Modelling and Simulation of Security-aware Task Scheduling in Cloud Computing Based on Blockchain Technology

Andrzej Wilczyński, Joanna Ko lodziej

PII: S1569-190X(19)30169-8
DOI: https://doi.org/10.1016/j.simpat.2019.102038
Reference: SIMPAT 102038

To appear in: *Simulation Modelling Practice and Theory*

Received date: 25 October 2019
Revised date: 5 December 2019
Accepted date: 6 December 2019

Please cite this article as: Andrzej Wilczyński, Joanna Ko lodziej, Modelling and Simulation of Security-aware Task Scheduling in Cloud Computing Based on Blockchain Technology, *Simulation Modelling Practice and Theory* (2019), doi: https://doi.org/10.1016/j.simpat.2019.102038

Modelling and Simulation of Security-aware Task Scheduling in Cloud Computing Based on Blockchain Technology

Andrzej Wilczyński[a,b,*], Joanna Kołodziej[c]

[a] *Cracow University of Technology, Department of Computer Science, Warszawska 24, 31-155 Cracow, Poland*
[b] *AGH University of Science and Technology, Department of Computer Science, al. Mickiewicza 30, 30-059 Cracow, Poland*
[c] *Research and Academic Computer Network (NASK), Kolska 12, 01-045 Warsaw, Poland*

Abstract

Although a lot of work has been done in the domain, tasks scheduling and resource allocation in cloud computing remain the challenging problems for both industry and academia. Security in scheduling in highly distributed computing environments is one of the most important criteria in the era of personalization of the cloud services. Blockchain became recently a promising technology for integration with the cloud clusters and improvement of the security of cloud transactions and access to data and application codes. In this paper, we developed a new model of the cloud scheduler based on the blockchain technology. Differently to the other similar models, we tried to offload the implementation of the blockchain modules. We developed a novel 'proof–of–schedule' consensus algorithm (instead of 'proof–of–work') and used the Stackelberg games for the improvement of the approval of the generated schedules. The developed model has been experimentally simulated and validated by using the new original cloud simulator. The proposed Blockchain Scheduler was also compared with other selected cloud schedulers. The experiments shows that the applied approach improved significantly the efficiency of prepared schedules, in most cases, simulator returns a schedule with better makespan than existing individual scheduling modules.

*Corresponding author
Email address: and.wilczynski@gmail.com (Andrzej Wilczyński)

Keywords: Cloud Scheduling, Blockchain, Stackelberg Game, Proof of Schedule

1. Introduction

Over the last years, scheduling problems in cloud computing remain the challenges for both academia and industry [1]. The main difficulties can be observed in the proper allocation of the tasks and virtual servers at the available resources and then the efficient management of such tasks and resources is the large scale of the cloud system, in which the local clusters may work under different administrators and using different cloud technologies and data and information transmission protocols[1]. Another issue nowadays is quite wide set of the scheduling criteria – sometimes the conflicting ones. While optimal schedule execution time and makespan are strictly related to the expectations of the end users and resource providers (the costs calculated for them depend on the time of calculating all submitted tasks and utilization of the physical resources in the cloud), energy consumption and security became the key factors of scheduling of computational tasks as well as data and information storage and processing in modern cloud system[2]. Although cloud computing is already not relatively new technology, the maintain of the security at all levels (data, application, network) remains still critical, mainly due to external data storage, communication through 'public' internet and 'multitenancy' of the cloud servers and administrators.

Recently, blockchain [3] is promoted as an effective and secure technology for the online financial operations through the communication solely between transaction network peers and without the involvement of third parties. By using blockchain, the data can be stored in the distributed relatively small databases instead of storing all data in a central data center. It may increase the security of the whole cloud system, because most of the damages from attacks on such databases can be easily locally prevented. Therefore, blockchain can be successfully utilized in various areas including the financial sector and the Internet of Things (IoT).

In this paper, we present our recent developments on the deployment of the generic blockchain architecture and algorithms for modelling and simulation of the connected distributed cloud clusters, where the secure schedules

[1] www.avinetworks.com/glossary/multi-cloud/

are generated and then executed. Differently to the existing solutions (see Sec. 2), we tried to optimize the implementation of the most important blockchain modules and keep only the most important ones – we resigned for instance from the time consuming 'proof–of–work' algorithm. Our secure scheduling model is based on iterative Stackelberg game [4] and developed new *proof–of–schedule* algorithm. The main aim of such model is to ensures the highest possible level of security in task processing and task execution according to the requirements of the cloud end users. We simulate our model through simple experiments provided with developed original *BCSchedCloudSim* simulator.

Our contributions in this paper are the following:

- the development of a new generic model of the secure cloud scheduler based on the blockchain (BC) architecture,
- the adaptation of the blockchain algorithm to the scheduling in clouds - development of a new *proof of schedule algorithm and integration with the BC architecture,*
- the development and implementation of the BCSchedCloudSim simulator.

The rest of the paper is organized as follows. In Sec. 2 we compared the work presented in this paper with related studies. Next, in Sec. 3 we define the main types of the scheduling problems and scheduling attributes. Security is addressed as one of the important and challenging issues in scheduling in clouds. Sec. 4 presents the essentials of the blockchain technology and typical architecture. In Sec. 5 we present the developed model of the secure blockchain-based scheduler with the detailed description of proof-of-schedule procedure. The results of simulation of the developed model and the experimental comparison of the proposed scheduler with the other cloud scheduling algorithms are presented in Sec. 7. We summarize our research and define the directions for the further research in the domain in Sec. 8.

2. Related Work

The security problem in computational clouds has been addressed already in many publications. Takabi et al. [5] defined the most important security

and privacy challenges in clouds which are related to various delivery and deployment models. An interesting survey of various security risks that pose the threats to the cloud is presented by Subashini et al. in [6]. Asma et al. [7] surveyed the cloud computing security issues. They classified the cloud security problems based on the service model types. Kolodziej and Xhafa[8], Song et. al,[9, 10], Grzonka et al.[11] proposed the security models in scheduling computational tasks in the cloud physical resources. All their models are based on *trust level* and *security demand* parameters, which in fact cannot be easily determined or measured in the realistic scenarios. It makes those scheduling models theoretical rather than applicable in real life.

Some researchers already considered blockchain as a promising technology for supporting the cloud scheduling, especially with security as the important criterion. Lokhandwala [12] used the decentralized blockchain network for the efficient allocation of the resources, optimization of the consumed energy and improvement of data security in the cloud. By using the smart contracts [13], the data are stored in the distributed databases in the blocks. The tasks are assigned to the data centers with the lowest load in the blocks. In the experimental part, the author used 'Shortest Job First' scheduling algorithm [14] for the minimization of the waiting time of response. The author focused on testing the blockchain network and allocation of the proper data servers (with the proper datasets). The author observed also a high inability to manipulate the data, which increases the security of the whole system. In this approach however, the execution of the basic blockchain modules is time and energy consuming, which is the weak aspect of that model.

Hong et al. in [15] discuss the problem of communication and task scheduling among users in device–to–device (D2D) network [16]. The main aim of their research was to reduce the average execution time of tasks mapped onto the network resources. The authors developed an innovative blockchain–based credit system that can be used for task scheduling to enforce justice among D2D network users. Their system consists of two main parts: the cooperative task scheduling to reduce the average task execution time among users and a blockchain-based credit system to ensure fairness in network.

Zhang et al. in [17] draws attention to the growing number of users in the clouds and the problem of providing them services through single providers. He proposes an approach where different providers work together to provide a satisfactory solution to the end users. Authors consider task scheduling across the clouds, based on genetic algorithm. They take into account the

impact of resource distance on the completion time and cost of tasks execution, regarding customer requirements in accordance with the established Quality of Service. In the simulation part, a comparison of the results of inter-cloud scheduling and scheduling with a single cloud for different amounts of tasks has been presented. As a result, they demonstrate that task scheduling between clouds is neither the best nor the worst approach, but nevertheless provides adequate computing resources, which are often lacking for individual providers.

3. Cloud Scheduling Problems

The main aim of scheduling in computational clouds is efficient mapping of tasks originated by applications or submitted to the system by the end users to a set of physical and virtual resources available in the cloud system [18]. The tasks and resources can be added and removed to and from the cloud. Scheduling in computational clouds remains a challenging NP-complete global optimization problem due to the large-scale heterogeneous architecture of the cloud network and co–existence of local geographically dispersed task dispatchers and resource owners working in different autonomous administrative domains. In particular, the well defined scheduling algorithms in computational cloud should [19]:

- ensure that all tasks are executed withing their deadlines and generate the expected high quality results,
- minimize the task execution times and – as the direct consequence – should minimize the scheduling real costs calculated for the end users[2],
- be aware of and prevent the temporary software failures.

Similarly to the computational grids, various types of the cloud scheduling problems can be defined based on the specified environmental attributes, such as the static, dynamic, fully decentralized or hierarchical cloud architectures, and scheduling attributes [20]. The most important scheduling attributes include:

- **tasks processing policy** – instantaneous vs. batch,

[2]This policy is usually called the 'pay-per-use' cloud paradigm

- **interrelations among tasks** – independence vs dependency.

Specification of the tasks processing policy is important in the identification of the particular scheduling problem. In the instantaneous mode, the tasks are scheduled as soon as they are arrived at the system. In batch scheduling, the submitted tasks are grouped into batches and the scheduler assigns each batch to the resources. The tasks may be independently sent and executed in the cloud system or they can be submitted as parallel applications with priority criteria and interrelations among the application components (usually modelled by a Directed Acyclic Graph (DAG) [21]).

Kolodziej in [22] defined the following notation for specification of the scheduling problems in distributed large–scale systems:

$$\alpha|\beta|\gamma, \tag{1}$$

where α determines the resource layer and cloud architecture type, β defines the task processing policy, and γ specifies the scheduling criteria. Using that notation, we may define the scheduling problem considered in this paper in the following way:

$$Rm, D|b, indep|\gamma, \tag{2}$$

where Rm denotes the independent (unrelated) machines/resources, D– decentralized cloud system, b – batch mode, *indep* independent scheduling. The main scheduling criterion was the total execution time of the tasks in the schedule, and additionally - makespan in the experimental section (Sec. 7.2). Another issue is security in scheduling, which is discussed in the following subsection.

3.1. Security aspects in cloud scheduling

Security, among the other scheduling aspects and criteria such as high availability of computing and data resources, optimization of the scheduling costs paid by the end-users, energy optimization in the cloud systems and personalization of the cloud services based on the end-users' requirements, is one of the most challenging problems in today's cloud cloud systems [11], [23]. "security' in cloud scheduling may be related to the various aspects and parameters in the scheduling process. On one hand, the end users may have some special individual requirements for the data protection and privacy policies and the execution of their tasks in the specially protected machines.

On the other hand, in order to guarantee the low possible costs of the task execution and high quality results of such execution (task calculation), the system cannot be too sensitive on the very specific individual requests of the users, which may be sometimes in conflict with the interests of the other cloud users. Another issue is to protect the system against the external attacks, where the malicious software or data is injected in the system and the generated schedule may be infected and destroyed during its execution.

A general security-aware IaaS cloud model developed by Kolodziej et al in [8] is based on the hierarchical multi–level architecture. In this model, security is considered as additional criterion in the scheduling process and cloud scheduler analyzes the security requirements for the execution of tasks and requests of the end users for trustful resources available within the system. The system brokers analyze 'reputation' indexes of the machines /cloud physical resources received from the resource managers/owners and send the allocation suggestions to the scheduler.

Fig. 1 shows the 3-level architecture of the security-aware IaaS cloud cluster. The *trust level* and *security demand* parameters are generated by aggregation of several scheduling and system attributes. Those parameters depend heavily on the security policy, accumulated resource or cloud cluster 'reputation', self–defense capability, attack history, special users' requirements, and peer authentication. Fig. 2 presents the major behaviour and intrinsic security attributes needed for the specification of trust levels of the cloud clusters and security demand of the cloud applications (see also [10]).

Song et al. in [9, 10] have developed a fuzzy–logic trust model, in which the scheduling attributes are aggregated into single scalar parameters, and *security demand vector* and *trust vector* are defined. There is a trust resource manager in the system, who maintains the status of the resources and monitors the execution of tasks assigned to those resources. Additionally, the *Machine Failure Probability* matrix, the elements of which, are interpreted as the probabilities of failures of the machines during the tasks executions due to the high security restrictions. These probabilities are calculated by using the negative exponential distribution function (see [22] for details).

The process of matching the trust level and security demand parameters is similar to that of a real-life scenario where users of some portals, such as Yahoo!, are required to specify the security level of the login session.However, the above fuzzy model and the calculation of the Machine Failure Probability matrix are rather theoretical than practical and it is very difficult to use the defined model in the simulation of the realistic cloud environments as well

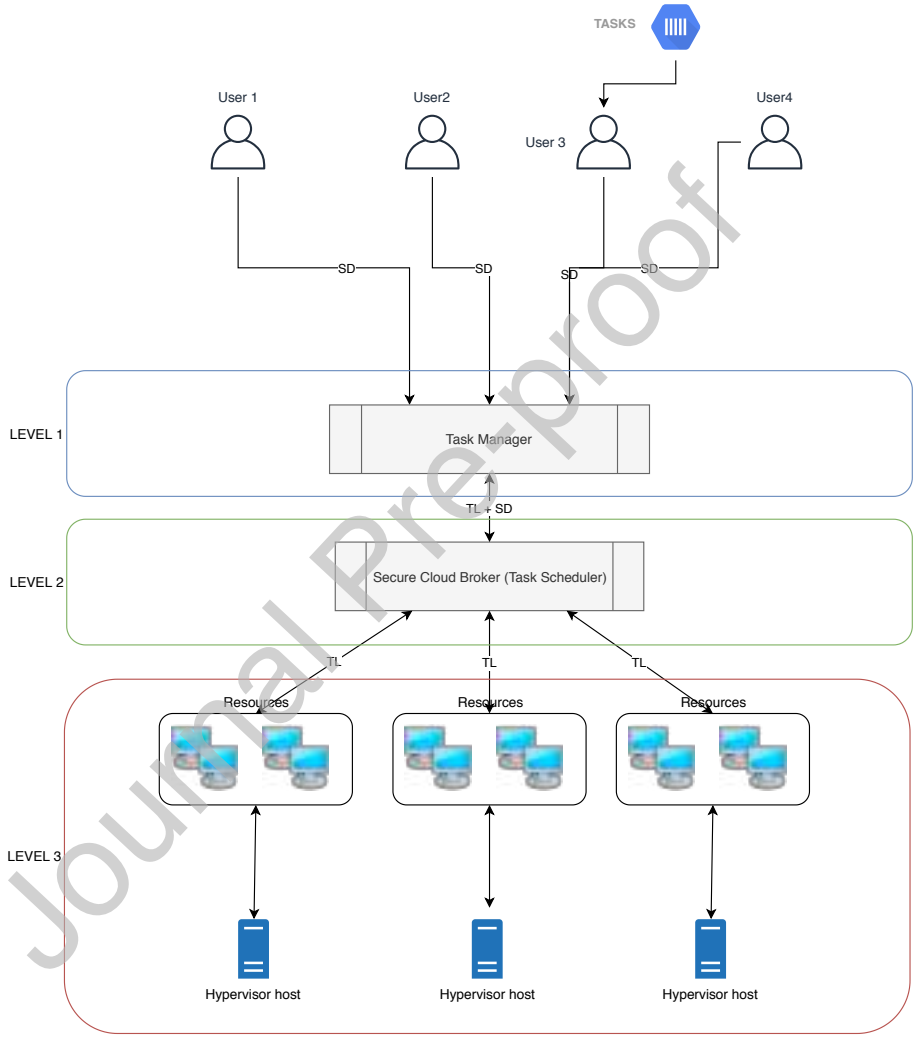

Figure 1: The model of secure IaaS cloud cluster

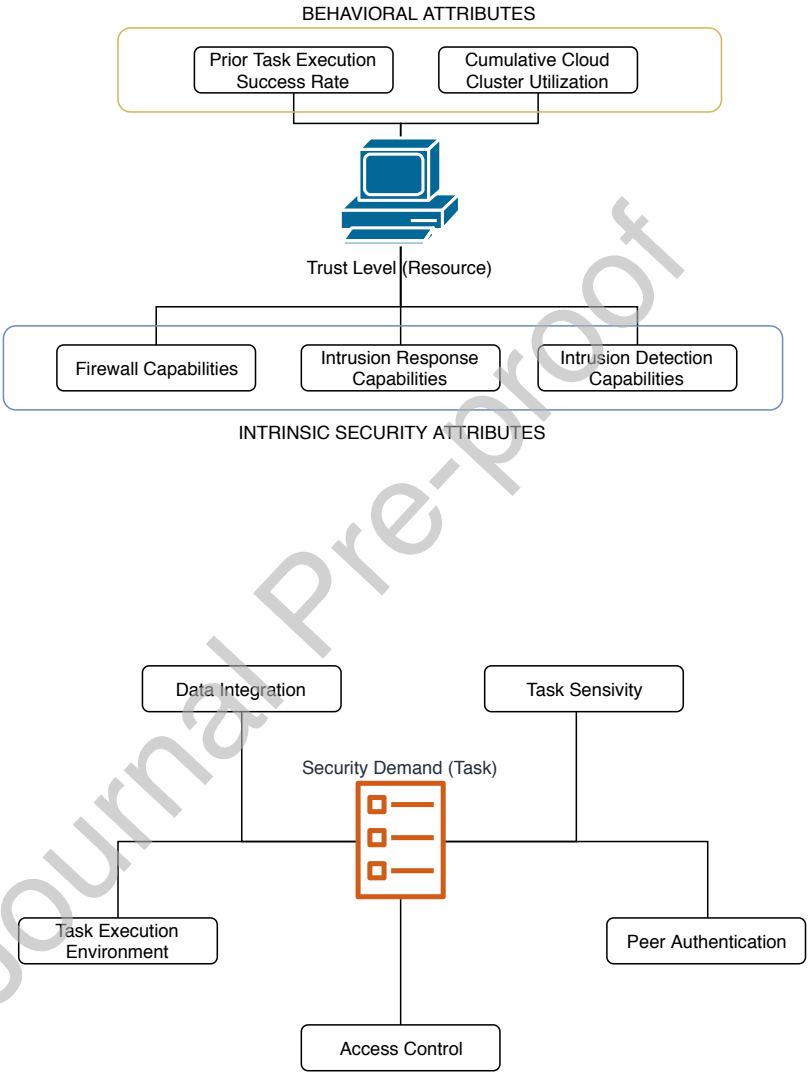

Figure 2: The major attributes affecting the trust level and security demand in cloud infrastructure

as the realistic implementations of the developed schedulers.

Recently, the blockchain technology became the promising solution for the improvement of the security in the cloud scheduling. The prototypes of the blockchain cloud systems are already proposed[3] [4], however the integration of the blockchain architecture and algorithms with the scheduling process to improve security as the scheduling criterion remains an open research problem. This paper presents the results of our research in this domain. We propose a new method to find the optimal schedule based on blockchain technology. We introduce a model of our approach and a simulator implemented for the needs of conducting experiments and simulations.

4. Blockchain Essentials

Blockchain (BC) is a relatively new technology, which has a big potential to be a promising methodology in solving the complex problems in high performance computing systems. There are many definitions of the blockchain architecture and technology. Most of them are based on the concept of Bitcoin model developed by Nakamoto in [24]. Following this model, Blockchain (BC) can be defined as a distributed ledger of records, which contains transactions confirmed by the cryptographic digital signatures and are grouped into blocks. The main features of BC can be specified as follows:

- **decentralization** – in BC there is no need for a central supervisor, who maintains the data consistency,the decentralized network based on consensus algorithms jointly confirms each transaction,
- **persistency** – transactions are validated, any attempt to approve transactions that are incompatible with established policies will be caught by confirming/mining nodes, blocks containing incorrect data are immediately detected,
- **anonymity** – each user in the network is assigned a generated address (hash), by means of which can perform operations,

[3] www.medium.com/eternacapital/blockchain-based-decentralised-cloud-computing-277f307611e1

[4] www.guardtime.com/blog/blockcloud-re-inventing-cloud-with-blockchains

- **auditability** – each transaction must refer to some previous transactions, thanks to which there is a possibility to trace and verify what has happened with the data being processed,
- **transparency** – transactions of any public address are available for inspection by every user having access to BC,
- **security** – chain of blocks are shared, tamper-proof, and can not be spoofed due to one-way cryptographic hash functions,
- **immutability** – data stored in BC are immutable, each entry in the ledger must be confirmed by the network, it can not be a secret operation; each block contains the hash of the previous block, which is generated on the basis of the data in the block, each even a minor change in the data will change the hash, which will cause that the other nodes intercept the modification and reject it.

4.1. Blockchain architecture and main modules

Based on the most popular taxonomy of the blockchain systems published by Lin et. al in [25], we can define the following three major classes of the BC networks:

- **public blockchain**, in which the external users have the ability to read and add records to the ledger,
- **private blockchain** network, which have an owner who determines the access to the blocks and ability to add and confirm the BC transactions
- **consortium blockchain** with a pool of selected nodes, that can add data to the chain, data reading can be open or private.

Each block in the BC usually consists of:

- block number,
- hash of the current block, which is generated from data contained within the block, usually using the Merkle tree (see Fig. 3), which guarantees that every change in the data in the previous blocks will change the hash of the following blocks,
- hash of previous block,

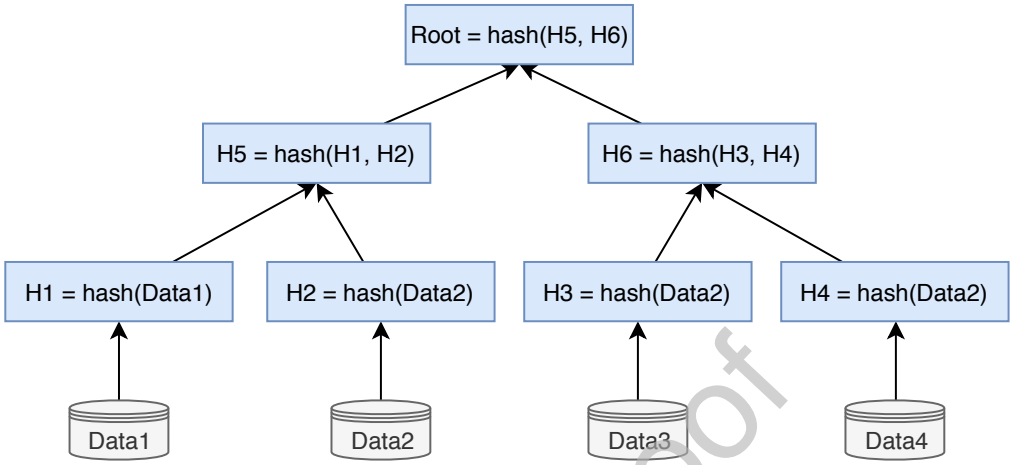

Figure 3: Merkle tree

- timestamp,
- list of transactions

The example of a typical chain of blocks is presented in Fig. 4.

4.2. Security aspects in blockchain

Security in BC environment is usually related to the protection of the transactions and data stored and transmitted in the system. Joshi et al. in [26] defined the mayor security-related procedures in BC as follows (see also [27]):

- **defense in penetration** – data protection procedures executed with many various parameters, very difficult for breaking,
- **minimum privilege** – low–level access to the data,
- **manage vulnerabilities** – checking security vulnerabilities and patching them,
- **manage risks** – identification and control of risks in the environment,
- **manage patches** – patching faulty parts of the source code.

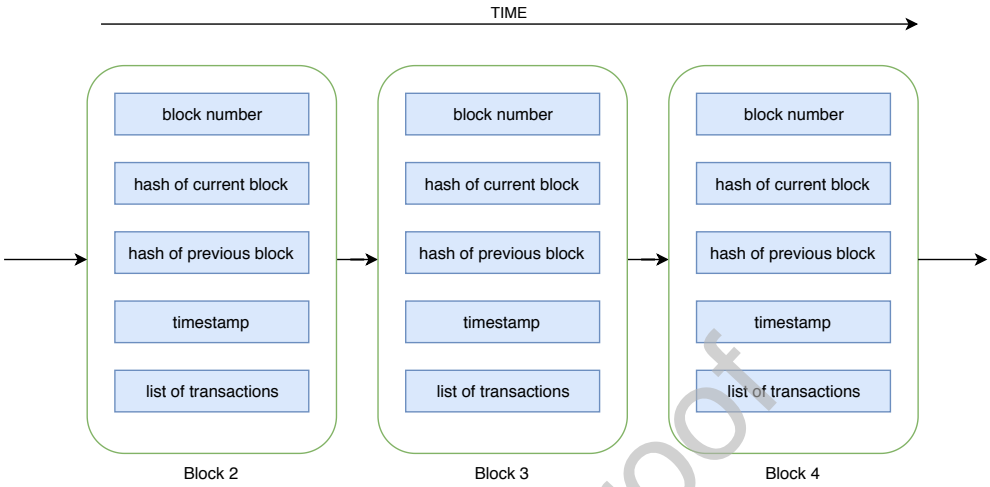

Figure 4: Abstract chain of blocks

In BC systems, the procedures for data security and for the verification of the nodes' ability to perform specific operations are the most commonly used security mechanisms. The method of establishing consensus in the network ensures an appropriate level of security, for instance, in PoW [27], to hack a network node would have to have at least 51% of computing resources which is practically impossible [28].

4.3. Blockchain vs. traditional cloud schedulers

Most of the existing scheduling technologies and algorithms usually generate the queue or simple array of tasks for allocation on each available machine/server separately. The end users – as the main requirement for the scheduler–define the shortest time to complete all submitted tasks. The end users usually do not have any additional instruments for the confirmation that the generated schedule is optimal. In order to solve partially that problem, we propose a blockchain-based scheduler, where the generated schedules must be approved by the neighbour nodes in the cloud cluster and the optimal schedule is proposed for the end users as the result of the cloud providers' internal agreement. Therefore, the end users cannot get the better offer from the other providers.

5. Secure Blockchain Cloud Scheduler

The scheduling problem considered in this paper is the batch independent problem defined by Eq. 2 in Sec. 3. The model of the blockchain scheduler is presented in Fig. 5. The nodes in the blockchain (BC) network communicate with each other and define a consensus on the 'correctness of the schedule'. Such schedule may be generated by any node in the network, that one who will generate the schedule first will start the transaction. If most of the BC nodes in the network confirm such 'correctness of the schedule', the schedule is saved in chain of blocks. It means that the schedule was approved for the execution in the cloud or may be directly sent to the user in the case it contains just the tasks from a given one user. The approved schedule can be executed in an optimal time slot. Each node may generate its own schedule and compare it with the schedule proposed by its predecessor in the defined sequence of the BC nodes.

5.0.1. Pool of Task Managers Requests

Task Manager in the BC network is responsible for transferring the request into the *request pool*. Each request contains characteristics of tasks, characteristics of virtual machines and digital signature of the Task Manager. The characteristics of tasks may be defined as the number of floating point operations or instructions needed to perform a given task. Machine characteristics parameter is specified by the number of floating point operations or instructions performed by a given machine per second. Those characteristics parameters can be delivered by the processor manufacturer or cloud service provider.

5.0.2. BC network Nodes

Each BC network node that want to participate in the generation of the schedule downloads one request from the pool of requests. Then, using the implemented in the node scheduling algorithm, the node generates a schedule and calculates its execution time. The prepared schedule, together with other data are placed in the transaction. Detailed data included in the transaction are:

- id,
- sender,
- recipient,

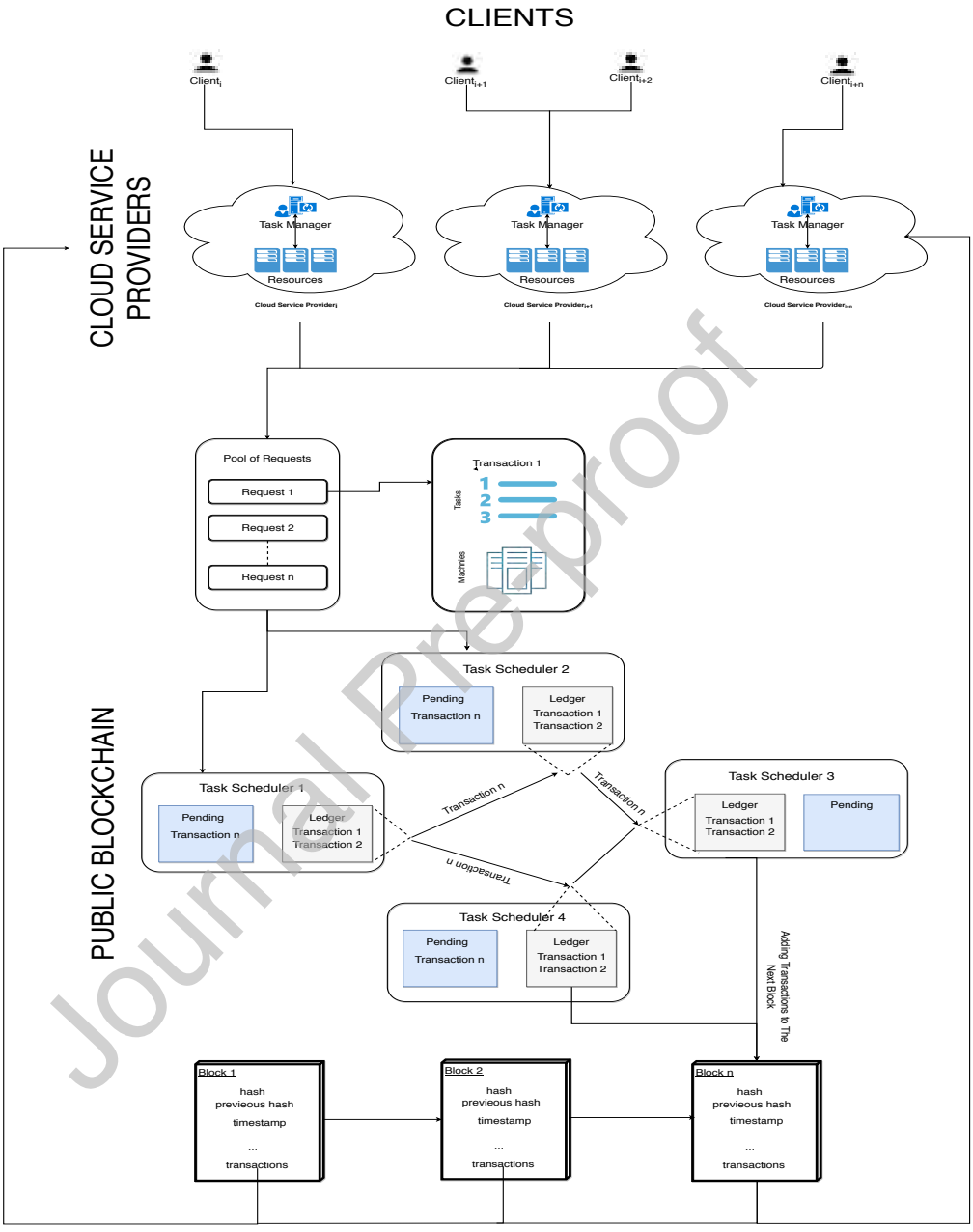

Figure 5: Scheduling process.

- signature,
- request id,
- schedule - contains an array of tasks to execute, an array of machines together with a list of tasks to be executed on each of them and the longest time of execution all tasks from one of the machines (it is the completion time of all tasks in all machines).

After the generation of the schedule and transaction, the transaction is sent for confirmation to the remaining BC nodes in the network in order to define a consensus by using the proof-of-schedule algorithm defined in Sec.5.1.

5.0.3. Chain of Blocks

The confirmed transaction is moved the BC block, when the consensus is achieved for the generated schedule. Each block consists of:

- the current block hash value,
- the previous block hash value,
- a timestamp,
- the Merkle tree root hash [29],
- a list of transactions,
- the size of the block parameter - such parameter defines the minimal number of operations or instructions within the whole block.

Having the appropriate number of transactions confirmed by the network, which is specified as a global system parameter, the block is added to the chain of blocks. A serious problem with generating the chains may occur, when different nodes in the BC network may try to add the multiple blocks at the same time, which could generate many variants of the chain of blocks. The solution of such problem is usually the selection by the system the longest chain of blocks of the confirmed transactions through the consensus procedure.

5.1. Proof of Schedule

In the proposed model of the BC cloud scheduler, the consensus procedure is defined as a *Proof of Schedule (PoSch)* algorithm especially designed for the secure scheduling in clouds.

Let us introduce first the following notation:

- $TS = \{ts_i, ts_{i+1}, \ldots, ts_l\}$ – a sequence of nodes in BC network involved in the approval of a given transaction,
- ts_i – initial node in TS which starts the transaction,
- ts_{i+1} – first node in TS, which confirms the transaction
- I – characteristics of tasks - the number of operations or instructions needed to execute the task,
- M – characteristics of machines – the number of operations or instructions executed by the machine per second,
- T_{ts_i} – execution time of the schedule generated by node ts_i,
- SF_{ts_i} – scheduling factor - the number of all I in the blockchain added by node ts_i,
- $T_{ts_{i+1}}$ - exeuction time of the schedule generated by node ts_{i+1},
- $SF_{ts_{i+1}}$ - scheduling factor - the number of all I in the blockchain added by node ts_{i+1}.

The initial node ts_i in the sequence TS generates a schedule by using the specified scheduling algorithm implemented in that node (it is assumed in this paper, that each node has the one specified scheduling algorithm, which may be the same in the whole network or – in more general case – it can be different in each node). The schedule is generated based on the task characteristics I and the machine characteristics M. The schedule along with its estimated execution time parameter is stored in the transaction record and then sent to the other nodes for approval. The values of the following parameters are estimated for the approval of the transaction, namely T_{ts_i}, $T_{ts_{i+1}}$, SF_{ts_i}, $SF_{ts_{i+1}}$. The remaining nodes $ts_{i+1}, \ldots, ts_l$, having the schedule generated by ts_i, may follow the action of ts_i and also generate their own schedules by using the locally implemented schedulers and estimate the execution times of

such schedules. Each node in TS confirms the schedule sent by its predecessor in TS, which does not mean that the predecessor prepared it itself, because it can be a schedule from the node, with whom the sending node previously lost the game. For instance, node ts_{i+1} confirms the schedule generated by the initial node ts_i. Such schedule confirmation/approval can be formally modelled by using the Stackelberg game model [22]. The game in details is defined in the following section.

5.1.1. Stackelberg game

Stackelberg game [4] is a non-symmetric game, where one player called *leader* has a privilege position and makes decisions first, when the other players – *followers* – follow his actions. In our BC cloud model, the *leader* will be node, which starts a transaction such as the node ts_i specified in the previous section. The $follower$ can be the next node ts_{i+1} in the node sequence TS. The node starting the transaction plays the Stackelberg game successive with all remaining nodes in TS (a single game is provided for the pairs of nodes: initial node –i.e. ts_i – and the node ts_j, where $j = i+1, \ldots, l$). Let us consider the Stackelberg game for the nodes ts_i ($leader$) and ts_{i+1} ($follower$). Being the *leader*, the node ts_i generates a schedule, which is then sent to ts_{i+1} for the verification and approval.

The approval 'decision' of the node ts_{i+1} can be made based on the schedule execution time T_{ts_i} delivered by the node ts_i. However, ts_{i+1} can estimate the time of the delivered schedule by using the local scheduling algorithm implemented in ts_{i+1} (we will denote such time by $T_{ts_{i+1}}$).

In the game scenario, there are two possible pure strategies, which may be chosen by the $follower$ ts_{i+1}:

- s_1 - choosing option 1 – (time T_{ts_i})
- s_2 - choosing option 2 – (time $T_{ts_{i+1}}$)

where s_1, $s_2 \in \{0, 1\}$

The game utility function for the $follower$ can be defined based on the scaling the scheduling factors, which in our model are treated as *confidence coefficients* and determined based on the history of adding transactions to blocks by given nodes. That scaling procedure is defined as follows:

$$\overline{SF_{ts_i}} = \begin{cases} 1 & \text{if } \max\{SF_{ts_{i+1}}, SF_{ts_i}\} = SF_{ts_i} \\ \frac{SF_{ts_i}}{SF_{ts_{i+1}}} & \text{if } \max\{SF_{ts_{i+1}}, SF_{ts_i}\} \neq SF_{ts_i} \end{cases}$$
$$\overline{SF_{ts_{i+1}}} = \begin{cases} 1 & \text{if } \max\{SF_{ts_i}, SF_{ts_{i+1}}\} = SF_{ts_{i+1}} \\ \frac{SF_{ts_{i+1}}}{SF_{ts_i}} & \text{if } \max\{SF_{ts_i}, SF_{ts_{i+1}}\} \neq SF_{ts_{i+1}} \end{cases} \tag{3}$$

Considering both strategies s_1 and s_2 and scaled confidence coefficients of players, the utility function for the $follower$ is defined in the following ways:

$$u(s_1, s_2, T_{ts_i}, T_{ts_{i+1}}) = T_{ts_i}\overline{SF_{ts_i}}s_1 + T_{ts_{i+1}}\overline{SF_{ts_{i+1}}}s_2 \tag{4}$$

In order to solve the game among the nodes ts_i and ts_{i+1}, we need to solve the following maximization problem:

$$\begin{cases} \underset{s_1,s_2}{\text{argmax}}\ u(s_1, s_2, T_{ts_i}, T_{ts_{i+1}}) \\ s_1 + s_2 = 1 \\ s_1, s_2 \in \{0, 1\} \end{cases} \tag{5}$$

There are many optimization methods, which can be used for solving the problem defined by Eq. 5. For our research, we used the simplex method [30] and generate the optimal strategies s_1 i s_2. However, for simplicity, the values of the strategies in fact can be defined as the binary values for the node ts_{i+1} , i.e.:

- in the case of option 1 – T_{ts_i} – $s_1 = 1$ and $s_2 = 0$,
- in the case of option 2 – $T_{ts_{i+1}}$ – if $s_2 = 1$ and $s_1 = 0$.

If node ts_{i+1} selects the schedule generated by ts_i, it sends it to the next node ts_{i+2} and informs the ts_i node about the schedule approval. Otherwise, ts_{i+1} – being the *leader* – creates a new transaction with a schedule generated by itself and sends such schedule to the node ts_{i+2}. The game is repeated as many times as the node receives confirmation from at least 50% of the items in the sequence TS.

Scheduling factors SF_{ts_i} and $SF_{ts_{i+1}}$ must be non–zero game, which means that they must be drawn randomly, if nodes participating in the game do not yet have such data because they are new in BC network. The initiating node will choose its time as correct without playing the game because it has no predecessor.

In the following subsection, we demonstrate the example of the Stackelberg game for the pair of nodes with an exemplary parameters.

5.1.2. Numerical results of the Stackelberg game

Let's consider a Stackelberg game with *leader* and one $follower$. The scheduling factor of the *leader* is 1500 instructions, and the schedule execution time time proposed by the *leader* is 15 seconds, the scheduling factor of the $follower$ is 8000, and the time proposed by him is 6 seconds. Using the notation introduced in Sec. 5.1.1, the settings for the game are defined n the following way:

- T_{ts_i} — 15 seconds,
- SF_{ts_i} – 1500 instructions,
- $T_{ts_{i+1}}$ — 6 seconds,
- $SF_{ts_{i+1}}$ — 8000 instructions.

Below we presented an example of the implementation of such a game with the determination of the game utility function, which is maximized during the game:

Scaling:

1. $max\{SF_{ts_i}, SF_{ts_{i+1}}\} = SF_{ts_{i+1}} = 8000$
2. $\overline{SF_{n_i}} = \frac{SF_{ts_i}}{SF_{ts_{i+1}}} = \frac{1500}{8000} = \frac{3}{16}$
   $\overline{SF_{ts_{i+1}}} = 1$
3. $u^n(s_1, s_2, T_{ts_i}, T_{ts_{i+1}}) = 15 \times \frac{3}{16}s_1 + 6 \times 1s_2$

The problem to be solved:

$$\begin{cases} \underset{s_1^n, s_2^n}{\text{argmax}} \ \frac{45}{16} s_1 + 6 s_2 \\ s_1 + s_2 = 1 \\ s_1, s_2 \in \{0, 1\} \end{cases} \tag{6}$$

Solution:

$$\begin{cases} u(s_1, s_2, T_{ts_i}, T_{ts_{i+1}}) = 6 \\ s_1 = 0 \\ s_2 = 1 \end{cases} \tag{7}$$

In the above example, the $follower$ is the game winner. It means that the follower node will not approve the the transaction sent for verification by the *leader*, but it will initiate a new transaction with its own schedule becoming the same a leader of the new game and will disseminates it across the network for confirmation.

5.1.3. Blocks mining

Transaction generated by a given node is *confirmed*, if at least 50% of the other nodes in the BC network will approve the schedule generated by such node. In such a case, the confirmed transaction is sent and added to the block. The block can contain many transactions. We assume in our model, that block can be added to the chain if the total number of operations or instructions needed to execute the tasks within all transactions in a given block will exceed BI, where BI is the parameter of the scheduler. In this paper, we set BI to 100 000:

$$\text{block is ready if} \qquad \begin{cases} \sum_{i=1}^{n} \ I_i \ \geq BI \\ BI = 100000 \end{cases}$$

Each transaction in a given ock must be validated by the transaction providers. They should use the cryptographic methods for 'signing' the transactions. Verification node check whether transactions are validated, or not. After validating all transactions in the block by $validators$, it is added to the chain. $Validators$ are the nodes that participated in the process of creating transactions and have expressed a desire to mine blocks. A leader is elected from the pool of validators based on its history of adding transactions to the

blocks (scheduling factors) at a given time and is defined by the following expression:

$$\begin{cases} L_t = max(TF_{(v_i,t)}, TF_{(v_{i+1},t)}, ..., TF_{(v_{i+n},t)}) \\ TF_{(v_i,t)}, TF_{(v_{i+1},t)}, ..., TF_{(v_{i+n},t)} \leq \frac{1}{2} BW_t \end{cases} \tag{8}$$

where

- L_t - a leader for adding the given block in the given time t
- $TF_{(v_i,t)}$ - trust factor - the sum of all I added to the blockchain by validator v_i during time t
- BW_t - the sum of all I added to the blockchain during time t

In our case, we set the value of time t to 30 days.

5.1.4. Profits for nodes

Rewarding a node for creating a transaction or participating in the process of confirming the transaction can be arbitrary. Usually in networks based on proof of stake, it is assumed that nodes charge a fee for the performing specific operations. The profits for the node for creating or confirming one transaction within an entire block NP_{tn} can be defined as follows:

$$NP_{tn} = SF_{tn}/N * 0.8 \tag{9}$$

where

- SF - the number of all I in the transaction tn.
- N - the number of all nodes confirming the transaction (including the creating node)

The node profit for mining the block NP_b can be defined as follows::

$$NP_b = SF_b * 0.2 \tag{10}$$

where

- SF_b - the number of all I in the block

The above formulas showing the profits for nodes are a proposition and can be freely modified, for example depending on whether the BC is private or public.

6. Blockchain secure cloud scheduler simulator

The model of secure cloud scheduler based on the blockchain mechanism defined in Sec. 5 was implemented for the experimental evaluation as the *Blockchain Secure Cloud Scheduler Simulator - BCSchedCloudSim*. The general concept of BCSchedCloudSim is presented in Fig. 6. The implementation of the most important components ofthe simulator, such as dedicated consensus algorithm PoSch, is original without re–using of the existing solutions such as Ethereum [31]. Ethereum supports the process of creating blockchain-based systems - dApps, such as MakerDAO, Augur, CanWork and many others [32].

6.1. MapDb database

For data storage, we used the MapDB [33] database which allows quick saving and reading of unordered data. MapDB combines the built-in database engine and the Java collection structure, and the data is processed as maps, lists and queues that can be stored on disk. MapDB is an open-source system. The sample code below shows the possible connection to the MapDb by using the DBMaker [34] tool available in Java. DBMaker allows to simplify significantly the whole connection procedure, cause it is in fact reduced just to the specification of the path to the data file, which can be stored in the blockchain (BC) network node. While the new (next) block is generated and saved, each node on the BC network must be updated.

```
1 DBMaker.Maker dbConnection = DBMaker.
2     fileDB(path).fileMmapEnable().fileLockDisable();
3 dbConnection = dbConnection.transactionEnable();
```

6.2. Networking

Communication between individual nodes in the BC network in the simulator is provided by using sockets on threads. In this approach the client must have two information:

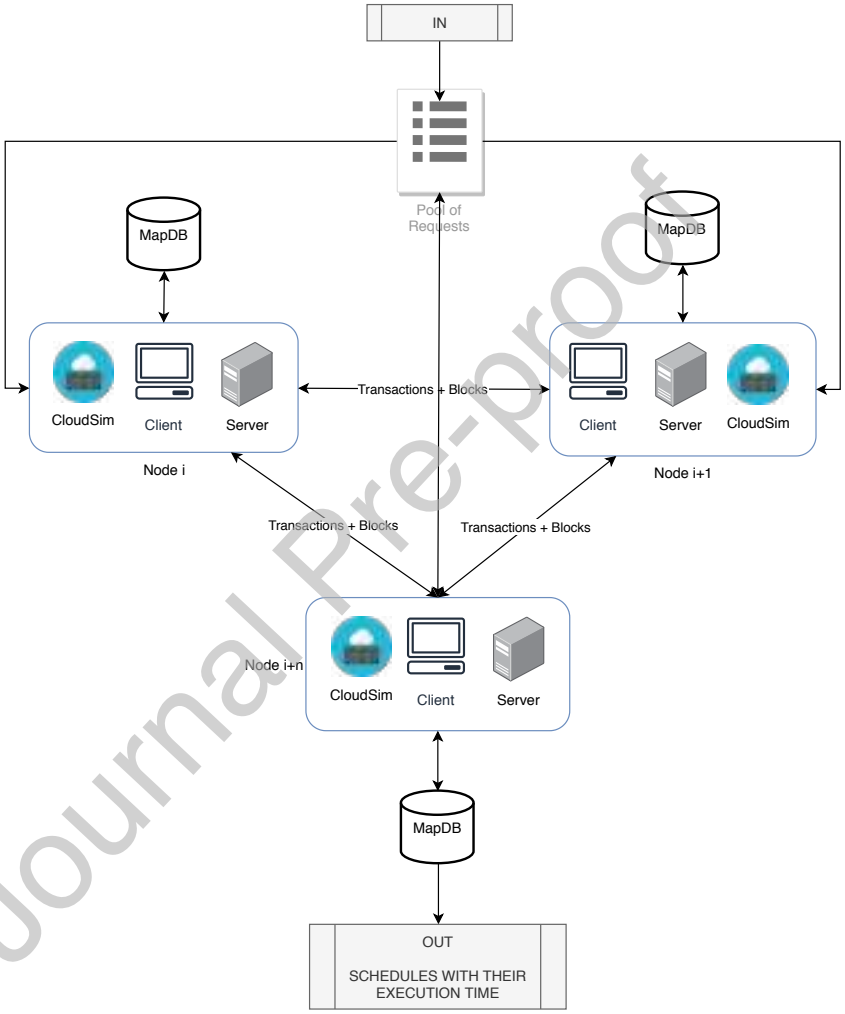

Figure 6: Scheme of Blockchain Secure Cloud Scheduler Simulator working

- server IP address,
- port number.

An example of socket-based communication is shown in Fig. 7.

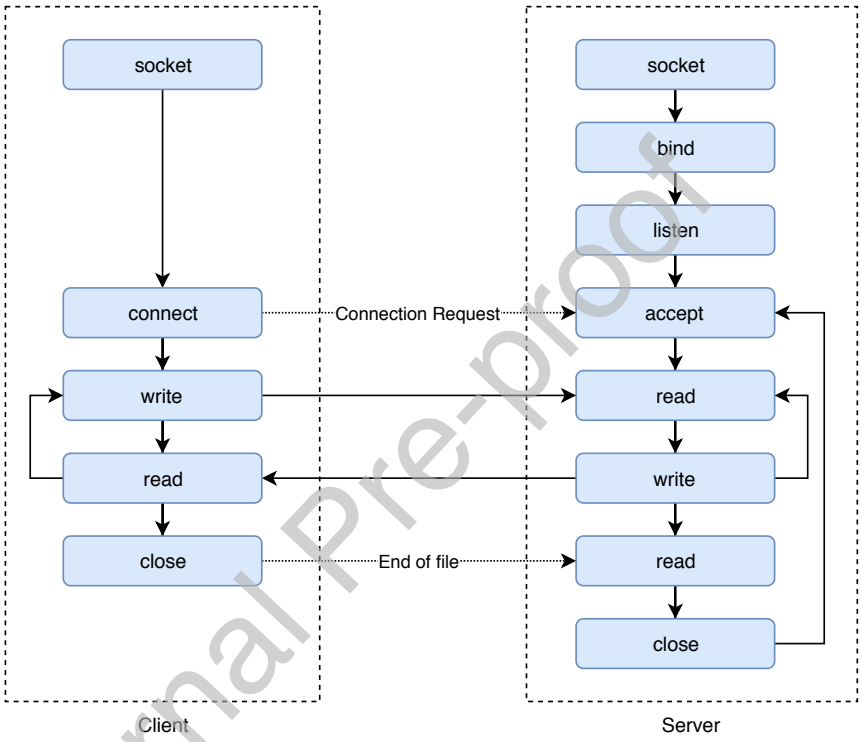

Figure 7: Socket-based communication

Two main classes, *Socket* and *ServerSocket*, were used to establish the connection. The *Socket* class is responsible for communication between the server and client so that it is possible to read and write messages. In turn, the *ServerSocket* class is used on the server side, the body of this class contains *accept*(), which blocks the console until the client connects. Below we present a sample of pseudocode of server and client:

Server:

```
ServerSocket serverSocket = new ServerSocket('port');
Socket clientSocket = serverSocket.accept();
```

Client:

```
Socket s = new Socket();
s.connect(
    new InetSocketAddress('server IP address'), 'port'), 100000
);
```

Each node in the BC network participating in the transaction confirmation process has a server and a client on its side. Every few seconds, each node sends a signal to its neighbour nodes about its current activity. According to this information, nodes send out transactions for approval only to the active nodes. Due to the use of sockets, data exchange in the network takes place very quickly.

6.2.1. Approval of the Transactions

The transaction is approved by using the algorithm and Stackelberg game based module defined in Sec. 5.1.1. We used the *org.apache.commons.math3* package for solving the maximizing problem with constraints defined by Eq. 5. The verification node in the BC network may accept or reject the transaction by using the following procedure:

```
s1Coefficient = leader.getTimeOfSchedule()*leader.
    getScaleSchedulingFactor();
s2Coefficient = follower.getTimeOfSchedule()*follower.
    getScaleSchedulingFactor();
...
Collection<LinearConstraint> constraints = new ArrayList<
    LinearConstraint>();
constraints.add(
    new LinearConstraint(new double[] { 1, 1 }, Relationship.EQ,
    1)
);
solver = new SimplexSolver();
solution = solver.optimize(...);

x = solution.getPoint()[0];
y = solution.getPoint()[1];
if (y == 1) {
    return true;
} else {
    return false;
}
```

The scaled scheduling factors of the *leader* and $follower$ are used for the calculation of the coefficients s_1 and s_2. Then the constraints are added and the process of finding a solution of the maximization problem is initialized. In the case of $y = 1$, the schedule generated by the $follower$ is of the better 'quality' than that one generated by the *leader*. It means that $follower$ won the game. In the case of win the *leader* (his schedule is better), *leader* receives the confirmation of the transaction from the $follower$.

6.2.2. Simulation of blocks mining

Blocks mining simulation is based on the model defined in Section 5.1.3. The node that participated in the generation and confirmation of most of the transactions during the specified period of time (in our case - 30 days) is selected from the pool of validators as a *leader*. The *leader* can validate and add a block to the blockchain. The pseudocode below is the implementation example of the above procedure of the *leader*:

```
public byte[] getLeader()
{
    byte[] maxTrustFactorNode = null;
    float maxTrustFactor = 0;
    float BCt = 1/2 * Node.getBlockchainTrustFactor(
    numberOfDayLimit);
    for (byte[] validator : list) {
        float TF = Node.getTrustFactor(validator,
    numberOfDayLimit, true);
        if (TF > maxTrustFactor && TF <= BCt) {
            maxTrustFactor = TF;
            maxTrustFactorNode = validator;
        }
    }
    return maxTrustFactorNode;
}
```

7. Experimental Analysis

In this section, we present the results of experiments conducted with the BCSchedCloudSim simulator. The experimental analysis was divided into

two parts. First we evaluate the blockchain-based scheduler in the scenario specified in Sec. 7.1. In the second part, we compare the efficiency of the BC-based scheduler with 4 selected cloud schedulers in Sec. 7.2.

7.1. Experimental evaluation of the BC-based cloud scheduler

The BC network was tested in the following scenario:

1. Four nodes have been defined in the blockchain network, as well as 8 transactions containing tasks and machines.
2. The node number 4 is started, which, without finding the genesis block, deals with its generation.
3. The remaining 3 nodes are started, the role of which will be primarily the confirmation of the transaction, the node number 1 is responsible for their creation and broadcasting.
4. Node number 4 takes 8 transactions from the pool, prepares a random schedule for them and broadcast them for confirmation.
5. Node 1 must obtain confirmation from at least 2 nodes (50% of the BC network), each confirmation node prepares its own random schedule and compares it with the one sent for verification.
6. After receiving the appropriate number of confirmations, node 1 prepares a block. Each block must contain at least 100 000 number of instruction or operations needed to execute the tasks, after collecting the appropriate number of transactions, the block is created.
7. The leader from the pool of validators is selected which confirms the correctness of the created block and places it in the blockchain.

7.1.1. Simulator parameters

Simulation on the proposed model were carried out on 4 nodes, responsible both for confirming the transaction and for mining blocks. Each node returns different randomly specified schedule for the transaction. Eight requests described in Tab. 1 have been added to the pool of requests. Each of values from Tasks column $(I_1, ..., I_n)$ represents one task and describes the number of operations or instructions needed to execute this task, and analogously each of values from Machines column $(M_1, ..., M_n)$ represents one machines and describes the number of floating point operations or instructions performed by this machine in 1 second. Each node is available on the defined host and port. Tab. 2 defines the numbers of hosts with which the

node currently has connections. The initial state we adopted in the simulation is an empty chain of blocks. Together with the activation of the first node in the BC network, block 0 (genesis block) is generated, which will then be sent to the rest of the BC nodes.

Table 1: Pool of Task Managers Requests

No.	Tasks $(I_1, ..., I_n)$	Machines $(M_1, ..., M_n)$
1.	(10000, 5000, 1000, 15000)	(3000, 30000, 10000)
2.	(8000, 1000, 40000)	(3000, 30000, 2500, 20000)
3.	(1000, 3000, 8000, 18000, 900)	(7000, 18000, 6000)
4.	(10000, 7500, 1000)	(1500, 6000)
5.	(20000, 4500, 3000)	(8000, 6000)
6.	(20000, 7500, 10000)	(1000, 14000, 2000)
7.	(10000, 18500, 1000)	(4000, 11000, 12000, 1000)
8.	(12000, 28500, 37000)	(2000, 15000, 12000, 1000)

Table 2: Nodes

No.	Host	Port	Connections
1.	127.0.0.1	7001	(2, 4)
2.	127.0.0.1	7002	(1, 3)
3.	127.0.0.1	7003	(2, 4)
4.	127.0.0.1	7004	(1, 3)

The simulation process described in this subsection was provided in three main stages.

7.1.2. Stage 1 - Genesis Block

We activated the node number 4 without collecting transaction from the pool. The node did not find the genesis block, so it was created according to statically defined rules in application code. The generated Genesis Block is shown below:

```
{
  "hash": "a02ba163f3a02db22fdd14b310119f
           1a4c2f9e4a773a573f757904dd5433d4dd",
  "previousHash": "0",
  "timeStamp": 1555428842347,
  "merkleRoot": "652b7b515656cfc956ll7a4268fef1f4
                 707aba3572a8bac81cba6561lc71b9ff",
  "transactions": [
    {
      "transactionId": [
        48
      ],
      "sender": [in bytes],
      "recipient": [in bytes],
      "schedule": {
        "tasks": [],
        "machines": [],
        "time": 0.0
      },
      "r": [in bytes],
      "s": [in bytes],
      "v": [28],
      "outputs": [
        {
          "id": [in bytes],
          "sender": [in bytes],
          "parentTransactionId": [
            48
          ],
          "schedulingFactor": 0.0,
          "timestamp": 1555428842347
        }
      ],
      "numberOfVerification": 0
    }
  ]
}
```


The block hash was generated using the Sha256 hash function from the "scheduler" string. As this is the genesis block, there is no preceding block so value of previous hash is equal to 0 and public keys of the sender and recipient were generated randomly. One transaction that contains an empty schedule is placed in the block. As you can see, the sender public key, recipient public

key and digital signature are saved using bytes, which makes it easier to implement and transfer data between nodes. On the listing, we hide values stored in bytes due to their length [in bytes]. We have used Elliptic Curve Digital Signature Algorithm (ECDSA) [35] to generate signature that is why we have three arrays of bytes r, s and v to save it. The schedule includes empty array of tasks and array of machines, the time of schedule and number of transaction verification are equal to 0 - there is no schedule in Genesis Block. In outputs, we have saved the sum of I within a given transaction, which later allows us to easily calculate the scheduling factor for the sender, which is the sum of all outputs in BC with the assigned public key of sender.

7.1.3. Stage 2 - Creating and confirming transactions

Node 4 is already running, now we are starting the rest of nodes. Nodes 2, 3 and 4 will listen to the network to confirm transactions, while node 1 will broadcast the transactions to confirm.

```
Node 1 log output:

receiving hb|heartbeat from another peer working on port: 7002
receiving hb|heartbeat from another peer working on port: 7004

Sending transaction to peers for accept...
sending tx|{...}
sending tx|{...}
...
```

It can be seen on the log output that node 1 gets signals about activity from nodes 2 and 4. Node 3 is active as well, but it is not connected with node 1. Node 1 gets 8 requests from the pool, prepares a random schedule for them, generates transaction with such data and sends them to confirm.

```
Node 2 log output:
...

Checking transaction...
Time of schedule verification: 41210468 [NS].
Task execution time: 21.0 seconds
Schedule incorrect, transaction rejected.

Checking transaction...
Time of schedule verification: 82199380 [NS].
Task execution time: 49.0 seconds
```

```
Schedule incorrect, transaction rejected.

Checking transaction...
Time of schedule verification: 289038093 [NS].
Task execution time: 26.0 seconds
Schedule incorrect, transaction rejected.

Checking transaction...
Time of schedule verification: 12366930 [NS].
Task execution time: 7.0 seconds
Schedule correct, transaction verified.

Checking transaction...
Time of schedule verification: 232778 [NS].
Task execution time: 26.0 seconds
Schedule correct, transaction verified.

Checking transaction...
Time of schedule verification: 345971 [NS].
Task execution time: 26.0 seconds
Schedule incorrect, transaction rejected.
...
```

The node 2 receives the prepared random schedules and compares them with its own. Each node has an empty history in BC, so the scheduling factor for each of them is randomized. Some transactions are rejected by the node and some are accepted. In the case of acceptance to node 1, a return message with transaction confirmation is sent.

```
Node 1 log output:

Transaction ... was verified.
Transaction ... was verified.
...
Transaction added to verified pool.
Transaction ... was verified.
...
Transaction added to verified pool.
...
```

After obtaining confirmations from the 50% network, in this case 2 nodes, the transaction is added to the verified transaction pool from it can be added

to the block. The transaction will not be considered as valid until it receives the appropriate number of confirmations. In the situation, where any of the nodes to which the transaction was sent failed and did not respond for a long time, it is necessary to resend the transaction to other nodes. In the case of the simulator there are only 4 nodes, but in practice the number of nodes in the network is much larger and is constantly changing. Due to that fact the transaction is sent to more nodes than required, while after receiving the right amount of confirmations, the rest of the response is ignored.

7.1.4. Stage 3 - Mining blocks

After reaching the minimum number of instructions or operations needed to execute the tasks within all transactions from the block (100 000), the mining process begins.

```
Node 1 log output:

Number of operations or instructions needed to execute the tasks
    within the created block: 127000.0
Transaction successfully added to the block.
Transaction successfully added to the block.
Transaction successfully added to the block.

Validator scheduling factor: 127000.0
Block mined!!! : fc0c15ed4d3b0f224beee3c6b2c9d44b
                 beb0e57a20e51e2ef32af7bb7a858e0c
Time of mining 8592327 [NS].
```

The node received confirmation for 3 transactions, each transaction was confirmed by at least 2 nodes (50% of the network). Total number of all operations or instructions needed to execute the tasks within this 3 transactions is equal to 127000, so a block can be created. All transactions are added to the block, then the trust factor is calculated for the mining node (validator). At this moment the BC consists only the genesis block, so TF of mining node considering the last 30 days is equal to the sum of all I from transactions created by it within the block that is currently being mine.

The fragment of the added block is presented below, one of the three transactions that the block contains is placed on the listing:

```
{
  "hash": "fc0c15ed4d3b0f224beee3c6b2c9d44
           bbeb0e57a20e51e2ef32af7bb7a858e0c",
```

```
"previousHash": "a02ba163f3a02db22fdd14b310119f1
                 a4c2f9e4a773a573f757904dd5433d4dd",
"timeStamp": 1555429630021,
"merkleRoot": "5f03029462fe8118d8ece16db7c541f
               5e4286cba3bf2120f1631b4934441b093",
"transactions": [
  {
    "transactionId": [in bytes],
    "sender": [in bytes],
    "recipient": [in bytes],
    "schedule": {
      "tasks": [
        {
          "id": 1,
          "numberOfOperations": 10000.0
        },
        {
          "id": 2,
          "numberOfOperations": 7500.0
        },
        {
          "id": 3,
          "numberOfOperations": 1000.0
        }
      ],
      "machines": [
        {
          "numberOfOperationsPerSecond": 1500.0,
          "tasksToExecute": []
        },
        {
          "numberOfOperationsPerSecond": 6000.0,
          "tasksToExecute": [1,2,3]
        }
      ],
      "time": 8.0
    },
    "r": [in bytes],
    "s": [in bytes],
    "v": [28],
    "outputs": [
      {
        "id": [in bytes],
        "sender": [in bytes],
        "value": -20.0,
```

```
            "parentTransactionId": [in bytes],
            "schedulingFactor": 18500.0,
            "timestamp": 1555429630021
          }
        ],
        "numberOfVerification": 2
      },
      ...
  ]
}
```

7.1.5. Discussion

Presented numerical results confirm the correctness of the implementation and its compliance with the algorithm described in Sec. 5. In the simulation, each node has a connection with 2 others.

At the beginning Genesis Block with hard-coded into classes string "scheduler" was generated. This block contains one empty transaction (without schedule). In the request pool, 8 requests were placed, all of them were collected by node 1, which prepared random schedules, generate transactions and sent them to other nodes for confirmation. All 4 nodes had an empty account at the moment, none of them prepared schedules that had been placed in BC, therefore their scheduling factors during the verification process have been chosen randomly. A correctly verified transaction had to be confirmed by at least 2 nodes. Node 1 received such number of confirmations for 3 transactions. Examples of transaction confirmation times are shown in the table below:

Table 3: Transaction confirmation times

No.	Time in [NS]
1.	41210468
2.	82199380
3.	289038093
4.	12366930
5.	232778
6.	345971

The average transaction confirmation time is 70898936 nanoseconds, which

gives about 0.07 seconds, so it is very short, which leads to a quick preparation of the schedule for the end customer, but it should be taken into account that these schedules were prepared randomly. Node 1, after added confirmed transactions to the block, obtained the appropriate amount of BI (127 000) needed to add a block to BC. Next, from the pool of validators involved in creating the transactions within given block (only node 1), the leader node is selected, which proceeds to the mining process. This process is not based on the computing power of the mining node, so it is more similar to Proof of Stake than to Proof of Work. The application of this approach has led to the fact that the mining process takes place very quickly and lasts about 0.009 seconds, thanks to which the energy needed to power the processing unit is not wasted. As a result of this process, a block consisting of 3 transactions containing the best schedules established by the network was created. The end user can get them using his public key. Each transaction contains tasks and machines on which they are to be executed in accordance with the identifiers of tasks placed in the *tasksToExecute* array.
Taking into account the results obtained, it can be clearly seen that the implementation has been done correctly and the results obtained are promising.

7.2. Comparative analysis of the performances of BC-based scheduler against to the selected cloud scheduling algorithms

The proposed scheduler has been tested and compared with individual algorithms according to the following scenario:

1. Five different schedulers were used for task scheduling, each of them is evaluated by the time of execution the last task from schedule (makespan). The same data is uploaded to each scheduler.
2. The task scheduling process is run on four different schedulers. Each of them uses a different algorithm, includes: First Come First Served (FCFS), Hybrid Heuristic based on Genetic Algorithm (HSGA), Round Robin (RR) and Shortest Job First (SJF) [36], [37], [38], [39].
3. The fifth one, blockchain-based task scheduling (BS) process is started. There are sixteen nodes defined in the BC network that use the same algorithms as the schedulers from point 2.

7.2.1. Simulator parameters

Experiments on the performance of the proposed model were carried out using the CloudSim simulator [40]. As a measure of the evaluation, the

time of completing the last task (makespan) was adopted. The simulator parameters are shown below:

- 1 Data Center,
- image size of VM - 10000 MB,
- memory of VM - 4096 MB,
- number of CPUs in VM - 4,
- computing capacity of VM - the number of floating point operations performed in one second by the virtual machine expressed in MFLOPS (milion floating point operations performed in one second), random value in the range 1 to 12,
- workload of task (I) - the number of operations or instructions needed to execute the task expressed in MFLO (milion floating point operations), random value in the range 100 to 1000,
- 16 nodes in the BC network, each 4 nodes use the same scheduling algorithms, so 4 different scheduling algorithms can be distinguished across the entire network,
- each transaction (schedule) must be confirmed by at least 8 nodes (50% of the network),
- 1000000 - the minimum number of operations or instructions needed to create a new block (sum of I),
- each node in BC initially has the same scheduling factor, which is equal to 2000.

Each simulation was carried out 32 times and then the average makespan value was pulled out. The experiment was conduct for different number of tasks and virtual machines. As can be seen in the Fig. 8, 9, 10 and 11 in three out of four experiments carried out, the proposed BS was the best. In second experiment with 300 task and 15 virtual machines, SJF turned out to be the best, the difference between BS and SJF is very small, the result may be due to the fact that the first block in the blockchain network was created by a node using a different algorithm than SJF, due to the difference in makespan

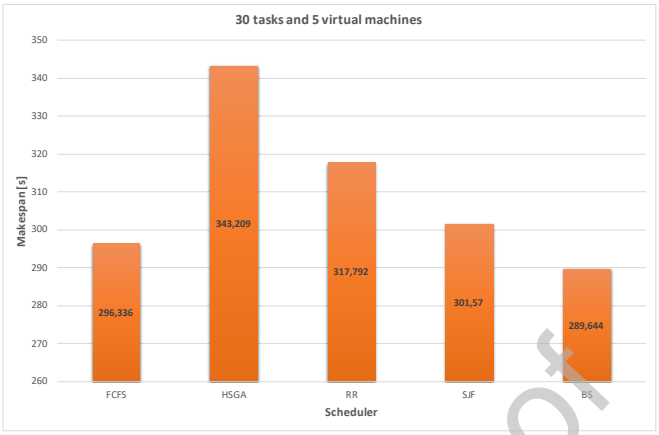

Figure 8: Evaluation of the model performance with 30 tasks and 5 virtual machines.

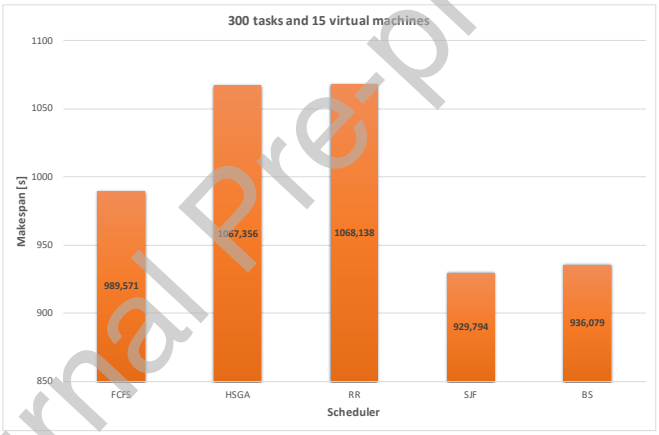

Figure 9: Evaluation of the model performance with 300 tasks and 15 virtual machines.

is small, comparing it with BS, the schedule was considered correct. In one case the result of BS is on the second place, but the client is more certain that the schedule was prepared correctly because it was confirmed by many nodes (not only one provider). To determine the significance of the results obtained, the two-tailed Wilcoxon Signed-Ranks Test for Paired Samples was also carried out as part of the experiment. For this test we use the following null hypothesis:

Null Hypothesis 1. *H_0: any differences between the worst result and BS result is due to chance (BS does not return the optimal result)*

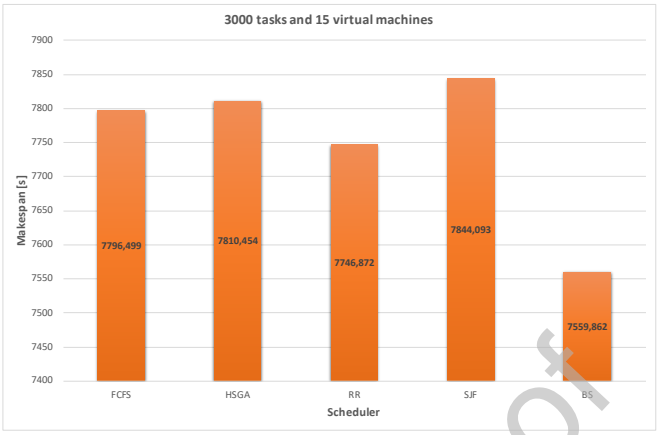

Figure 10: Evaluation of the model performance with 3000 tasks and 15 virtual machines.

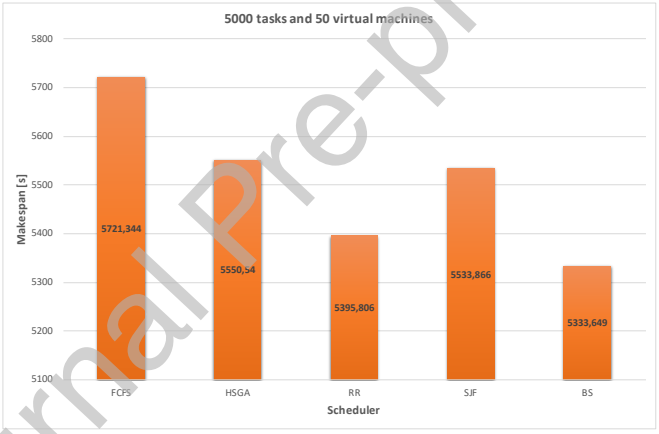

Figure 11: Evaluation of the model performance with 5000 tasks and 50 virtual machines.

The test was conducted for each experiment, selecting as a sample the worst result returned by a single scheduling module and the result returned by BS. Table 4 presents the test results:

where:

- T_+ - Positive Sum
- T_- - Negative Sum
- T - Test Statistic

Table 4: Wilcoxon Signed-Ranks Test for Paired Samples results

Experiment	Samples with σ	T_+	T_-	T	n
30 tasks, 5 VM	HSGA (94.26) & BS (72.78)	377	151	151	32
300 tasks, 15 VM	RR (244.082) & BS (174.527)	402	126	126	32
3000 tasks, 15 VM	SJF (493.501) & BS (502.938)	373	155	155	32
5000 tasks, 50 VM	FCFS (580.278) & BS (486.646)	398	130	130	32

- n - number of samples where difference was not equal to 0

The critical value for the T statistic from the Wilcoxon Signed-Ranks Table for $n = 32$ and $\alpha = .05$ is equal to $T_{crit} = 159$, which gives the range $[0, 159]$. Since T statistic of each performed test is in the range $[0, 159]$, we can in any case reject the null hypothesis, and therefore conclude that there is a significant difference between result obtained from single scheduling module and BS scheduler. This confirms that the network is working properly and that the nodes in the network work together to achieve the best result.

8. Conclusions and Future Work

In this paper, we presented the model of the new cloud scheduler based on the blockchain technology. The reason of our interest in blockchain architecture and algorithms was the improvement of the security in the task allocation, data storage and transmission among the cloud nodes and clusters and easier generation of the optimal schedules, which are approved by the nodes in the communicating cloud clusters. That scheduling optimization problem has been solved by using the asymmetric Stackelberg game model with the schedule execution time as privileged criterion. We developed specially designed simulator of the validation of the proposed blockchain cloud scheduler under various combinations of parameters. The developed scheduler was also evaluated and compared with the other popular cloud schedulers in a simple experimental analysis. The provided experiments confirmed the effectiveness of the blokchain technology in supporting the generation and secure execution of the optimal schedules.

The most important observations of the experiments carried out are:

- in most cases, the proposed BS returns a schedule with better makespan than existing individual schedulers,

- the greater the number of tasks to be executed, the more favorable schedule the BS returns (greater differences in makespan result into greater differences in costs),
- a different number of virtual machines does not have a significant impact on the returned result, BS uses the same scheduling algorithms as individual modules, but its advantage is that many different algorithms are compared with each other and the best one is selected.

The presented model and simulations are the first steps in our research on the security aspects in cloud scheduling and data and resource allocation in the distributed computational environments. In our future research, first we would like to consider the graphs of the interconnected tasks (modelled by DAGs) We also would like to consider the very realistic scenario, where different cloud clusters may work under different cloud technologies, which will be the example of the multi-cloud system.

Acknowledgment. Joanna Kołodziej's work is a part of the CYBERSECIDENT/-369195/I/NCBR/2017 project supported by the National Centre of Research and Development in the frame of CyberSecIdent Programme.

References

[1] R. Buyya, a. et., Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation Comp. Syst. 25 (2009) 599–616.

[2] M. Zbakh, M. Essaaidi, P. Manneback, C. E. Rong, Cloud Computing and Big Data: Technologies, Applications and Security, Springer Vlg., 2019. doi:10.1007/978-3-319-97719-5.

[3] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, H. Wang, An overview of blockchain technology: Architecture, consensus, and future trends, 2017. doi:10.1109/BigDataCongress.2017.85.

[4] A. Wilczyński, A. Jakóbik, J. Kołodziej, Stackelberg security games: Models, applications and computational aspects, Journal of Telecommunications and Information Technology 3 (2016) 70–79.

[5] H. Takabi, J. Joshi, G.-J. Ahm, Security and privacy challenges in cloud computing environments, IEEE Security and Privacy 8 (2010) 24–31.

[6] S. Subashini, V. Kavitha, A survey on security issues in service delivery models of cloud computing, Journal of network and computer applications 34 (2011) 1–11.

[7] A. Asma, A. Mousmi, M. , Cloud computing security issues, International Journal of Application or Innovation in Engineering & Management 1 (2012) 141–147.

[8] J. Kołodziej, F. Xhafa, Integration of task abortion and security requirements in ga-based meta-heuristics for independent batch grid scheduling, Computers and Mathematics with Applications 63 (2012) 350–364.

[9] S. Song, K. Hwang, Y. Kwok, Trusted grid computing with security binding and trust integration, J. of Grid Computing 3 (2005) 53–73.

[10] S. Song, K. Hwang, Y. Kwok, Risk-resilient heuristics and genetic algorithms for security assured grid job scheduling, IEEE Tran. on Computers 55 (2006) 703–719.

[11] D. Grzonka, A. Jakóbik, J. Kołodziej, S. Pllana, Using a multi-agent system and artificial intelligence for monitoring and improving the cloud performance and security, Future Generation Computer Systems 86 (2018) 1106–1117.

[12] F. A. Lokhandwala, A Heuristic Approach to Improve Task Scheduling in Cloud Computing using Blockchain technology, Master's thesis, Dublin, National College of Ireland, 2018. URL: `http://trap.ncirl.ie/3299/`.

[13] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, F. Wang, Blockchain-enabled smart contracts: Architecture, applications, and future trends, IEEE Transactions on Systems, Man, and Cybernetics: Systems (2019) 1–12.

[14] S. Chowdhury, Survey on various scheduling algorithms, "Imperial Journal of Interdisciplinary Research (IJIR) 3 (2018) 4.

[15] Z. Hong, Z. Wang, W. Cai, V. C. M. Leung, Blockchain-empowered fair computational resource sharing system in the d2d network, Future Internet 9 (2017) 85.

[16] U. N. Kar, D. K. Sanyal, An overview of device-to-device communication in cellular networks, ICT Express 4 (2018) 203 – 208.

[17] M. Zhang, Y. Yang, Z. Mi, Z. Xiong, An improved genetic-based approach to task scheduling in inter-cloud environment, in: 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015, pp. 997–1003. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.187.

[18] H. Karatza, Performance of clouds - issues and research directions, in: CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science, Porto, Portugal, 18 - 21 April, 2012, 2012, p. 05.

[19] I. Moschakis, H. Karatza, Parallel job scheduling on a dynamic cloud model with variable workload and active balancing, in: 16th Panhellenic Conference on Informatics, PCI 2012, Piraeus, Greece, October 5-7, 2012, 2012, pp. 93–98. doi:10.1109/PCi.2012.16.

[20] R. Annette, A. Banu, S. Shriram, A taxonomy and survey of scheduling algorithms in cloud: Based on task dependency, In: International Journal of Computer Applications 82 (2013) 20–26.

[21] G. Stavrinides, H. Karatza, Scheduling real–time dags in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes, Future Generation Comp. Syst. 28 (2012) 977–988.

[22] J. Kołodziej, Evolutionary Hierarchical Multi–Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems, Springer, 2012.

[23] D. Grzonka, J. Kołodziej, J. Tao, S. Khan, Artificial neural network support to monitoring of the evolutionary driven security aware scheduling in computational distributed environments, Future Generation Computer Systems 51 (2015) 72–86.

[24] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, Cryptography Mailing list – metzdowd.com, 2009.

[25] I.-C. Lin, T.-C. Liao, A survey of blockchain security issues and challenges, I. J. Network Security 19 (2017) 653–659.

[26] A. Joshi, M. Han, Y. Wang, A survey on security and privacy issues of blockchain technology, Mathem. Foundat. of Comput. 1 (2018) 121–127.

[27] A. Wilczyński, A. Widłak, Blockchain networks security aspects and consensus models, Journal of Telecommunications and Information Technology 2 (2019) 46–52.

[28] 51% attack, https://www.investopedia.com/terms/1/51-attack.asp, 2019.

[29] D. Yaga, P. Mell, N. Roby, K. Scarfone, Blockchain Technology Overview, National Institute of Standards and Technology Internal Report 8202 – Draft NISTIR 8202, 2018.

[30] J. . Reeb, S. Leavengood, Using the Simplex Method to Solve Linear Programming Maximization Problems, Oregon State University, Extension Service, 1998.

[31] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, cryptopapers.net, Ethereum Project Yellow Paper, 2014.

[32] State of the dapps ranking the best ethereum dapps, https://www.stateofthedapps.com/rankings/platform/ethereum, 2019.

[33] J. Kotek, MapDB, Release 2.0, MapDB, 2016.

[34] Dbmaker, http://www.mapdb.org/javadoc/latest/mapdb/org/mapdb/DBMaker.html, 2019.

[35] D. Johnson, A. Menezes, S. Vanstone, The elliptic curve digital signature algorithm (ecdsa), International Journal of Information Security 1 (2001) 36–63.

[36] N. Xoxa, M. Zotaj, I. Tafa, J. Fejzaj, Simulation of first come first served (fcfs) and shortest job first (sjf) algorithms, International Journal of Computer Science and Network 3 (2014) 444–449.

[37] A. Rajguru, S. Apte, A performance analysis of task scheduling algorithms using qualitative parameters, International Journal of Computer Applications. 74 (2013) 33–38.

[38] M. Fahmy, CloudSim Task Allocation and Scheduling, github.com/michaelfahmy/ cloudsim-task-scheduling, 2017.

[39] M. Fahmy, HSGA: a hybrid heuristic algorithm for workflow scheduling in cloud systems, github.com/Deeksha96/Task-Scheduling-for-Cloud-Computing-Using-Genetic-Algorithm, 2018.

[40] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, , R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Journal SoftwarePractice & Experience 41 (2011) 23–50.