Experimental Evaluation of the Virtual Environment Efficiency for Distributed Software Development

Pavel Kolyasnikov[1]
Russian Academy of Education,
Russia

Evgeny Nikulchev[2], Iliy Silakov[3], Dmitry Ilin[4]
MIREA–Russian Technological University
Moscow, Russia

Alexander Gusev[5]
Kuban State University
Krasnodar, Russia

***Abstract*—At every software design stage nowadays, there is an acute need to solve the problem of effective choice of libraries, development technologies, data exchange formats, virtual environment systems, characteristics of virtual machines. Due to the spread of various kinds of devices and the popularity of Web platforms, lots of systems are developed not for the universal installation on a device (box version), but for a specific architecture with the subsequent provision of web services. Under these conditions, the only way for estimating the efficiency parameters at the design stage is to conduct various kinds of experiments to evaluate the parameters of a particular solution. Using the example of the Web platform of digital psychological tools, the methods for experimental parameter evaluation were developed in the article. The mechanisms and technologies for improving the efficiency of the Vagrant and Docker cloud virtual environment were also proposed in the paper. A set of basic criteria for evaluating the effectiveness of the configuration of the virtual development environment has been determined to be rapid deployment; increase in the speed and decrease in the volume of resources used; increase in the speed of data exchange between the host machine and the virtual machine. The results of experimental estimates of the parameters that define the formulated efficiency criteria are given as: processor utilization involved (percentage); the amount of RAM involved (GB); initialization time of virtual machines (seconds); time to assemble the component completely (Build) and to reassemble the component (Watch) (seconds). To improve the efficiency, a file system access driver based on the NFS protocol was studied in the paper.**

Keywords—Distributed software development; virtual development environment; increase development efficiency; virtual machines; vagrant; Docker; NFS; webpack

I. Introduction

Currently, the virtual cloud development environments are widely used in the team disturbed development of large projects [1]. This technology uses virtualization and virtual machines configuration management tools [2] to apply the necessary parameters and install the required components with the automation of the synchronization process, configuration and launch of the development environment.

The use of virtual development environments allows developers to avoid differences between the local development environments and the final platform, and greatly simplify the installation and configuration of environments on new machines.

The paper examined a system for the preparation of virtual development environments in Vagrant [3], which allows creating reproducible virtual development environments [4] and reduces the number of difficulties that can arise for the reason of the incompatibility of software and hardware used by developers [5].

The use of a development management system facilitates simultaneous distributed work on several components of the developed software [6, 7] and also automates the process of installing and configuring all the necessary components of the development environment [8-10]. The processes of updating and modifying the components used are also simplified [11], since it suffices only to make changes to the configuration files.

The paper contains the results of a study of increasing the efficiency in virtual software development of a digital psychological research platform [12]. At use of the virtual machines structure close to real servers, following performance problems were discovered [13]: low data exchange rate, as well as instability of work.

The aim of the paper is the development of research methodology and the development of mechanisms for increasing the efficiency of the virtual development environment in the distributed development of large software systems.

The paper consists of seven sections: the 1st is the Introduction, the 2nd is devoted to the formulation of the problem of describing the initial version of the development environment, the 3rd section contains the description of research methods and parameters for evaluating the effectiveness; the 4th section presents the results of experimental evaluation of the initial version of the development environment; the 5th section provides the development of alternative options aimed at improving efficiency; the 6th section contains the evaluation of the effectiveness of the alternatives; the 7th section shows the results.

The paper defines a set of criteria, estimation methods and suggests mechanisms for increasing the efficiency of the virtual work environment in distributed software development.

II. Source Data

The structure of virtual machines, close to the structure of servers involved in the project, is used as the original development environment (Fig. 1). With this approach, each component of the platform corresponds to a separate virtual machine configuration.

In the original development environment, the following virtual machine parameters were chosen for the components:

- API Server: OC ubuntu/xenial64, version 20180424.0.0, 1 CPU, 1024 MB RAM;
- Researcher Account: OC ubuntu/xenial64, version 20180424.0.0, 2 CPU, 1024 MB RAM;
- Psychotest Player: OC ubuntu/xenial64, version 20180424.0.0, 2 CPU, 512 MB RAM.

When developing using a similar structure of the working environment, there is a need to simultaneously run several virtual machines for simultaneous operation of several interconnected components. During the practical use of the developed structure of the virtual development environment on the computers of developers, a significant reduction in performance was noticed. Further observations revealed the following problems of the development environment:

- high workload on the developers' working machines;
- low data exchange rate of virtual machines with the main system;
- high component build time;
- work instability due to increasing loads;
- the need for manual execution of a large number of operations when starting the environment.

During the development of large projects, the number of components being developed increases. At use of such a structure, it can lead to an even greater decrease in computer performance and a decrease in the efficiency of developers [14]. In addition, the development also requires the use of additional software (development environment, web browser, network data entry drivers [15]), which also leads to a significant increase in the workload of the developers' machines, a decrease in the stability of the system and an increase in software execution time involved in the development.

The low data exchange speed of the main system and the virtual machine, observed in using the developed structure of the virtual environment, may be due to two factors: a significant increase in the load on the developer’s machine, but also the driver used by the virtual machine to access the parent file system.

It is also worth to highlight the problem of the need to perform a large number of repetitive actions manually in the development environment start. It also takes considerable time due to the need to wait for the end of each virtual machine before to start the rest of the components. Due to the increasing load on the developer's work machine, this time can also increase.

The problems described above have a significant impact on the speed of software development due to high performance losses and time consumption [16]. It is worth to note that with the expansion of the overall structure of the project, parallel development of several components becomes almost impossible due to an even greater increase in resource requirements.

Thus, it can be concluded that it is necessary to examine approaches to create development environments for software developers and improve their structure. Based on the problems found during the observations, the main requirements for the desired solution for the software development environment were identified:

- rapid deployment;
- increasing the speed and lowering the resources used;
- increasing the speed of data exchange between the host machine and the virtual machine.

It is necessary to evaluate alternative technologies and develop a solution that meets the requirements described above. In the study, it is also necessary to make an experimental assessment of the used and alternative solutions, and to evaluate the expediency of switching to an alternative structure of the development environment.

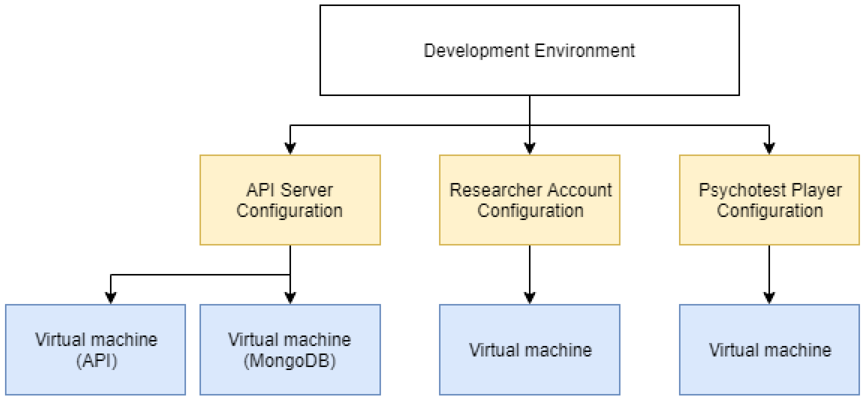

Fig. 1. The Schema of the Original Development Environment.

III. Research Methods

For the experiments described in this paper, measurements of the following indicators were used:

- CPU resources usage (percent);
- the amount of RAM usage (GB);
- virtual machine startup time (seconds);
- component full build time (Build) (seconds);
- component rebuild time (Watch) (seconds).

All experiments were conducted on the same working machine with the following configuration:

- motherboard: Dell 00TMJ3;
- processor: Intel Core i5-5250U;
- RAM: DDR3 12 GB, frequency 800 MHz;
- drive: Samsung SSD 860 EVO 500 GB;
- operating system: Windows 10.

For each component of the virtual development environment, 10 experiments were conducted to measure each of the indicators. To estimate the deviation of the obtained values, Student's coefficient with confidential probability P=0.95 was used.

To measure the processor and RAM resources used, a bash script was developed that compared the figures before the virtual development environment was launched with the figures after the virtual machine started up completely. To get data about the required indicators, the system command “WMIC” was used in the script code. This command provides the possibility of getting the necessary data through a command interface.

All measurements of the indicators were carried out in the waiting state of the virtual machine – after its complete launch and during the work of key components (web server, database server, etc.).

To estimate the virtual machines launch time, the system utility “time” was used. This utility was launched with Vagrant as a prefix in the start up command (“time vagrant up”) and provides information on the time spent for execution after completion. If it is necessary to launch several components of the development environment, launching the necessary Vagrant containers was carried out in parallel.

To estimate the execution time of build tasks (complete “build” assembly and reassembly in “watch” mode), the data provided after the task was completed with the tool for building web applications (Webpack) was used.

IV. Assessment of the Original Development Environment

With the aim of estimating the performance and working speed of the used environment, an experiment was conducted, including measurements of the time taken to start up, as well as an assessment of the CPU resources and RAM used. The results of the experiment are presented in Table I.

According to the data obtained during the experiment, it is concluded about the objectivity of the problems described during the observations, which reinforces the need to find alternative solutions.

TABLE I. Results of Assessing Resource usage by Development Environments

Configuration	**VM count**	**Startup time, sec**	**CPU usage, %**	**RAM usage, GB**
Initial configuration (API Server)	2	289,4 ± 2,8	24,3% ± 6,7%	1,5 ± 0,1
Initial configuration (Researcher Account)	1	118,5 ± 13	21,8% ± 6%	0,5 ± 0,08
Initial configuration (Psychotest Player)	1	155,3 ± 19,1	16,1% ± 3%	0,7 ± 0,04
Initial configuration (API Server, Researcher Account)	3	347,4 ± 4,4	35,4% ± 9,1%	2 ± 0,2
Initial configuration (API Server, Psychotest Player)	3	290,1 ± 4,5	29% ± 5,3%	2 ± 0,06
Initial configuration (3 components)	4	404,8 ± 4,1	49% ± 5%	2,2 ± 0,18

V. Development of Mechanisms for Increasing the Efficiency of the Environment

Due to the high load arising from the use of virtual machines on the basis of Vagrant, it is advisable to consider alternative technologies for the organizing development environments. As an alternative, the Docker technology [17] was considered, which can also be used to organize synchronized development environments [18].

The Docker system is based on the use of abstraction with the help of the built-in Linux kernel virtualization capabilities for isolating various components used within the stand-alone containers with separate environments [19].

Originally, this approach was aimed at delivering the developed software solutions to the server, but later the platform also began to be used for replacing virtual development environments. At the same time, Docker can act as an independent solution [20], or work as the basis of a Vagrant-based solution, thereby replacing the use of virtual machines.

To estimate Docker as an alternative to Vagrant, a number of criteria were identified for comparing these technologies. The results of comparing are shown in Table II.

It can be concluded that both technologies provide comparable advantages for different approaches for similar tasks execution. Using Docker to synchronize and quickly set up the virtual development environment can potentially be a valid solution due to the advantages of containerization and the solutions used within the Docker implementation. However, this approach also obliges to develop a new structure of components using Docker containers.

TABLE II. COMPARING VAGRANT AND DOCKER

Criterion	Vagrant	Docker
License	MIT	Apache 2.0
Developer	Hashi Corp.	Docker, Inc.
Platform supported	Linux, Windows, MacOS	Linux, Windows, MacOS
Purpose	Virtual development environment	Containerization of applications and automation of tasks, components delivery to the server
Ease of use	Starting the environment with the use of one command	An understanding of the container system and dependencies is necessary; launching the environment with one command in combination with Vagrant
Ease of configuration	Configuration file written on Ruby	Native format of configuration file
Container structure	The container includes all the dependencies specified in the configuration. The container is only the runtime environment	Each component and its dependencies are separate containers

In the context of the developed platform, an important task was to maintain maximum proximity to the work systems environment on remote servers. In this case, changing the structure of components for use in the form of Docker containers inevitably causes a discrepancy between the developer's environment and the server system environment, and also makes it difficult to support an existing solution in the context of the developed platform. Therefore, the use of containerization technology becomes impractical.

Another possible solution in this case may save Vagrant as the basis of the chosen solution, but using a single virtual machine to run all the components.

As an alternative solution, it is worth to consider a configuration based on a single virtual environment in Vagrant. In this case, each component runs within one virtual machine, using also a single modular Vagrant configuration (Fig. 2).

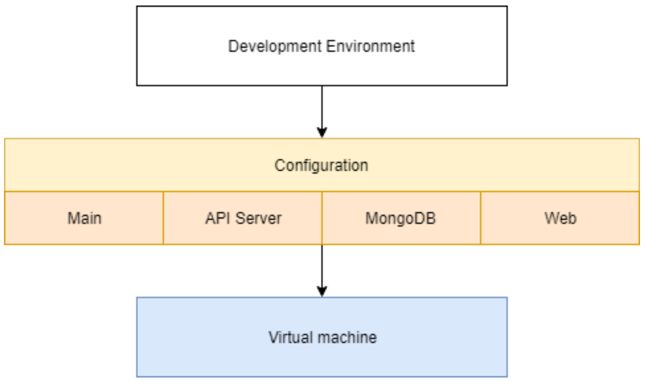

Fig. 2. Alternative Structure of Development Environment.

Use of such a structure should reduce the load on the computer by use of only one virtual machine. At the same time, the use of a modular configuration based on the original solution should simplify support and minimize discrepancies with the server environment.

To compare the original and alternative structures of the development environment, an experimental assessment of the resources consumed was carried out (Table III). As a virtual machine configuration, the alternative solution was based on ubuntu/xenial64 OS, version 20180424.0.0 with 2 CPUs, 2048 MB RAM.

With the aim of increasing the efficiency, it is also worth to examine alternative driver of data exchange with parent system based of NFS protocol [18], since such a solution can significantly increase the speed of working with the file system and the tasks execution [21].

An additional driver is necessary to use NFS on the machines with Windows OS management [22]. In addition, it is necessary to involve bindfs [23] extension that allows to transfer the access rights from parent system.

Due to the architectural feature of the NFS protocol, which does not provide an implementation of system signals in file changes tracking [24], it is also necessary to modify the configuration of the Webpack software [25] used to build web components. The following changes were made:

- to ensure compatibility with NFS, the "watchOptions.poll" option was installed, implementing a workaround of monitored files at a specified time interval;
- to exclude unchangeable library files from the build, the "watchOptions.ignore" option was used.

Additional measurements were taken for the time spent to build a component after configuration changes (Table IV).

TABLE III. MEASUREMENTS RESULTS OF THE DEVELOPMENT ENVIRONMENTS PERFORMANCE

Configuration	VM count	Startup time, sec	CPU usage, %	RAM usage, GB
Initial configuration (3 components)	4	404,8 ± 4,1	49% ± 5%	2,2 ± 0,18
Alternative configuration (3 components)	1	210,3 ± 4,6	23,8% ± 7,7%	1,3 ± 0,13

TABLE IV. RESULTS OF COMPONENT BUILDING TIME (SEC)

Configuration	Researcher Account		Psychotest Player	
	Build	Watch	Build	Watch
Initial configuration (3 components)	120,4 ± 2,7	5 ± 0,4	131 ± 3,1	4,4 ± 0,1
Alternative configuration (3 components)	100,7 ± 2,5	3,6 ± 0,3	82,4 ± 2,4	3,3 ± 0,1
Improved alternative configuration (3 components)	90,1 ± 2,4	1,2 ± 0,3	79 ± 1,3	1,1 ± 0,2

Such configuration changes are also caused an start time increase (from 210.3 to 227.7 seconds in average), associated with the addition of the waiting time for NFS driver starting and the mounting the directory in virtual machine, and also used CPU resources increase (from 24% to 29% in average). However, indicators of resources consumed still remain significantly lower compared with the full launch of all components in original development environment. In this case, a decrease in the time spent on the components build operations was observed in using the proposed improvements.

VI. Results of the Experiments

Experiments were conducted to estimate the launch time and the load on the computer for various configurations of the virtual development environment. The results of the experiments are given in Table V.

The results of experiments on evaluating the execution time of the complete component assembly (Build) and component reassembly (Watch) for the Platform components of the Researcher Account and Psychotest Player are presented in Table VI.

From the results of the experiments it can be seen that the proposed alternative configuration exerts a significantly lower load on the CPU (Fig. 3). An improved alternative configuration uses slightly more processor resources, which is associated with the use of additional driver, but even in this case, the load is much lower than with the full launch of the original development environment.

Similar changes are also noticeable in the used RAM comparing (Fig. 4).

TABLE V. The Results of the Evaluation of Development Environment Start up Time and Performance

Configuration	**VM count**	**Startup time, sec**	**CPU usage, %**	**RAM usage, GB**
Initial configuration (API Server)	2	289,4 ± 2,8	24,3% ± 6,7%	1,5 ± 0,1
Initial configuration (Researcher Account)	1	118,5 ± 13	21,8% ± 6%	0,5 ± 0,08
Initial configuration (Psychotest Player)	1	155,3 ± 19,1	16,1% ± 3%	0,7 ± 0,04
Initial configuration (API Server, Researcher Account)	3	347,4 ± 4,4	35,4% ± 9,1%	2 ± 0,2
Initial configuration (API Server, Psychotest Player)	3	290,1 ± 4,5	29% ± 5,3%	2 ± 0,06
Initial configuration (3 components)	4	404,8 ± 4,1	49% ± 5%	2,2 ± 0,18
Alternative configuration (3 components)	1	210,3 ± 4,6	23,8% ± 7,7%	1,3 ± 0,13
Improved alternative configuration (3 components)	1	227,7 ± 3,4	28,7% ± 5,4%	1,2 ± 0,09

TABLE VI. The Results of the Evaluation of the Component Build Time

Configuration	**Researcher Account**		**Psychotest Player**	
	Build	**Watch**	**Build**	**Watch**
Initial configuration (API Server)	101,9 ± 2,9	3 ± 0,4	86,1 ± 2,1	3,1 ± 0,1
Initial configuration (Researcher Account)	100,6 ± 2,5	2,7 ± 0,4	91,8 ± 2,4	3,7 ± 0,1
Initial configuration (Psychotest Player)	99 ± 2,4	3,6 ± 0,4	89,3 ± 3	3,7 ± 0,1
Initial configuration (API Server, Researcher Account)	98,7 ± 2,2	3,7 ± 0,5	119,2 ± 2,4	4 ± 0,2
Initial configuration (API Server, Psychotest Player)	94,6 ± 2,7	4,1 ± 0,4	137,7 ± 3,6	4 ± 0,2
Initial configuration (3 components)	120,4 ± 2,7	5 ± 0,4	131 ± 3,1	4,4 ± 0,1
Alternative configuration (3 components)	100,7 ± 2,5	3,6 ± 0,3	82,4 ± 2,4	3,3 ± 0,1
Improved alternative configuration (3 components)	90,1 ± 2,4	1,2 ± 0,3	79 ± 1,3	1,1 ± 0,2

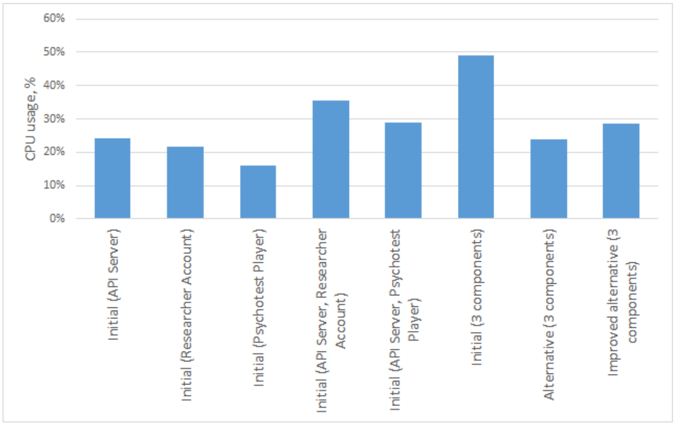

Fig. 3. Average CPU Load, in Percent.

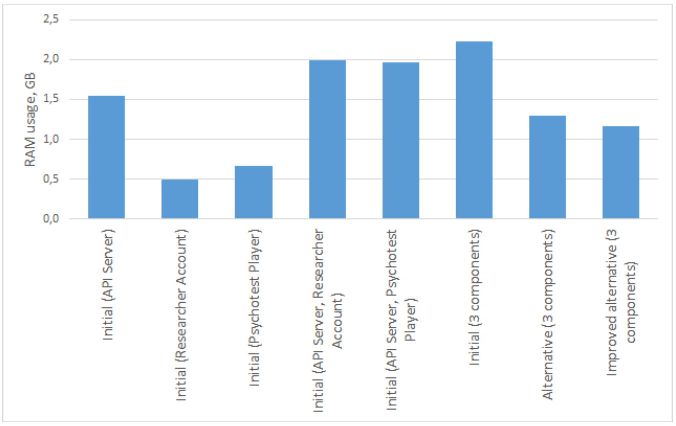

Fig. 4. Average RAM used, GB.

Reducing the CPU load and the number of used virtual machines led to a significant decrease in the average time required to start the virtual development environment (Fig. 5). The improved alternative configuration is lower that alternative configuration without additional modifications, like in case with the load on the processor. This may also be due to the use of an additional driver and the need to wait for its initialization.

However, improved alternative configuration leads in performance in components build (Fig. 6), and also shows significantly better performance indicators in rebuild (Fig. 7). Such improvements are a direct consequence of using the NFS driver, which increases the speed of data exchange with the host machine. It significantly reduces the time required to build components.

Therefore, the acquired improved alternative configuration of the development environment for Vagrant shows a significant reduction in the load on the processor and RAM in comparison with the full launch of the original development environment. It also shows a significant acceleration of the component building process, which reduces the waiting time on the part of the developers and thereby increases their efficiency.

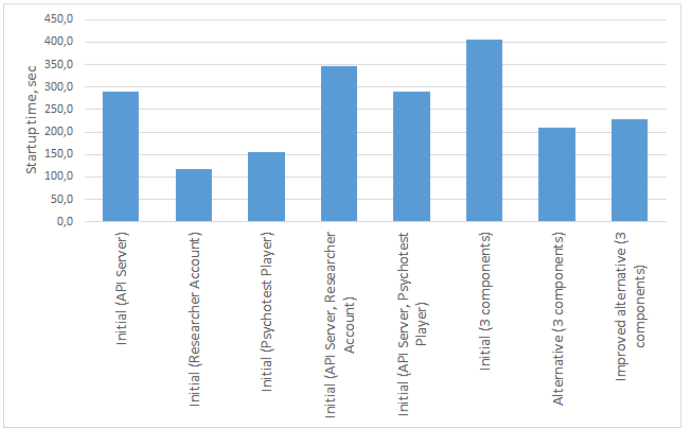

Fig. 5. Average Startup Time, Sec.

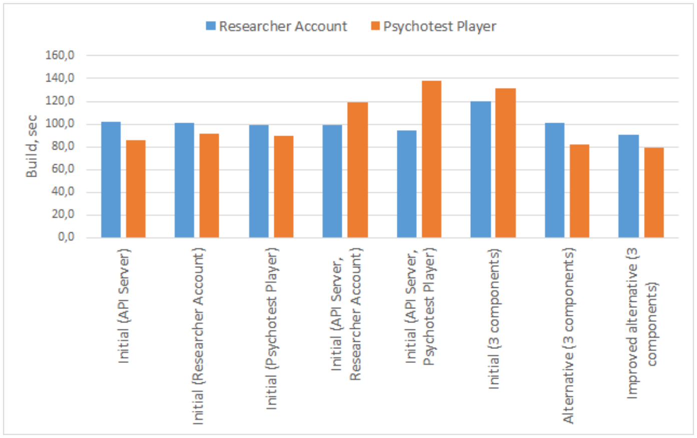

Fig. 6. Average Execution Time for the Complete Assembly of the Components for the Researcher Account and the Psychotest Player (Build), Sec.

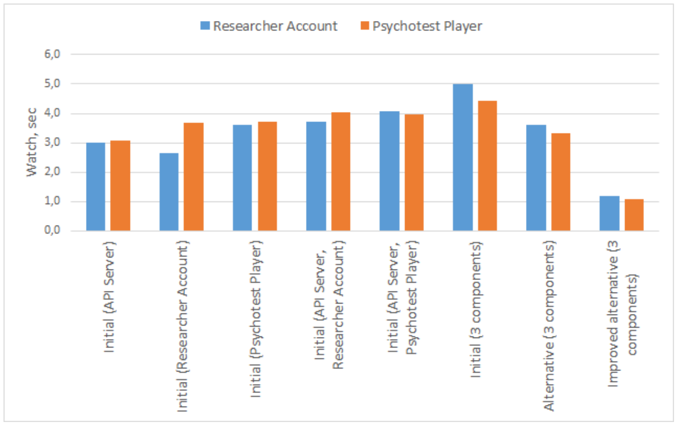

Fig. 7. Average Execution Time for the Reassembly of the Components for the Researcher Account and the Psychotest Player (Watch), Sec.

VII. Discussion

In the process of research and development of mechanisms to improve the effectiveness of the original software development environment, it was found that the alternative configuration of Vagrant based on a single virtual machine for all existing platform components provides a significant reduction in both the startup time and computer load.

Both Vagrant and Docker can be used to organize a software development environment. It is reasonable to select between them with respect to the technologies used at the server in order to ensure that the development and the deployment environments are as identical as possible.

Additional research may be conducted to measure the increase in the number of components of the platform and compare the performance of the original and alternative Vagrant configurations. A separate study can provide a more detailed consideration of the Docker container technology as a solution for the organization of the software development environment.

When choosing technologies and preparing the software development environment, it is necessary to consider the whole architecture of the application being developed and the particular features of its operation. It makes sense to choose Vagrant when using virtual machines and the Docker in the case of the container technologies. It should also be borne in mind that the hybrid design can be implemented with both the technologies used. Although, one shouldn't exclude the use of other technologies and configurations. In this regard, to assess the performance of the software development environment, it is necessary to conduct separate experiments and develop one's own testing methodology, which can be completely different from that presented in this article.

Switching from several virtual machines to the single one may not be allowed when configuring certain development environments. For example, if it is imperative to isolate a particular component of the system. In this case, the increase in the productivity of the software development environment might not be achieved and the developer will have to look for alternative approaches to solve this problem.

VIII. Conclusions

Due to the increase of web projects development complexity and working environments setting up, there is a need to use virtual development environments.

At use of virtual development environment, which structure contains an autonomous virtual machine for each component, some problems with high resource consumption and a reduction in the performance of the developers' working machines were noticed, which made it necessary to consider alternative solutions.

The paper described an analysis of the reasons of performance reduction at use of development environments based on virtualization. The main technologies used for the development of virtual development environments are considered, and an improved structure is proposed. In addition, an experimental assessment of the original and alternative solutions was made.

The assessment of the configuration of the original software development environment showed that using a separate virtual machine for each of the components of the psychological platform to get as close as possible to the server structure was not an effective approach. At the developers' computers, there was a significant drop in performance, a low data exchange rate with the VM and a high component assembly time.

In the search for a solution, the containerization technology based on Docker and the alternative configuration of Vagrant using a single virtual machine for all existing components were considered. However, using Docker requires developing a completely new component structure and will cause the developer's environment to differ from the server environment. Therefore, the alternative configuration of Vagrant is the most preferred option within the framework of the developed platform. The measurements showed that the use of the alternative configuration significantly reduced the startup time and reduced the load on the computer.

As an efficiency enhancement, a driver based on the NFS protocol was applied and the configuration of the Webpack system was modified. The measurements showed that the use of the NFS entailed a slight increase in the time for launching the virtual machine but reduced the time for assembling the components of the platform.

It was shown that in the case of the open digital platform for mass psychological research the application of the alternative Vagrant configuration with the NFS driver and the Webpack optimization provided a significant performance boost compared to the original configuration.

The implemented alternative solution based on a single development environment showed significantly lower resource consumption, as well as a reduction in the tasks building time with the help of the driver for accessing file system based of the NFS protocol.

Therefore, the use of a virtual development environment based on a single virtual machine using the NFS driver can significantly reduce the workload of the developers' computers. This increases rapidity and reduce time consumption, which improves the developers' efficiency.

Conducted studies, including the stages of parameter estimation, the introduced characteristics and criteria, can be the basis for the formation of a methodology for the experimental evaluation of the software development environment configurations, which would allow choosing effective solutions at the design stage.

Acknowledgment

The work was financed by the Ministry of Science and Higher Education of Russia, project 25.13253.2018 / 12.1 "Development of the technological concept of the Data Center for Interdisciplinary Research in Education".

References

[1] Caballer M., Blanquer I., Moltó G., de Alfonso C. (2015) Dynamic management of virtual infrastructures, Journal of Grid Computing, 13(1), 53-70. doi: 10.1007/s10723-014-9296-5

[2] Giannakopoulos I., Konstantinou I., Tsoumakos D., Koziris N. (201) Cloud application deployment with transient failure recovery, Journal of Cloud Computing, 7(1), 11. doi: 10.1186/s13677-018-0112-9

[3] Vagrant, 2019. Available at: https://www.vagrantup.com/ (accessed 27.03.2019).

[4] Spanaki P., Sklavos N. (2018) Cloud Computing: Security Issues and Establishing Virtual Cloud Environment via Vagrant to Secure Cloud Hosts. In Computer and Network Security Essentials. Springer, pp. 539-553. doi: 10.1007/978-3-319-58424-9_31

[5] Hashimoto M. (2013) Vagrant: Up and Running: Create and Manage Virtualized Development Environments. O'Reilly Media Inc, 2013.

[6] Xuan N. P. N., Lim S., Jung S. (2017) Centralized management solution for vagrant in development environment, In Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication. ACM,. art. no. 37. doi: 10.1145/3022227.3022263

[7] Thompson C. (2015) Vagrant virtual development environment cookbook. Packt Publishing Ltd.

[8] Mouat A. (2016) Using Docker: Developing and Deploying Software with Containers. O'Reilly Media Inc.

[9] Sammons G. (22016) Learning Vagrant: Fast programming guide. CreateSpace Independent Publishing Platform.

[10] Peacock, M. (2015) Creating Development Environments with Vagrant. Packt Publishing Ltd.

[11] Iuhasz G., Pop D., Dragan I. (2016) Architecture of a scalable platform for monitoring multiple big data frameworks, Scalable Computing: Practice and Experience, 17(4), 313-321. doi: 10.12694/scpe.v17i4.1203

[12] Nikulchev E., Ilin D., Kolyasnikov P., Belov V., Zakharov I., Malykh S. (2018) Programming Technologies for the Development of Web-Based Platform for Digital Psychological Tools, International Journal of Advanced Computer Science And Applications, 9(8), 34-45. doi: 10.14569/IJACSA.2018.090806

[13] Kashyap S., Min C., Kim T. (2016) Opportunistic spinlocks: Achieving virtual machine scalability in the clouds, ACM SIGOPS Operating Systems Review, 50(1), 9-16. doi: 10.1145/2903267.2903271

[14] Saikrishna P. S., Pasumarthy R., Bhatt N. P. (2017) Identification and multivariable gain-scheduling control for cloud computing systems, IEEE Transactions on Control Systems Technology, 25(3), 792-807. doi: 10.1109/TCST.2016.2580659

[15] Li J., Xue S., Zhang W., Qi Z. (2017) When i/o interrupt becomes system bottleneck: Efficiency and scalability enhancement for sr-iov network virtualization, IEEE Transactions on Cloud Computing, Early Access. Doi: 10.1109/TCC.2017.2712686

[16] Basok B.M., Zakharov V.N., Frenkel S.L. (2017) Iterative approach to increasing quality of programs testing, Russian Technological Journal, 5(4), 43-12.

[17] Docker, 2019. Available at: https://www.docker.com/ (accessed 27.03.2019).

[18] Chen M., Bangera G. B., Hildebrand D., Jalia F., Kuenning G., Nelson H., Zadok E. (2017) vNFS: maximizing NFS performance with compounds and vectorized I/O, ACM Transactions on Storage. 13(3), 21. doi: 10.1145 / 3116213

[19] Kane S.P., Matthias K. (2018) Docker: Up & Running: Shipping Reliable Containers in Production. O'Reilly Media Inc, 2018.

[20] Peinl R., Holzschuher F., Pfitzer F. (2016) Docker cluster management for the cloud-survey results and own solution, Journal of Grid Computing, 14(2), 265-282. doi: 10.1007/s10723-016-9366-y

[21] Krieger, M. T., Torreno, O., Trelles, O., & Kranzlmüller, D. Krieger M. T. et al. (2017) Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows, Future Generation Computer Systems, 67, 329-340. doi:10.1016/j.future.2016.02.008

[22] Vagrant WinNFSd–GitHub, 2019. Available at: https://github.com/ winnfsd/vagrant-winnfsd (accessed 27.03.2019).

[23] Vagrant bindfs – GitHub, 2019. Available at: https://github.com/gael-ian/vagrant-bindfs (accessed 28.03.2019).

[24] Dani S. A. (2017) JavaScript by Example. Packt Publishing.

[25] Webpack, 2019. Available at: https://webpack.js.org/ (accessed 28.03.2019).