the interaction of processes in the trace, which makes it possible to analyze the performance of web services in detail.

Improving the performance of distributed knowledge-based systems by directly measuring performance includes solving the following problems:

1. Choosing a criterion (metric) for evaluating the performance of a web service.
2. Collect performance data.
3. Visualization of performance data.
4. Identify performance problems.
5. Finding ways to fix performance problems.
6. Troubleshoot performance problems.

As noted above, the metric of measuring performance can be considered such diverse indicators as total execution time, parallel efficiency, scalability, memory requirements, bandwidth, etc.

Collecting performance data consists of tracing the calls to the web service functions. Depending on the selected criterion, one or several launches of the web service with representative data and certain parameters are performed on a particular computing system (or different systems).

The visualization of the collected performance data is usually done when identifying performance problems and finding ways to fix them will be done “manually”. There are various tools for visualizing performance data, for example, space-time diagrams, function statistics, and others [11, 12].

Apache JMeter, HP Loadrunner, Microsoft Visual Studio, TSUNG are used for performance testing [13, 14, 15, 16].

Apach JMeter is a load testing tool developed by the Apache Software Foundation. The advantages of the tool include: conducting tests with complex logic and correlation of dynamic parameters; performance testing of web applications, including APIs, web services, and database connectivity; the ability to reproduce the behavior of several users with parallel threads.

HP LoadRunner has a distributed architecture (VuGen, Controller, Load Generator, Analysis). Provides a set of tools for working with various protocols. Provides detailed logging of the actions of each virtual user.

Microsoft Visual Studio provides the following advantages: storage of test results in MS SQL Server database and their convenient structuring; no restrictions on the number of virtual users with a license.

TSUNG - support for HTTP, WebDAV, SOAP, PostgreSQL, MySQL, LDAP, XMPP; convenience in testing API requests; load distribution on the cluster from client machines.

This review reviewed existing web service testing systems to improve system performance. Table 1 shows a comparative analysis of the considered systems.

Table 1.
Comparison of performance testing tools

The challenge of increasing productivity	Apache JMeter	HP Loadrunner	Microsoft Visual Studio	TSUNG
1. Choice of metrics				
2. Collecting performance data	+	+	+	+
3. Performance data visualization	+	+		
4. Identifying performance problems		+		+
5. Finding ways to fix problems	+			
6. Troubleshooting				

3. Models and methods of knowledge representation for performance analysis

A web service in a distributed knowledge-based system (Fig. 1) is considered as a set of interacting processes:

$$PR = \{pr_1, pr_2, ... pr_N\}. \quad (1)$$

The interaction of processes is carried out using actions initiated processes in a certain sequence:

$$PR_i \Rightarrow A_i = \langle a_{ij} \rangle; i = 1,2,..., N; j = 1,2,...M_i. \quad (2)$$

Each action of the process consists in calling the functions of the web service:

$$F = \{f_k\}, k = 1,2,..., K. \quad (3)$$

Each function is characterized by input and output arguments:

$$f_k \Rightarrow FA_k^{IN} = (fa_{k1}^{IN}, \dots fa_{kL_k^{IN}}^{IN}); k = 1, \dots, K$$
$$f_k \Rightarrow FA_k^{OUT} = (fa_{k1}^{OUT}, \dots fa_{kL_k^{IN}}^{OUT}); k = 1, \dots, K \quad (4)$$

Thus, each process action is characterized by a web service function, the values of its arguments when called and terminated:

$$a_{ij} = \left\langle f_k, Faval_k^{IN}, FAval_k^{OUT} \right\rangle; i = 1,2,\dots,N; j = 1,2,\dots,M_i; f_k \in F$$
$$FAval_k^{IN} = (av_{k1}^{IN}, av_{k2}^{IN}, \dots av_{kL_k^{IN}}^{IN}); k = 1,\dots,K \quad (5)$$
$$FAval_k^{OUT} = (av_{k1}^{OUT}, av_{k2}^{OUT} \dots fa_{kL_k^{IN}}^{OUT}); k = 1,\dots,K$$

There are several messaging models that are defined by the syntax and semantics of function *F*.

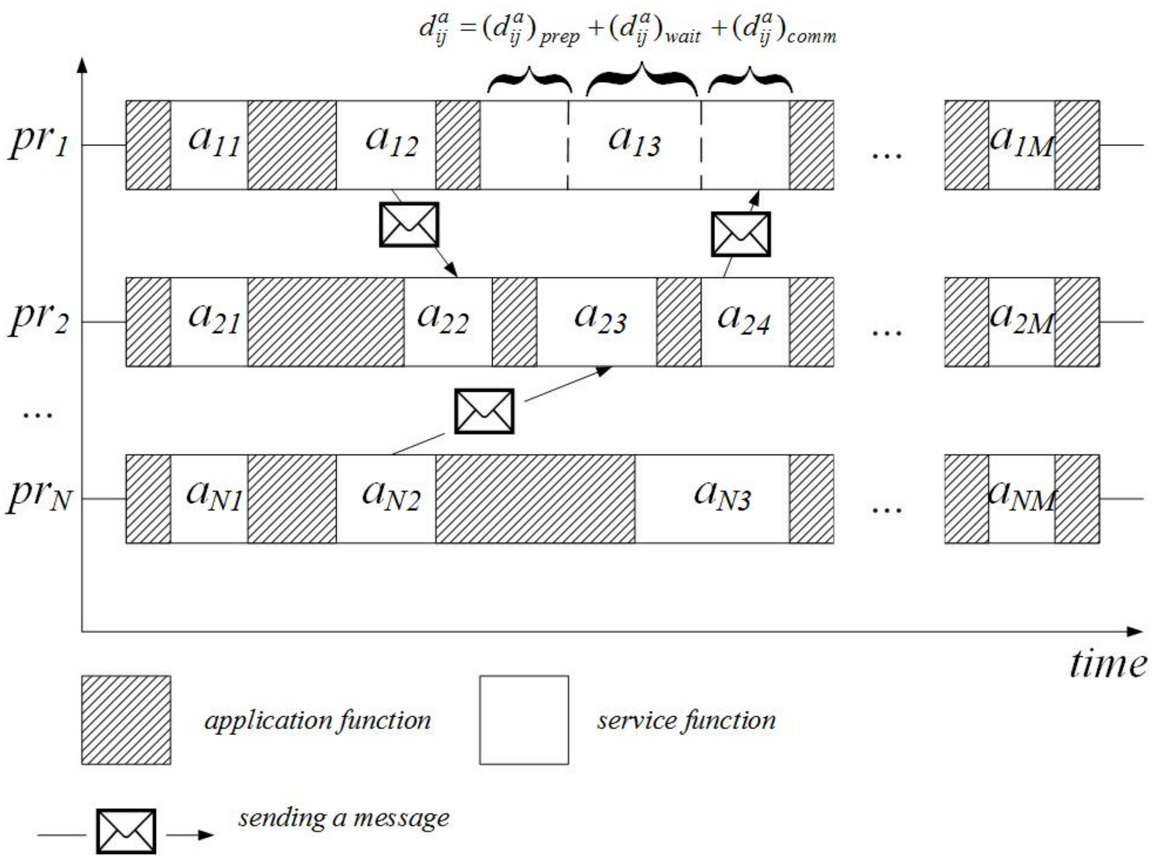

Figure 1: Model messaging web services

The duration of actions to call a web service consists of the following components:

$$d_{ij}^a = (d_{ij}^a)_{prep} + (d_{ij}^a)_{wait} + (d_{ij}^a)_{comm}, \quad (6)$$

were $(d_{ij}^a)_{prep}$ - is the duration of data preparation for transmission / reception of messages;

$(d_{ij}^a)_{wait}$ - the duration of waiting for the web service to be ready;

$(d_{ij}^a)_{comm}$ - duration of transmission / reception of messages over the network.

4. Identifying performance problems

The identification of performance problems based on the analysis of the collected data is determined by the selected criterion [17]. The search for ways to eliminate the identified performance problems can be carried out at different levels: the application level, the level of the web service functions, the level of the computing system, etc. [18, 19]. To fix problems at the application level, you need to change the source code of the service.

The process of performing the listed tasks can be repeated (for example, if there are several performance problems) until the required performance is achieved. However, it is recommended that you start by fixing the problems that will yield the most results, so productivity tools should provide information about the extent to which performance problems affect the overall running time of the application.

Thus, the work proposes an expert methodology for solving the set task, including the following stages:

- systematization of typical productivity problems and establishment of the reasons for their occurrence;
- development of recommendations to eliminate these causes;
- a description of the identified performance problems, rules for determining the causes of their occurrence and recommendations for their elimination in the knowledge base.

Events are recorded in the track when calling action functions. A simple event is represented as:

$$e = \left\langle f, et, EPval, t, d, cs \right\rangle, \quad (7)$$

where $f \in F$ - action function;

et - the type of event that sets its parameters $et \Rightarrow EP = (ep_1,...,ep_K)$;

$EPval = (v_1,...,v_K)$ - event parameter values;

t - time of the event;

d - event duration;

cs - service code.

The following action tracing rules must be set to execute tracing:

$$a = \left\langle f, FAval^{IN}, FAval^{OUT} \right\rangle \xrightarrow{rule} e = \left\langle f, et, EPval, t, d, pr, cs \right\rangle, \quad (8)$$
$$rule = \left\langle f, et, EPtemp \right\rangle$$

where $f \in F$ - function that triggers the tracer to generate a rule event;

et - the type of event that sets its parameters $et \Rightarrow EP = (ep_1,...,ep_K)$;

$EPtemp = \langle ep_i, Expr_i, kind_i \rangle; i = 1,...,N$ - description of event parameters and a method of obtaining their values, where:

ep_i - event parameter;

$Expr_i : FAval^{IN} \times FAval^{OUT} \to v_i$ - function for calculating the parameter value v_i based on the arguments of the called function;

$kind_i \in \{in, out\}$ - character of parameter (*in* - input, i.e. calculated only on the basis of the input arguments of the function; *out* - output, i.e. formed on the basis of the output arguments of the function or input and output).

The values of the *t, d, pr* and *cs* parameters of the event are generated by the tracer.

Trace rules are specified for the subset of functions whose call actions cause the described performance problem. This provides selective tracing of activities. And the parameters of the events logged in the trace contain only the information that is relevant for identifying the described performance problem.

The work uses the following model of the problem of web services performance:

$$pb = \left\langle pd, dur, TrRulesAnRules, REC(A^{INFO)} \right\rangle, \quad (9)$$

were pd - verbal description of the problem;

dur - the duration of the appearance of this problem;

$TrRules$ - rules for tracing the action of the problem;

$AnRules$ - rules for recognizing the problem;

REC - recommendations for its elimination;

$A^{INFO} = \left\{ \left\langle f, t, d, pr_i\ , cs_i \right\rangle \right\}$ - a description of the actions that led to the problem.

The rule for identifying a performance problem is represented as:

$$ce = \langle cet, CEPval, ME \rangle \xrightarrow{pbrule} pb = \left\langle pd, dur, REC(A^{INFO)} \right\rangle$$
$$pbrule = \langle cet, \varphi, pd, RECtemp, L_{dur} \rangle \quad , \qquad (10)$$

were *cet* – the type of compound event as a contender for the identified performance problem;

CEPval – the condition for the occurrence of the problem (if the value of the function is 1, then the problem exists and not otherwise);

pd – a verbal description of the problem determined by the rule;

RECtemp :*CEPval* x *REC* – recommendation template to fix the problem;

Ldur – function for calculating the duration of the performance problem.

It is concluded that there is a performance problem *pb,* when the conditions of the rule are met. The software environment recursively extracts simple events from the trace and generates information about a set of actions based on them.

The rules for identifying performance problems are described using a sequence of ontological axioms that have the syntax:

```
declare problem for <event name>
when
< logical expression >
parameters(
name = "< problem name >",
description = "< description of the problem >",
advice = "< recommendation >",
duration = < duration expression >);
```

where *declare problem f*or is a description of the rule for identifying a performance problem for an applicant event cet;

when – condition for the occurrence of the problem (as a logical expression over the parameters of a composite event);

name – name of the problem;

description – verbal description of the pb problem;

advice – recommendation template for its resolution $RECtemp$;

duration is an expression for calculating the duration of the manifestation of this problem L_{dur}.

The analysis flowchart is illustrated in Fig. 2. The initial data for the analysis is the service message trace, formed as a result of the trace and containing simple events, and the analysis result is a set of identified performance problems. The analysis is carried out in two stages:

1. Constructing compound events as candidates for specific types of performance problems.
2. Identify performance problems from multiple constructed composite events.

For a practical solution to the problem of increasing productivity, special systems are being developed that allow evaluating the performance of web services and identifying the most "heavy" parts of the code (performance problems). Based on these data, the expert can make a decision to optimize the program code in order to reduce losses in these areas. A common disadvantage of such systems is that the task of identifying performance problems remains quite difficult due to the large volume of information being processed and the complexity of the links that generate individual problems. In addition, the performance problems identified by the expert and the decisions made to eliminate them in such systems do not persist, while most of the problems are, as a rule, of a fairly typical nature and can manifest themselves in other parallel applications. The practical value of this methodology is in the development of a system capable of accumulating expert knowledge on identifying typical performance problems of distributed knowledge-based systems and ways to eliminate them.

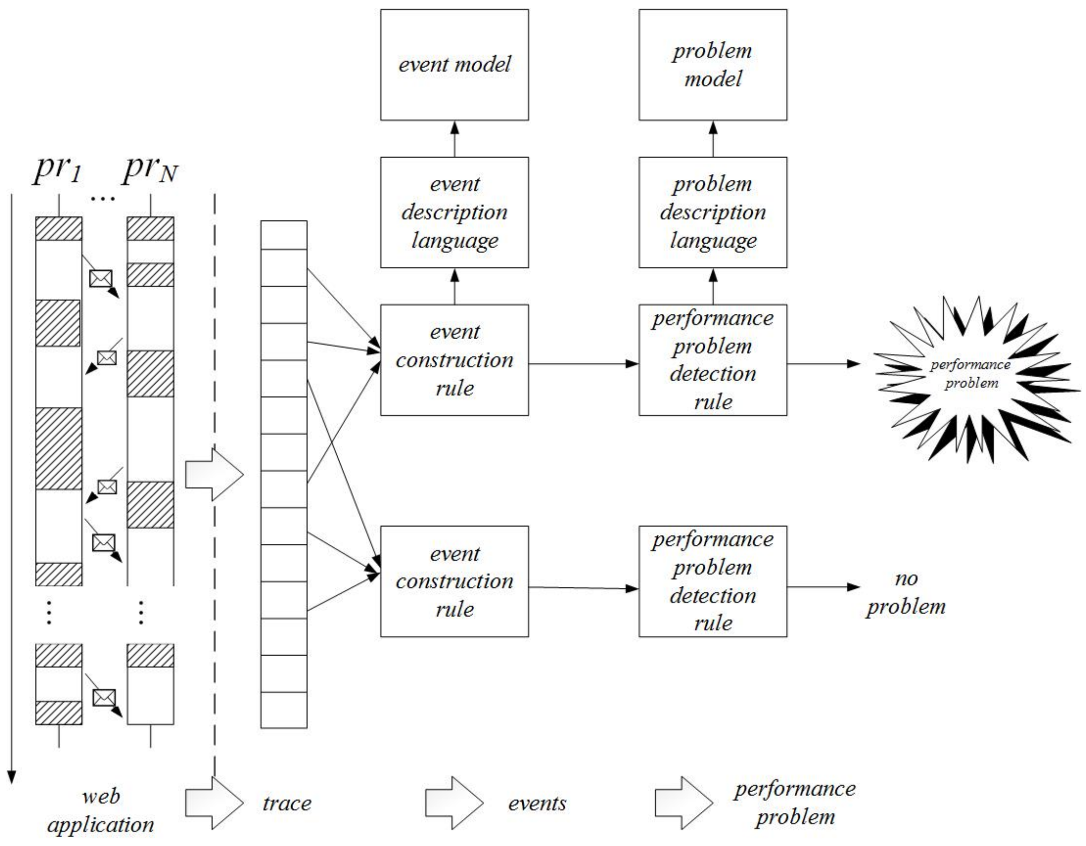

Figure 2: Performance analysis flow chart

All stages of performance analysis are implemented in a software environment for working with ontologies and the knowledge base of distributed knowledge-based systems. Descriptions of typical performance problems were included in the knowledge base of the system (Table 2).

Table 2
Performance challenges for distributed knowledge-based systems

Problem type	Description
The problem of knowledge engineering	- service ontology error - search error
Bandwidth problem	- the limit of service operations has been exceeded - the time of using the service has been exceeded
The concurrency problem	- late message sending - late message reception - sync error - memory access delays
Failure problem	- errors in the service - errors in incoming parameters - network / port errors
The resource problem	- not enough RAM - insufficient physical memory

Code problem	- program code errors - output errors
The problem of knowledge engineering	- service ontology error - search error

The general scheme of the system operation is shown in Fig. 3. Working with the system consists of two stages: preparatory (training stage) and application stage.

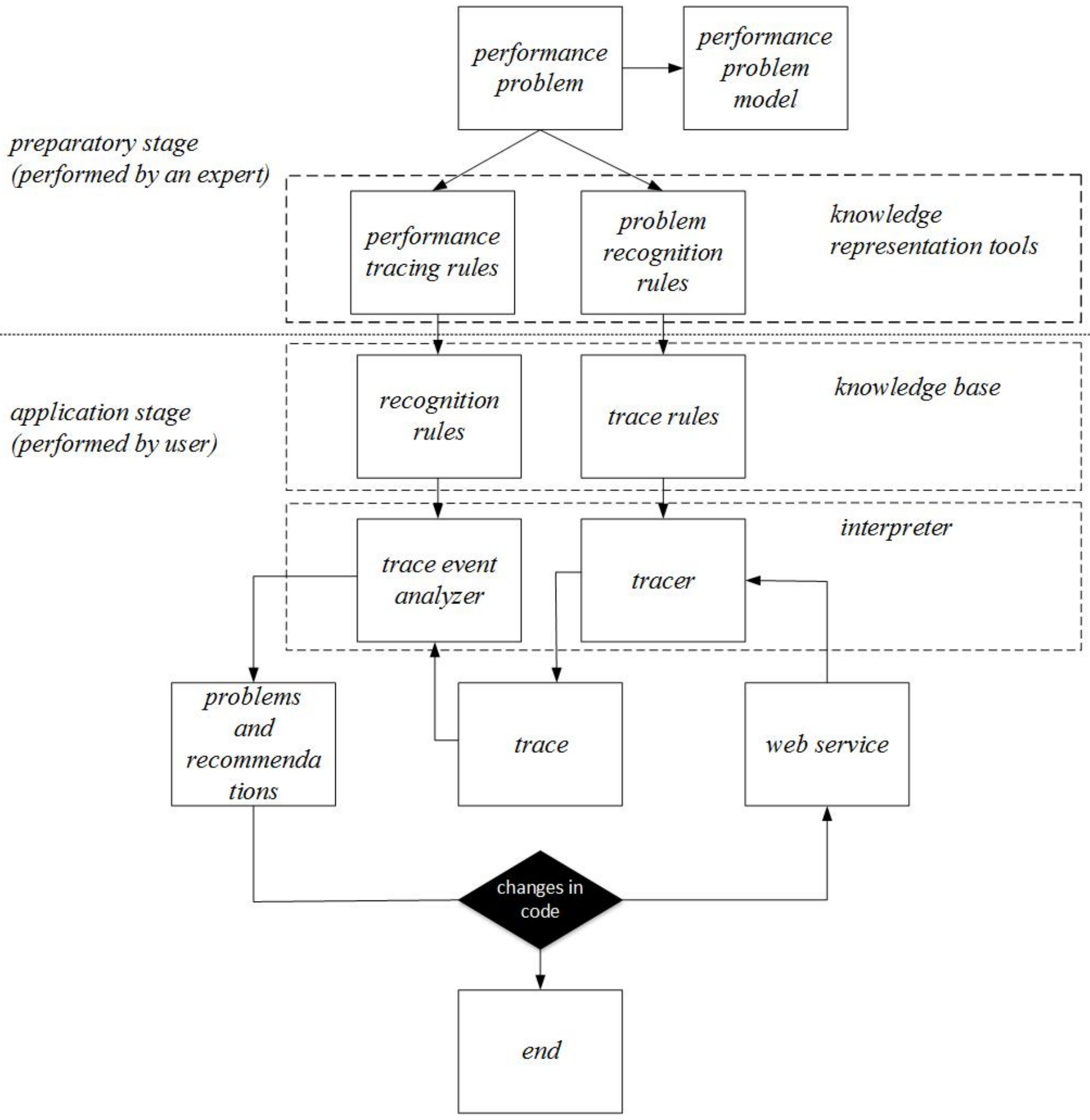
performance problem
performance problem model
preparatory stage (performed by an expert)
performance tracing rules
problem recognition rules
knowledge representation tools
application stage (performed by user)
recognition rules
trace rules
knowledge base
trace event analyzer
tracer
interpreter
problems and recommenda tions
trace
web service
changes in code
end

Figure 3: System operation diagram

The preparatory stage is performed by an expert after identifying the next performance problem. At this stage:

1. The expert describes the actions of the processes that (or a combination of which) can lead to the emergence of the identified problem. Actions are described in the form of a description of the functions of web services, the use of which is associated with the execution of these actions and the rules for tracing these functions.
2. The expert describes the rules for tracing actions and the rules for recognizing the identified problem.
3. The rules described by the expert are added to the knowledge base of the system.

The application phase is performed by the user. At this stage:

1. The system tracer is launched along with the application.
2. During the execution of the application, the tracer, guided by the tracing rules, generates a trace in which all the necessary actions of the application processes are recorded in the form of a sequence of events.
3. After executing the application, the trace analyzer, guided by the rules for recognizing known problems, determines the presence of problems, generates a list of them and recommendations for their elimination.

5. Conclusions

The concept of performance of distributed knowledge-based systems is given, the goals of performance analysis are listed, and it is noted that of a sufficiently large number of known performance criteria (metrics), the execution time of messaging exchange functions is most often used as the main indicator of overhead loss of web services.

This article discusses existing methods for analyzing the performance of distributed systems and methods for measuring performance. It is noted that for web services, direct performance measurement with tracing of web service function calls is most often used. The scientific novelty of the work lies in the presentation of the model of the performance problem of web services as a set of process actions, certain combinations of which, under certain conditions, can lead to an identifiable performance problem.

An overview of existing tools for testing the performance of web services is given. It is noted that most modern tools are based on visualizing performance data, and the rest of the tasks (identifying problems, finding ways and fixing problems) are solved manually by the user. Currently, systems are being developed that allow solving problems of identifying performance problems and finding ways to eliminate them in an automatic mode. It is necessary to set a fixed set of problems that they can analyze, as well as add new problems, without reprogramming the source code. The practical value of this methodology is in the development of a system capable of accumulating expert knowledge on identifying typical performance problems of distributed knowledge-based systems and ways to eliminate them.

All stages of the performance analysis methodology are implemented in a software environment for working with ontologies and a knowledge base of distributed knowledge-based systems. Descriptions of typical performance problems were included in the knowledge base of the system. Thus, the proposed model for representing knowledge in the form of an ontology capable of accumulating expert knowledge.

6. References

[1] N. Kulykovska, S. Skrupsky, T. Diachuk. A model of semantic web service in a distributed computer system. CMIS-2020 Computer Modeling and Intelligent Systems: Proceedings of the Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020) CEUR-WS.org, online. (2020) 338-351.

[2] B. Gregg. Systems Performance: Enterprise and the Cloud. First edition, Upper Saddle River (New Jersey): Prentice Hall, 2013.

[3] N. Herbst [et al.]. Quantifying Cloud Performance and Dependability: Taxonomy, Metric Design, and Emerging Challenges. ACM Trans. Model. Perform. Eval. Comput. Syst. 3, 4, Article 19 (2018) 36 pages. doi: 10.1145/3236332.

[4] A. Balalaie, A. Heydarnoori, P. Jamshidi. Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Softw. 33, 3 (2016) 42-52.

[5] D. Lilja. Measuring computer performance: A Practitioner's Guide, Cambridge: Cambridge University Press, 2000.

[6] A. Grama. Introduction to Parallel Computing, 2nd edition, Boston: Addison-Wesley, 2003.

[7] W. Serrai, A. Abdelli, L. Mokdad, Y. Hammal. Towards an efficient and a more accurate web service selection using MCDM methods. Journal of Computational Science. Volume 22 (2017) 253-267. doi:10.1016/j.jocs.2017.05.024.

[8] L. Li, Allen D. Malony. Automatic Performance Diagnosis of Parallel Computations with Compositional Models. Proceedings of the IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA, USA (2007) 1-8. doi: 10.1109/IPDPS.2007.370401.

[9] S. Tsafarakis, Th. Kokotas, A. Pantouvakis. A multiple criteria approach for airline passenger satisfaction measurement and service quality improvement. Journal of Air Transport Management. Volume 68 (2018) 61-75. doi:10.1016/j.jairtraman.2017.09.010.

[10] R. Jain. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling, New York: Wiley-Interscience (1991).

[11] A. Ghasempour. Internet of Things in Smart Grid: Architecture, Applications, Services, Key Technologies, and Challenges Inventions 4, no. 1: 22 (2019). doi:10.3390/inventions4010022.

[12] A. Sunardi, A. Suharjito. MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based. Procedia Computer Science. Volume 157 (2019) 134-141. doi:10.1016/j.procs.2019.08.150.

[13] Apache JMeter™, 2021. URL: https://jmeter.apache.org/

[14] LoadRunner Professional. Simplified project-based performance load testing solution to quickly identify abnormal application behavior, 2021. URL: https://www.microfocus.com/en-us/products/loadrunner-professional/overview

[15] Visual Studio, 2021. URL: https://visualstudio.microsoft.com/ru/

[16] Tsung 1.7.0 released, 2021. URL: http://tsung.erlang-projects.org/

[17] P. McMullan. An Expert System Approach to Data Distribution and Distribution. Proceedings of the 5th International Conference on Parallel Computing Technologies, St. Petersburg (1999) 487-488.

[18] C. Seragiotto [et al.] On Using Aksum for Semi-Automatically Searching of Performance Problems in Parallel and Distributed Programs. Proceedings of the Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing (2003) 385-392.

[19] S. Subbotin, A. Oliinyk, V. Levashenko, E. Zaitseva. Diagnostic rule mining based on artificial immune system for a case of uneven distribution of classes in sample. Communications - Scientific Letters of the University of Zilina Vol. 18(3) (2016) 3–11.