Application of user dew agent in hybrid-computing environments

A A Pashinin and V G Bogdanova

Matrosov Institute for System Dynamics and Control Theory SB RAS, Lermontov St. 134, Irkutsk, Russia, 664033

apcrol@gmail.com

Abstract. In recent years, research in Dew computing as a new layer in the vertical hierarchy of scalable computing has been developing. Directions for the development of this technology include research into the possibilities of Dew computing and its applications. Our research belongs to the second direction. We use this technology in a package of applied microservices for solving complex problems of the qualitative study of binary dynamic systems based on the Boolean constraint method in a hybrid cloud environment. This environment includes on-premises, high-performance, and cloud resources. In this paper, the usage of Dew computing technology in a hybrid cloud environment is discussed. A user dew agent installed on an on-premises resource is proposed to provide the possibility of the applied microservices package collaboration with cloud services in a hybrid cloud environment. The advantages of using a user dew agent are considered. The architecture, functionality, operation modes, and the installation of the dew agent are presented.

1. Introduction

In recent years, beyond well-known technologies such as Cloud and Fog computing, a new paradigm has emerged, Dew Computing (DC). This paradigm combines the concept of Cloud computing with the capabilities of on-premises resources. DC is oriented on a microservice approach in a vertical hierarchy of heterogeneous distributed computing [1]. Dew-Server (DS) is one of the Dew-architecture components. Its main functions are providing cloud services for on-premises resources and synchronizing the DS database with the cloud database.

The objective of our research is to develop software that provides the use of DC in an applied microservices package (AMP) for solving complex problems of the qualitative study of binary dynamic systems (BDS) based on the Boolean constraint method [2] in a hybrid cloud environment. Following this method, a Boolean model of the required BDS property is constructed using the BDS dynamic description and the property specification. For BDS functioning on a finite time interval, the obtained Boolean model is in the form of Boolean constraints or a Quantified Boolean formula. These problems are correspondently in the complexity classes NP and PSPACE. Efficient SAT or QSAT solvers are required for verifying the Boolean constraints satisfiability or QBF truth. The software for constructing property models and verifying their satisfiability is implemented as microservices. The advantage of the AMP implementation based on the microservice-oriented technology is described in [3]. AMP microservices are divided into intensive- and non-intensive using computational resources. In the first case, their installation requires high-performance resources. In the second, local user resources are sufficient. The SAT and QSAT solvers require high-performance computations in dependency on the problem dimension. These characteristics determine the use of a hybrid infrastructure for their deployment, consisting of cloud and on-premises resources, including on-premises clusters. The use of DC technology in such infrastructure ensures the following possibilities:

– The availability of computational microservices installed on local computers during the temporary absence of the Internet;
– Scaling of computing to cloud resources when peak loads occur (if there is a network connection);
– Automatic synchronization of cloud and local resources when the connection is restored.

The HPCSOMAS agents provide access to AMP and the control of computations. These agents are implemented using the class library of the HPCSOMAS-MSC platform [4]. In this study, we propose a User Dew Agent (UDA) based on the class of HPCSOMAS-agent with extended functionality. UDA is intended for supporting data synchronized distributed AMP-based computing in a hybrid cloud environment. These new UDA possibilities include analogical DS functions mentioned above.

This paper has the following structure. Section 2 discusses publications close to the research topic. In Section 3, the architecture and functional possibilities of the UDA are presented. The UDA functioning is described in Section 4. In Section 5, several aspects of the proposed software are considered. Section 6 provides a conclusion on the results achieved during the study.

2. Related work

DC definitions covering various aspects of this new computing paradigm are given in [1, 5-7]. In [1], a hierarchy of scalable distributed computing is presented, in which DC is at the ground layer. In [6], the goal of DC is defined, which is to ensure the full realization of the potential of local computers (non-cloud resources) and cloud services. In [7], the description of DC includes features, such as independence and collaboration of dew- and cloud services. The first feature is the ability to work without an Internet connection, and the second is the ability of software installed on on-premises resources to automatically synchronize information with the cloud when this connection is restored [8]. In [9], a detailed comparison of post-cloud technologies is given by some criteria, and the difference between cloud and dew computing models is considered. In the first case, online applications are launched from users' computers using cloud services. In the second, part of the functionality and data are transferred to on-premises resources while using the potential of cloud services and distributed computational resources remains possible. In [5], the Cloud-Dew architecture is presented, which is an extension of the client-server architecture, and a special kind of web server, DS, is proposed. The differences between DS and the usual server, namely, servicing only one client on whose resource the DS is installed, the need for periodic synchronization, and interaction with the network only for the synchronization process, are discussed in [10].

In [11], three research directions in dew computing are identified: early research, DC capabilities, and DC applications. The studies of DC in Industrial Internet of Things (IIoT) are considered in [12, 13]. In [12], the feasibility of IIoT devices as the dew devices is examined. The research, evaluation, analysis, and discussion of smartwatches for the DC environment are shown in [13]. Our research is devoted to scientific computations in DC. In our previous studies [14], DC technology was used for solving the problem for the parametric synthesis of the stabilizing regulator for controlled dynamic objects in a heterogeneous distributed computing environment. As noted in [11, 12, 13], this study is referred to the using DC technology in high-performance computing. In the current study, we propose the DC-based UDA providing access for the AMP for solving problems of qualitative analysis and parametric synthesis of BDS based on the Boolean constraint method in a hybrid cloud environment [15].

3. Architecture and the functional possibility of the UDA

UDA provides the following possibilities:

- A web-interface for the user problem formulation;
- Formation of the execution scheme of this problem (a further task for brevity);
- Organizing of launch and monitor the execution of sub-tasks of this problem;
- Access to AMP services, the user knowledge base, computational results on external HPCSOMAS-agents (in the cloud or on-premises clusters, etc.), and automatic synchronization of these data with correspondent data on local user resources.

The UDA architecture is presented in figure 1. The UDA has a web-interface and executable agents for performing its commands. UDA subsystems can be divided as follows: functioning in the same way as HPCSOMAS-agent subsystems, new and modified ones. *Synchronization*, *Authentication*, and *Authorization* managers are new. These managers are involved in ensuring data synchronization and provision of cloud services for on-premises resources.

New and modified UDA managers are intended for performing the following functions:

- *Synchronization manager* synchronizes user data (files, calculation results) on UDA with data on HPCSOMAS agents on resources.
- *Configuration manager* provides tools to customize the agent.
- *Task distribution* and *Task managers* control the execution of tasks, respectively, only on the on-premises or external resources.
- *File manager* performs the control of files. Files can be both local and synchronized with other agents.
- *Authentication* and *Authorization managers* provide a fast connection via access keys that the user can download from HPCSOMAS agents and install on the UDA.

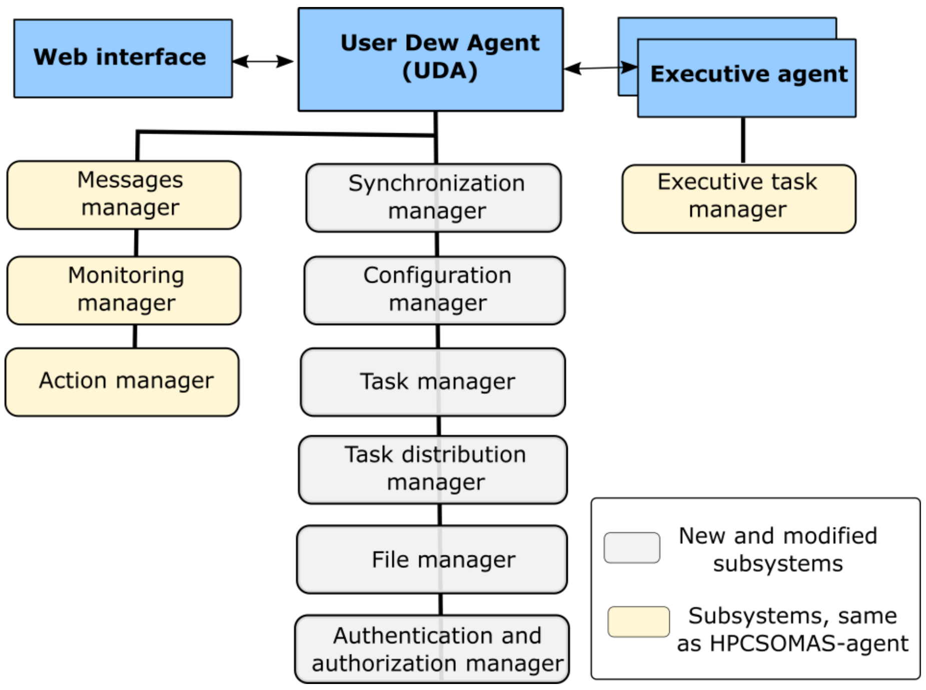


Figure 1. The architectural components of the UDA.

The advantage of using the UDA on an on-premises computer is the ability to run computationally uncomplicated tasks on services locally, even without access to the Internet. Also, an evident advantage is the ability to synchronize user data, such as downloaded files and computation results, from remote resources and saving them in the absence of communication or other situations that lead to data loss.

The UDA, unlike the HPCSOMAS agent, only sends tasks and does not accept them from other agents. After starting on the user's computer, the UDA runs in the background mode and, if necessary, checks the need for data synchronization with a specified frequency.

The UDA notifies the user about the status of tasks received from him and directed to other agents. After task completion, these agents return UDA results. If it is not online, the results are stored in the user folder on this agent. After the connection is restored, the UDA will notify the agents about this fact, synchronize all previously not transferred results, and send the user a notification.

3.1. UDA functioning modes

The UDA has two modes of processing task, depending on its intended use:

- Local assembly mode of subtasks. In this mode, the UDA collects intermediate results and launches the following subtask with the obtained results. This mode is suitable when part of the computations is performed directly on the user's computer.
- Task clone mode. This mode provides the creation of a duplicate task on an external resource of the computing infrastructure. The UDA submits the task clone for processing to the agent of this computational resource.

By default, the UDA uses clone mode if it is not involved in computations, and the task can be performed on agents of external resources.

3.2. The UDA implementation and installation

The UDA is implemented as a servlet that runs within the Apache TomEE web server [16]. The user can download the installation wizard for the local client of the HPCSOMAS-MSC. This wizard carries out the following actions. The Apache TomEE is installed on the user's computer, with previously installed UDA and some AMP computational microservices in the servlet container. These microservices are non-resource intensive and compatible with the user operating system installed on his computer. During installation, the user only needs to select a port for his work and the path to the folder where agents will store working files.

The user can connect his UDA to HPCSOMAS agents on remote computing resources and use its web interface in the future to send tasks for computations. UDA user interface is based on JSP and JavaScript technologies [17]. The HTTP protocol is used to transfer commands and exchange messages. The UDA is connected to external agents in the same way as agents on computing resources communicate with each other – using access key-files. Such files contain data for authorization, names, and network addresses of servers. The user authorized in the computing resource agent can upload the authorization key to his account. After that, the key is used to connect this agent to its local UDA.

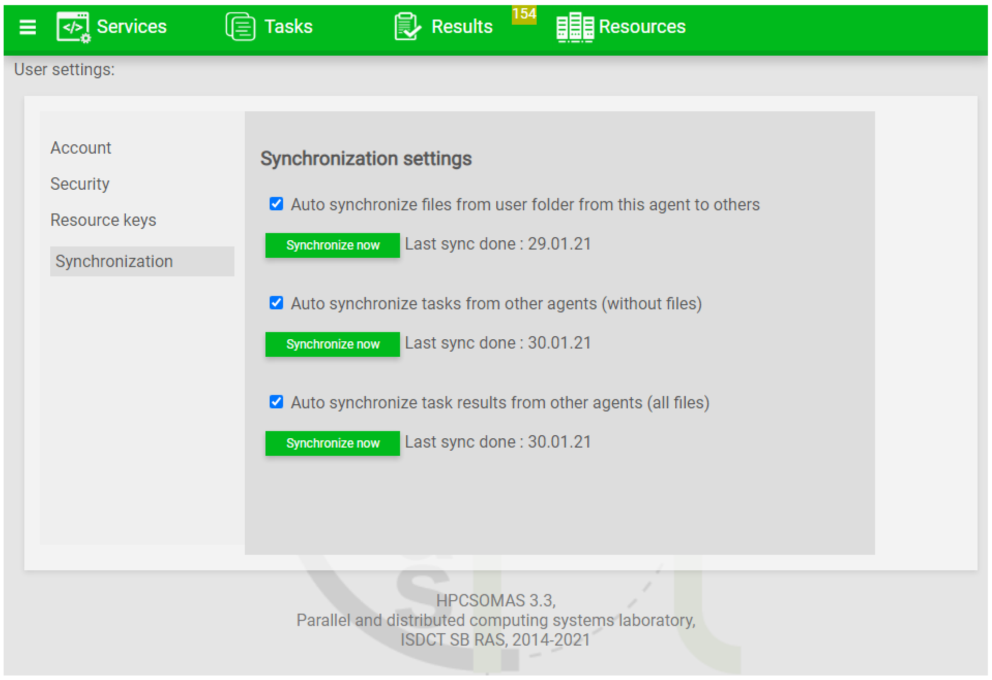
3.3. UDA settings

UDA web-interface includes the following sections: Services, Tasks, Results, and Resource (Figure 2). In the Services section, access to the available computational microservices is provided. Launched and completed tasks are listed in section Tasks. In section Resources, available resources characteristics are shown. The Results section contains the results of both this agent and all others connected using resource keys. In the section Results (figure 2), there are 154 results of completed tasks, which can be viewed. User settings are edited on the side panel. The UDA provides the following special user settings (Figure 2):

- Resource keys – access keys to other agents for downloading lists of available services, task results, and user files;

– Synchronization – settings for automatic synchronization of user files and completed tasks.

All task result files can be downloaded but stored physically on this or other agents, depending on the settings.



User settings:

Account
Security
Resource keys
Synchronization

Synchronization settings

☑ Auto synchronize files from user folder from this agent to others

Synchronize now Last sync done : 29.01.21

☑ Auto synchronize tasks from other agents (without files)

Synchronize now Last sync done : 30.01.21

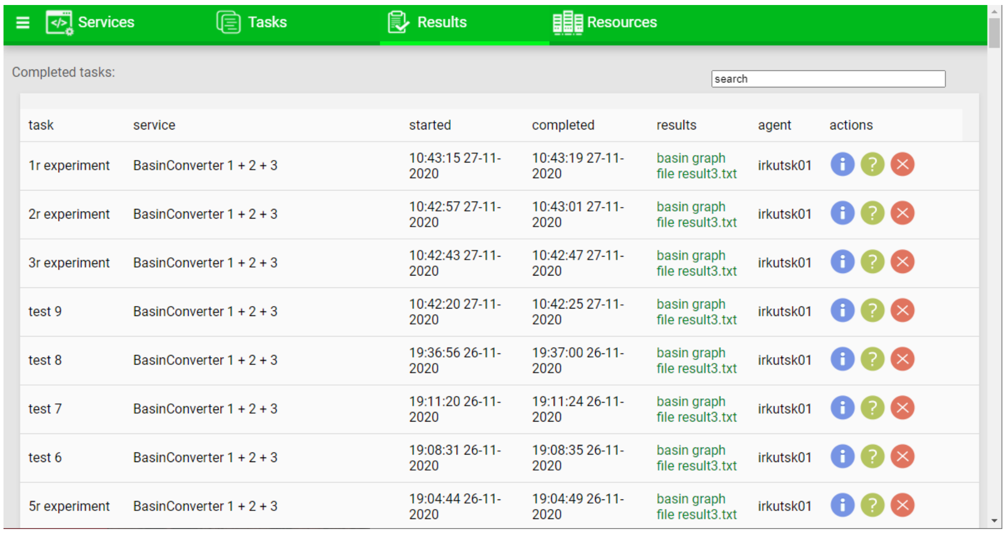
☑ Auto synchronize task results from other agents (all files)

Synchronize now Last sync done : 30.01.21

HPCSOMAS 3.3,
Parallel and distributed computing systems laboratory,
ISDCT SB RAS, 2014-2021

Figure 2. UDA web-interface.

In figure 3, the example of listed result files is shown. The search bar allows finding the completed task by the name, service, result file, and the agent on which it ran. The action (denoted by "**i**") allows viewing the following task characteristics: name and ID, user name, queue and completion time, computational subtasks created at runtime, etc. The action (denoted by "**?**") allows viewing task parameters.



Completed tasks:

task	service	started	completed	results	agent	actions
1r experiment	BasinConverter 1 + 2 + 3	10:43:15 27-11-2020	10:43:19 27-11-2020	basin graph file result3.txt	irkutsk01	i ? ✕
2r experiment	BasinConverter 1 + 2 + 3	10:42:57 27-11-2020	10:43:01 27-11-2020	basin graph file result3.txt	irkutsk01	i ? ✕
3r experiment	BasinConverter 1 + 2 + 3	10:42:43 27-11-2020	10:42:47 27-11-2020	basin graph file result3.txt	irkutsk01	i ? ✕
test 9	BasinConverter 1 + 2 + 3	10:42:20 27-11-2020	10:42:25 27-11-2020	basin graph file result3.txt	irkutsk01	i ? ✕
test 8	BasinConverter 1 + 2 + 3	19:36:56 26-11-2020	19:37:00 26-11-2020	basin graph file result3.txt	irkutsk01	i ? ✕
test 7	BasinConverter 1 + 2 + 3	19:11:20 26-11-2020	19:11:24 26-11-2020	basin graph file result3.txt	irkutsk01	i ? ✕
test 6	BasinConverter 1 + 2 + 3	19:08:31 26-11-2020	19:08:35 26-11-2020	basin graph file result3.txt	irkutsk01	i ? ✕
5r experiment	BasinConverter 1 + 2 + 3	19:04:44 26-11-2020	19:04:49 26-11-2020	basin graph file result3.txt	irkutsk01	i ? ✕

Figure 3. Viewing and processing result

4. UDA functioning

The UDA servlet has a functioning cycle (figure 4) in the main thread. In this figure, dotted arrows represent sending messages and data. All actions in this cycle occur in order of priority. It eliminates problems of simultaneous data processing and regularizes the prioritization of decision making. Received from the web interface or HPCSOMAS agents, a command or message is registered in the UDA Messages manager if it is impossible to respond immediately.

The main actions of the cycle are the following:

- Processing commands and messages received from HPCSOMAS agents or the web-interface;
- Initializing tasks based on the user problem formulation in local assembly mode and creation of local subtasks;
- Sending computational subtasks execution proposals to connected HPCSOMAS agents with suitable computational microservices;
- Analyzing requests for subtasks execution from HPCSOMAS agents;
- Sending subtasks (in local assembly mode) or tasks (in clone mode) to selected HPCSOMAS agents (figure 5);
- Running subtasks on local computational microservices by UDA Executive agent;
- Checking the status of subtasks executed on local computing services and solving failure problems:
  - Task time limit is achieved;
  - Respond to a request is not receive;
- Checking the status of user tasks on HPCSOMAS agents for testing its operability;
- Processing the results of completed subtasks (received locally or remotely) and updating local tasks;
- Synchronizing user files for HPSOMAS agent and UDA in bilateral or unilateral order, if necessary;
- Unilaterally synchronizing of task lists for UDA and connected HPCSOMAS agent;
- Informing the connected HPCSOMAS agent that the UDA is online (the user's computer could be in sleep mode or disconnected from the Internet for some time).

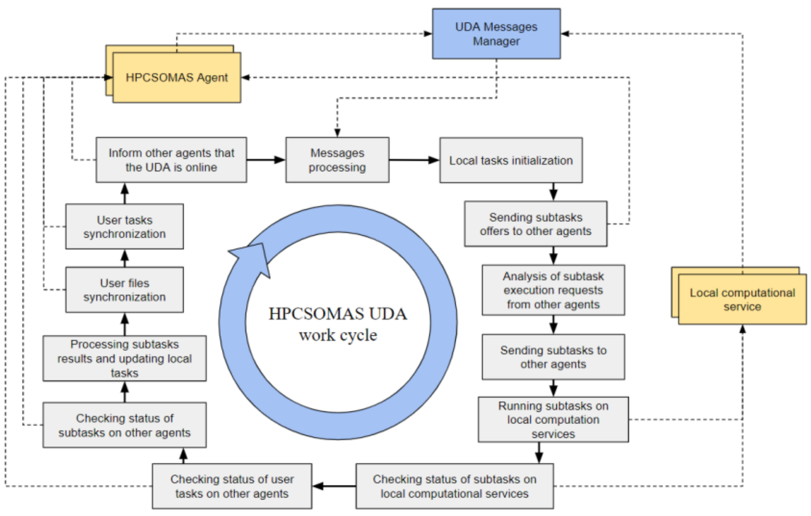

UDA Messages Manager
HPCSOMAS Agent
Inform other agents that the UDA is online
Messages processing
Local tasks initialization
User tasks synchronization
Sending subtasks offers to other agents
HPCSOMAS UDA work cycle
Analysis of subtask execution requests from other agents
Local computational service
User files synchronization
Processing subtasks results and updating local tasks
Sending subtasks to other agents
Checking status of subtasks on other agents
Running subtasks on local computation services
Checking status of user tasks on other agents
Checking status of subtasks on local computational services

Figure 4. The cycle of the functioning UDA.

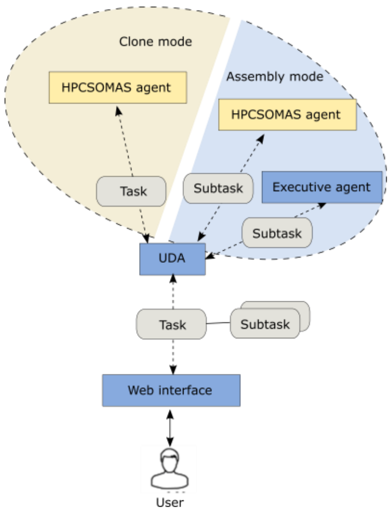


Figure 5. Task distribution.

5. The example of using UDA
In the example, the problem for searching cycles of a given length *k* in BDS is used.

5.1. Problem formulation
In [2], a synchronous autonomous BDS is considered, the vector-matrix equation of which has the form

$$x^t = F(x^{t-1}), \qquad (1)$$

where *x* is the state vector of the dimension *n* ($x \in B^n$, $B = \{0,1\}$), $t \in T = \{1,2,...,k\}$ is the discrete time, and *F*(*x*) is the vector function of logic algebra called the transition function ($F: B^n \to B^n$). Let us define the trajectory $x(t, x^0)$ of (1) for each initial state $x^0 \in B^n$ as the finite sequence of states $x^0, x^1, ..., x^k$ from the set B^n. The state x^i is a successor of the state x^{i-1}, and x^{i-1} is a predecessor of the state x^i. In an autonomous BDS, each state has only one successor, and the number of predecessors of this state can vary from zero to 2^{n-1}. A cycle of the length *k* is a closed trajectory ($x^0 = x^k$) in which all two states are pairwise distinct.

In the example's problem, it is required to find all cycles of the given length *k*. For solving this problem based on the Boolean constraint method [2] the following actions must be performed:

- Building Boolean constraint $L(x^{t-1}, x^t) = \vee_{i=1}^{n}(x_i^t \oplus F_i(x^{t-1}))$ based on BDS dynamic description (1);
- Building Boolean model using *L* and the property specification of the presence of *k*-length cycles;
- Verifying Boolean model satisfiability.

Following [2], a cycle of length *k* (if it exists) is a solution to the Boolean equation

$$\Phi(x^0, x^1, ..., x^k)\big|_{x^0 = x^k} \vee R(x^0, x^1, ..., x^{k-1}) = 0, \qquad (2)$$

where the condition for pairwise differences of all states of the set *C* of a cycle of length *k* is given by the expression

$$R(x^0, x^1, ..., x^{k-1}) = \vee_{\substack{1 \le q \le k-1, \\ k \bmod q = 0}} \wedge_{i=1}^{n} y_i^q = 0, y_i^q = (x_i^0 \cdot \overline{x_i^q} \vee \overline{x_i^0} \cdot x_i^q).$$

For the above actions, the corresponded AMP microservices are required.

5.2. Features of the implementation of microservices for solving the problem
The searching for cycles of the given length *k* is performed for BDS of high-dimension of state vector *n* = 288 (stream cipher Trivium). The description of the BDS dynamics has the following form [18]:

$$\begin{aligned}
&x_1^t = x_{288}^{t-1} \oplus x_{287}^{t-1} \cdot x_{286}^{t-1} \oplus x_{243}^{t-1} \oplus x_{69}^{t-1} \\
&x_{94}^t = x_{93}^{t-1} \oplus x_{92}^{t-1} \cdot x_{91}^{t-1} \oplus x_{171}^{t-1} \oplus x_{66}^{t-1} \\
&x_{178}^t = x_{177}^{t-1} \oplus x_{176}^{t-1} \cdot x_{175}^{t-1} \oplus x_{264}^{t-1} \oplus x_{162}^{t-1} \\
&x_i^t = x_{i-1}^{t-1}, \\
&i \in \{2,3,...,93,95,...,177,179,...,288\}, t \in \{1,...,k\}.
\end{aligned}$$

Boolean constraints for this dynamics description are generated using the SageMath convertor ANF = 0 $\rightarrow$ CNF = 1 [19]. Sage Math is supplied with a free license in the following forms:

1) Online service performing small calculations;
2) A package of programs installed on Windows;
3) An image of a virtual machine on Linux.

We use the third form for the microservice *Sage Executor* implementation. Apache TomEE, UDA, and microservices are installed on the image with Sage Math. For such an approach, it is enough to launch this image in Virtual Box and provide it with external access via IP to connect the agent from this virtual machine to any other HPCSOMAS agents, UDA, or use it through the browser.

The microservice *Cycle BM* for building a Boolean model for the dynamic property (the presence of *k*-length cycles) is implemented based on the ordinary program in the C++ language. *Sage Executor* and *Cycle BM* are non-resource intensive and may be installed on UDA.

Microservices *BM SAT*-X verifying Boolean model satisfiability are implemented based on the AllSAT solver. In the case of small and medium BDS dimensions, the sequential AllSAT solver [20] is used (launched by the non-resource intensive microservice *BM SAT-S*). In another case, parallel AllSAT solver HPCAllsat [21] developed by authors is used (launched by the resource-intensive microservice *BM SAT-P*). The *BM SAT-P* requires a high-performance resource for its running.
Several microservices are implemented for the result post-processing, for example, *POST TR-T* (table post-processing) and *POST TR-G* (graph post-processing). The service Grapher [4] is used for the result visualization.
Sage Executor, *Cycle BM*, *BM SAT-S*, *POST TR-T*, *POST TR-G*, and *Grapher* are non-resource intensive and may be installed on UDA. The *BM SAT-P* must be installed on the SMP cluster.

For the installation of the HPCSOMAS cloud resource agent (CRA), the virtual dedicated server (VDS) [22] was used (figure 6).

The UDA uses the local assembly mode for solving the considered problem. The UDA launches *Sage Executor* for obtaining the BDS dynamic description. Microservices *Cycle BM*, *BM SAT-P*, *POST TR-T* are launched for *k* from 35 to 45. The *BM SAT-P* is used due to the large BDS dimension. The search state space is 2^{288} for considered BDS. The *POST TR-G* is launched for preparing data for visualization by the *Grapher* (figure 7).

All computational results are synchronized for the UDA and cloud user data. Required cloud services were made available for UDA. All steps of solving the problem for searching *k*-length cycles are automated.

5.3. Features of our approach to solving the problem

Let us briefly describe the features of our approach to solving this problem in comparison with existing ones. In [18, 23, 24], SAT-approach to solving the considered problem is presented.

Based on an approach [18], Boolean constraints for finding a path of the given length *k* are formed dynamically, followed by a call to the standard SAT solver and analysis of the resulting solution to identify a cycle of length *k*. The process is repeated iteratively. Such organization of the algorithm prevents the parallel solution of the problem. In [23], based on a standard SAT solver, a specialized SAT solver focused on cryptography problems was implemented. In [24], algebraic equations describing *k* iterations of the cryptographic algorithm are generated in the ANF format manually or automatically, and the ANF is converted to CNF. Then the SAT solver is called. The authors note that methods of converting ANF to CNF can affect the efficiency of the subsequent solution.

Our approach is based on the Boolean constraint method (BCM) [2]. In contrast to [23], we use the standard AllSAT solver [20]. For BDS of high dimension, the developed parallel solver [21] is used. In comparison to [23, 24], the Boolean model based on BCM includes a BDS dynamics description (1) and, additionally, a property specification of the presence of *k*-cycles (2). Unlike [24], we use a sparse strategy for ANF to CNF converting [26].

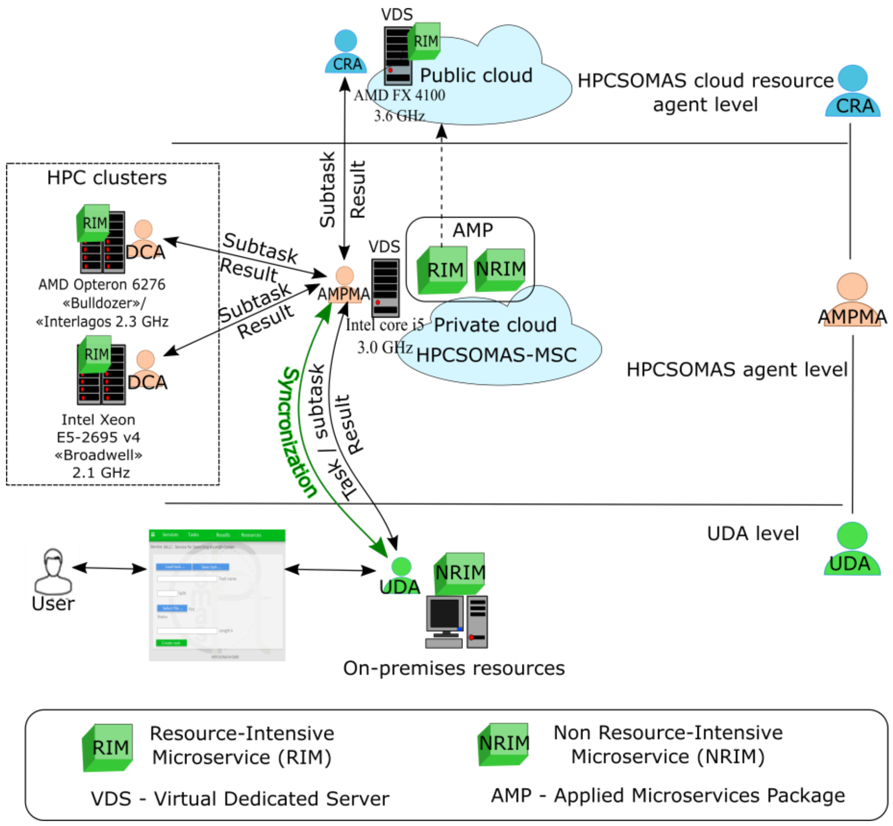


Figure 6. Synchronized distributed computation using UDA.

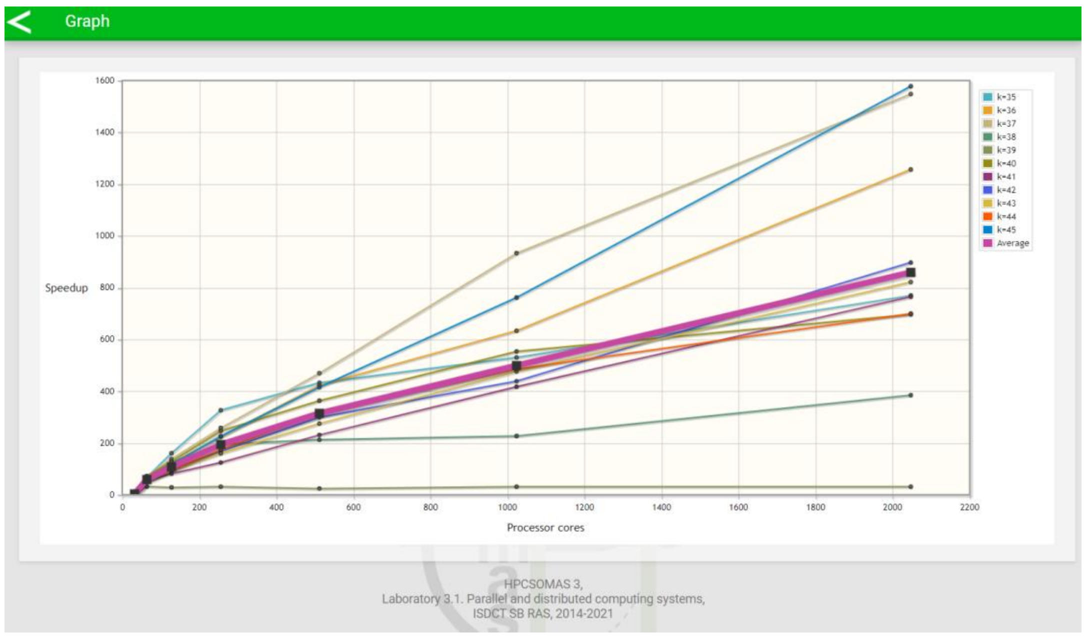


Figure 7. The visualization of the runtime speedup for increasing processor number cores.

In detail, computational experiments for the considered problem are presented in [21]. In particular, the comparison of the BCM-based approach with the approach described in [18] is given. The experiment is conducted on the on-premises resource (Intel Core i7-2670QM, 2.2 GHz) for solving the considered problem sequentially. In this experiment, the BCM-based solving problem reached a speedup 2.5 times on average (for *k* from 1 to 30) in comparison with the approach proposed in [18]. In [24], the considered problem is solved in parallel for *k* from 1 to 31 on 44 cores (Intel(R) Xeon(R) CPU E5-2699 v4 2.20 GHz). The runtime for *k* = 31 is 51m 54.39s (for Trivium) as shown in [24]. In our case, the runtime of the parallel HPCAllsat [21] running on 32 core (Intel(R) Xeon(R) CPU E5-2695 v4 2.10 GHz) for *k* = 31 is 34m 56.03s (for Trivium) using the BCM-based approach.

A sequential solving of the considered problem on a local resource is acceptable for *k* less than 32. In the example, *k* is increased from 35 to 45. In this case, additional resources of the high-performance cluster or cloud are required. UDA forms and sends subtasks to HPCSOMAS agents. As subtasks are completed, UDA synchronizes and visualizes results (figures 6 and 7). In this study, we consider the UDA implementation features on finding cycles of the given length *k* for Trivium. We use this problem for UDA testing due to its high dimension and not focus on its cryptographic aspects.

6. Conclusion

The user Dew-agent was developed for using Dew Computing technology in a package of applied microservices for solving problems of qualitative analysis and parametric synthesis of binary dynamic systems based on the Boolean constraint method in a hybrid computational environment.

Based on this technology, the user Dew-agent automates providing a choice of the resource for computations in the hybrid computing environment, the availability of cloud services, and synchronization of both computations results and other user data.

Acknowledgments

The study was supported by the Ministry of Science and Higher Education of the Russian Federation, project no. 121032400051-9. The authors would like to thank Irkutsk Supercomputer Center of SB RAS for providing the access to HPC-cluster "Akademik V.M. Matrosov" [26].

References

[1] Skala K, Davidovic D, Afgan E, Sovic I and Sojat Z 2015 Scalable distributed computing hierarchy: Cloud, Fog and Dew Computing *OJCC* **2**(1) 16–24

[2] Oparin G, Bogdanova V and Pashinin A 2019 Qualitative analysis of autonomous synchronous binary dynamic systems *MESA* **10**(3) 407-419

[3] Oparin G A, Bogdanova V G, Pashinin A A and Gorsky S A 2019 Microservice-oriented approach to automation of distributed scientific computations *Proc. of the 42st Int. Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2019, Opatija, Croatia, May 2019* pp 253–258

[4] Oparin G A, Bogdanova V G and Pashinin A A 2020 Automated tools for the development of microservice compositions for hybrid scientific computations *Proc. of the 2nd Int. Workshop on Information, Computation, and Control Systems for Distributed Environments, ICCS-DE, Irkutsk, Russia, July 6-7, 2020: CEUR-WS Proceedings* vol 2638 pp 201–213

[5] Wang Y 2015 Cloud-dew architecture *Int. J. Cloud Computing* **4**(3) 199–210

[6] Wang Y 2016 Definition and categorization of Dew Computing *OJCC* **3**(1) 1–7

[7] Ristov S, Cvetkov K and Gusev M 2016 Implementation of a horizontal scalable balancer for Dew Computing services *Scalable Computing: Practice and Experience* **17**(2) 79–90

[8] Wang Y and Leblanc D 2016 Integrating SaaS and SaaP with Dew Computing *Proc. of IEEE Int. Conf. on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), Atlanta, GA* pp 590–594

[9] Abdelsamea A, Nassar S M and Eldeeb H 2020 The past, present and future of scalable computing technologies trends and paradigms: A survey *Int. Journal of Innovation and Applied Studies* **30**(1) 199–214

[10] Fisher D E and Yang S 2016 Doing more with the Dew: A new approach to Cloud-Dew

architecture *OJCC* **3**(1)

[11] Wang Y, Skala K, Rindos A, Gusev M, Yang S and Yi P 2017 Dew Computing and transition of Internet computing paradigms *ZTE Communications* **15**(4) 30–37

[12] Garrocho C T B, Cavalcanti C F M da Cunha and Oliveira R A R 2020 Performance evaluation of Industrial Internet of Things services in devices of Cloud-Fog-Dew-Things computing *Proc. of the X Brazilian Symposium on Computing Systems Engineering (SBESC), Florianopolis, Brazil, 2020* pp 1–8

[13] Garrocho C T B and Oliveira R A R 2020 Counting time in drops: views on the role and importance of smartwatches in dew computing *Wireless Netw* **26** 3139–3157

[14] Oparin G A, Bogdanova V G, Gorsky S A and Pashinin A A 2017 Service-oriented application for parallel solving the parametric synthesis feedback problem of controlled dynamic systems *Proc. of the 40th Int. Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, 2017* pp 353–358

[15] CSCC: Practical Guide to Cloud Computing. [Online]. Available: https://www.omg.org/cloud/delivera-bles/CSCC-Practical-Guide-to-Hybrid-Cloud-Computing.pdf

[16] Apache TomEE. [Online] Available: https://tomee.apache.org/

[17] Hall M and Brown L 2003 Core servlets and Javaserver pages: Cory technologies. [Online]. Avalable: https://freecomputerbooks.com/Core-Servlets-and-Javaserver-Pages.html

[18] Dubrova E and Teslenko M 2016 A SAT-Based algorithm for finding short cycles in shift register based stream ciphers *IACR Cryptology ePrint Archive* p 1068

[19] SageMath. [Online]. Available: https://www.sagemath.org/

[20] Toda T and Soh T 2016 Implementing Efficient all solutions SAT solvers *ACM Journal of Experimental Algorithmics* **21**(1) 1–44

[21] Bogdanova V G, Gorsky S A and Pashinin A A 2020 HPC-based parallel software for solving applied Boolean satisfiability problems *in Proc. of the 43rd Int. Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, 2020* pp 1006–1011

[22] First VDS. [Online]. Available: https://firstvds.ru/

[23] Soos M, Nohl K and Castelluccia C 2009 Extending SAT solvers to cryptographic problems *Theory and Applications of Satisfiability Testing - SAT 2009* vol 5584, ed. O Kullmann (Berlin, Heidelberg: Springer)

[24] Dudzic W and Kanciak K 2020 Using SAT solvers to finding short cycles in cryptographic algorithms *Int. Journal of Electronics and Telecommunications* **66**(3) 443–448

[25] An ANF to CNF Converter using a Dense/Sparse Strategy. [Online]. Available: https://doc.sagemath.org/html/en/reference/sat/sage/sat/converters/polybori.html

[26] Irkutsk Supercomputer Centre of SB RAS. [Online]. Available: http://hpc.icc.ru