AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF COMPUTER SCIENCE, ELECTRONICS AND TELECOMMUNICATIONS

DEPARTMENT OF COMPUTER SCIENCE

PhD Thesis

Blockchain-based task scheduling in computational clouds

Andrzej Wilczyński

Supervisor:	Professor Joanna Kołodziej, Ph.D., D.Sc.
Co-supervisor:	Agnieszka Jakóbik, Ph.D.

Kraków, 2020

Acknowledgements

I would like to express my gratitude to my supervisor Professor Joanna Kołodziej for substantive care, possibility to expand my scientific knowledge and helpful remarks during creating this dissertation.

I am also very grateful to Agnieszka Jakóbik, Ph.D., co-supervisor, for her unique help and fruitful discussions that contributed to the submission of this thesis.

I would also like to thank my tutor Aleksander Byrski, Ph.D., D.Sc., for his supporting during this important stage in my scientific development.

My gratitude is also addressed to colleagues from Department of Computer Science, Cracow University of Technology, for many interesting conversations and opportunities to learn.

Last but not least, I would like to thank my wife, Iza, who constantly sustained me in the pursuit of completing this dissertation, to my whole family and friends for support and faith, that gave me strength and motivation to act.

Abstract

Dynamic development of computer science stands behind an increase in demand for the various services provided via cloud computing. Problem is, many customers want to use the same services at the same time. This compels cloud service providers to improve task scheduling in order to achieve the desired quality and pace of the services, in accordance with the provisions of the Service Level Agreements. The aim of task scheduling is to create a precise schedule according to which tasks will be executed on the particular resource at the determined time. Cloud service providers must take into account the specific requirements of their end clients that are inevitably connected with the quality of results and, therefore, with the performance of the implemented task scheduling systems. However, scheduling is hardly limited to minimising costs and maximising efficiency because it also affects aspects related to security. The importance of the above issues in modern IT systems require innovative solutions and constant improvements. Thus, a new approach to finding the optimal schedule was proposed, based on blockchain technology and called *Secure Blockchain Scheduler*.

Blockchain operation consists in maintaining a joint and collective ledger of records in a digital form, distributed over the network, in the same copies. A key aspect in blockchain network is to determine which user can publish records, and this requires the implementation of a consensus model. In this dissertation, a consensus called *Proof of Schedule* was proposed. Based on Stackelberg game, it regulates checking and adding new blocks to the blockchain and determines how to validate schedules stored in transactions. Such an approach must result in the competition between different schedule providers, won by the one who takes account of the client's requirements faster and prepares an optimal schedule to meet them.

Security issues are unavoidable during the preparation of any schedule. Their ever-changing nature requires special attention and continuous improvement. To examine whether the proposed solution is safe, the *security level* of the schedule was defined whose value depends on three probabilities: probability of machine failure during tasks execution due to high security restrictions, probability of sending a false or incorrect schedule by the scheduler and probability of an unauthorized manipulation or modification of the prepared schedule.

As a part of the dissertation, the *Blockchain Secure Cloud Scheduler Simulator* was implemented in which the proposed consensus algorithm was used. The application of the simulator included conducting comparative simulations to evaluate the properties of the proposed blockchain scheduler against some competitive scheduling modules. The results demonstrate that the blockchain scheduler significantly improved the efficiency and security of the prepared schedules. The proposed approach extends the possibilities of using different scheduling modules by the end-users. By delegating the preparation of the schedules, providers can get benefits only for that, without having to execute tasks from customers.

Streszczenie

Dynamiczny rozwój dziedziny informatyki skutkuje ciągłym wzrostem popytu na różne usługi świadczone przez chmury obliczeniowe. Zapotrzebowanie to rodzi różnego rodzaju problemy, zwłaszcza gdy wielu klientów chce korzystać z tych samych usług w tym samym czasie. Sytuacja ta zmusza dostawców do skupienia sie na odpowiednim harmonogramowaniu zlecanych zadań w celu osiągnięcia pożądanej jakości i tempa świadczenia usług, zgodnie z umowami określającymi gwarantowany poziom ich świadczenia. Celem szeregowania zadań w chmurach obliczeniowych jest stworzenie dokładnego harmonogramu, zgodnie z którym zadania będą wykonywane na określonym zasobie w danym czasie. Dostawcy usług muszą brać pod uwagę szczególne wymagania swoich klientów, które są związane przede wszystkim z jakością otrzymywanych wyników i minimalizacją kosztów. Harmonogramowanie zadań w chmurach obliczeniowych nie ogranicza się jednak tylko do tych czynników, bardzo ważne są również aspekty związane z bezpieczeństwem. Znaczenie tych zagadnień w nowoczesnych systemach IT wymaga innowacyjnych rozwiązań i ciągłych ulepszeń, dlatego w ramach tej rozprawy doktorskiej zaproponowane zostało nowe podejście do znajdowania optymalnego harmonogramu zgodnie z którym świadczone będą usługi. Rozwiązanie to zostało nazwane *Secure Blockchain Scheduler* i jest oparte na technologii bockchain.

Blockchain jest to zdecentralizowana księga rekordów przechowywana w formie cyfrowej, dystrybuowana przez sieć w tych samych kopiach. Kluczowym aspektem w sieci blockchain jest ustalenie, który użytkownik może publikować nowe rekordy, co wymaga wdrożenia modelu konsensusu. W ramach rozprawy zaproponowano algorytm *Proof of Schedule*, oparty na grze Stackelberga. Konsensus ten reguluje dodawanie nowych bloków do księgi i określa sposób walidacji harmonogramów zapisanych w transakcjach. Podejście to wymusza konkurencyjność pomiędzy różnymi dostawcami harmonogramów. Wygrywa ten, który uwzględniając wymagania klienta najszybciej przygotuje optymalny i bezpieczny harmonogram.

Kwestie bezpieczeństwa podczas przygotowywania harmonogramu i ciągle zmieniające się normy z tym związane również wymagają szczególnej uwagi i doskonalenia. W celu weryfikacji czy zaproponowany harmonogram spełnia odpowiednie wymogi w tym zakresie zdefiniowano poziom bezpieczeństwa harmonogramu jako *security level*. Na wartość *security level* wpływają trzy czynniki: prawdopodobieństwo awarii maszyny podczas wykonywania zadań, prawdopoobieństwa wysłania przez moduł harmonogramujący fałszywego lub niepoprawnego harmonogramu oraz prawdopodobieństwo nieautoryzowanej modyfikacji przygotowanego harmonogramu.

W ramach rozprawy zaimplementowano symulator *Blockchain Secure Cloud Scheduler*, w którym wykorzystano zaproponowany algorytm konsensusu. Symulator został wykorzystany do przeprowadzenia eksperymentów porównawczych. Zwracane przez niego wyniki są optymalne i

spełniają założone wymogi bezpieczeństwa, a na tle wyników zwracanych przez inne moduły harmonogramujące wypadają najlepiej. Delegowanie samego przygotowania harmonogramów pozwala na czerpanie korzyści finansowych przez dostawców za sam harmonogram bez konieczności jego wykonywania. Rozszerza to znacznie możliwości wykorzystania dostępnych na rynku modułów harmonogramujących zadania przez użytkowników końcowych.

Contents

Chapter 1

Introduction

This chapter introduces the main topics of the research provided in the dissertation, cloud computing essentials and motivation of work. The scheduling problems in computational cloud were defined here and the main research hypothesis verified in the dissertation was formulated. The chapter ends with a short description of the organisation of the thesis and contents of the chapters.

1.1. Task scheduling in cloud computing

Dynamic development of the research in computer science leads to an increase in demand for computational resources such as computational and data servers, warehouses, databases, networks or services dedicated to data analysis and exploration. Cloud computing paradigm has been addressed as a methodology and computing services and architecture to manage with these challenges. The term *'cloud computing'* is very general nowadays. The classical definition of *'cloud computing'* (CC) is proposed by Buyya et al. [1]. They defined CC as an extension of the grid infrastructure consisting of data centres, where the capabilities of business applications are provided as services that can be accessed through the network. Cloud service providers receive profits for enabling their customers to access such services. On the other hand, consumers are motivated by the reduction of the related costs. Cloud computing is not a completely new model or paradigm but rather an evolution of previously developed models and technologies, such as:

– Computational Grid [2] - a system model composed of many connected computers in the distributed clusters [1] that cooperate in a large-scale network, which ensures multiplication of computing power and reliability of such a solution,

– Virtualization - an approach that allows the virtualization of the available resources in such a way that many computing units are visible as one large computing unit. There is no need in the grid to overhaul the hardware infrastructure to obtain more computing power, and both the infrastructure and computing power are optimally used. There are many software tools that allow virtualizing machines, for instance: VMware, KVM, Xen [3] or OpenStack Platform [4] which is a more recent and innovative solution. Computational Grid may refer to the hardware resources as well as to the data layer that provides a simpler interface and methods for accessing data. There can be many sources of data, but the user who relies on this data will see one abstract layer [5],

– Utility Computing Network[6] - a model for providing specific resources on-demand and estimating fees based on their consumption,

– Service-Oriented Architecture (SOA) [7] - an approach to software development in which the main focus is on the defined services that meet the user's requirements.

The world-leading cloud providers such as Google, Microsoft or Amazon, initially used clouds in running their internal business operations. However, after the building of large data centres and data servers farms in many countries [8], [9], they noticed a broader potential of the solution and started offering the external enterprises the previously unused resources or services such as data storage or data processing. There are various methods of classification of the cloud environments. Based on the access to the cloud services and resources, we may classify clouds into the following three categories:

– public clouds - externally available services everyone can use for a fee depending on the application,

– private clouds - based on the infrastructure supporting only one entity; they are typically used by the companies processing sensitive data they do not want to send outside their internal systems,

– hybrid clouds - where processing and storage tasks are partially performed by the public cloud, and partially by private systems.

The most popular model of the cloud environment defines CC as a multilayer system [10], where the following layer-stack can be specified:

– Infrastructure as a Service (IaaS) - the bottom layer of the system, it provides the client with IT infrastructure such as software, hardware or servicing,

– Platform as a Service (PaaS) - the middle layer, it provides ready-to-use and customized applications without the need to purchase hardware or software licenses,

– Software as a Service (SaaS) - the upper layer, it provides users with specific software features, such as e-mail access or calendar.

Services dedicated to meeting the different needs of their end-users may be crucial for the business operations of the consumers. Therefore it is necessary to guarantee the appropriate level of their provision, which is usually regulated through the Service Level Agreement (SLA) [11] brokered between the providers and consumers. As many clients may want to use the same services at the same time, the providers must schedule tasks to achieve the desired quality and pace of service, in accordance with the provisions of the SLA. On the other hand, from the perspective of the providers, it is important to minimize the maintenance costs by shutting down resources where the currently unused services are running. To do it quickly, proper scheduling of tasks is necessary.

1.2. Scheduling problems in computational clouds

A basic task scheduling model is shown in Fig. 1.1. In the figure, clients directing requests to the cloud can be seen. The requests are collected by the cloud broker (task scheduler) responsible for decomposing requests for smaller tasks and directing them to virtual machines. After the tasks are executed, the results are returned to the broker who subsequently passes them to the cloud client.

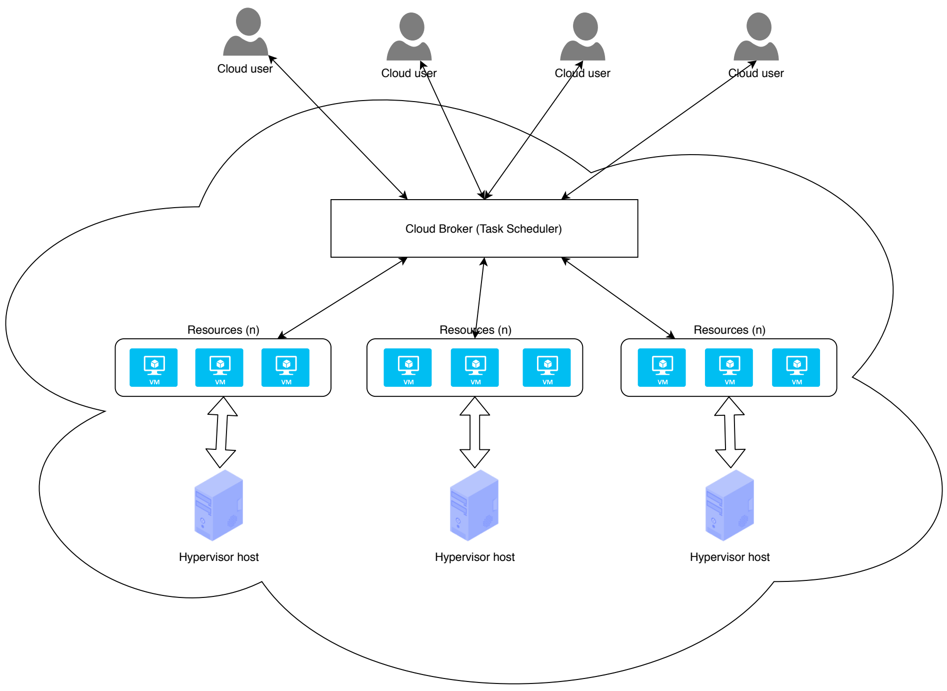

Fig. 1.1. Basic task scheduling process in cloud computing [12]

The aim of task scheduling is to build a schedule that determines when to execute each task and which resources should be selected to do it. For instance, tasks must be scheduled when there is a need to execute a number of calculations provided by the users and deliver the results within a certain time. To ensure a guaranteed Quality of Service (QoS) [13] to the clients, it is necessary to make as efficient mapping of tasks to the given resources as possible; otherwise, the clients will not pay for them. The task scheduling is, therefore, considered as one of the burning issues to tackle in cloud computing systems.

1.2.1. Types of scheduling problems and metaheuristics used to solve them

Task scheduling in computational clouds is a very complex problem. In fact, it is a set of problems and definition of the concrete problem may be formulated based on various cloud scheduling attributes, namely:

– the environment (static or dynamic),

– cloud architecture (centralized, decentralized or hierarchical),

– task processing policy (immediate or batch),

– tasks' interrelations (independency or dependency).

Karatza et al. [14] defined the following instances of the cloud scheduling problems:

– bag-of-tasks scheduling - jobs consisting of independent tasks that can be processed in parallel,

– gang scheduling - jobs consisting of tasks that often communicate with one another, which can be processed in parallel,

– Directed Acyclic Graph (DAG) scheduling [15] - jobs consisting of tasks with a significant order of execution (workflow); tasks can be planned on different system nodes,

– real-time scheduling - composed of jobs in which the deadlines for executing tasks are defined,

– fault-tolerant scheduling - jobs in which there is a high probability of software failures that may prevent the execution of the schedule.

Annette et al. [16] use a simpler classification and distinguish between dependent and independent tasks that can be defined as follows:

– dependent tasks - tasks that must be executed in a certain order; some of them need the results calculated by other tasks,

– independent tasks - tasks that are not mutually dependent and there is no need to execute them in the defined order.

In general, the problem of mapping tasks on largely extensive computational resources in the cloud is one of the problems referred to as NP-hard problems [17]. It is very difficult to provide the optimal solution in polynomial time for this category of problems. However, there are some meta-heuristic [18] algorithms that can provide sub-optimal solutions to this type of problem. Among these techniques are for instance [19]:

– QoS-based task scheduling algorithms - the schedule is evaluated with regard to the various criteria pertaining to the Quality of Service parameters,

– Ant Colony Optimization - algorithms based on arbitrary searches using positive criticism system and imitating the behaviour of real ant colonies in nature,

– Genetic Algorithm (GA) based task scheduling - algorithms based on genetic algorithms and neural networks,

– Particle Swarm Optimization (PSO)- algorithms based on population behaviour and quite popular due to the ease, adequacy and wide range of their use,

– Fuzzy-based task scheduling - based on fuzzy algorithms.

1.2.2. Scheduling criteria

There are many algorithms dedicated to task scheduling, each of them evaluated according to multiple criteria. These criteria may be desirable either for the client or for the provider. Most of them are defined as optimizing criteria, but the constraints and conditions referring to the security issues are also included. Fig. 1.2 presents an example of a schedule arranged for 14 tasks where 3 machines are available for their execution; the example will be evaluated using some different criteria.

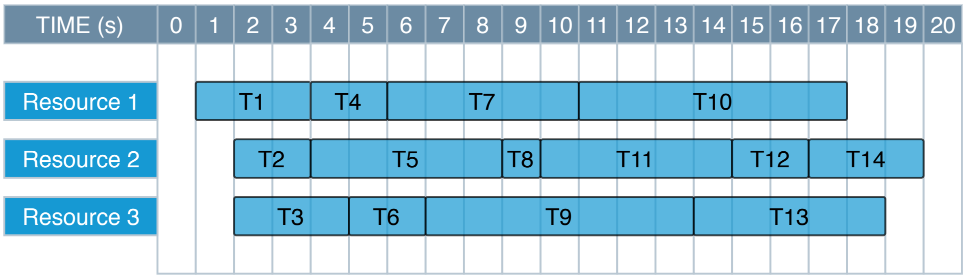

Fig. 1.2. Example of schedule

Optimization criteria are related to the optimization of the schedule in order to execute it faster and cheaper for the clients or to do the individual tasks at a predetermined time. Some of the criteria for task scheduling methods, are listed below:

1. Makespan - the time of finishing the last task from the batch; the smaller the makespan is, the faster the tasks are completed:

$$makespan = max\{ET_i, ET_{i+1}, ..., ET_n\} \tag{1.1}$$

where

ET_i – the ending time of the task i

n – number of tasks in the batch

Makespan for the schedule from Fig. 1.2 is 19 s (ending time of task number 14).

2. Flowtime - the sum of the ending times of all tasks from the batch; this metric describes the response time to the client for the submitted task, and its minimization means a reduction in the average response time of the entire schedule:

$$flowtime = \sum_{i=1}^{n} ET_i \tag{1.2}$$

where

ET_i – the ending time of the task i

n – number of tasks in the batch

Flowtime for the schedule from Fig. 1.2 is:
ET1 + ET2 + ... + ET13 + ET14 = 3 + 3 + 4 + 5 + 8 + 6 + 10 + 9 + 13 + 17 + 14 + 16 + 18 + 19 = 139 s

3. Economic cost - the total sum the client has to pay to the provider for the resource utilization

$$Economic\ Cost = \sum_{i=1}^{m} (C_i * T_i) \tag{1.3}$$

where

C_i – the cost of 1 second of utilization the resource i

T_i – time in which the resource i is utilized

m – number of resources

Assuming that the cost per 1 second of utilization is equal to: Resource 1 = 100, Resource 2 = 150, Resource 3 = 200, the economic cost for the schedule from Fig. 1.2 is:
100 * 17 + 150 * 19 + 200 * 18 = 8150

4. Resource utilization - maximizing the utilization of the resources, this metric is very important for the provider whose profit raises with the reduction of time gaps when the machine is not utilized:

$$Resource\ Utilization = \frac{\sum_{i=1}^{m} TR_i}{makespan * m} \tag{1.4}$$

where

TR_i – the time of completion of all the tasks by the resource i
m – number of resources

Resource utilization for the schedule from Fig. 1.2 is:
(17 + 19 + 18) / 19 * 3 = 0.95

5. Deadline constraint - defines the time limit within which the task or batch must be executed.

Apart from that, there are many other criteria, described in detail by Kalra et al. [17]. These are, for instance, tardiness, waiting time, turnaround time, fairness, throughput, priority constraint, dependency constraint, budget constraint etc.

1.3. Research goals and motivation

Task scheduling is indispensable to every distributed system. It allows for the appropriate use of the available resources and is usually a key element in the functioning of cloud computing. In the earlier sections, I briefly discussed the types of scheduling problems, metaheuristics dedicated to solving them, and the methods of their evaluation. These three facets of the issue set out the basic goals and challenges faced by the scheduling modules. In other words, the question remains how to assure the clients that their tasks are executed with due diligence and according to their expectations, sometimes within a specified time limit. By formulating the requirements and directing them to the cloud, customers demand high-quality results and safety for the smallest possible fee. Cloud providers are, therefore, compelled to improve the performance of task scheduling systems and the quality of results. At the same time, they must take into account the specific requirements of their clients, such as execution or storage of tasks in the specific geographic locations or protecting them from the unauthorized publication. However, scheduling is not limited to minimizing costs and maximizing efficiency of schedulers. It also concerns such aspects as security, minimizing energy consumption or deadlines for completing tasks. All these challenges clearly show the importance of task scheduling in modern computational clouds and growing demand for innovative solutions and improvements.

Taking into consideration all the problems raised above, one can state that there is much space for improvement in CC. Moreover, the emerging issues may be solved by virtue of the increasingly popular blockchain technology. Blockchain (BC) is a decentralized computer network, devoid of a central management unit, which is used to store and send information about the transactions concluded on the internet [20]. The essence of the blockchain operation is to maintain a joint and collective ledger of records in a digital form, distributed over the network, in the same copies. This technology can be used in various transactions, e.g, in trade, in the electricity market or as a virtual currency. Its main advantage is the possibility of confirming transactions without the agency or involvement of public trust institutions. Since the transactions stored in blockchain are irreversible, at the currently available technology and computing power of units they are very resistant to nefarious modifications. It is estimated that breaking a blockchain network would require computing power equal to half the internet. As such, BC can play a huge role in economics and contribute to the development of the economy. However, its potential can also be used more directly: to enhance the selection of the best schedules for the particular clients and to increase security in the systems they use.

1.4. Research hypothesis and contributions

The following research hypothesis is formulated in this thesis:

"Blockchain-based cloud schedulers are efficient security-aware methods of planning the mapping the tasks into the cloud resources with respect to the end-users requirements."

Modern scheduling systems in CC focus on the reduction of the time of scheduling and, consequently, clients' costs. Nowadays, customers have also requirements related to the security issues, performance of their tasks, and the quality of results they receive. Sometimes they want their tasks to be executed with a machine located in a specific place. It is, of course, challenging for service providers to meet such requirements. In order to verify the defined research hypothesis, in this thesis, a new model of security-aware cloud scheduler that allows to fulfil the cloud end-users requirements and be beneficial for the cloud service and resource providers was defined. The model is based on blockchain technology [20] and Stackelberg games [21] described in more detail in Chapters 4 and 5.

The original research contributions of the thesis include:

– definition of a new taxonomy of the cloud scheduling (Section 2.1),

– definition of a new 'security level' scheduling criterion (Section 3.3),

– development of a new original blockchain-based scheduler (Section 6.1),

– definition of a new algorithm *Proof of Schedule*, based on Stackelberg game, dedicated to establishing a consensus in the blockchain-based cloud network used for execution of the schedules (Section 6.2),

– development and implementation of a new Blockchain Secure Cloud Scheduler Simulator - a tool for conducting experiments and comparing the proposed solution with other available scheduling modules (Section 6.3).

The research conducted in this dissertation significantly contributes to the development of the area related to the topic of task scheduling in distributed computational clouds, in particular to all aspects related to the security of schedules.

1.5. The structure of the dissertation

The dissertation is organized as follows. Chapter 1 contains a description of the importance of task scheduling in cloud computing and the related problems to solve, being the direct motivation for picking up the topic of the thesis.

Chapter 2 provides an overview of the scholarly literature considering the problems related to task scheduling in computational clouds. The particular approaches are compared with each other, taking into account the aspects that are most crucial during the evaluation of schedules. Subsequently, the topic of the dissertation is placed in the considered research area.

Chapter 3 provides the characteristics of virtual machines and tasks and specifies how to obtain them (Section 3.2). Section 3.3 defines a new security-related criterion for the schedule evaluation called *'security level'*. The following sections directly concern the central problem considered in the dissertation and comprise the criteria relevant to the evaluation of the proposed solution, together with the adopted restrictions.

Chapter 4 comprises a general description of blockchain technology, including such aspects as: the types of BC network, its architecture and the resulting advantages, models of consensus achievement in the networks, and the security that BC technology guarantees.

Chapter 5 is an introduction to Stackelberg games. First of all, the key concepts related to game theory (such as player, strategy, payoff, and equilibrium) are described and the types of games briefly characterized. Then, the Stackelberg game and its special variation used in the dissertation to solve the particular problem are defined.

The proposed solution in the form of the *Secure Blockchain Scheduler* is discussed in Chapter 6. The presentation of the system model based on blockchain technology is followed by the method for determining consensus on the network, called *'Proof of Schedule'* (Section 6.2). The individual parts of the implemented simulator used to conduct the experiments are described at the end of the chapter (Section 6.3).

The results of the experiments are presented and discussed in Chapter 7. The initial section (Section 7.1) contains the tests of Stackelberg game mechanism conducted on various input datasets. Then, taking into consideration the adopted assumptions, each stage in the implemented *Blockchain Secure Cloud Scheduler Simulator* is checked, which includes the creation of the genesis block, the creation and confirmation of transactions, and adding transactions to the block and block to the blockchain (Section 7.2). Finally, in Section 7.3, the results from the proposed *Secure Blockchain Scheduler* are compared with the results obtained from 4 other known schedulers.

The thesis is summarized in Chapter 8 along with a short discussion on the perspectives of further research in the domain.

Chapter 2

State of the Art in Cloud Scheduling

In this chapter, the state-of-the-art in cloud scheduling was surveyed. A new taxonomy of the cloud scheduling was defined as the result of the provided comprehensive comparison analysis of the selected schedulers. That taxonomy is extended in the further sections by the definition of the novel blockchain-based scheduler, which efficiency is justified in the experimental analysis in Chapter 7.

2.1. Taxonomy of the cloud schedulers

In recent years, many methods for task scheduling have been proposed. There is a lot of scholarly papers relevant to task scheduling in CC, there are also different taxonomy classification according to which cloud schedulers are classified. Rodriguez and Buyya in [22] identify taxonomy, according to which the scheduling model is studied based on the four features, namely: (i) task-VM mapping dynamicity, (ii) resource provisioning strategy, (iii) scheduling objectives and (iv) optimization strategy. To conduct the analysis of state of the art, new taxonomy for cloud schedulers based on two criteria optimality and security have been proposed. These criteria have been chosen because they are important to end-users and schedule providers, but they do not always have to be met. According to the proposed taxonomy, cloud schedulers can be divided into:

– most popular schedulers, with particular emphasis on the optimization of the execution time of schedule,

– schedulers based on blockchain technology, with particular emphasis on the security-related aspects of schedule.

Each article selected for discussion was evaluated according to the following criteria:

– measures - measures used to evaluate the efficiency or performance of the proposed algorithms,

– security - safety related to preparing schedule and task processing,

– specific requirements of clients - taking into account the individual conditions of end users, such as a deadline or low price,

– purpose - the purpose or reason for applying a given approach, for instance, to optimize costs for the client.

The first step was to analyze research in which the main focus was on optimizing the time of schedule, then articles where attention was also paid to security issues, at the end approaches where blockchain technology was used.

Pandey et al. in [23] discussed the topic of scheduling in applications (software) delivered as services. These services are usually provided as a subscription in the pay-as-you-go model in which the client must pay for the usage time. Thus, keeping costs low depends on some objective factors: minimization of the cost of execution, load balancing of the utilized resources, or minimization of execution time. The study included a solution to minimize the total cost of application execution in the resources provided by Amazon and GoGrid using PSO method. In addition to the optimization of the execution time as such, the authors also took into account the costs resulting from the transfer of data between the resources. In the experimental part, they showed that PSO-based algorithm could achieve three times cost savings as compared to existing *'Best Resource Selection'* algorithm based on minimum execution time, selecting the resource with the maximum cost. The evaluation of the proposed approach is as follows:

- measures - cost of completing the application;
- security - not considered;
- specific clients requirements - not considered;
- purpose/reason - optimization of costs incurred by the client.

Zhang et al. [24] raised the problem of the increasing number of cloud computing users and their servicing by single providers. They referred to the idea of inter-cloud [25] and the fact that single providers have limited resources: if they are completely used, it is necessary to borrow the services/resources from other cloud providers. Zhang et al. focused on task scheduling across the clouds, see Fig. 2.1. They proposed the expansion of the genetic algorithm for scheduling independent tasks by adapting the Gene Space Balance Strategy, which optimizes the generation of the initial population. They took into account the influence of distance on the completion time of the tasks and the cost of their execution, and adjusted the scheduling target according to different QoS requirement of clients. In the simulation part, they presented a comparison between the results of inter-cloud scheduling and single cloud scheduling, for the various number of tasks. It turned out that the result of planning for inter clouds was neither the best nor the worst. The evaluation of the proposed approach is as follows:

- measures - cost and completion time;
- security - not considered;
- specific clients requirements - QoS;
- purpose/reason - limited resources of a single provider.

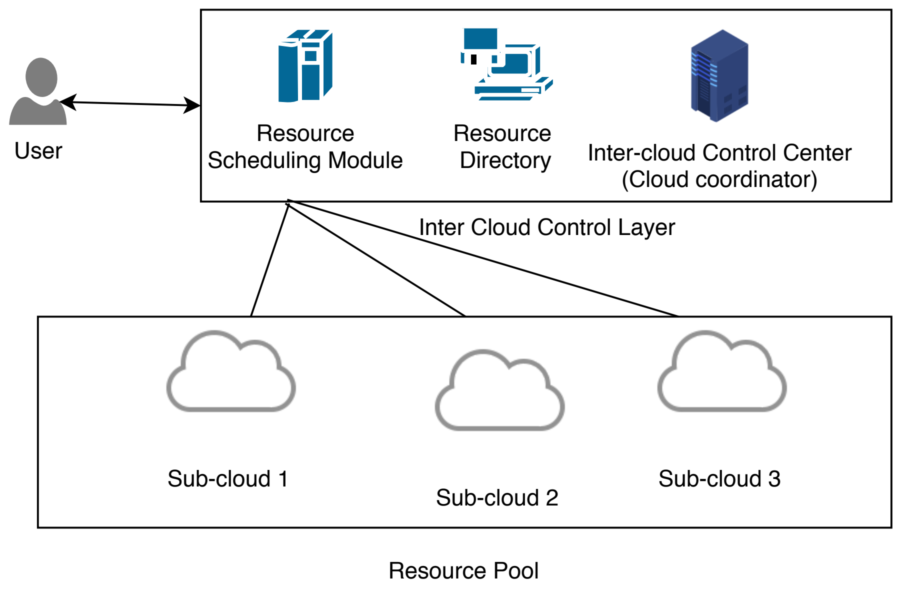
User
Resource Scheduling Module
Resource Directory
Inter-cloud Control Center (Cloud coordinator)
Inter Cloud Control Layer
Sub-cloud 1
Sub-cloud 2
Sub-cloud 3
Resource Pool

Fig. 2.1. Inter-cloud task scheduling architecture proposed by Zhang et al. [24]

Jalaparti et al. [26] have studied a model taking into account the interaction between cloud clients and efficiency when using the cloud. They revealed that despite the isolation of virtualization provided by virtualization techniques, complex interactions could occur between clients sharing the same cloud resources, and therefore the client's job could take more or less time dependent on how much the cloud was loaded. Subsequently, they proposed a model, based on existing theoretical game models, that records various aspects of cloud computing such as prices, resource requests from customers, interactions between clients' tasks and interaction between clients and cloud providers. As a solution, they defined a new class of games called Cloud Resource Allocation Games (CRAG). In CRAG, the problem of resource allocation in clouds is tantamount to a classic non-cooperative game, where cloud clients (players) selfishly try to maximize their utility. In the experimental part, some simulations were made to investigate how various Nash and Stackelberg equilibria [21] work in practice. Additionally, a comparison with a solution not based on the theory of games (Round Robin) was provided. The results showed that the proposed methods were effective. Compared to the values obtained for the Nash and Stackelberg equilibria, the turnaround time in the case of the Round-Robin approach was between 15% and 500% worse . A similar solution related to the optimization of resources using

Stackelberg games was proposed by Jakóbik and me in Jakóbik et al. [27] where some security issues were also considered. The evaluation of the proposed approach is as follows:

- measures - total cost to the system, the maximum cost of a machine and maximum turnaround time;
- security - not considered;
- specific clients requirements - not considered;
- purpose/reason - taking into account the interaction between users in the cloud and minimization of costs.

The next approach for task scheduling was described in Garg et al.[28], where a novel mechanism for task scheduling in grids was presented. The authors applied the principles of the auction to properly allocate resources to parallel applications, taking into account also the specific requirements of end users. Apart from satisfying QoS requirements of the users, their main goal was to ensure the maximum use of resources and to minimize the impact on waiting time and slowdown. The meta-heuristics they proposed, called Double Auction-inspired Meta-scheduling, is a sequence of three stages: collection, valuation and matching. The first stage is responsible for gathering information about the resources and applications, for instance, QoS requirements. Second of all, the valuation is calculated for all applications. Finally, the application is adjusted to the resources based on the previously prepared valuation. In the evaluation part, the authors compared their algorithm with five other well-known solutions. According to the results, the algorithm proved beneficial for both users and resource providers. The mechanism could plan from 8 to 15 percent more user applications than the other ones and had a higher success rate indicating the level of compliance with deadlines. The evaluation of the proposed approach is as follows:

- measures - success ratio, urgency vs. success ratio and number of deadlines missed;
- security - not considered;
- specific clients requirements - QoS and deadline;
- purpose/reason - maximum utilization of resources and meeting deadlines.

Another approach based on auctions was presented by Borjigin et al. [29]. Their study addresses the topic of Network Function Virtualization paradigm (NFV) and introduces some new ideas related to the planning and management of network resources. NFV makes use of the virtualized network functions (VNFs), i.e. virtualized tasks separated from the network hardware arranged by network service providers. The development of NFV faces various technical problems in the service of VNF.

Firstly, hardware and software can be operated by different service providers, which increases latency or unstable throughput. Secondly, it is difficult to coordinate providers managing virtual resources in such a way as to ensure adequate service performance. Thus, the authors presented a double auction-based resource scheduling method that allows the appropriate utilization of resources and ensures high performance of services on the NFV market. As it is shown in Fig. 2.2, their model includes the participation of both suppliers and customers in auctions. Borjigin et al. proposed three algorithms to optimize profits in data center networks by providing a bidding price and asking price. The first algorithm presents the details of the double-auction process. The second algorithm is associated with the process of price adjustment, which ensures the profits of both the buyer and the seller. The third algorithm is able to calculate such a price to attract customers. In the simulation part, these algorithms were compared with a single-auction model. The results showed that they increased the profits of customers and resource providers.

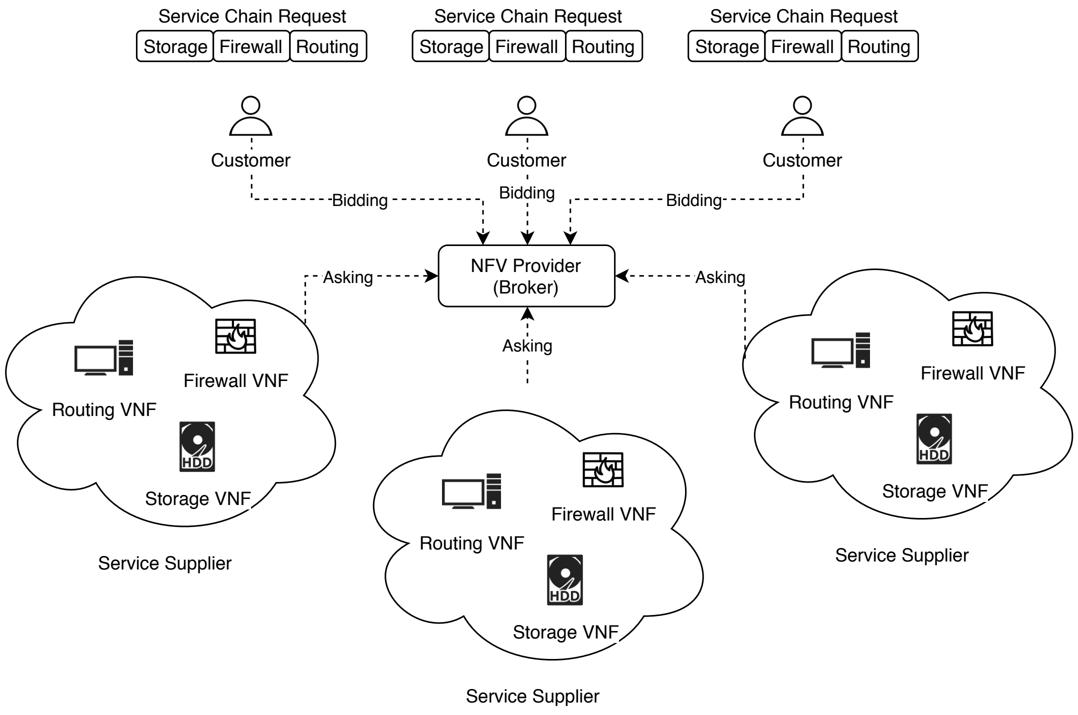

Fig. 2.2. Double auction between service suppliers and customers in an NFV market by Borjigin et al. [29]

This approach is interesting because it introduces an element of competitiveness. However, it is used to allocate specific resources to users and not to task scheduling on those resources. The evaluation of the proposed approach is as follows:

- measures - profits for customers and suppliers;
- security - not considered;

- specific clients requirements - not considered;
- purpose/reason - minimization of customer costs and maximal utilization of resources.

Kołodziej and Xhafa [30] proposed a scheduling model simultaneously allowing aggregation of task abortion and ensuring security requirements, which are the criteria for the cumulative objective function along with makespan and flowtime. They defined a meta-broker being responsible for checking the security conditions and availability of resources. The level of security in their approach is determined on the basis of trust level (tl) parameters defined for the resources and security demand (sd) defined for the tasks. These parameters depend mainly on the specific requirements of the user, security policy, history of attacks or the ability to self-defence. They are described in more detail in [31]:

– security demand - related to tasks, specified for each task in the job, refers to data integration, task sensitivity, peer authentication, access control and task execution environment, is defined as a vector:

$$SD = [sd_j, sd_{j+1}, \dots, sd_n] \tag{2.1}$$

where

sd_j	– one of security demand parameters, assumes a value within the range [0,1], where 0 represents the lowest and 1 the highest security requirements for execution task j
n	– number of tasks in the job

– trust level - related to resources, specified for all resources in the system, this metric determines the level of client trust to the resource manager, refers to prior task execution success rate, cumulative grid cluster utilization, firewall capabilities, intrusion detection capabilities, intrusion response capabilities, is defined as a vector:

$$TL = [tl_i, tl_{i+1}, \dots, tl_m] \tag{2.2}$$

where

tl_i	– one of trust level parameters assumes a value within the range [0,1], where 0 represents the riskiest and 1 fully trusted machine i
m	– number of resources

On the basis of the sd and tl, it is possible to assess whether the condition of ensuring security is met and, consequently, whether the task can be successfully executed on a given machine. It means that

$sd_j \leq tl_i$ for a given (j, i) task-machine pair. In the experimental section, the authors compared the results of scheduling carried out in 2 different modes: a secure mode where all the security conditions and resource uncertainty are verified for the task-machine pairs and a risky mode where all risky and failing conditions are ignored. The measurement of the makespan showed that, in comparison to the classic approach, some scheduling algorithms performed better in risky mode when put in Grid environments having medium or large size. On the other hand, the secure mode brought the best results in all grid instances. The referred article addresses security issues but its scope is fairly narrow and theoretical. It does not discuss such issues as checking the inviolability of tasks and results, unauthorized modification or correctness of the prepared schedule. The evaluation of the proposed approach is as follows:

- measures - makespan and flowtime;
- security - trust level and security demand;
- specific clients requirements - not considered;
- purpose/reason - dynamics of the Grid systems and taking into account security requirements.

Li et al. [32] proposed security and cost-aware scheduling (SCAS) algorithm for different types of tasks in computational clouds, intended to minimize the total cost of workflow execution while meeting the assumed deadline and risk rate limits. Their approach was based on the application of meta-heuristic PSO to create a workflow schedule with tasks mapped to the resources and to the type and number of virtual machines that should be used. To protect the tasks against snooping, alteration and spoofing attacks, the authors used three security services: authentication service, integrity service and confidentiality service. Each task can require all three types of security measures, with the security levels depending on the user's specification. In the experimental section, four different algorithms were tested against three workflows. Then, the impact of security services and risk coefficient were examined. The results confirmed the effectiveness and practicality of the used algorithm. The evaluation of the proposed approach is as follows:

- measures - execution time and cost;
- security - authentication service; integrity service and confidentiality service;
- specific clients requirements - deadline;
- purpose/reason - minimize the total workflow execution cost.

Another scheduler is defined by Jakóbik et al. in [33]. The authors present an innovative architectural model based on a multi-agent scheme and security-aware meta scheduler controlled by genetic

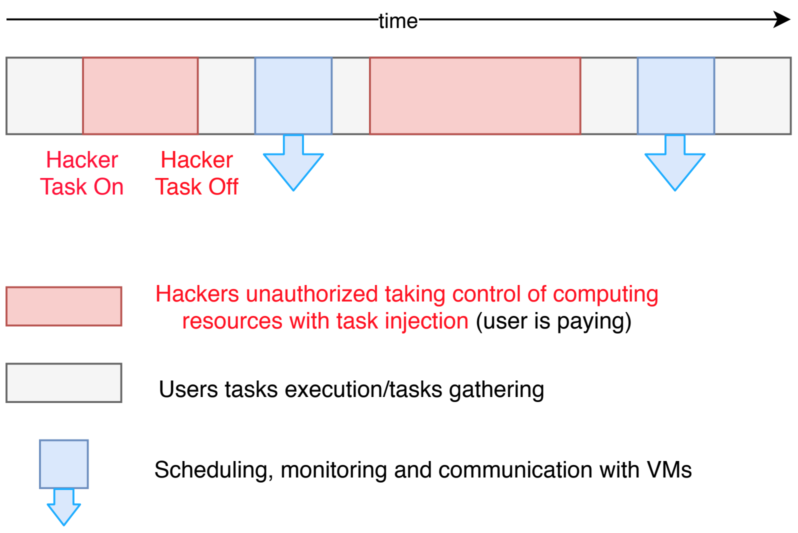
time
Hacker Task On
Hacker Task Off
Hackers unauthorized taking control of computing resources with task injection (user is paying)
Users tasks execution/tasks gathering
Scheduling, monitoring and communication with VMs

Fig. 2.3. Task injection attack proposed by Jakóbik et al. [33]

heuristics. The authors focus on the safety of task scheduling in cloud computing and described its behaviour in the event of a task injection attack. Namely, they considered a situation in which an attacker, logged in as an authorized consumer, tries to send an unauthorized task (see Fig. 2.3). This, in turn, triggers a response from the system in the form of an alert sent to the correct place: its verification takes place before task scheduling. In addition, the authors proposed two models supporting users security requirements, a scoring model that allows task scheduling only on virtual machines that have an appropriate level of security, and a model that takes into account the time needed for cryptographic operations associated with each specific task. These models are similar to those described by Kołodziej and Xhafa [30]. In the experimental part, the influence of non-deterministic time intervals for the scheduling process on the environment performance was examined, and the makespan for different security levels was calculated. The results showed the effectiveness of the proposed models and their increasingly positive impact on the system's safety. The evaluation of the proposed approach is as follows:

- measures - makespan;
- security - sd and tl;
- specific clients requirements - not considered;
- purpose/reason - prevention of task injection attacks.

On the other hand, one should mention a relatively recent study by Lokhandwala [34] which is particularly related to the topic of this dissertation because its author resorted to blockchain technology to solve the problem of task scheduling, In the Lokhandwala's approach, a decentralized blockchain

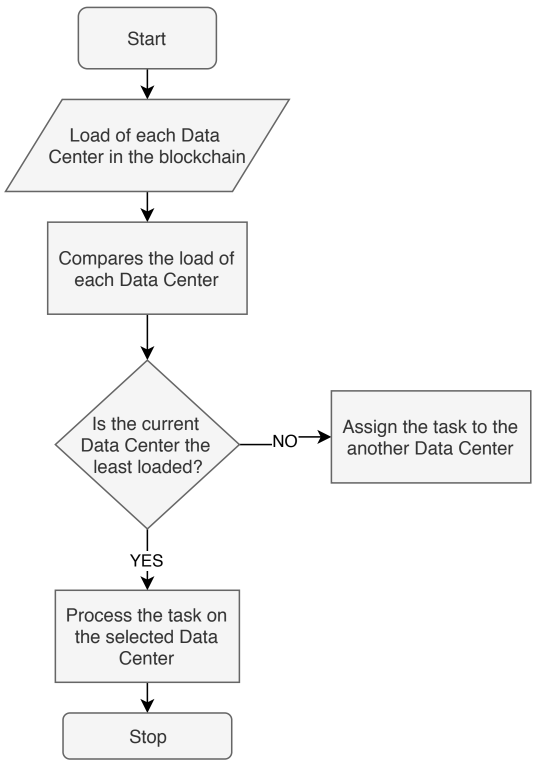
Start
Load of each Data Center in the blockchain
Compares the load of each Data Center
Is the current Data Center the least loaded?
NO
Assign the task to the another Data Center
YES
Process the task on the selected Data Center
Stop

Fig. 2.4. Smart contract algorithm proposed by Lokhandwala [34]

network was used to allocate resources more efficiently, which resulted in the reduction of the consumed energy and, consequently, costs. A load of data centres stored in blocks is checked using smart contracts [35]. Then, the tasks to be executed are assigned to the data centres with the least load. The algorithm on which the smart contract was based is shown in Fig. 2.4. In the experimental part, the correctness of the blockchain network was first checked and, subsequently, the solution was evaluated. To conduct experiments *Shortest Job First* (SJF) algorithm was applied whose main purpose was to minimize the waiting time of virtual machine (VM) response. However, the author did not measure the actual impact of the method on the waiting time of VM response, which would require its comparison with one of the classical methods not based on blockchain. The focus was more on testing the functioning of the blockchain network as such, which included assigning the tasks to the appropriate data centres and the security issues, i.e. blockchain resistance to manipulate the data. Lokhandwala concluded that the use of blockchain was more suitable for data storage than calculating the load of Data Centres. It was because the block mining process turned out to be very energy-consuming due to the chosen consensus algorithm (which probably should have been different for the case). The evaluation of the proposed approach is as follows:

- measures - waiting time;
- security - no possibility to manipulate data;

- specific clients requirements - not considered;
- purpose/reason - allocate the resources more efficiently, consuming less energy.

Hong et al. [36] discussed the problem of communication and task scheduling among users in device-to-device network (D2D) [37] so as to effectively reduce the average time of task execution. Their idea consisted in the use of wasted computing power of mobile devices, which are typically in the idle state with nothing but notification listeners and other low energy consumption applications activated.

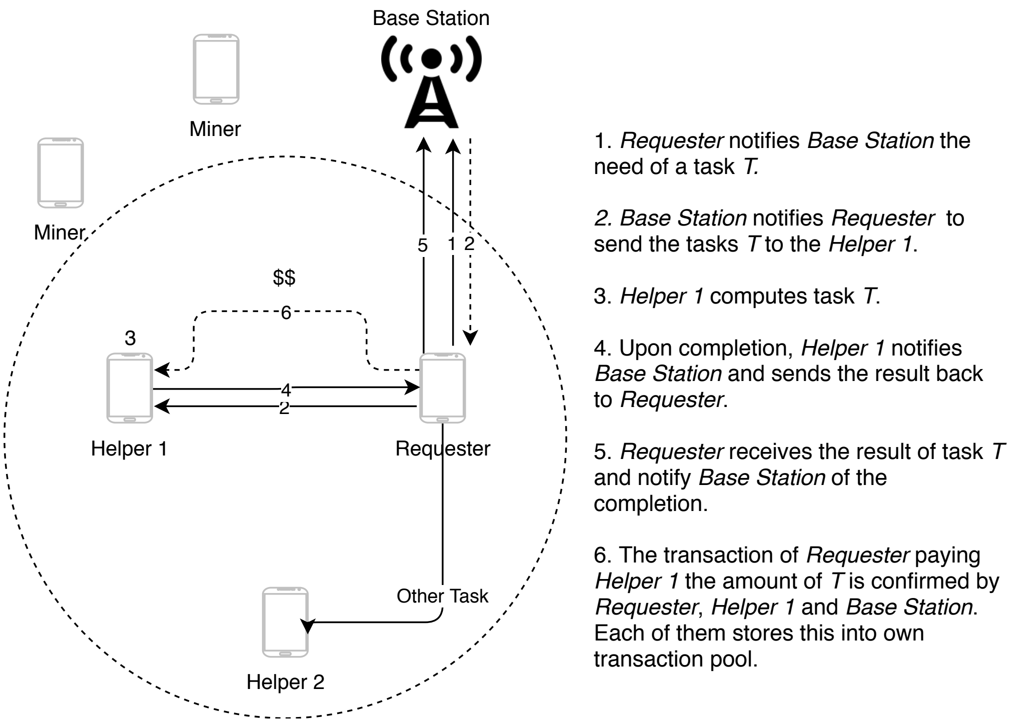
Base Station
Miner
Miner
$$
6
3
4
2
5
1
2
Helper 1
Requester
Other Task
Helper 2
1. *Requester* notifies *Base Station* the need of a task *T*.
2. Base Station notifies *Requester* to send the tasks *T* to the *Helper 1*.
3. *Helper 1* computes task *T*.
4. Upon completion, *Helper 1* notifies *Base Station* and sends the result back to *Requester*.
5. *Requester* receives the result of task *T* and notify *Base Station* of the completion.
6. The transaction of *Requester* paying *Helper 1* the amount of *T* is confirmed by *Requester*, *Helper 1* and *Base Station*. Each of them stores this into own transaction pool.

Fig. 2.5. Task scheduling in D2D network proposed by Hong et al. [36]

The possibility of using these dormant resources together with the storage can, indeed, lead to highly profitable and profitable cooperation in the area of executing tasks in D2D networks. There are, however, some doubts if task scheduling in such systems is fair to everyone. It may look unfair if the users contributing a lot of their computational resources to others receive little being in dire need. Hence, Hong et al. proposed an innovative blockchain-based credit system that can be used for task scheduling to enforce justice among D2D network users. Their solution consists of two parts: the cooperative task scheduling to reduce the average task execution time among the users, and a blockchain-based credit system to ensure fairness in the network. The system model and the principle of its operation are presented in Fig. 2.5. The authors checked the impact of various initial credit provided to each user, different maximum waiting times, task sizes, and time elapsed on the performance. According to the results, the proposed model significantly shortens the average task execution time for the requesters in D2D networks. The evaluation of the proposed approach is as follows:

- measures - waiting time;
- security - not considered;
- specific clients requirements - not considered;
- purpose/reason - shorten the time of task execution in D2D networks.

2.2. Comparison and problems that require further improvement

The analysis of the literature conducted in the previous section reveals that there are many solutions for task scheduling in computational clouds. The authors of each study focused on different aspects of the issue. A comparison of their methods is presented in Table 2.1.

When it comes to optimizing makespan, all the presented algorithms are decent, but the adopted approach has one general disadvantage. As only one provider to whom the request by the client is addressed participates in the preparation of a schedule, there is no certainty that the produced schedule is optimal and correct. The client must simply trust that the provided algorithms are the reliable and optimal solution; their confrontation with the ones offered by other providers is impossible. Moreover, providers are typically reluctant to share information concerning task scheduling and its progress: such information is considered as confidential. Introduction of an element of rivalry and forcing many providers to compete with each other in terms of optimization of schedule execution time would no doubt give the users much more certainty that their tasks are scheduled in a manner optimal for their requirements. However, to achieve this, one must model the process where many providers can participate in the scheduling without disclosing their algorithms. The aspect of competitiveness has been addressed by Garg et al. [28] and Borjigin et al. [29]. Although these studies discuss the solutions quite strictly related to the auctions and allocation of resources, they show the benefits that may bring the implementation of competitiveness into the task scheduling process.

Some authors focused their attention on security issues. Kołodziej and Xhafa [30] and Jakóbik et al. [33] put under scrutiny, among others, security demand and trust level parameters determined mainly on the basis of the unique users' requirements, security policy or attack history. Their studies, although very theoretical and concise, do not take into account the possibility of falsification or modification of the schedule. As far as security is concerned, it should be ensured that the data is not manipulated. In other words, the user must be sure that the prepared schedule is correct. Otherwise, most likely after launching the procedure of its execution, it fails, leaving the client empty-handed and with a fresh bill for resources utilization to pay. Some of the approaches take into consideration the specific requirements of users such as the deadline for the specific tasks or QoS, these parameters improve customer satisfaction and their application is justified.

Paper	Measures	Security	Specific Clients Requirements	Purpose/Reason
approaches related to the optimization of the execution time and costs				
[23]	cost of completing the application	not considered	not considered	optimization of costs incurred by the client
[24]	cost, completion time	not considered	QoS	limited resources of a single provider
[26]	the total cost to system, the maximum cost of a machine, maximum turnaround time	not considered	not considered	taking into account the interaction between users in the cloud, minimization of costs
[28]	success ratio, urgency vs. success ratio, number of deadlines missed	not considered	QoS, deadline	maximal utilization of resources, meet deadlines
[29]	profits of customer and suppliers	not considered	not considered	minimization of customer costs, maximal utilization of resources
approaches related to the security issues				
[30]	makespan, flowtime	trust level, security demand	not considered	dynamics of the Grid systems, taking into account security requirements

[32]	execution time, cost	authentication service, integrity service, confidentiality service	deadline	minimize the total workflow execution cost
[33]	makespan	security demand, trust level	not considered	prevention of task injection attacks
blockchain-based approaches				
[34]	waiting time	no possibility to manipulate data	not considered	allocate the resources more efficiently, consuming less energy
[36]	waiting time	not considered	not considered	shorten the time of task execution in D2D networks

Table 2.1. Comparison of selected solutions for task scheduling

Finally, there are two approaches making use of blockchain technology. Lokhandwala [34] applied it to store information on the availability of a given data center and the possibility of sending further tasks to it. The author also emphasized the security this technology provides. Hong et al. [36] used blockchain in a different manner, i.e. in D2D network to synchronize and execute tasks on different mobile devices. Their approach clearly shows the reliability of blockchain technology and the benefits that blockchain network decentralization can bring. The remuneration model presented there can be included in the scheduling of tasks to reward schedule providers.

As can be seen, the problem of task scheduling is neither a new area of research nor a niche issue. However, it still needs many improvements and contributions, especially in terms of optimization and security aspects.

Chapter 3

Problem of Task Scheduling in Computational Clouds

In this chapter, a scheduling problem considered in the dissertation has been formally defined with relation to the attributes posed in Section 1.2. Then, the task and machines used in the proposed model are defined as general terms. Finally, the criteria and assumptions regarding evaluation of the proposed approach are presented.

3.1. Problem formulation

In order to formulate the instance of the problem of task scheduling in computational clouds, the following terms need to be clarified:

– task/job - a single task to execute,

– batch - set of tasks/jobs to execute; there may be dependencies between the tasks in the batch

In this dissertation, the problem of independent batch static scheduling [16] is considered. This means that there are no dependencies between the tasks to be executed; the tasks are processed in batch and the characteristics of the virtual machines used to execute the tasks do not change during the execution. The considered problem can be formulated as follows:

$$\begin{cases} Minimize(makespan/flowtime/economic\ cost/resource\ utilization) \\ Maximize(SL(schedule)) \end{cases} \tag{3.1}$$

where:

$makespan$ – described in Eq. 1.1

$flowtime$ – described in Eq. 1.2

$economic\ cost$	– described in Eq. 1.3
$resource\ utilization$	– described in Eq. 1.4
$/$	– means *'or'*
SL	– described in Section 3.3

The partial goals of the proposed approach are:

- execution of the schedule in the shortest possible time,
- minimization of costs incurred by the client,
- meeting specific user requirements, such as maximum cost, security aspects etc. defined by customers,
- ensuring that the prepared schedule is correct, checked by several different providers of schedule,
- limit the possibilities of falsifying the prepared schedule,
- using multiple scheduling modules at the same time without the necessity to set up secure connections between them; it forces competition between schedule providers who do not need to reveal their scheduling algorithms.

The considered independent batch static scheduling was solved in the following steps:

1. Obtaining jobs to be executed along with the specific requirements of the client.
2. The choice of virtual machines to execute tasks, taking into account the specific requirements (e.g. geographical location of physical servers or security aspects) of the client and define the expected SL.
3. Preparation of the characteristics of tasks and virtual machines ($[wl_1, \ldots, wl_n]$, $[cc_1, \ldots, cc_m]$).
4. Preparation of the schedule for executing tasks from batch on the available virtual machines by many different schedule providers.
5. Getting an optimal schedule, taking into account the expected level of security.
6. Allocation of tasks to virtual machines.

3.2. Tasks and machines

Tasks and machines are the main elements of each task scheduling model. Tasks are defined and sent to the cloud by the end-users, while machines or computing units are resources provided by the cloud resource providers. Both constituents must be appropriately characterized so that to define how many tasks can be performed on a given machine at a given time. In one of the most popular models, tasks are described by computational needs and concern, among others, the execution of the job, service or set of services. On the other hand, the machines are described by their computing abilities, regardless of the characteristics associated with memory or disk space. They can be represented by a virtual resource, a single computer or increasingly popular mobile devices [38]. One of the most popular notations that describes a task and a machine was presented by Kołodziej [31]. In the proposed model, tasks are independent and, together with machines, characterized only by the attributes related to the computing capabilities of a resource. In the above approach, the following assumptions are made:

– n – number of tasks in a batch,

– m – number of machines available to execute a given batch of tasks,

– $N = \{1, \ldots, n\}$ - the set of task numbers from the batch,

– $M = \{1, \ldots, m\}$ - the set of machine numbers.

Considering the above assumptions, tasks and machines can be defined as follows:

– Task j - described by workload expressed in floating point operations (FLO) and marked as wl_j, $[wl_1, \ldots, wl_n]$, it is a workload vector for all tasks in the batch,

– Machine i - described by computing capacity expressed in floating point operations performed in one second (FLOPS) by the machine and marked as cc_i, $[cc_1, \ldots, cc_m]$, it is a computing capacity vector for all machines available to execute a batch.

Task workload can be estimated according to the user's specification, historical statistics or obtained from system predictions [39]. Computing capacity of the machine can be estimated on the basis of the results obtained by benchmarks [40], [41], for instance, using Linpack Benchmark [42].

3.3. Schedule security level

Ensuring an appropriate security level is a very important problem in the process of preparing schedules. There are many security issues that require special attention and continuous improvement, the main ones are:

- difficulties in ensuring privacy and data confidentiality,
- possibility of falsifying the schedule by providers,
- unauthorized modification of results,
- the vulnerability of computing units to failures,
- performing a given task on a machine with specific security parameters (antivirus, firewall, etc.).

A realistic task scheduling model should take into account the above factors and simulate their occurrence. The proposed solution defines security of the schedule as a security level (SL). The SL value consists of three main factors: $P^{failure}$, P^{fake} and $P^{hacking}$ [43].

$P^{failure}$ is the probability of machine failure during tasks execution due to high security restrictions. $P^{failure}_{i,j}$ for specific machine i and task j was defined by Kołodziej [31] as follows:

$$P^{failure}_{i,j} = \begin{cases} 0 & sd_j \leq tl_i \\ 1 - e^{-\alpha(sd_j - tl_i)} & sd_j > tl_i \end{cases} \tag{3.2}$$

where

α – failure coefficient defined as a global parameter

sd_j – described in Eq. 2.1

tl_i – described in Eq. 2.2

Considering the above probability of machine failure during execution of task j on the machine i, $P^{failure}$ for the schedule is defined as follows:

$$P^{failure} = \frac{\sum_{j=1}^{n} P^{failure}_{i,j}}{n} \tag{3.3}$$

where

j – task number

i – machine number

n – number of tasks in the batch

P^{fake} is the probability that the scheduling module will send a false or incorrect schedule, with the assumed value within the range $[0, 1]$, where 0 represents the lowest and 1 represents the highest probability of schedule falsification by the schedule provider.

$P^{hacking}$ is the probability of manipulation, i. e. modification of the prepared schedule by unauthorized entities, with the assumed value within the range $[0, 1]$, where 0 represents the lowest and 1 represents the highest probability to modify results by unauthorized entities.

Taking into account the above three factors, SL has been defined as follows:

$$SL = 3 - P^{failure} - P^{fake} - P^{hacking} \tag{3.4}$$

SL takes values within the range $[0, 3]$, where 0 indicates a very low level of security of the schedule (very risky schedule) and 3 indicates a very high level of security (very secure schedule). Promising technology and methodology to improve the security aspects of scheduling is blockchain technology, which is described in the next chapter.

3.4. Model evaluation

Metrics for model evaluation must address the issues pertaining to the optimization of the schedule, mainly its execution time, which has a large impact on the costs incurred by the client and also security issues. The proposed model in the experimental part was evaluated according to the following criteria:

– makespan,

– flowtime,

– economic cost,

– resource utilization,

– SL.

The above criteria were all described in detail in Sections 1.2.2 and 3.3. Besides, the following assumptions regarding the evaluation of the adopted solution were made:

– communication between different data centres, responsible for task execution, was omitted,

– all data needed to complete the task were either located in computing units or their collection did not affect the execution time,

– only the schedule preparation process was considered, the monitoring of its execution was not included,

– getting fees by cloud service providers for participating in preparing schedules was omitted.

Chapter 4

Introduction to the Blockchain Technology

In this chapter, the blockchain essentials were presented. The terminology for blockchain technology is defined here as:

– **blockchain** - the actual ledger,

– **blockchain technology** - a term describing the technology in its most general form,

– **blockchain network** - network in which a blockchain is being used,

– **blockchain system** - system based on blockchain technology,

– **blockchain node** - an individual system in blockchain network,

– **blockchain user** - a person or entity which is using the blockchain network.

Few existing taxonomies of the blockchain network were described here. Then, the generic model blockchain network with its main components and basic algorithms for achieving consensus was defined.

4.1. Definition of blockchain

The recent digital evolution has contributed to the increase of the volume, velocity and variety of data available on the Internet. In the era of Information and Communication Technology (ICT, one struggles with such problems as collecting and optimally managing such large amounts of data. Research and engineering challenging tasks related to the ICT systems are: ensuring secure network communication of users, data processing and data storage without the nefarious involvement of third parties. The financial aspects are also crucial. One of the possible solution of such problems may be the application of the blockchain BC technology and networks for development of the new models of the networks and ICT systems, especially in the cases where security aspects are important [44]. There are many definitions of the blockchain (the actual ledger). Most of them refer to the origins of the blockchain technology that evolved from Bitcoin created by Satoshi Nakamoto in 2008 [45]. Blockchain is a distributed ledger of records, which contains cryptographically signed transactions that are grouped into blocks. The essentials in the blockchain technology described by Zheng et al. [46], and Tasca and Tessone [47] can be specified as follows:

– decentralization - in all standard transaction systems, there is typically a central unit called the supervisor, confirming the compliance of the transaction and records it in the system. Since such a core unit must handle all the requests and approve them, it is often a bottleneck that determines the efficiency of the entire system. BC lacks that problematic central supervisor because the decentralized nature of the BC system based on consensus algorithm jointly confirms each transaction, maintaining data consistency,

– persistency - since the transactions are validated, any attempt to approve transactions being incompatible with the established policies are immediately detected by confirming/mining nodes; blocks containing incorrect data are immediately detected, too,

– anonymity - each user in the network is assigned a generated address (hash) by means of which they can perform operations. This address does not allow unambiguous identification of a real user,

– auditability - each transaction must refer to some previous transactions; hence there is a possibility to trace and verify what has happened with the processed data. For instance, in the bitcoin network one can check how the balance of a given user has changed since the beginning of its existence in the system,

– transparency - transactions of any public address are available for inspection by every user having access to BC; each user of the public network has the same rights,

– security - chain of blocks are shared, tamper-proof, and cannot be spoofed due to one-way cryptographic hash functions. The security of transactions is ensured by the use of cryptographic methods. Roughly speaking, users can send data only if they have a private key. The private key is applied to generate a signature, which in, turn, serves to confirm that transaction was requested by a specific user and to prevent it from being changed,

– immutability - data stored in BC are immutable; each entry in the ledger must be confirmed by the network, so it cannot 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. Therefore, even a minor change in the data results in the change of the hash and consequent interception and rejection of the modification by the other nodes.

4.2. Blockchain taxonomy

Although blockchain is a relatively new technology, there are few different blockchain taxonomies defined in the literature. Lin and Liao [48] divided blockchain technologies into three types, depending on the character of data availability:

– public blockchain - everyone has access to the transaction and can participate in the process of obtaining a consensus; the examples of such a network are bitcoin or ethereum [49], see Fig. 4.1,

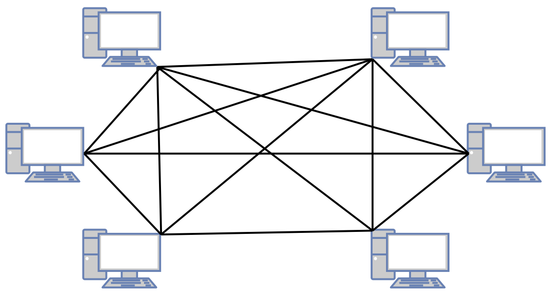

Fig. 4.1. Public blockchain

– private blockchain - not every node can participate in blockchain network and read the ledger; instead, access is limited and strict management of access rights is implemented. A private BC is shown in Fig. 4.2,

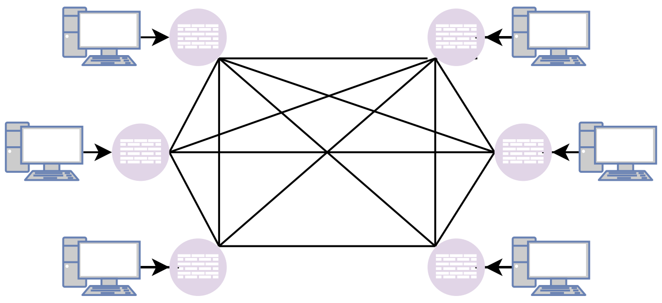

Fig. 4.2. Private blockchain

– consortium blockchain - some pre-selected nodes have direct access to BC and only the nodes from the consortium are allowed to add data; the data to be viewed can be open or private. A consortium BC is shown in Fig. 4.3.

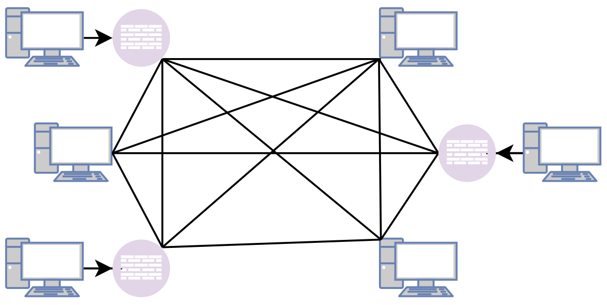

Fig. 4.3. Consortium blockchain

Another taxonomy is defined for the blockchain network. Cohn et al. in [50] defined the following two classes according to the authorization criterion:

– permissioned blockchains - proprietary networks used by specific persons or entities, for instance, a group of cooperating banks that process financial transactions,

– permissionless blockchains - open networks to which anyone can access and use data located there.

When it comes to basic functionality and smart contracts [35], Hileman and Rauches [51] considered two types of blockchain networks:

– stateless blockchain - *'transaction-optimes'*, networks limited to the functionality of the chain in terms of the computational complexity that they can perform (e.g. bitcoin),

– stateful blockchain - *'logic-optimised'*, networks that have expandable functionality in terms of expressing computation (e.g. dApps in ethereum [52]).

Blockchain technology is beneficial and the particular type of blockchain network should be chosen depending on the service that is offered and the market needs.

4.3. Blockchain network

BC network is often defined as the distributed network as it is demonstrated in Fig.4.4. The blockchain network is usually composed of the following main elements:

– nodes - individual systems that store blockchain and ensure transactions are valid,

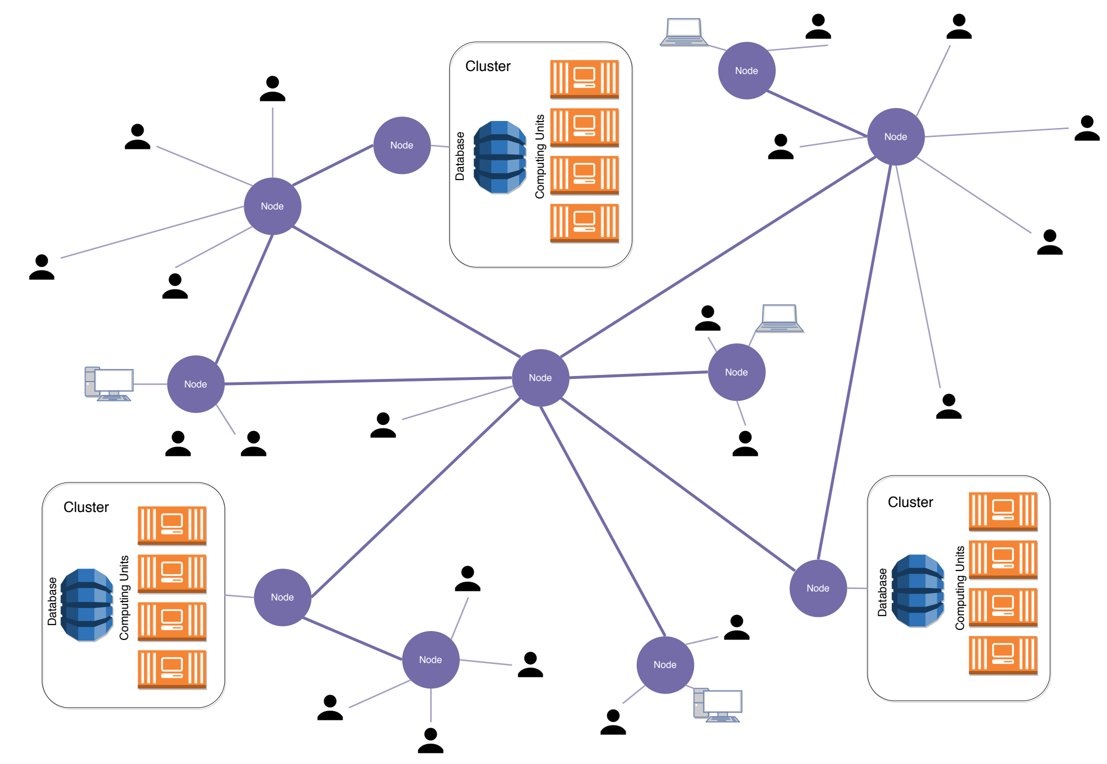

Fig. 4.4. Decentralized blockchain network

– users - persons or entities that can read the ledger.

Blockchain network is peer-to-peer (P2P) [53] network, where each node can communicate with each other without the need of use a central information exchange point. There is no general standard in creating blockchain network that would allow communication between all blockchain networks. For now, each blockchain is created separately, and communications between different blockchain network requires special workarounds. Each BC network works on predefined rules that are agreed by all nodes in the network. These rules include conditions for adding and validating transactions as well as the mechanism of interaction between participating nodes. All common communication rules used by the BC network are called the blockchain protocol.

4.3.1. Blockchain components, protocols and algorithms

Blockchain (the actual ledger) is a chain of blocks, and each block contains a hash digest of the previous block. The block consists of a block header and data block. Each blockchain implementation is different and may alter in some fields. One of the basic data sets is presented by Yaga et al. in [54] where the following fields exist:

1. Block Header:

   – block number,

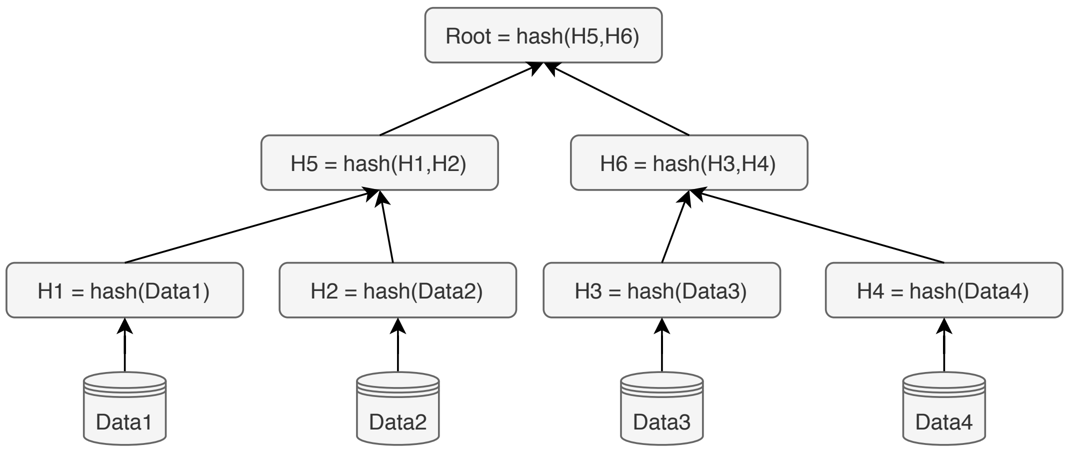

Fig. 4.5. Merkle tree

- hash of a current block - hash generated from the data contained in the block and previous blocks, usually determined using the Merkle tree, see Fig. 4.5; any modification of the data in the block will change the hash,
- hash of a previous block,
- timestamp,
- nonce value - the value usually used in BC based on *Proof of Work* (PoW, see Section 4.3.2. Its finding usually consists in solving the hash function, which allows adding the block to the chain.

2. Block Data:

   (a) a list of transactions - a single transaction usually consists of:

   - inputs - the input data are usually digital assets to be sent; the source of the asset (its origin) is located here, for instance, in the case of cryptocurrencies, it is amount of sent money,
   - outputs - in the outputs usually the recipient of digital assets is defined, along with how many digital assets are receiving and conditions that must be met to spend this value,

   (b) other data.

Each ledger starts with a block called Genesis Block, each block must be added to the chain after it. The genesis block defines the initial state of the system. An example of chain of blocks is shown in Fig. 4.6. The blockchain operation comprises a few simple steps:

1. The sending node prepares new data as a transaction and broadcasts it over the network.

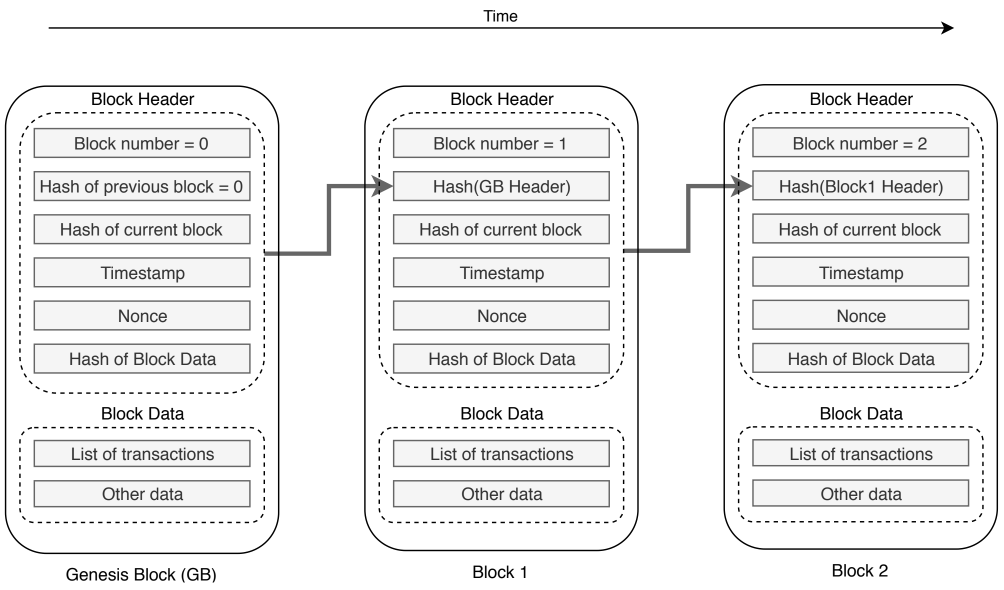

Fig. 4.6. Generic chain of blocks

2. The receiving node verifies the transaction and the data included in it. Each transaction must be signed and authorized using asymmetric cryptography [55]. The private key is used to sign transactions, while the public key is used to identify the user (user address) and verify the signature generated with the private key. With this mechanism, it is possible to check whether the user who sent the message is its author. The procedure of signing and verification of the signature is shown in Fig. 4.7. Whenever the transaction and data pass the validation process, the node responds to the sending node and saves the transaction in the block.

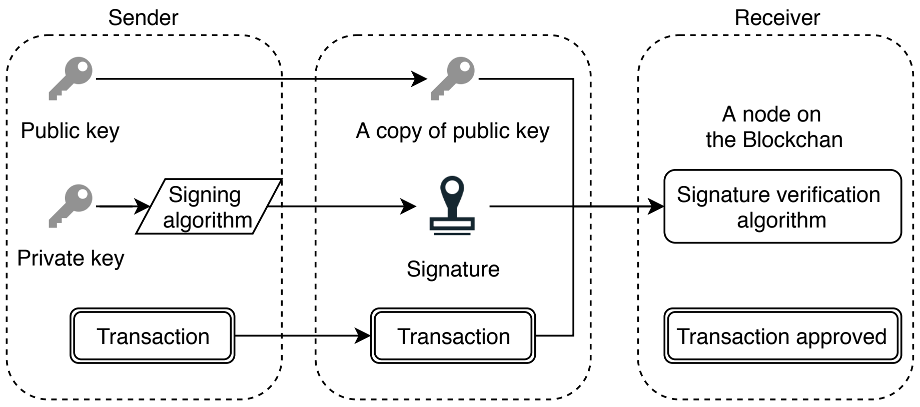

Fig. 4.7. The procedure of signing and verification of the signature

3. After saving the appropriate number of validated transactions in a local block, nodes start the block confirmation procedure in accordance with the consensus adopted in the network.

4. The block is saved in the chain after the execution of consensus algorithm (block confirmation by the appropriate number of nodes).

5. Every node in the BC network must locally save the approved block and include it in its chain.

In every BC network, many blocks may be published at about the same time. Consequently, the existence of different versions of the blockchain in various places is possible. This can happen for a variety of reasons, for instance due to network latency between nodes. To ensure consistency, such a situation must be resolved quickly. This type of problem is shown in Fig. 4.8. As can be seen in the figure, a conflict occurs after adding the block $i+2$. The block added by node A contains transactions 1, 2 and 3 while the block added by node B contains transaction 1, 2 and 4. Therefore, it is not the same block. Such differences do not result from the poor completion of the blocks. In fact, they arise from the almost simultaneous confirmation of different transactions by the nodes. Most BC networks deal with this problem while waiting for the addition of the next block by one of the nodes. It is assumed that the *'longer blockchain'* wins. In other words, the winner is the node that is quicker to add the next block after the block including the above conflict.

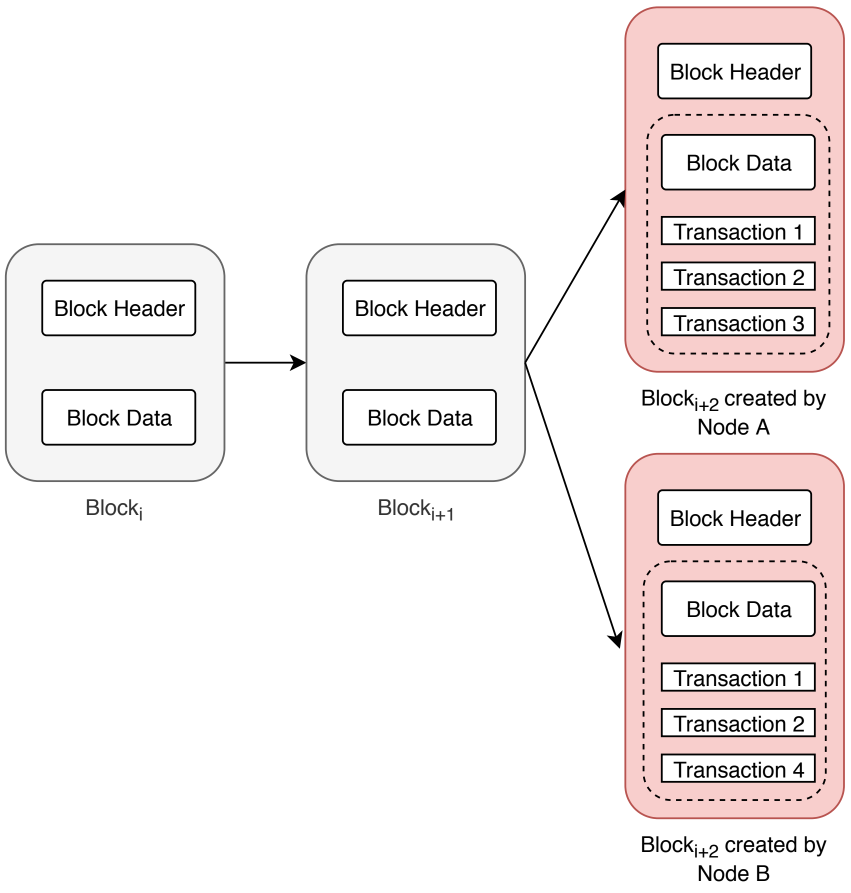

Fig. 4.8. Blockchain in conflict

4.3.2. Consensus models

A key aspect of blockchain technology is to determine which user can publish the next block. This requires the implementation of a consensus model. There are many different techniques for obtaining consensus in blockchain networks. The most important of them are presented below:

– Proof of Work (PoW) - this model is the most popular method of obtaining a consensus in BC network. The node can add a block after solving a computationally intensive puzzle or cryptographic function, which can only be done by brute force [56]. In PoW, the probability of mining a new block by a node depends on the ratio of its computing power intended to solve the puzzle to the total computing power of all miners connected to the network. An example of puzzle was presented below. The node using the hash function SHA-256 [57] must find a hash value meeting the following criteria:

  SHA256("schedule" + "nonce") = hash value starting with "09"

  The nonce value is a numeric value that is added to the string *'schedule'*. After each hash calculation the nonce value is changed. This operation is repeated until the hash value has the form of string beginning with "09".

  SHA256("schedule" + "113") = f8ac5c8094a5ebe334ebe4ba1cde6e29acc718743575933 d6c622406177a6aa4 - means “not solved”

  SHA256("schedule" + "114") = a8007fd4ef5eded7a095d958b3af9e89b48b1bc3a313555 d140f8aa400eb7a6a - means “not solved”

  SHA256(schedule + "115") = 09ce2827da9ebc62bc2491ce96bdf366044247f17860826 2154d6eefb8f40721 - means “solved”

  The above puzzle is not difficult, but its complexity increases with each addition of a subsequent character to string 09. This model is adopted in such networks where suspicion prevails over trust. Although it works well, its obvious disadvantage is the consumption of a very large amount of energy to solve the puzzles. This type of consensus has been used in such networks as Bitcoin or Litecoin (cryptocurrencies).

– Proof of Stake (PoS) - this model combines generating blocks with the possession of a certain amount of digital assets in BC. The selection of a node to perform a function of a validator checking if the next block can be added to the chain is based on the number of assets it includes:

the more assets a node has, the more likely it is to be selected. The strategy is, therefore, based on the assumption that nodes with more assets can provide more reliable information than those having fewer assets. Usually, tokens are used to determine the number of assets. Assuming that a node in the network has a maximum of 100 tokens and the node has 20 tokens, it has a 20% chance to become a validator and mine a block. Such an approach may lead to a problem related to the monopolisation of the network, where the node with large assets accumulates the assets faster than others. Therefore, in some solutions, limitations associated with adding blocks are applied. After mining the block, the node must wait some time before confirming the next one. Other solutions introduce limitations of the lifetime of tokens: they are only valid for a specified time. Contrary to PoW, PoS does not need to consume much energy to solve puzzles and works more economically. However, in order to choose a validator, nodes have to merge into groups, which causes centralization. This approach is used, for instance, in Decred [58] or Peercoin [59]. The algorithm proposed in this dissertation called *Proof of Schedule* is derived from this consensus model.

– Proof of Authority - in this model, nodes are not asked to solve puzzles or mathematical problems. Instead, the network includes hard-configured units called *'authorities'*, which are authorized to add new blocks and secure the blockchain network. This strategy tends to work well in private or consortium blockchains. Authorities receive a set of private keys that have special permissions in the network. The networks based on proof of authority may, nevertheless, have some issues concerning the distribution of mining load between signers and the control of the frequency of mining.

4.3.3. Security aspects in BC networks

Security in BC networks is provided by means of advanced cryptographic techniques, various methods for determining consensus and, above all, the core aspect of this technology, i.e. immutability of data. Joshi et al. in [60] defined several important security principles that should be followed when building and supporting a blockchain network:

– defence in penetration - the use of multiple layers of security is more effective than the application of a single layer,

– minimum privilege - access to data should be limited to the lowest possible level,

– manage vulnerabilities - security vulnerabilities ought to be constantly checked and corrected,

– manage risks - the risk should be regularly assessed and managed at an environmental level,

– manage patches - faulty system components should be corrected, tested and developed as patches for the successive versions of the application.

Blockchain systems are based on many techniques that ensure an adequate level of security. In practice, consensus models are responsible for providing the appropriate level of network safety. For example in PoW to make the network fake, the node would have to possess at least 51% of the computing resources of the entire network to be able to falsify the information contained in the blocks, which is hardly possible in practice [20].

Chapter 5

Introduction to the Stackelberg Games

This chapter introduces the mathematical backgrounds of the game theory, where the model of player, strategy, payoff function and equilibrium state are formally defined. Then, the popular types of symmetric and asymmetric games are presented with a special focus on the Stackelberg asymmetric game, which is used in solving the problems defined in this dissertation.

5.1. Game theory

Game theory is a formal methodology for the analysis of interactions between intelligent players, decision makers, agents or corporations. It is an useful tool for solving problems in economics, business, law, politics and many other areas. Game theory may be also successfully applied for solving the problems, where security aspects are very important [61]. IT systems very often use game theory to determine optimal decisions [21]. In general, game is defined for a set of players. Each player has the set of strategies, which determines the player's actions during the game and are used for calculation of the benefits for the players as the values of the game payoff function. The solution of the game is the equilibrium state, which is often defined as the global optimum of such function. Formally, the n-players game can be defined as follows:

$$\Gamma_n = ((N, \{S_i\}_{i \in N}, \{H_i\}_{i \in N}) \tag{5.1}$$

where:

$$\begin{aligned}
N = \{1, \ldots, n\} &\quad \text{– the set of players} \\
\{S_1, \ldots, S_n\}\ (card S_i \geq 2; i = 1, \ldots, n\ ) &\quad \text{– the set of strategies of the players} \\
\{H_1, \ldots, H_n\}; H_i : S_1 \times \cdots \times S_n \to \mathbb{R}; \forall_{i=1,\ldots,n} &\quad \text{– the set of payoff functions of the players}
\end{aligned}$$

Strategy is defined as a set of planned actions of a given player. The game strategies can be divided into following two main categories [62]:

– pure strategy - it is a deterministic plan of action of the player i during the game. In this dissertation, a set of all pure strategies for the player i is defined by S_i. The profile of pure strategies in the n-players game Γ_n is determined by the following vector of players' strategies:

$$s = \left[s_1, s_2, ..., s_n\right], s_i \in S_i; (i = 1, 2, \ldots, n). \tag{5.2}$$

– mixed strategy – assuming that $S_i = s_{i_1}, s_{i_2}, \ldots, s_{i_m}$ is the finite set of m pure strategies of the player i, ΔS_i is the simplex over S_i and ΔS_i is the set of all probability distributions over S_i, the mixed strategy of player i is defined as the following vector: $\sigma_i \in S_i \subset \Delta S_i$:

$$\sigma_i = \{\sigma_i(s_{i_1}), \sigma_i(s_{i_2}), ..., \sigma_i(s_{i_m})\}, \tag{5.3}$$

where:

$\sigma_i(s_i)$ – the probability that the player i is playing according to the strategy s_i

Payoffs are numerical values of the payoff function defined for all layers. In many game-theoretical models, such function defines the game. Tadelis [62] defined the expected payoff of player i in 2-players game as:

$$H_i(s_i, \sigma_{-i}) := \sum_{s_{-i} \in S-i} \sigma_{-i}(s_{-i}) H_i(s_i, s_{-i}) \tag{5.4}$$

where the player i chooses a pure strategy $s_i \in S_i$, his opponent chooses a mixed strategy $\sigma_{-i} \in \Delta S_{-i}$ and $H_i(s_i, s_{-i})$ is the payoff function calculated for the player i.

In the game Γ , the players strive for maximizing the expected payoff, so they must select the optimal strategy. The optimal result of the game is a situation where none of the players has the motivation to change strategy after considering the choice of the opponent [63]. A set of strategies (one strategy for each player, chosen so that no player can receive higher payoff due to strategy change) is determined by the Nash equilibrium [64] or Nash equilibrium point [65] and can be defined as an n-dimensional vector $(\bar{s}_1, \ldots, \bar{s}_n)$ of strategies if:

$$H_i(\bar{s}_1, \ldots, \bar{s}_n) = \max_{s_i \in S_i} H_i(\bar{s}_1, \ldots, \bar{s}_{i-1}, s_i, \bar{s}_{i+1}, \ldots, \bar{s}_n) \ . \tag{5.5}$$

for all $i = 1, \ldots, n$

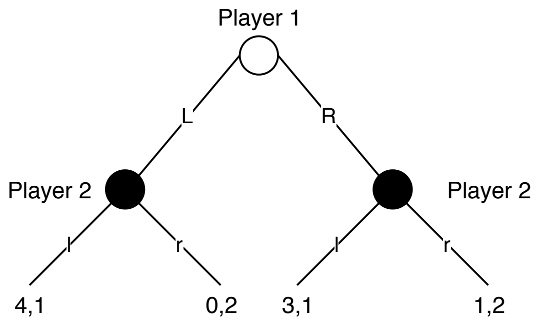

Fig. 5.1. Decision tree in extensive form game [70]

5.2. Types of games

There are various games classifications presented in the research papers.. Bhuiyan [66] define the classes of games according to the following criteria: number of players, rationality of players, cooperation, normal or extensive form, and being zero-sum or non-zero-sum game. Durosimi [67] also distinguished between simultaneous and sequential move games and symmetric and asymmetric games. Based on the published popular categories, games can be divided into the following classes:

1. Normal form and extensive form games [68], [69]:
   - normal form game - the game is described in tabular form. The presented matrices show the strategies adopted by various players and their possible payoffs,
   - extensive form game - the game is described in the form of a decision tree; these types of games help in handling events that may occur accidentally. Sample decision tree where player 1 moves first and player 2 after them is shown in Fig. 5.1.
2. Simultaneous and sequential move games [71]:
   - simultaneous move game - games in which players move simultaneously; alternatively, they do not move at the same time, but in such a situation, the players who make the later move are not aware of the actions of the players who made the earlier move. Normal form games are often used to represent simultaneous games,
   - sequential move game - games in which later-moving players have some knowledge about the actions of earlier-moving players. It does not have to be full knowledge of the earlier players' actions; for instance, it can be restricted to one specific action. An extensive form game is often used to represent sequential games.
3. Cooperative and non-cooperative games [72], [73]:
   - cooperative game - games in which players choose their strategy through negotiation and agreement with other players,

  - non-cooperative game - games in which players strive to maximize their payoffs and adopt their strategy accordingly.

4. Constant sum, zero sum and non-zero sum games [74], [75]:
   - constant sum game - a game in which the sum of the results of all players remains constant, even if the results for individual players are different,
   - zero sum game - this is a special case of a constant sum game in which the profit of one player is always equal to the loss of the other player. An example of a zero sum game is chess, where the profit of one player results in the loss of the other one,
   - non-zero sum game - a game in which the sum of the results of all players is not equal to zero. A non-zero sum game can be transformed into a zero-sum game, an example of zero sum game is shown in Table 5.1.

	A	B
A	2,-2	-1,1
B	0,0	3,-3

Table 5.1. A zero sum game

5. Symmetric and asymmetric games [76]:
   - symmetric game - a game in which the strategies adopted by all players are the same; the decisions in a symmetrical game depend on the strategies used and not on the players' decisions. Even if the players are exchanged, the decisions remain the same,
   - asymmetric game - a game in which the strategies adopted by the players are different and thus the strategy adopted by one player can give different benefits to other players.

5.3. Stackelberg games

Stackelberg game [77] is an example of the asymmetric game in which two roles of players can be distinguished: *leader* and *follower*. The leader is the first to make an action by choosing the best strategy for himself, then the follower makes his action considering what the leader did. Many players can participate in such a game. The leader is the only player who makes a move at the first level. Then, the followers react to his movement trying to minimize the cost function in relation to what the leader did. At the end, the leader updates his strategy to minimize the total cost of the game. The solution of that game is called Stackelberg equilibrium. In this state, the followers observe the strategy s_l

of the leader and responds with strategies $f(s_l) : s_l \to s_f$, which are optimal for the expected payoffs. Solving the Stackelberg game is equivalent to the solution of a maximization problem with constraints. In general, Stackelberg game can be formulated as follows:

$$\begin{cases} \max\limits_{s_l, s_f} H_l(s_l, s_f) \\ s_f \in \arg\max\limits_{s_f} \{\Pi_f(s_l, s_f) : g_f(s_l, s_f) \leq 0\} \\ g_l(s_l, s_f) \leq 0 \\ (s_l, s_f) \in S \end{cases} \tag{5.6}$$

where H_l and H_f denote the payoff functions defined for the leader and the follower, respectively [78]. The strategy chosen by the leader can be described as $s_l = (s_l^1, \ldots, s_l^T) \in S_l$, where s_l^t is a particular decision of the leader at the time t and S_l is a set of possible strategies for the leader. The strategy chosen by the follower can be described as $s_f = (s_f^1, \ldots, s_f^T) \in S_f$, where s_f^t is a particular decision of the follower at the time t and S_f is a set of possible strategies for the follower. The combination of the leader and the follower decisions for any period of time t can be given by $s^t = (d_l^t, d_f^t) \in S^t$, where $S^t = S_l^t \; x \; S_f^t$ is the set of all alternative decisions available to the leader and follower at time t. The set of all possible choices of the leader and the follower throughout the game determines S, where:

$$S = \underset{t=1}{\overset{T}{\mathrm{H}}} S^t \tag{5.7}$$

Usually, there is more than one optimal strategy for the follower and each of them has a different effect on the payoff for the leader. Two types of Stackelberg equilibrium were distinguished:

– Strong Stackelberg Equilibrium (SSE) - followers break the ties in favour of the defender and choose the optimal scenario for the leader,

– Weak Stackelberg Equilibrium (WSE) - followers choose the worst case scenario in terms of benefits for the leader.

In the literature, SSE-related solutions are usually provided, because they exist in all Stackelberg games, unlike WSE. Formally, a pair of strategies $(s_l, f(s_l))$ is defined as Strong Stackelberg Equilibrium if the following conditions are met [79]:

1. The leader plays best-response:

$$H_l(s_l, f(s_l)) \geq H_l(s_l', f(s_l')) \tag{5.8}$$

for all the strategies s_l' of the leader.

2. The follower plays best-response:

$$H_f(s_l, f(s_l)) \geq H_f(s_l, s'_f) \tag{5.9}$$

for all strategies s'_f of the follower.

3. The follower breaks ties in favour of the leader:

$$H_l(s_l, f(s_l)) \geq H_l(s_l, s'_f) \tag{5.10}$$

for all the optimal strategies s'_f of the follower.

According to Ungureanu [80], this type of strategy game should be counted among the Generalized Stackelberg games. The author emphasized the fact that in this type of game moves of the players do not take place simultaneously and players choose their strategies in a known order. The game is defined as a sequential because in each stage one player chooses his strategy. To ensure the selection of the optimal strategy, player solves the optimization problem. This process can be described as follows:

1. The first player chooses his strategy $s^1 \in S_1$ and informs the second player of his choice.
2. The second player chooses his strategy $s^2 \in S_2$ and informs the third player about the choices s^1 and s^2, and so on.
3. At the end, n^{th} player chooses his strategy $s^n \in S_n$ after knowing the choices $s^1 ... s^{n-1}$ of previous players.

When the player $p \in N$ makes a move, players from 1 to $p - 1$ are the leaders or predecessors of player p, on the other hand, players $p + 1, ..., n$ are followers or successors of the player p. Players have full information about the choices of predecessors but have no information about the choices of their successors. Only the p^{th} player $(p < n)$ has full information about the sets of strategies and payoff functions of all players $p, p + 1, ..., n$. The Stackelberg equilibrium requires induction on the reverse sequence of players, through which the player n calculates his best move mapping and players $n - 1, n - 2, ..., 2, 1$ calculate their best moves Stackelberg mappings. The problem of calculating Stackelberg equilibrium in such a game, where we have more than two players is NP-hard [81].

The scenario close to the one presented by Ungureanu is considered for the topic of this dissertation, where Stackelberg game is used to choose the best schedule. This is an issue of linear programming (LP) [82], an optimization problem with constraints that can be solved using the simplex method [83]. This method is very efficient in providing an optimal solution to the puzzles containing thousands of decision variables and restrictions. It uses an iterative algorithm to find the optimal solution, and also provides the information about slack variables (unused resources) and shadow prices (alternative costs) that can be useful in conducting sensitivity analysis [84].

Chapter 6

Secure Blockchain Scheduler

This chapter presents a description of the proposed *Secure Blockchain Scheduler* model based on the BC technology, in which the method of establishing consensus in the network is the key issue. The initial sections define the consensus establishment algorithm as *Proof of Schedule* procedure based on the Stackelberg game. Then, one provides the characteristics of the most important modules and parameters of the developed *Blockchain Secure Cloud Scheduler Simulator* implemented for experimental evaluation of proposed model.

6.1. Blockchain scheduler model

Task scheduling systems available on the market (e.g., Amazon ECS [85]) use many variations of scheduling algorithms (FCFS [86], SJF [87] etc.) that are, nevertheless, not publicly available. Cloud service providers such as Amazon Web Services (AWS) [88] or Azure [89] use their own algorithms to prepare schedules. The clients, wishing to use their services, send the task to them and can only trust that it will be done according to their expectations at the lowest possible cost. The internal algorithms used by service providers are not always optimal in taking account of the specific customer requirements. As part of this dissertation the new *Secure Blockchain Scheduler* (BS) model, which is based on the BC technology was developed. This is a new method of generation of an optimal schedule, where both the specific customer needs and security requirements are taken into account. To make the model available to every scheduler provider who would like to participate in the preparation of the schedule, the public blockchain was selected. This type of approach leads to competition between different providers. The provider who prepares the schedule meeting the client's requirements faster, wins. In this case, the so-called inter-clouds [25] approach is referred which assumes the possibility of cooperation between clouds whenever they need additional services or computing resources. The described model is based on the similar idea according to which the providers can use the services offered by other providers in order to obtain the best schedule. Communication between different clouds providers is determined by a public blockchain, which is decentralized and does not require the use of special protocols for information exchange. Therefore, the units specializing only in one field and not necessarily providing other services can freely participate in the process of generation of schedules. The general concept of the developed model is presented in Fig. 6.1. The main actors in that model are clients (end users) and cloud service providers. Its main elements are the pool of requests, transactions, nodes participating in the transaction approval process and chain of blocks in which transactions with prepared and confirmed schedules are located and stored.

6.1.1. Clients and cloud service providers

In the model, clients formulate their requirements and direct them to the cloud. These can be jobs related to running the executable file or executing a fragment of the source code that solves some mathematical problem. Together with a specification of their needs, they can also provide some specific conditions, for instance:

– executing tasks on a machine with special security parameters (firewall, antivirus, etc.) due to data sensitivity and confidentiality,

– data processing only on servers located in a defined geographical location due to legal reasons,

– quick execution of tasks regardless of costs or the exact opposite.

Cloud service providers collect requests from clients, which are then forwarded to *Task Managers* (TMs). Based on the received data describing the tasks and the specific requirements of the client, TM who has knowledge about the available resources/virtual machines (VMs) in his cloud, chooses the ones that will be the most suitable for their execution. The choice is made taking into account requirements in terms of security, the physical location of machines, the short waiting time for results or a small budget of the client. After that, task manager performs a description of the selected VMs and tasks to be executed.
Each VM is characterized using two factors:

– computing capacity cc (Section 3.2),

– trust level tl (Eq. 2.2).

Each task is characterized using two factors:

– workload wl (Section 3.2),

– security demand sd (Eq. 2.1).

In addition to the specification of machines and tasks, mainly on the basis of security aspects specified by the client, TM also sets the value of the expected SL parameter for the schedule to be

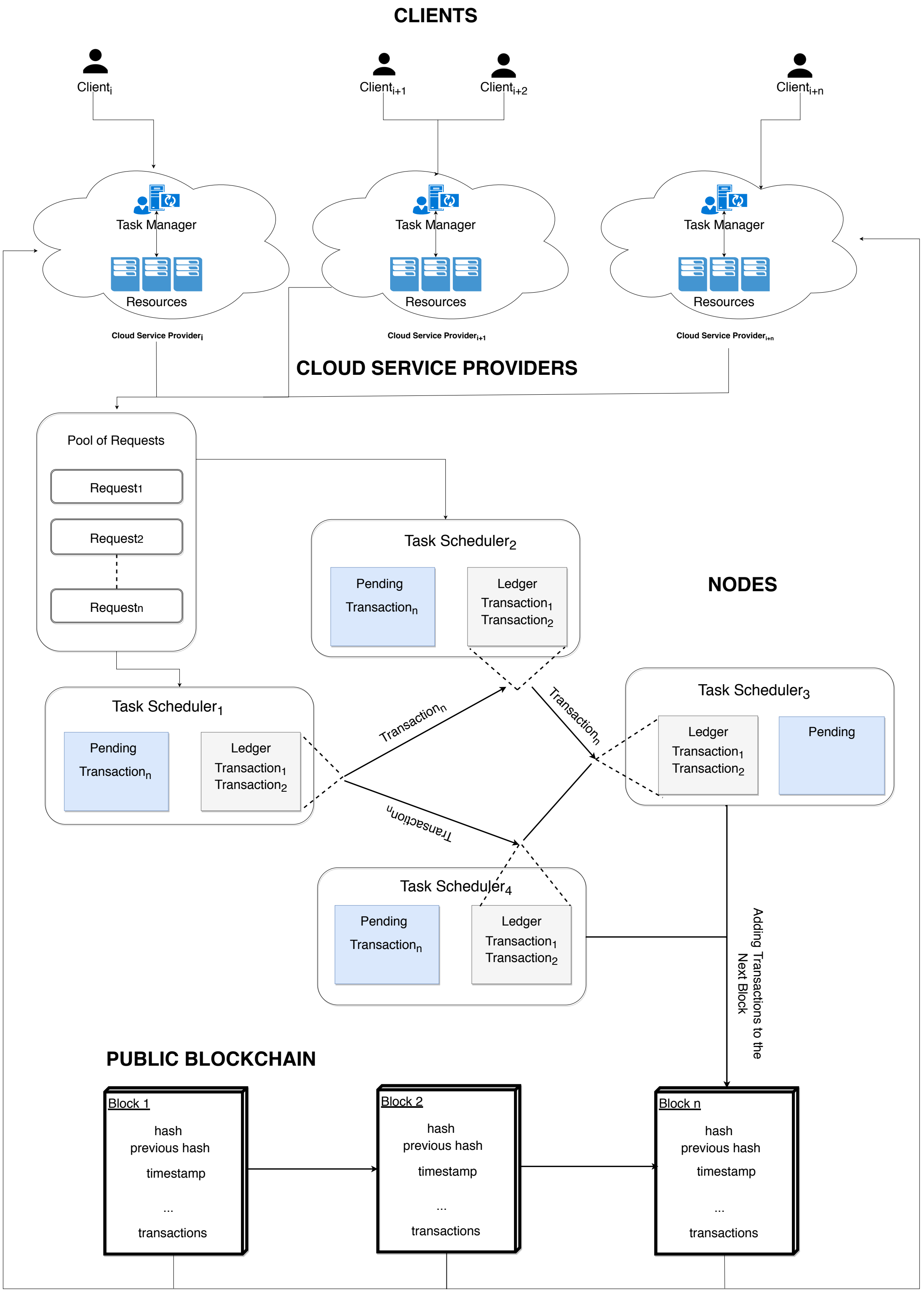

Fig. 6.1. Secure Blockchain Scheduler Model

prepared. If the SL value is equal to 0 then all proposed schedules prepared by the nodes are accepted, otherwise only those whose SL value is greater than or equal to the value given by TM are considered as correct. The SL parameter assumes values in the range [0,3] and the higher its value, the longer the schedule preparation because the majority of prepared schedules are rejected. On the other hand, the higher the SL value, the more secure the schedule is.

6.1.2. Pool of requests

After preparing the characteristics of the tasks and machines and defining the expected SL, all the information is defined as the request, which is then sent and collect in the request pool. At this stage, the request is also signed by the task manager, i.e. the future recipient of the target transaction containing the prepared schedule. A body of such request is shown in the Fig. 6.2. The request pool is generally available, and task managers from different clouds can place requests in it. Nodes located in the blockchain network select those requests for which they would like to generate the schedule.

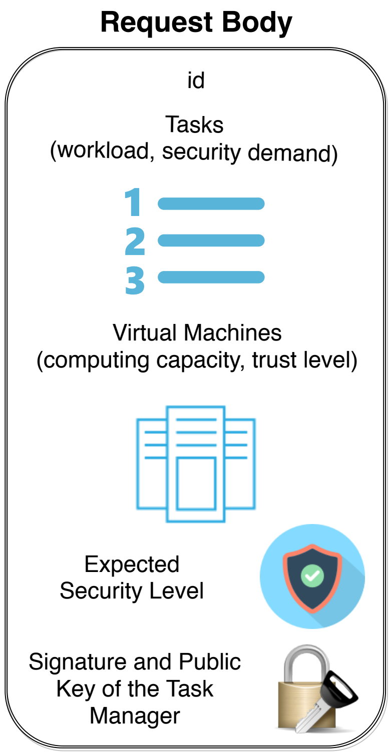

Fig. 6.2. Body of request from the pool of requests

6.1.3. Nodes and transactions

The initiating node retrieves the request from the request pool and prepares the schedule for the tasks and machines described in it according to its own scheduling algorithm. After the preparation,

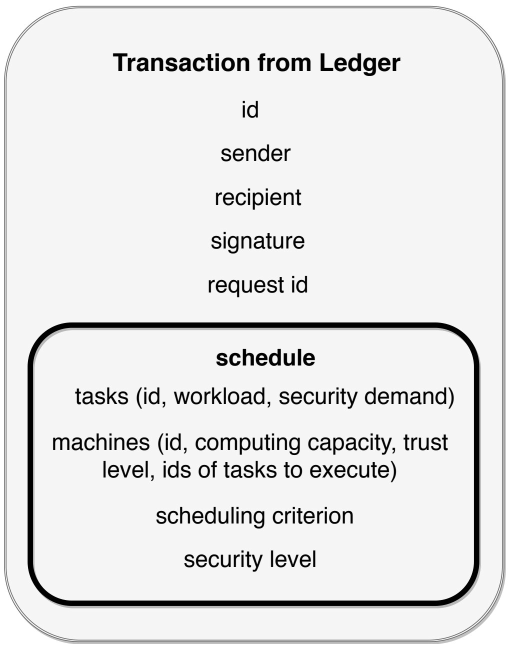
Transaction from Ledger
id
sender
recipient
signature
request id
schedule
tasks (id, workload, security demand)
machines (id, computing capacity, trust level, ids of tasks to execute)
scheduling criterion
security level

Fig. 6.3. Body of the transaction

it calculates one of the scheduling criteria, i.e. makespan or flowtime or economic cost or resource utilization. The obtained results, together with data from request, are placed into the transaction which is then broadcast over the blockchain network for confirmation. The fully prepared transaction is shown in Fig. 6.3 and contains such information as:

– id - transaction id generated on the basis of data contained therein,

– sender (PKN) - the public key of the node preparing the transaction,

– recipient (PKTM) - the public key of the task manager creating the request,

– signature (DS) - digital signature made by the node,

– request id - id of the request sent by the task manager,

– information about the prepared schedule:

  - tasks - id, workload and security demand of the tasks;
  - machines - id, computing capacity, trust level and ids of tasks to execute; each machine includes the assigned ids of tasks that should be executed on it;
  - scheduling criterion (SC) - makespan, flowtime, economic cost or resource utilization.

In the described transaction, SC is the factor that determines whether the schedule is optimal. This factor will be replaced by makespan, flowtime, economic cost or resource utilization (Section 1.2.2).

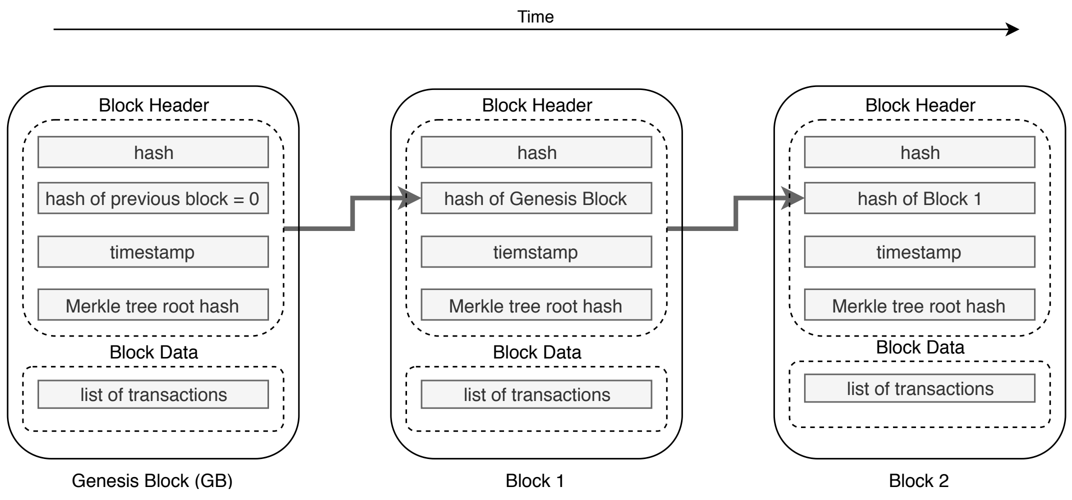

Fig. 6.4. The first few blocks from the chain of blocks of the model

In the experimental part, the model will be evaluated for each of these criteria. After obtaining the appropriate number of transaction confirmations, the node places it in the block. Before adding the transaction to the block, the SL value of the prepared schedule is calculated. If the value of SL is, at least, at the security level specified by TM, the transaction is added to the block. Otherwise, it is omitted because its further processing and saving in the chain of blocks would be pointless.

6.1.4. Chain of blocks

The block is created and validated after collecting a sufficient number of transactions and the number of required transaction confirmations is defined as a global parameter for the entire BC network. Each block consists of:

– the block hash value (Bhv) - calculated on the basis of previous block hash value, a timestamp and the Merkle tree root hash value,

– the previous block hash value (PBhv),

– a timestamp (Tim),

– the Merkle tree root hash value (MTRhv) (see Fig. 4.5), generated using the SHA-256 hash function [57],

– a list of transactions - transactions with prepared schedules, containing the information presented in the Fig. 6.3.

Once all the block building requirements are met, it is mined by mining node and distributed across the BC network. During block propagation, some conflicts may occur due to the fact that several nodes may try to add many blocks at the same time. As a result, different versions of the chain of blocks may be provided (see Fig. 4.8). Such situations are resolved by adopting a rule to wait for the next added block and invariably recognize a longer sequence of blocks as official; in other words, the first node successful in adding the next block prevails. The algorithm for establishing consensus between nodes, verifying the correctness of the schedule and mining new blocks is characterized in Section 6.2. After confirming the chain throughout the BC network and recognizing it as official (a fragment of such a chain is shown in the Fig. 6.4) task managers can load the schedule prepared for them. They obtain it using their public key; each transaction with prepared schedule is addressed to the TM whose PKTM was given in the pool of requests. Having the schedule, TM can allocate tasks to the resources and monitoring them. At this moment, the whole process ends.

6.2. Proof of schedule

The model proposed in the previous section requires an appropriate consensus algorithm dedicated to regulating schedule checking and adding new blocks to the chain. The proposed consensus algorithm has been defined as *Proof of Schedule* (PoSch) [90] and is described in the following sections.

6.2.1. Generalized Stackelberg game

In the approval of the schedule (transaction) nodes (task schedulers) from i to l take part. Node ts_i based on the pool of tasks and their workloads and the pool of virtual machines and their computing capacities prepares the most optimal schedule according to its own algorithm and gives its adopted scheduling criterion, let's assume that in this case it is makespan. After the preparation, schedule is placed in the transaction and disseminates across the network. The nodes that receive the transaction for confirmation also prepare the schedule according to their own algorithms and calculate its makespan. The following notation will be used to present the described procedure:

- $TS = \{ts_i, ts_{i+1}, \ldots, ts_l\}$ - the set of nodes involved in confirming transaction,
- wl_j - characterization of the task j (see Section 3.2),
- $wl(schedule)$ - the sum of vector elements $[wl_1, \ldots, wl_n]$, defined for all tasks from schedule,
- ts_i - node initiating the transaction,
- ts_{i+1} - first node confirming the transaction,
- M_{ts_i} - makespan of the schedule determined by node ts_i,
- $M_{ts_{i+1}}$ - makespan of the schedule determined by node ts_{i+1},

– SF_{ts_i} - scheduling factor of the ts_i node, the sum of $wl(schedule)$ for all schedules in the blockchain added by node ts_i,

– $SF_{ts_{i+1}}$ - scheduling factor of the ts_{i+1} node, the sum of wl_{sch} for all schedules in the blockchain added by node ts_{i+1},

– BW_t - the sum of $wl(schedule)$ for all schedules added to the blockchain within a given period of time t.

Each subsequent node verifies the schedule sent by its predecessor, which does not mean that the predecessor prepared it itself (it can be actually a schedule from the initiating node). First, it prepares the schedule according to its own algorithm and then it calculates its makespan. The aforesaid schedule confirmation/approval can be formally modelled using Generalized Stackelberg Game described in Section 5.3. In such a game, there are many players and the game is defined as sequential with the choice of strategy by one player in each stage. In first stage, the game takes place between nodes $ts_i - leader$ and $ts_{i+1} - follower$; the assumption is that the node ts_i has already made its move. The next move is performed by node ts_{i+1}, which has two options to choose:

- M_{ts_i} - makespan proposed by the $leader\ ts_i$;
- $M_{ts_{i+1}}$ - makespan calculated by its own algorithm.

The $follower$ chooses one of two pure strategies (see Eq. 5.2) defined as:

- s_1 - choosing makespan M_{ts_i};
- s_2 - choosing makespan $M_{ts_{i+1}}$;

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

To determine the utility function for the follower, the scheduling factors, which are treat as confidence coefficients must be scaled, which is carried out as follows:

$$\overline{SF_{ts_i}} = \begin{cases} 1 & \text{if max } \{SF_{ts_{i+1}}, SF_{ts_i}\} = SF_{ts_i} \\ \frac{SF_{ts_i}}{SF_{ts_{i+1}}} & \text{if max } \{SF_{ts_{i+1}}, SF_{ts_i}\} \neq SF_{ts_i} \end{cases}$$
$$\overline{SF_{ts_{i+1}}} = \begin{cases} 1 & \text{if max } \{SF_{ts_i}, SF_{ts_{i+1}}\} = SF_{ts_{i+1}} \\ \frac{SF_{ts_{i+1}}}{SF_{ts_i}} & \text{if max } \{SF_{ts_i}, SF_{ts_{i+1}}\} \neq SF_{ts_{i+1}} \end{cases} \tag{6.1}$$

The utility function for the $follower$ depends on both strategies s_1 and s_2 and scaled confidence coefficients of players:

$$u(s_1, s_2, M_{ts_i}, M_{ts_{i+1}}) = M_{ts_i}\overline{SF_{ts_i}}s_1 + M_{ts_{i+1}}\overline{SF_{ts_{i+1}}}s_2 \tag{6.2}$$

Looking for a strategy is tantamount to solving the following problem:

$$\begin{cases} \underset{s_1,s_2}{\operatorname{argmax}}\, u(s_1, s_2, M_{ts_i}, M_{ts_{i+1}}) \\ s_1 + s_2 = 1 \\ s_1, s_2 \in \{0, 1\} \end{cases} \tag{6.3}$$

The problem 6.3 is a maximization problem with constraints. For the needs of this dissertation, it was solved using the simplex method [91]. Once the solution, i. e. strategies s_1 and s_2 , are found, node ts_{i+1} chooses:

– the option M_{ts_i}, if s_1 = 1 and s_2 = 0,

– the option $M_{ts_{i+1}}$, if s_2 = 1 and s_1 = 0.

If node ts_{i+1} loses the game (it chose the option M_{ts_i}), it sends transaction from node ts_i to the next node ts_{i+2} and notifies the node ts_i about the correctness of its schedule, so the next stage of the sequential game is taking place, where node ts_i remains the leader and the follower is the next node ts_{i+2}. Otherwise, node ts_{i+1} initiates a new transaction according to its own schedule, becoming a $leader$ and sends it for verification to the node ts_{i+2}, which is the first $follower$. The game is carried out until the node receives confirmation from the appropriate number of TS sequence items. The fewer confirmations the system requires, the more transactions it can produce in a given time period. On the other hand, the more confirmations, the more secure and reliable the system is in comparison to the other systems [92]. Bearing this in mind, this value should be selected individually depending on the system implementation. In the proposed approach, the minimum number of transaction confirmation (MTC) is defined as a global parameter during blockchain initialization. SF_{ts_i}, $SF_{ts_{i+1}}$ coefficients must be non-zero, so if nodes participating in the game do not yet have such data, they must be randomly selected. Similarly, the initiating node ts_i, having no predecessor, will choose its time of schedule.

6.2.2. Blocks mining

The block can contain many transactions. It is ready to be confirmed if the sum of $wl(schedule)$ for all schedules within all block transactions exceeds the set value:

$$\text{block is ready to mine if} \quad \textstyle\sum_{i=1}^{p} \; wl(schedule_i) \; \geq B_{wl} \tag{6.4}$$

where

$wl(schedule)$ – described in Section 6.2.1

B_{wl} – minimum block workload defined as a global parameter set during blockchain initialization

p – number of transactions in the block

When the appropriate B_{wl} indicator is obtained, the mining node can add a block to the blockchain. Each transaction in the block must be validated. Validation is intended to check whether the providers of the transaction inputs have cryptographically signed the transaction. The signature verifies if they have the right to transfer of funds for participation in the process of preparing the schedule. Confirmation nodes check the published block, verifying that each transaction it contains has been validated, after which the block can be added to the blockchain. The subsequent blocks are added by nodes called $validators$. $Validators$ are nodes that have so far participated in the transaction creation process and have also expressed a desire to mine a block. One leader is selected from the pool of validators. The election is made on the basis of adding transaction history of a node covering a specified period of time. The election criterion is defined as:

$$\begin{cases} L_t = max\{TF_{(v_i,t)}, TF_{(v_{i+1},t)}, ..., TF_{(v_r,t)}\} \\ \forall i = v_i, v_{i+1}, ..., v_r \forall t : \; TF_{(i,t)} \leq \frac{1}{2} BW_t \end{cases} \tag{6.5}$$

where

L_t – a leader selected to mine a given block

$TF_{(v_i,t)}$ – trust factor of the validator v_i, the sum of $wl(schedule)$ for all schedules added to the blockchain by validator v_i within a given period of time t defined as a global parameter set during blockchain initialization

BW_t – described in Section 6.2.1

r – number of validators

The above approach originates from the idea of *Proof of Stake* (see Section 4.3.2, namely from its specific case referring to the hybrid of coin ageing systems and delegate systems [54]. A node can become a leader only if its current trust factor does not exceed the value of BW_t, which protects BC network against the so-called 51% attack (or majority attack) to which such systems are vulnerable [93].

6.2.3. Profits for task schedulers

Nodes involved in creating or confirming transactions and block mining can be rewarded in a variety of ways. They usually charge a fee for performing specific action. One of the possibilities is presented below. According to it, nodes (task schedulers) can receive profits for participating in operations on which the functioning of BC network is based. The task scheduler profit for creating or confirming one transaction (schedule) $P(schedule)$ within an entire block can be defined as follows:

$$P(schedule) = \frac{wl(schedule)}{CN * 0.8} \tag{6.6}$$

where

$wl(schedule)$	– described in Section 6.2.1
CN	– the number of all nodes confirming the schedule (including the creating node)

However, profit for mining the block $P(block)$ can be defined as follows:

$$P(block) = wl(block) * 0.2 \tag{6.7}$$

where

$wl(block)$	– the sum of $wl(schedule)$ for all transactions in the block

The proposed reward approach is largely dependent on whether the blockchain is public or private. In the case of this dissertation, blockchain is public, but it can be freely changed to private. Therefore, the profits model may also change depending on the given implementation.

6.3. Blockchain Secure Cloud Scheduler Simulator

The requirements of the proposed solution, discussed in the earlier sections, and the use of a dedicated PoSch consensus algorithm had a consequence: the implementation of the proposed system could not be based on the existing solutions. Such solutions supporting the production of systems based on blockchain technology, such as Ethereum [94] at the time of simulator implementation was not sufficiently developed. The simulator has been implemented as a proof of concept using Java language in version 1.8 [95] and was called *Blockchain Secure Cloud Scheduler Simulator* (BCSchedCloudSim). The implementation was created using Maven [96], where the name of the project, version and dependencies with external libraries are specified in the $pom.xml$ file. This section presents the most important elements of the simulator.

6.3.1. MapDB database

As a data storage in the simulator, the MapDB [97] database was used, which allows saving and reading unstructured data in a very simple way. MapDB is an open-source solution with the embedded Java database engine and collection structure. Being one of the most efficient Java databases, it enables the use of such elements as maps, sets, lists or queues and provides such functions as ACID transactions, snapshots or incremental backups.The pseudocode responsible for connecting to the database by means of the DBMaker object is presented below, in Listing 6.1:

Listing 6.1. Connecting to the database using the DBMaker object

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

As can be seen, the use of the database is limited to providing the path to the file in which the data is saved. The file is saved directly to the local disk of the given BC network node. The database is used to store of a pool of unconfirmed transactions, blocks, or user accounts with their public keys. After the data is saved by the individual node, an event informing about that is propagated in the network to update local databases of the other nodes.

6.3.2. Networking

Communication between nodes in the network has been implemented using java sockets and threads. In socket programming [98] the client must have two information:

– server IP address,

– port number.

An example of one-way socket-based communication is shown in Fig. 6.5.

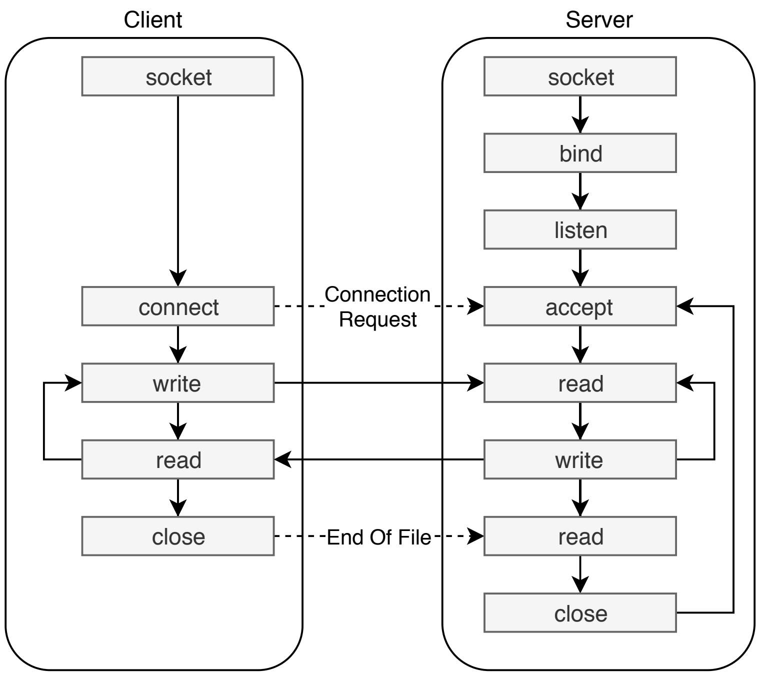

Fig. 6.5. One-way socket-based communication

Two main classes $Socket$ and $ServerSocket$ have been used to establish connections. The $Socket$ class is used for communication between the client and server so that to read and write messages. The $ServerSocket$ class is used on the server-side since it has an $accept()$ method that blocks the console until the client connects, and returns the $Socket$ instance after it is established. The following Listings 6.2 and 6.3 are samples of the server and client creation pseudocode.

Listing 6.2. Creating Server

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

Listing 6.3. Creating Client

```
Socket s = new Socket();
s.connect(new InetSocketAddress('server IP address'), 'port'), 10000);
```

Each node participating in the transaction confirmation process has both a client and a server on its side. After establishing the connection, every few seconds it sends a signal to the nodes with which it has previously established a connection to check if they are still active and not turned off. Transactions are sent only to the currently active nodes. Before sending, the $Transaction$ object is converted into its JSON representation. Once received, it is converted back to the object, which is done using the GSON library [99]. Most of the data in objects such as keys or node signatures have been saved in byte format to facilitate their transmission. The use of this type of data exchange guarantees very fast and reliable communication.

6.3.3. Transaction approval

The transaction confirmation process is based on an algorithm using the Stackelberg game presented in Section 6.2.1. The maximization problem with constraints described there was solved using the Commons Math library [100] that includes the $SimplexSolver$ object. On the Listing 6.4 is presented the pseudocode of the method that checks whether the transaction can be accepted (or rejected) by the verification node

Listing 6.4. Pseudocode of the method responsible for transaction approval

```
double s1Coefficient = leader.getSchedule().getMakespan()*
        leader.getScaleSchedulingFactor();
double s2Coefficient = follower.getSchedule().getMakespan()*
        follower.getScaleSchedulingFactor();

LinearObjectiveFunction f = new LinearObjectiveFunction(
        new double[] { s1Coefficient, s2Coefficient }, 0
);
Collection<LinearConstraint> constraints = new ArrayList<LinearConstraint>();
constraints.add(new LinearConstraint(
        new double[] { 1, 1 }, Relationship.EQ,  1)
);
SimplexSolver solver = new SimplexSolver();
PointValuePair solution = solver.optimize(
        new MaxIter(100),
        f,
        new LinearConstraintSet(constraints),
        GoalType.MAXIMIZE,
        new NonNegativeConstraint(true)
);

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

First the coefficients s_1 and s_2 are determined by means of the scaled scheduling factors of the $leader$ and $follower$ together with makespans calculated for the schedules prepared by them. With the use of the prepared coefficients, an object of the $LinearObjectiveFunction$, list of restrictions and a $SimplexSolver$ object is created and the process of finding a solution to the maximization problem is initiated. If y is equal to 1, then the schedule prepared by the $follower$ is better, which means that the $leader$ loses the game and the transaction is rejected. Otherwise, the $leader$ wins and receives confirmation of the correctness of the transaction from the $follower$.

6.3.4. Electing a leader and mining blocks

The process of mining blocks is based on Eq. 6.5. After reaching the appropriate number of transactions in the block, i.e. meeting Eq. 6.4, the leader is selected. The node that most often participated in creating and confirming transactions within a specified time is elected from the validator pool as a leader and, consequently, granted permission to attach a prepared block to the blockchain.

Listing 6.5. Electing a leader from validators pool

```
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;
}
```

As can be seen on the pseudocode presented in Listing 6.5, in method $getLeader()$ based on $TF_{(v_i,t)}$ the leader is selected. The time for which its trust factor is calculated is defined as a global system parameter.

6.3.5. Starting the node

This section contains a sample of pseudocode that shows the launch of one node instance, see Listing 6.6. First, the local port of the node is defined and a list of nodes in the network is prepared. Then, a thread is started, whose task is to occasionally send a signal to the other nodes informing about the node's activity.

Listing 6.6. Starting node instance

```
int localPort = 7301;
prepareNodeList();

new Thread(new PeriodicHeartBeat(serverStatus, localPort)).start();

Context context = new Context();

localNode = new blockchain.core.Node(context, wallet, serverStatus, localPort);
localNode.start();

ServerSocket serverSocket = new ServerSocket(localPort);
while (true) {
    Socket clientSocket = serverSocket.accept();
    localNode.addTransactionToPool(...);
    new Thread(new HeartBeatReceiver(clientSocket, serverStatus, localPort)).
        start();
}
```

A context object is created that contains a local database with blocks and pool of transactions. In the next two lines, node is started using the next thread and the server is created. The $accept()$

method waits for the clients. If clients connect with the local node port, it returns an instance of $Socket$. The node can then send the transaction to confirm what is happening by means of the $addTransactionToPool(...)$ method. Finally, a thread is started that receives a signal about the activity of all nodes in the BC network.

Chapter 7

Experiments

This chapter presents the results of the simple experimental evaluation of the developed simulator. The experiments have been conducted on the workstation with the *Intel Core i5 2.7 GHz* processor and 8 GB of RAM memory. First, the mechanism of the proposed Stackelberg game is verified. Then, the tests of the implemented BCSchedCloudSim and comparative simulations are carried out, evaluating the proposed blockchain scheduler together with some other available schedulers. The chapter ends with a short summary of the achieved results and the conclusions from the experimental analysis.

7.1. Numerical results of the implemented Stackelberg game

This section presents the numerical results of playing Stackelberg game proposed in Section 6.2.1. The goal of these experiments is to examine the properties of the proposed game, particularly the impact of input data on the game result. The analysis of numerical results was conducted under the following conditions:

– the game involves two players, the $leader$ and the $follower$,

– makespan M is used as SC,

– each player has scheduling factor SF and proposes his own M, the game is simulated for different SF (see Section 6.2.1) and M values,

– the $leader$ proposes a schedule and the $follower$ verifies it, the result of the game is its rejection or not.

Numerical results for the two chosen scenarios are presented below. In the first scenario, scheduling factors of both players are equal in order to show how the network behaves at the beginning when the SF values of all nodes are the same or very similar. In the second one, different scheduling factors and makespans were adopted to illustrate the operation of the BC network, which had been running for some time, and the SF values are different, depending on the activity of a given node.

Scenario 1 - scheduling factors of both players are equal

The parameters of the first scenario of the simulation are as follows:

- M_{leader} - 6100 sec
- SF_{leader} - 100 MFLO
- $M_{follower}$ - 6000 sec
- $SF_{follower}$ - 100 MFLO

As can be already seen, both the $leader$ and $follower$ have the same SF equal to 100 MFLO. According to Eq. 6.1, scheduling factors are first scaled:

1. $max\{SF_{leader}, SF_{follower}\} = 100$
2. $\overline{SF_{leader}} = 1$
   $\overline{SF_{follower}} = 1$

According to Eq. 6.2, the utility function for the $follower$ takes the form of:

$$u(s_1, s_2, M_{leader}, M_{follower}) = 6100 \times 1s_1 + 6000 \times 1s_2$$

According to Eq. 6.3, the problem to solve is:

$$\begin{cases} \underset{s_1,s_2}{\operatorname{argmax}}\ 6100s_1 + 6000s_2 \\ s_1 + s_2 = 1 \\ s_1, s_2 \in \{0,1\} \end{cases} \Rightarrow \begin{cases} u(s_1, s_2, M_{leader}, M_{follower}) = 6000 \\ s_1 = 1 \\ s_2 = 0 \end{cases}$$

The solution indicates that the winner of the game is the $leader$, which means that his schedule was approved by the $follower$. Table 7.1 presents the results of several simulations for different makespans and the same scheduling factors. As demonstrated above, if the scheduling factors are the same for both players, any schedule proposed by the leader with a smaller makespan than the one proposed by follower will be rejected. On the other hand, any schedule with a larger or equal makespan will be accepted. Consequently, players with equal SF factors have limited confidence in each other and do not accept schedules being better according to their opponents; they only accept solutions that are just as good or worse than their algorithm. It is also seen that if SF factors are equal, their size does not affect their decisions.

No.	$SF_{leader,follower}$	M_{leader}	$M_{follower}$	Schedule approved
1.	100	10	6000	✗
2.	100	5900	6000	✗
3.	100	5999	6000	✗
4.	100	6000	6000	✓
5.	100	6100	6000	✓
6.	300	6100	6000	✓
7.	100	6500	6000	✓
8.	1000	7000	7000	✓

Table 7.1. Stackelberg game results for different M and the same SF

Scenario 2 - different scheduling factors and makespans

The parameters of the second scenario of the simulation are as follows:

– M_{leader} - 6000 sec

– SF_{leader} - 284 MFLO

– $M_{follower}$ - 5700 sec

– $SF_{follower}$ - 300 MFLO

As we have already seen, both the $leader$ and the $follower$ have different SF and M. According to Eq. 6.1, scheduling factors are first scaled:

1. $max\{SF_{leader}, SF_{follower}\} = SF_{follower} = 300$
2. $\overline{SF_{leader}} = \frac{SF_{leader}}{SF_{follower}} = \frac{284}{300} = \frac{71}{75}$
   $\overline{SF_{follower}} = 1$

The utility function for the $follower$ is determined by the Eq. 6.2:

$$u(s_1, s_2, M_{leader}, M_{follower}) = 6000 \times \tfrac{71}{75} s_1 + 5700 \times 1 s_2$$

According to Eq. 6.3 the problem to solve is:

$$\begin{cases} \underset{s_1,s_2}{\operatorname{argmax}}\ 5680 s_1 + 5700 s_2 \\ s_1 + s_2 = 1 \\ s_1, s_2 \in \{0,1\} \end{cases} \Rightarrow \begin{cases} u^n(s_1, s_2, M_{leader}, M_{follower}) = 5700 \\ s_1 = 0 \\ s_2 = 1 \end{cases}$$

No.	SF_{leader}	$SF_{follower}$	M_{leader}	$M_{follower}$	Schedule approved
1.	100	300	6000	5700	✗
2.	270	300	6400	5700	✓
3.	284	300	6000	5700	✗
4.	284	300	6150	5700	✓
5.	285	300	6000	5700	✓
6.	305	300	5600	5700	✗
7.	306	300	5600	5700	✓
8.	311	300	5500	5700	✓

Table 7.2. Stackelberg game results for different SF and M

The solution indicates that the winner of the game is the $follower$, which means that it rejects the schedule proposed by the $leader$. Table 7.2 presents the results of several simulations for different scheduling factors and makespans. As can be seen, the comparison between scheduling factors of the leader and follower indicates the following relation: the lower the scheduling factor of the leader is, the higher the makespan value of the schedule proposed by the follower must be to compel him to consider the schedule as correct (see example 2 and 3). On the other hand, the higher the leader's scheduling factor is, the lower his schedule makespan can be, which is particularly evident in examples number 6 and 7.

The numerical results presented in this section prove that the higher the scheduling factor of a given player is, the more trust other players have in it. Players with the same SF have limited confidence in each other and prefer their own solutions. Thanks to this behaviour, it is certain that the BC network based on the proposed game operating in accordance with the standards adopted by the nodes, the prepared schedules are correct and their rejection rate is relatively low.

7.2. Evaluation of the Blockchain Secure Cloud Scheduler Simulator

In this part of the experiments, the proposed BCSchedCloudSim was checked for the correct execution and verification of operations, such as creating or approving transactions or blocks. The time of performing individual operations was also examined. The main goal for this simulation was to verify the blockchain mechanism, assuming that schedules placed in transactions are randomly generated. Security, correctness and the optimum were not verified. The tests were carried out according to the following scenario:

1. There are several nodes in the blockchain network that have access to the pool of requests thrown by task managers.

	Tasks		Virtual Machines	
No.	$[wl_1, \ldots, wl_n]$ in [MFLO]	$[sd_1, \ldots, sd_n]$	$[cc_1, \ldots, cc_m]$ in [MFLOPS]	$[tl_1, \ldots, tl_m]$
1.	[980, 924, 868, 566, 839]	[0.94, 0.23, 0.56, 0.40, 0.80]	[9.019445, 9.843292, 6.745162, 7.520450, 1.841657]	[0.76, 0.10, 0.93, 0.30, 0.32]
2.	[170, 189, 651]	[0.21, 0.73, 0.32]	[3.4275, 2.346821]	0.88, 0.87]

Table 7.3. The example of characteristics of tasks and virtual machines from the request

2. One of the nodes is started, which begins the search for the Genesis Block; the Genesis Block does not yet exist, so it is created and propagated by the node across the network.

3. The remaining nodes in the network are started. One of the BC network nodes retrieves several entries from the pool of requests and prepares transactions by placing random schedules within them.

4. After the transactions are ready to send, the initiating node broadcasts them to other nodes that are responsible for verification.

5. Confirmation nodes receive broadcasted transactions, then generate their own random schedules and compare them with those sent by the initiating node, which results in their rejection or approval.

6. After obtaining the appropriate number of confirmed transactions, the initiating node creates a block.

7. A leader is selected from the pool of validators that verifies the proposed block and adds it to the chain.

Simulation parameters

Tests checking the correct implementation of the BCSchedCloudSim were carried out with the following parameters:

- size of the BC network $= 16$ nodes. This is the maximum that can be achieved with the parameters of the computer on which the simulator is running, but it is enough to conduct the experiments effectively and correctly;
- data from the pool of requests is randomly generated, the example of characteristics of tasks and virtual machines from the request is shown in Table 7.3. In the first point, the workload is defined for 5 tasks and computing capacity for 5 virtual machines, in the second one, the workload for 3 tasks and computing capacity for 2 virtual machines was defined. For each task, security demand is defined, and similarly, trust level for each virtual machine;
- expected security level defined in each request $= 0$. SL is omitted, because it does not affect the operation of the simulator;
- minimum number of transaction confirmation (MTC) $= 8$. It is 50$\%$ of nodes of the simulated BC network, this quantity ensures that the system is secure, the client receives the optimal schedule and does not have to wait too long to get the result;
- $B_{wl} = 10000$ MFLO. At such selected value and workloads of tasks from the requests, about $2 - 5$ transactions will be placed in the block;
- number of records in the pool of requests $= 16$. Enough to build at least one block with the assumed B_{wl} value and generated workloads presented in Table 7.3;
- time for which TF of validators is determined $=$ 30 days. Limiting the time to 30 days prevents the problem of network centralization [101];
- each node is running on a different port on the same host; the configuration of nodes is presented in the Table 7.4. Node is described by the host, port and node numbers to which it has a connection.

No.	Host	Port	Connections
1.	127.0.0.1	7000	[2, …, 16]
…	…	…	…
16.	127.0.0.1	7016	[1, …, 15]

Table 7.4. Configuration of nodes defined in the simulator

The initial state adopted in the simulation is an empty blockchain. The study was divided into stages in which individual processes take place.

Stage 1 - Genesis Block Generation

The first node that is started is node number 1. It begins the search for the Genesis Block in the database. This block does not yet exist, so it is created according to the statically defined code in the application. The generated block is shown in the Listing 7.1:

Listing 7.1. Generated Genesis Block Body

```
{
  "hash": "a02ba163f3a02db22fdd14b310119f1a4c2f9e4a773a573f757904dd5433d4dd",
  "previousHash": "0",
  "timeStamp": 1568135430934,
  "merkleRoot": "9a20c547e80200cdd8c9dfef9fac68d1ae772466cd57900c741fdce04c79bc6b
      ",
  "transactions": [
    {
      "transactionId": [49],
      "sender": [in bytes],
      "recipient": [in bytes],
      "schedule": {
        "tasks": [
          {"id": 1, "workload": 960.0, "securityDemand": 0.21},
          {"id": 2, "workload": 761.0,"securityDemand": 0.82},
          {"id": 3, "workload": 990.0, "securityDemand": 0.35},
          {"id": 4, "workload": 333.0, "securityDemand": 0.64},
          {"id": 5, "workload": 610.0, "securityDemand": 0.12}
        ],
        "machines": [
          {"id": 1, "computingCapacity": 4.575484, "trustLevel": 0.98,
          "tasksToExecute": [5,2,1,3]},
          {"id": 2, "computingCapacity": 5.25538, "trustLevel": 0.04,
          "tasksToExecute": [4]}
        ],
        "makespan": 216.47055215237637,
        "securityLevel": 0.916549650474798
      },
      "r": [in bytes],
      "s": [in bytes],
      "v": [28],
      "outputs": [
        {"id": [in bytes],"sender": [in bytes], "parentTransactionId": [49],
        "schedulingFactor": 3654.0, "timestamp": 1568135430934}
      ],
      "numberOfVerification": 0
    },
    {"transactionId": [50] ...},
    {"transactionId": [51] ...},
    {"transactionId": [52] ...}
  ]
}
```

The block hash was generated from the word "$scheduler$" using the SHA-256 hash function. Because this is the first block in the chain, it has no preceding block and the previous hash value is 0. The block contains 4 transactions with randomly generated schedules. Based on these transactions, MTRhv was generated according to the algorithm shown in Fig. 4.5. Private keys of the schedule creators placed in transactions have been defined in the code, so they are known and can be used in the later tests. PKN, PKTM and DS are stored using bytes, which greatly facilitates their placement in the blockchain and transmission between nodes. Due to their length, they are hidden in the presented listing under the name [in bytes]. The Elliptic Curve Digital Signature Algorithm (ECDSA) [102] was used to generate the electronic signature, which is saved using three elements: r, s and v. The schedule contains data about the tasks and machines on which they are to be executed. In this case, makespan was used as SC, which is also saved in the transaction along with the SL calculated for the prepared schedule. The key of the sender and the scheduling factor calculated on the basis of the workloads of the tasks $wl(schedule)$ are saved in the outputs of each transaction. Quick access to this information is facilitated by calculating SF for individual nodes.

Stage 2 - Creating and confirming transactions

In the first step, node number 1 was started and the Genesis Block was created. Now, the other nodes are launched. In this simulation, node number 2 is responsible for creating and sending transactions while the role of the other nodes is to verify and confirm them.

Listing 7.2. Log output of the node sending the transaction to other nodes for verification

```
Receiving hb|heartbeat from another peer working on port: 7000
Receiving hb|heartbeat from another peer working on port: 7002
⋮
Receiving hb|heartbeat from another peer working on port: 7016

Starting scheduler...
Initialising...
Simulation completed.
Makespan: 216.47055208219845

Sending transaction to peers for accept...
```

```
12 sending tx|{...}|7000
13 sending tx|{...}|7002
14 ⋮
15 sending tx|{...}|7016
16
17 Transaction was verified.
18 Transaction was verified.
19 ⋮
20 Transaction added to verified pool.
21 Transaction was verified.
22 ⋮
23 Transaction added to verified pool.
24 Transaction was verified.
```

As seen in the Listing 7.2, node 2 receives signals about the activity of the other nodes in the BC network. Then starts the process of preparing the schedule, which places in the transaction. In line number 9, makespan value for the schedule proposed by node 2 is presented. Schedule is generated 16 times, always using the same data, so the node propagates in the network 16 transactions. However, the prepared schedule is not always the same as before. After sending it waits for responses, in some cases transactions are rejected and in some cases confirmed. After obtaining confirmations from 8 nodes in the network (MTC = 8), the transaction is added to the pool of verified transactions, from which it is then added to the block. The next Listing 7.3 describes the process from the perspective of the other nodes for checking received transactions:

Listing 7.3. Log output of the node receiving the transaction for verification

```
Checking transaction...
Starting scheduler...
Initialising...
Simulation completed.
Leader makespan: 188.47838557820745
Leader scheduling factor: 3654.0
Follower makespan: 216.4705522738141
Follower scheduling factor: 3654.0
Duration of the Stackelberg game: 332626 [NS].
Schedule incorrect, transaction rejected.
⋮
Checking transaction...
Starting scheduler...
Initialising...
Simulation completed.
Leader makespan: 144.9039913383999
```

```
17 Leader scheduling factor: 3654.0
18 Follower makespan: 188.47838557820745
19 Follower scheduling factor: 3654.0
20 Duration of the Stackelberg game: 310990 [NS].
21 Schedule correct, transaction verified.
```

Nodes receiving transactions to confirm first prepare a schedule for transmitted data according to its own algorithm. Then the Stackelberg game described in Section 6.2.1 and verified in Section 7.1 is carried out. The example shows the makespan proposed by node 2 (leader) and its SF, as well as the makespan calculated for the verification node (follower) schedule and its SF. SF values are equal, according to the transactions placed in Genesis Block. Transactions with schedules of the same $wl(schedule)$ value have already been placed there, both for node 2 and other nodes in the network. This prevents the randomization of SF value which would occur if the node had not added or verified a transaction before. The duration of the Stackelberg game is presented in line number 20. As can be seen, depending on the game result, the transaction in some cases is rejected and in some - accepted.

Stage 3 - Block mining

After obtaining the appropriate value of B_{wl} (10000), the process of block mining begins. Four transactions are added to the block, each of them confirmed by at least 8 nodes. After obtaining the appropriate value of B_{wl}, according to Eq. 6.5, a leader is selected from validators. L_t is chosen on the basis of TF determined for the last 30 days. In the described case, the leader was the node whose TF_t was equal to 18270 MFLO. Log output from the validator is shown in Listing 7.4.

Listing 7.4. Log output of block validator

```
Transaction successfully added to the block.
Transaction successfully added to the block.
⋮
Transaction successfully added to the block.

Number of transactions in block: 4
Validator trust factor: 18270.0
Block mined!!! : 677036ac572566376a389166c8f68858da97b729db4ba2c366a764f5f820d259
```

At the end of the log output, there is information that the block has been correctly added to the blockchain, along with its hash. Subsequently, this block has been propagated in the network to update local databases of other nodes in the BC network.

The three-step process described above, leading from the creation of the Genesis Block, through the creation and confirmation of transactions, to the final addition of the second block, confirms the correct operation of the simulator. The transaction verification process was carried out in accordance with the assumptions of the network consensus algorithm described in Section 6.2. Sixteen transactions were propagated in the network, and all were correctly read by nodes. BCSchedCloudSim has been tested for 16 simultaneously running nodes, which confirms the reliability of the presented results. The nodes correctly exchanged information between each other and verified the schedules according to the proposed Stackelberg game. Duration of the game was also measured; the results are shown in Fig. 7.1. The time for solving the optimization problem described in Eq. 6.3 was measured,

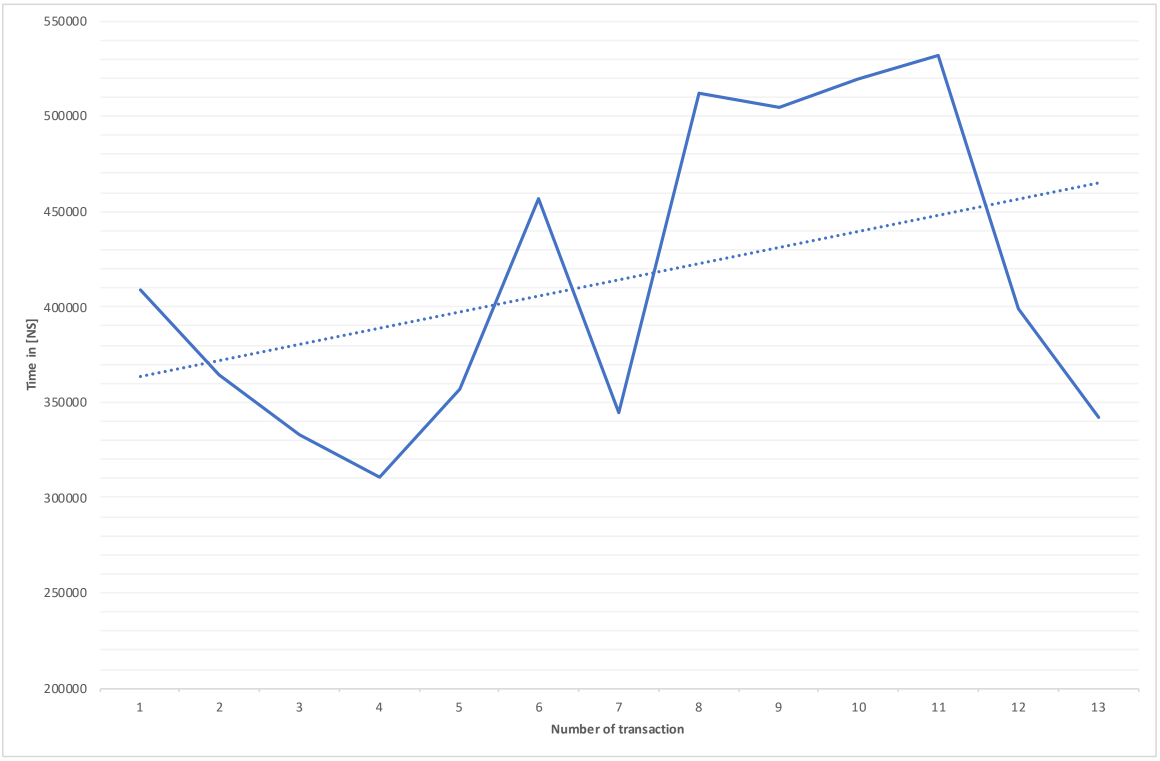

Fig. 7.1. Time of solving the optimization problem using the simplex method during transaction verification

not the entire transaction verification time, which depends on the time of the schedule preparation. The values range from 300000 to 550000 nanoseconds, so the solver used in the implementation works quickly enough not to delay the entire process. The presented results confirm the correctness of the simulator implementation, which was used to conduct experiments comparing the proposed solution with the existing ones described in the next section.

7.3. Evaluation of the proposed Blockchain Scheduler in comparison with the existing scheduling modules

This part of the experiment presents some comparisons of the proposed (BS) with other existing modules dedicated to task scheduling in computational clouds. Tests were carried out using a simulator evaluated in the previous section, and their goal was to check whether in optimal conditions the proposed solution worked better than other similar solutions. For the purpose of experiments, four different algorithms for task scheduling were implemented in the simulator. These were:

- First Come First Served (FCFS) [86];
- Shortest Job First (SJF) [87];
- Round Robin (RR) [103];
- Hybrid Heuristic Method based on Genetic Algorithm (HSGA) [104].

The algorithms enumerated above are all used by the existing scheduling modules. They are based on the implementations available in [105], [106], where CloudSim simulator [107] was used to test the execution of the prepared schedules.

Simulation scenario

The scenario of the conducted experiments is presented below:

1. The task scheduling process is run on 4 different scheduling modules; each module uses one of the RR, FCFS, HSGA or SJF algorithms.
2. BS is launched and 16 nodes are defined in the BC network. Each of them uses one of the four algorithms used by the scheduling modules described in point 1 to prepare the schedule. As a result, BS saves the best schedule selected by the network in the blockchain.
3. Schedules from points 1 and 2 are carried out for different datasets (different number of tasks and different number of virtual machines).
4. Each of the prepared schedules is evaluated according to different metrics. Points 1 and 2 are repeated to evaluate the schedule in terms of the values of makespan, flowtime, economic cost and resource utilization; the security level of the schedule is not taken into consideration (expected SL = 0). The value of each criterion is calculated on the basis of the results of schedule execution using the CloudSim simulator.
5. Points 1 and 2 are repeated for different expected SL, taking into account the makespan as a schedule evaluation criterion.

Simulation parameters

In order to conduct the simulation, test datasets were prepared in the form of task and virtual machines characteristics (workloads and computing capacities) together with the security demand and trust level parameters. Tasks characteristics ($[wl_1, \ldots, wl_n]$, $[sd_1, \ldots, sd_n]$) and virtual machines characteristics ($[cc_1, \ldots, cc_m]$, $[tl_1, \ldots, tl_m]$) were generated according to the Gaussian distribution:

$$\mathcal{N}(\mu, \sigma^2) \tag{7.1}$$

where:

μ – mean

σ^2 – variance of random variable

In the literature, there are many cases where test data is generated for several different scenarios (different number of machines and tasks). The number of virtual machines usually ranges between 32 and 256, while the number of tasks from 512 to 4096. Kołodziej [31] provided four different scenarios where different numbers of tasks and machines were defined and data was generated according to the normal distribution for machines $\mathcal{N}(5000, 875)$ and for tasks $\mathcal{N}(250000000, 43750000)$, respectively. The real characteristics of the computational units can be found on the webpages [108], [109], [110] where different processors are compared. Taking account of the experiments conducted by Kołodziej, it was decided to generate 2 different datasets of characteristics of virtual machines and 2 different datasets of characteristics of tasks, whose key parameters are presented in Tables 7.5 and 7.6. Datasets were generated applying the Commons Math library [100] using the $NormalDistribution$ class.

	Dataset 1	Dataset 2
Number of VM	32	128
Distribution of computing capacity (cc) values	$\mathcal{N}(7, 4)$	
Measure of cc	MFLOPS	
Minimum value of cc	1	
Maximum value of cc	12	
Distribution of trust level (tl)	$\mathcal{N}(0.5, 0.04)$	
Minimum value of tl	0.2	
Maximum value of tl	1	

Table 7.5. Characteristics of virtual machines

	Dataset 1	Dataset 2
Number of tasks	1024	4096
Distribution of workloads (wl) values	$\mathcal{N}(600, 90000)$	
Measure of wl	MFLO	
Minimum value of wl	100	
Maximum value of wl	1000	
Distribution of security demand (sd)	$\mathcal{N}(0.8, 0.0225)$	
Minimum value of sd	0.6	
Maximum value of sd	0.9	

Table 7.6. Characteristics of tasks

Due to the fact that 2 datasets of characteristics of tasks and 2 datasets of characteristics of virtual machines were generated, each task scheduling process took place on 4 different datasets:

- 32 VM and 1024 tasks;
- 32 VM and 4096 tasks;
- 128 VM and 1024 tasks;
- 128 VM and 4096 tasks.

BCSchedCloudSim parameters were set according to Table 7.7. The tested network consisted of 16 nodes, each tested dataset was thrown as a request into the pool of requests 96 times. Because of the initiating node must obtain confirmation from 8 other nodes to consider the transaction as valid the parameters were mostly the same as those adopted in the previous section for validation blockchain network implementation.

Number of records in the pool of requests	96
MTC	8
B_{wl}	1000000 MFLO
Time for which TF of validators is determined	30 days
Initial value of SF of each node in the BC network	583329 MFLO
Failure coefficient α (see. Eq. 3.3)	2.5

Table 7.7. BCSchedCloudSim configuration

Table 7.8 presents the CloudSim simulator parameters that were adopted during the simulation. One data centre with many virtual machines is simulated, each with 4096 MB of RAM and 4 cpus.

1 data center	
Size of VM image	10000 MB
Memory of VM	4096 MB
Number of CPUs in VM	4

Table 7.8. CloudSim configuration

Each schedule preparation and calculation metrics by 4 different scheduling modules based on the schedule execution in the CloudSim simulator was repeated 48 times. Then, the obtained results were compared with those returned by BS, where 96 requests with the same data were placed.

Evaluation of the Secure Blockchain Scheduler based on various scheduling criteria

According to the previously presented scenario, several experiments were carried out to assess the proposed solution. The parameters for each experiment presented in this section are in Table 7.9. As can be seen in this part of experiments, the security level criterion was omitted and will be considered in the next section. First, the impact of the applied solution on the obtained results was examined using makespan as a criterion for schedule evaluation. The algorithm for calculating the value of makespan was presented in Eq. 1.1. The results of the Exp. 1 are shown in Fig 7.2. The shorter the time necessary to execute the entire schedule, the better the schedule was. As demonstrated in the obtained results, in most cases the best result was achieved by BS. In three cases, the best makespan was returned. In one case, the returned makespan was not the best but it is not the worst either. Discrepancy between the minimum and maximum values returned by the scheduler is the smallest in the case of BS, which confirms the stability of its operation.

No.	SC	Expected SL	Size of the BC network
Exp. 1	makespan	0	16 nodes
Exp. 2	flowtime	0	16 nodes
Exp. 3	economic cost	0	16 nodes
Exp. 4	resource utilization	0	16 nodes

Table 7.9. Experiments outline - various scheduling criteria

In the next experiment (Exp. 2), flowtime was adopted as the criterion for assessing the schedule. The method of calculating flowtime was described in Eq. 1.2. The lower the flowtime, the lower the average response time of the entire schedule is. Therefore, the lower the flowtime value, the better the schedule must be. The results presented in Fig. 7.3 show that the HSGA and BS modules perform

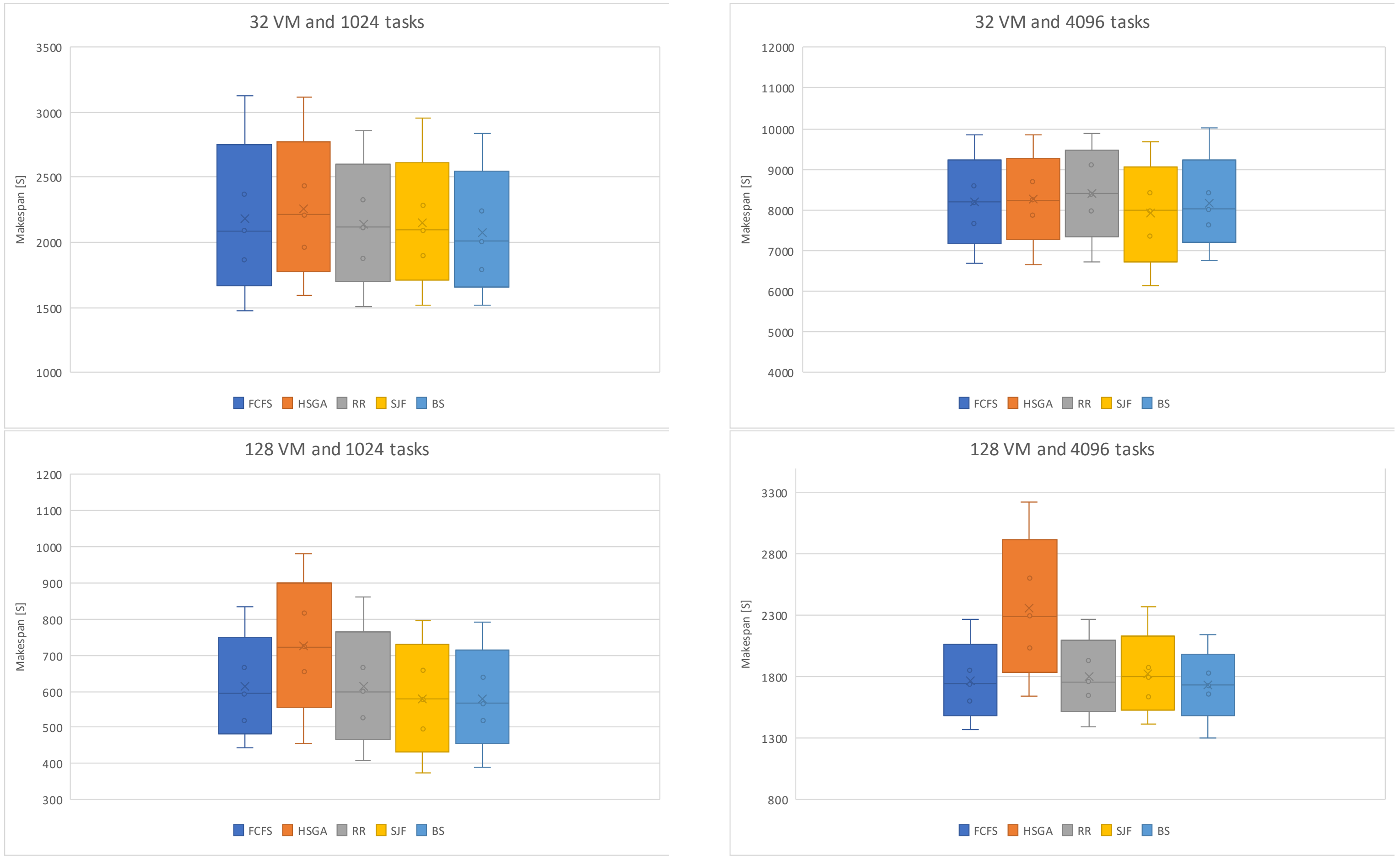

Fig. 7.2. Exp. 1 - evaluation of the model performance using makespan

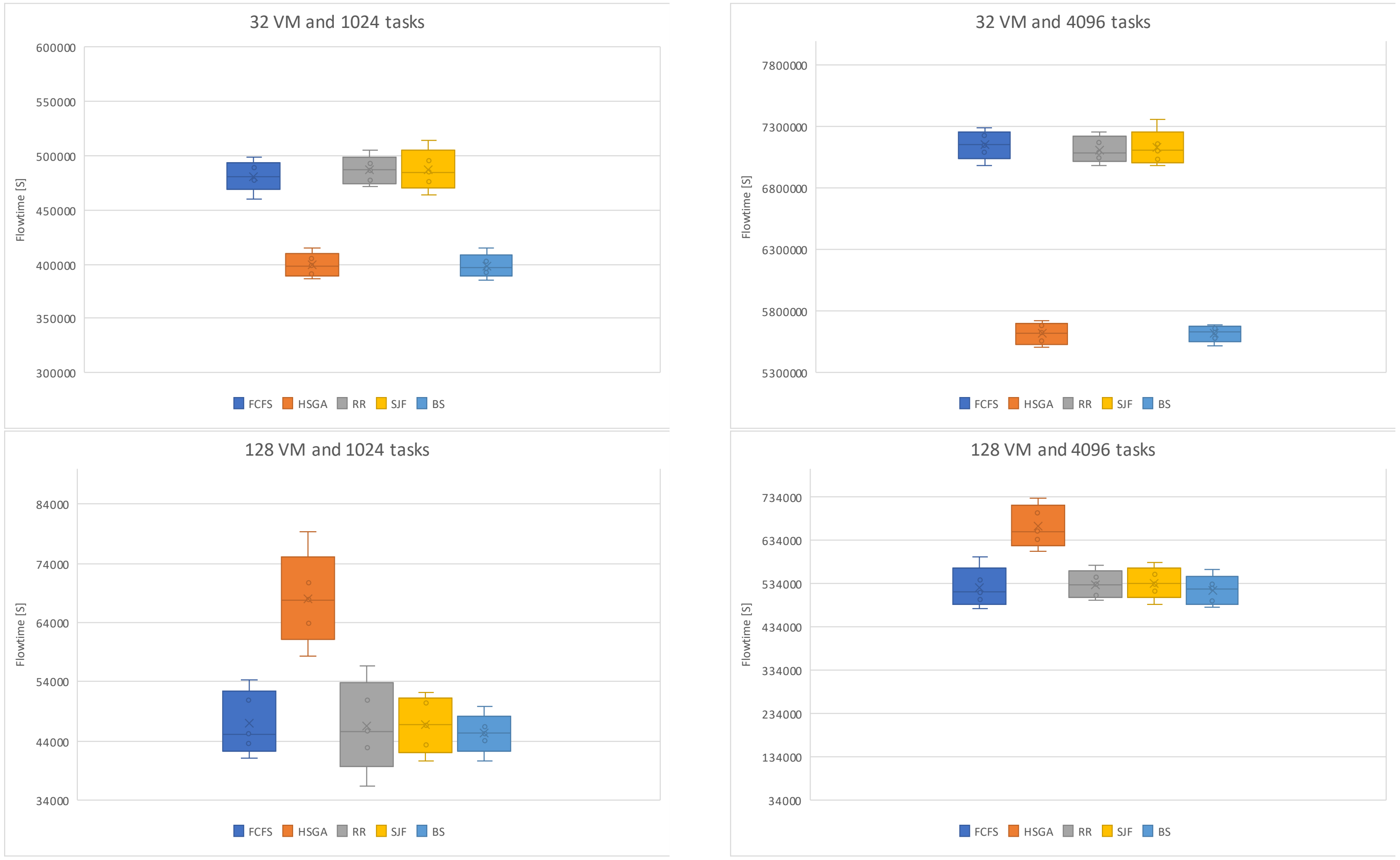

Fig. 7.3. Exp. 2 - evaluation of the model performance using flowtime

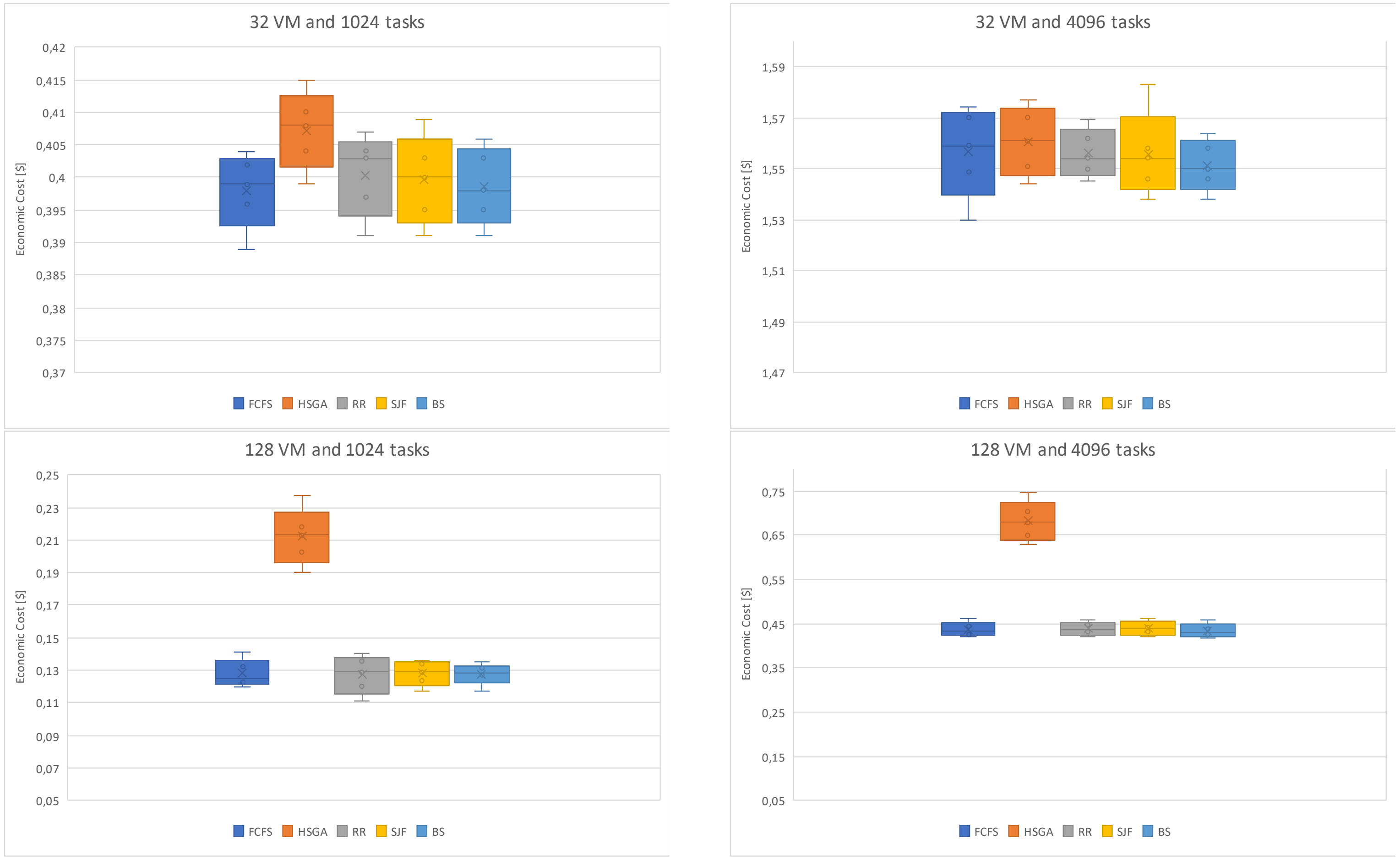

Fig. 7.4. Exp. 3 - evaluation of the model performance using economic cost

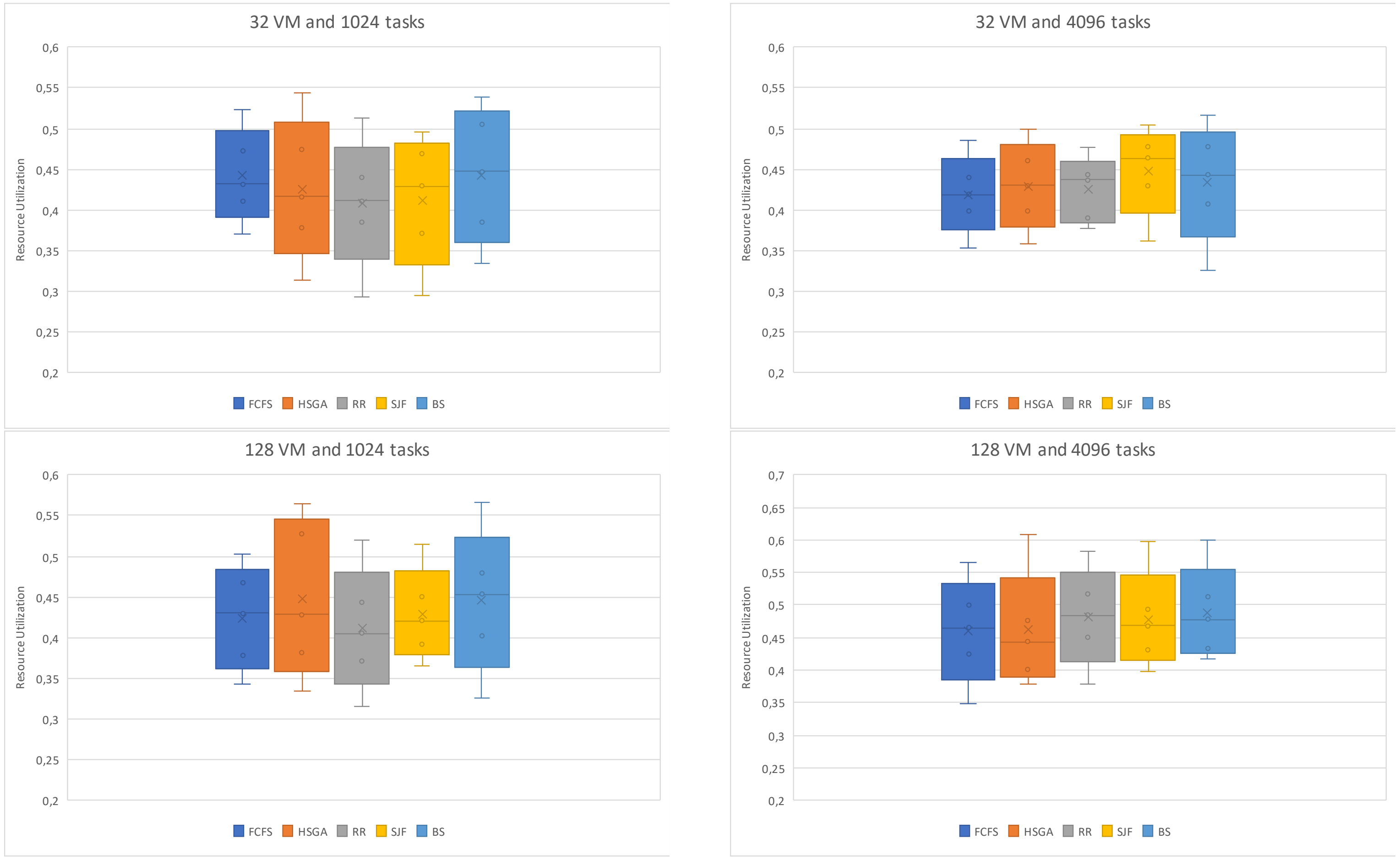

Fig. 7.5. Exp. 4 - evaluation of the model performance using resource utilization

best with fewer machines and tasks. However, as the number of tasks and virtual machines increases, HSGA module returns worse and worse results, which is not the case with BS. With larger numbers of virtual machines and tasks, BS is still the leader and provides the best results.

In Exp. 3 the economic cost criterion was focused on in schedule evaluation. The algorithm of calculating economic cost is shown in Eq. 1.3. To calculate its value, it was assumed that the cost of 1 second of resource utilization C_i is equal to $\$0.0000142$, this cost was estimated based on the prices presented on the AWS website [111]. It was also assumed that the cost of using each virtual machine is the same. Accordingly, the less a customer has to pay for the schedule, the better the schedule is. The results of this simulation are presented in Fig 7.4. It can be seen that the costs of executing simulated schedules are between $\$0.10$ and $\$1.5$. Considering this criterion, each scheduler returned similar results. Only the schedule returned by HSGA for more virtual machines and tasks proved more expensive. However, in three cases BS returned the most favourable result. Only in one case, FCFS was better.

In the last experiment (Exp. 4), resource utilization by the schedule was assessed. The method of its calculation is presented in Eq 1.4. This measure is important from the providers' point of view since the better utilization of resources and reduction of time gaps enables them to gain greater profits. Therefore, the higher the value of resource utilization, the better the schedule is. The results presented in Fig. 7.5 show that differences in the resource utilization of prepared schedules are not large, while the discrepancy in the returned minimum and maximum results is quite large. This time BS was the best in two cases, whereas the SJF module also brought satisfactory solutions.

To determine the significance of the obtained results , the two-tailed Wilcoxon signed-ranks test [112] for paired samples was also conducted as part of experiments. The test was carried out for Exp. 1, where makespan was used as a SC, which in most papers is considered as the most important criterion for schedule evaluation. The study was performed by selecting the worst result returned by a single scheduling module as the first sample and the result returned by BS as the second sample. The following null hypothesis was adopted:

Null Hypothesis 1. H_0: *any differences between the worst makespan returned by single scheduling module and the makespan returned by BS are due to chance*

The results of the test are shown in Table 7.10.

VM and tasks	Makespans with σ	$W+$	$W-$	W	Wn
32 VM, 1024 tasks	HSGA (341.354) & BS (329.543)	851.5	324.5	324.5	48
32 VM, 4096 tasks	RR (747.717) & BS (694.133)	921	255	255	48
128 VM, 1024 tasks	HSGA (107.029) & BS (90.764)	1105	71	71	48
128 VM, 4096 tasks	HSGA (376.586) & BS (139.968)	1164	12	12	48

Table 7.10. Two-tailed Wilcoxon signed-ranks test for Exp. 1

where:

$W+$	– positive sum of test
$W-$	– negative sum of test variable
W	– Test statistic
Wn	– number of samples where difference was not equal to 0 variable

The critical value for the W Test statistic from the Wilcoxon signed-ranks table [113] for $Wn = 48$ and $\alpha = .05$ is equal to $W_{crit} = 396$, which gives the range $[0, 396]$. Since W statistic of each performed test is in the range $[0, 396]$, the Null Hypothesis 1 is rejected, so it can be concluded that there is a significant difference between makespans obtained from single scheduling module and BS scheduler.

Evaluation of the Secure Blockchain Scheduler based on the security level of the schedule

In the previous section, the security level criterion was omitted. In this section, the results of experiments assessing primarily this aspect are presented. The parameters of each of the experiments carried out are presented in Tab. 7.11. The SL was calculated on the basis of Eq. 3.4. This equation

No.	SC	Expected SL	Size of the BC network
Exp. 5 and 6	makespan	1.5	16 nodes

Table 7.11. Experiments outline - expected security level equal to 1.5

takes into account three factors: $P^{failure}$, P^{fake} and $P^{hacking}$ [43].

$P^{failure}$ is calculated based on the prepared schedule according to Eq. 3.3 and determines the probability of machine failure during task execution due to the high security requirements specified for this task. This factor is estimated the same for each scheduler used in the experiments.

The value of the next factor P^{fake} arbitrates the probability that the schedulers send a false or incorrectly prepared schedule. In the experiments carried out for the purposes of this dissertation, four scheduling modules were used that directly return the results, and a fifth one in which the results are processed via a BC network. In the case of a single schedule module (FCFS, HSGA, RR and SJF) whose schedule is not verified by anyone else, it can be assumed that P^{fake} is equal to 0.5. However, in the case of BS, where the schedule is checked by other nodes from the network, the value of P^{fake} is determined by the formula:

$$P^{fake} = 1 - \frac{N_c}{N_v} \tag{7.2}$$

where

N_c	–	number of all confirmations that the schedule is correct obtained by the node from the verification nodes
N_v	–	number of all verifications of the schedule regardless of whether the answer was positive or negative

Assuming that there are 16 nodes in the network and MTC is equal to 8, it can be seen that in the case of BS this factor assumes a value within the range $[0, 0.5]$, depending on how many requests the node must send to the network to receive 8 confirmations.

The value of the last factor $P^{hacking}$ indicates the probability of manipulation or modification of the prepared schedule by unauthorized entities. In the case of a single schedule module (FCFS, HSGA, RR and SJF), it can be assumed that $P^{hacking}$ is equal to 0.5. For BS, this value depends on TF_t of the attacking node. The higher the TF_t the block adding node has, the greater the probability that it may launch a majority attack [114] to modify parts of the blockchain. The $P^{hacking}$ value in that case is calculated according to the following formula:

$$P^{hacking} = \begin{cases} 0.5 & TF_t \geq \frac{1}{2}BW_t \\ \frac{TF_t}{BW_t} & TF_t < \frac{1}{2}BW_t \end{cases} \tag{7.3}$$

where

TF_t	–	the sum of $wl(schedule)$ for all schedules added to the blockchain by attacking node within a given period of time t (in this simulation t is equal to 30 days)
BW_t	–	described in Section 6.2.1

The Eq. 7.3 shows that $P^{hacking}$ in the case of BS assumes a value within the range $[0, 0.5]$, depending on how high the attacker's trust factor is in the moment of adding the block to the blockchain.

Exp. 5 and 6 show how much advantage BS module has over the other scheduling modules when it comes to the security level criterion. The results of these experiments are presented in Fig. 7.6 and 7.7. As can be seen, the difference in the number of virtual machines and tasks is not significant because the results are very similar. However, it is worth noticing that BS has a significant advantage over the other modules. In the case of BS these are values around 2 while in the case of other schedulers the values are close to 1.5. In the simulation, the expected SL was set to 1.5 and all schedulers managed to achieve such a result.

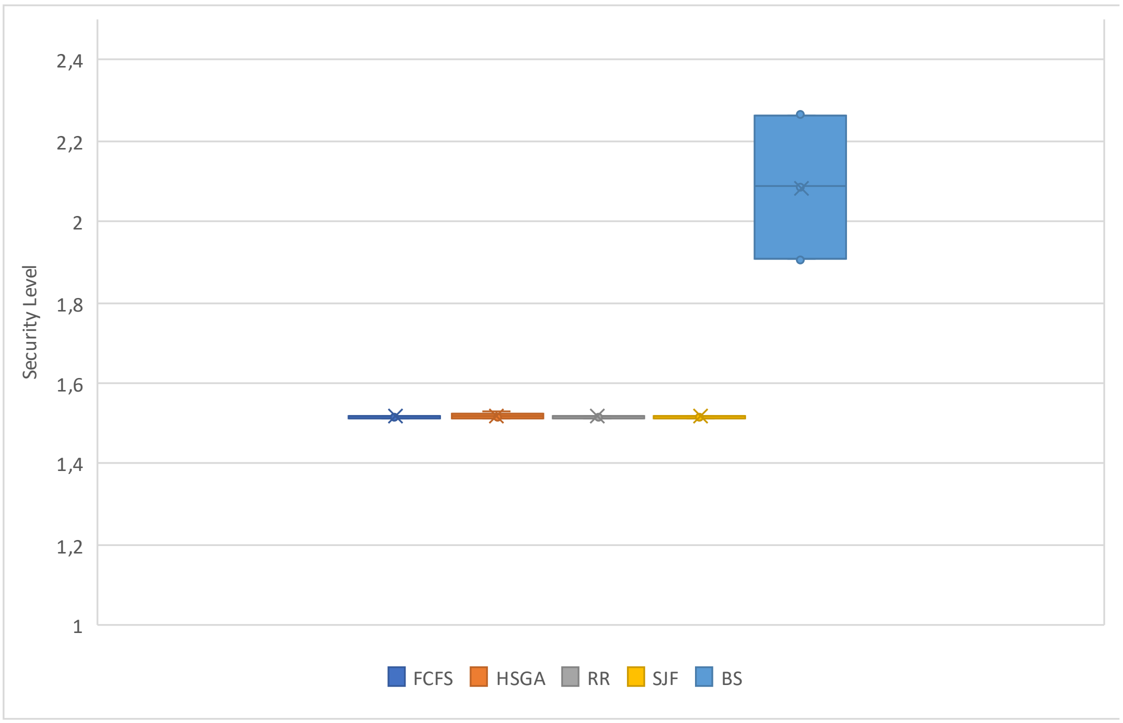

Fig. 7.6. Exp. 5 - evaluation of the model performance using security level - 32 VM and 1024 tasks

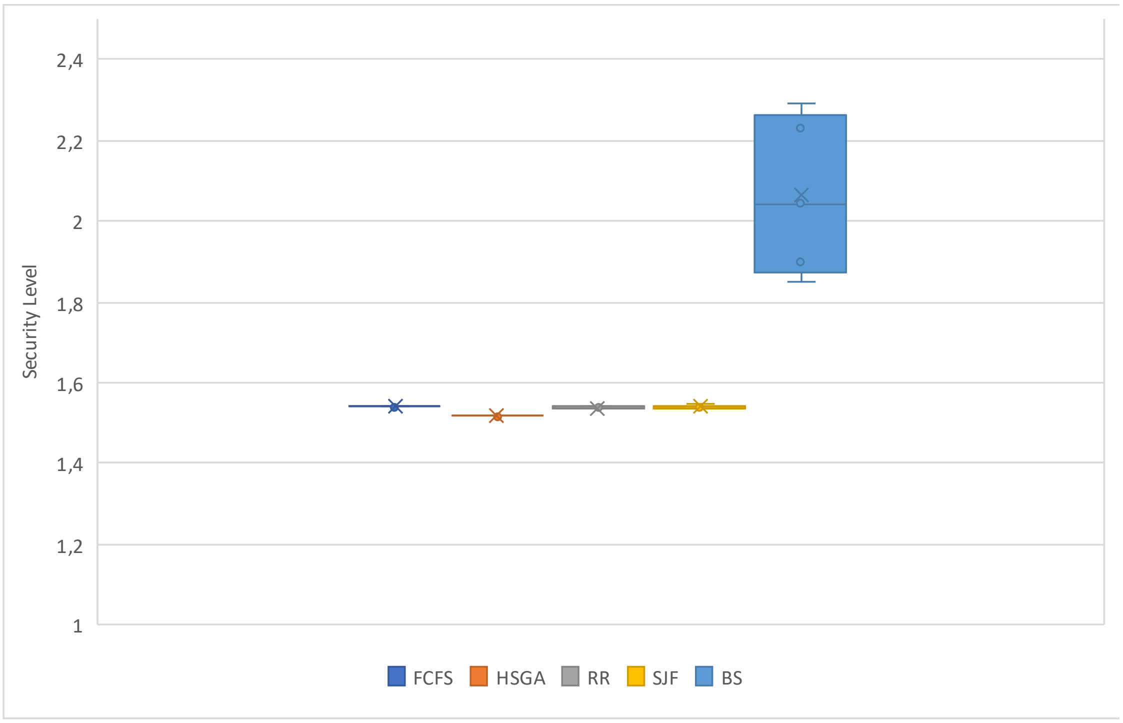

Fig. 7.7. Exp. 6 - evaluation of the model performance using security level - 128 VM and 4096 tasks

No.	Expected SL	Size of the BC network	MTC	Initial value of BW_t	Scheduler	$P^{failure}$	P^{fake}	$P^{hacking}$	Security Level
Exp. 7	1.5	16	8	2333316	FCFS	0.484	0.5	0.5	1.516
					HSGA	0.485	0.5	0.5	1.515
					RR	0.482	0.5	0.5	1.518
					SJF	0.482	0.5	0.5	1.518
					BS	0.484	0.1	0.325	2.091
Exp. 8	2	16	8	2333316	FCFS	0.489	0.5	0.5	✗
					HSGA	0.484	0.5	0.5	✗
					RR	0.484	0.5	0.5	✗
					SJF	0.483	0.5	0.5	✗
					BS	0.488	0.1	0.325	2,087
Exp. 9	2	4	2	2333316	BS	0,483	0	0,45	2,067
Exp. 10	2	8	4	2333316	BS	0,485	0,05	0,4	2,065
Exp. 11	2	12	6	2333316	BS	0,492	0,071	0,325	2,112
Exp. 12	2	16	8	4666632	BS	0,486	0,156	0,174	2,184
Exp. 13	2	16	8	6999948	BS	0,479	0,136	0,119	2,266

Table 7.12. Experiments outline - various SL and BC network configuration

In the subsequent experiments, simulations of various blockchain network configurations were carried out to check its impact on BS security level. In each of them, makespan was adopted as a SC for the schedule verification by the BC network. The number of virtual machines was set to 32 and the number of tasks to 1024. The results are shown in Tab. 7.12, together with the individual factors that make up SL. Exp. 7 proves that the security level of the schedule prepared by BS is about 0.5 larger than in other cases. On the other hand, in Exp. 8 the expected value of security level was set to 2 and no schedulers except BS was able to obtain this value. In the experiments 9, 10 and 11, the effect of changing the number of nodes and MTC in the BC configuration on the P^{fake} value was checked. The results demonstrate that the P^{fake} value raises with the number of nodes in the network, which means that for larger networks there is a higher transaction rejection rate before it receives the appropriate number of confirmations. The last two experiments, 12 and 13, show the impact of the chain of blocks size, specifically the BW_t value, on the $P^{hacking}$ factor. It turns out that the more transactions and blocks are in the chain, the lower the $P^{hacking}$ value is, which means that the security level of schedules grows with the length of blockchain and number of transactions placed in it. The highest SL achieved value was 2.266, a very good result compared to the 1.518 obtained by the other modules.

To confirm the significance of the results obtained in this part of experiments, where the security level of the schedule was taken into consideration, the two-tailed Wilcoxon signed-ranks test for paired samples was conducted for Exp. 7; it was also carried out for Exp 1. The schedule with the lowest SL value returned by a single scheduling module was selected as the first sample, and the schedule returned by BS as the second sample. The following null hypothesis was adopted:

Null Hypothesis 2. H_0: *any differences between the value of SL of the schedule returned by single scheduling module and the value SL returned by BS are due to chance*

The results of the test are shown in Table 7.13.

VM and tasks	SL with σ	$W+$	$W-$	W	Wn
32 VM, 1024 tasks	SJF (0.001) & BS (0.005)	0	1176	0	48

Table 7.13. Two-tailed Wilcoxon signed-ranks test for Exp. 7

The critical value for the W Test statistic from the Wilcoxon signed-ranks table for $Wn = 48$ and $\alpha = .05$ is equal to $W_{crit} = 396$, which gives the range $[0, 396]$. Since W statistic of the performed test is in the range $[0, 396]$, the Null Hypothesis 2 is rejected. Hence, it can be concluded that there is a significant difference between the value of SL of the schedule returned by single scheduling module and the value of SL of the schedule returned by BS scheduler.

7.4. Summary

In the experiments, the proposed Stackelberg game was first tested, according to which the schedules sent between nodes in the BC network were verified. The obtained numerical results confirmed its compliance with the assumptions described in Section 6.2.1. It has been proven that considering both the adopted schedule evaluation criterion and the history of a given node and its previous activities in the network (adding transactions), results in making decision whether the schedule is correct or not in appropriate cases. If the node is new in the network, it must first earn a proper reputation. Otherwise, the schedules prepared by it are not taken seriously by other nodes, which ensures a high level of correctness of schedules and the elimination of false schedules with an underestimated value of the evaluation criterion.

In the next stage, BCSchedCloudSim was checked, which was implemented to conduct experiments comparing BS with other available scheduling modules. All elements of the blockchain were described, from the creation of Genesis Block, through transactions up to the final block mining. The results confirmed the compliance of the simulator with the proposed model for determining consensus in the BC network. The average time needed to carry out the game was also measured, and the results confirmed that this was not a system bottleneck. In fact, the time needed to solve the optimization problem was acceptable and had not a major impact on the speed of receiving the requested schedules by TMs.

After verification of the proposed game and the prepared simulator, some experiments were carried out to evaluate the performance of the proposed solution with other currently available ones. Using different schedule evaluation criteria, BS was shown to return the best results in most cases. Sometimes the obtained results were not the best, but always close to that status. This was due to the fact that, in the simulated environment, schedules good enough to avoid rejection by the network appeared faster than the best ones proposed a little later. The larger the network, the more time it takes to obtain the right amount of transaction confirmations. Hence, the number of such cases is reduced with the increase of the number of nodes in the network. This indicates that the larger the BC network is, the better results are returned. Apart from that, security level criterion defined in this dissertation came under close scrutiny. It included, among others, the assessment of the parameters of machine security on which tasks are executed and the possibility of falsification of schedules and data manipulation by unauthorized entities. $P^{failure}$, P^{fake} and $P^{hacking}$ factors were implemented on the currently available scheduling modules and the proposed blockchain scheduler. In the experiments, the value of expected security level, defined as TMs in requests which are then processed by nodes, was set to 1.5 and then to 2.0. FCFS, HSGA, RR and SJF modules were able to return the schedule with the appropriate security level only in the first case, when the expected SL was 1.5. The SL value returned by BS in each experiment was well above the value returned by the competitive modules. Finally, the influence of the blockchain network configuration, i.e. the size of the BC network and MTC on the value of this criterion was verified. It turned out that the P^{fake} factor increased proportionally to the increment in the number of nodes in the BC network, assuming that the MTC was equal to 50% of the number of nodes. On the other hand, $P^{hacking}$ factor decreased with the growing number of transactions stored in blocks. Nevertheless, P^{fake} grew slower than $P^{hacking}$ decreased. It can be, therefore, concluded that there is a positive correlation between the increase of the blockchain and the increase of the value of the SL.

BS meets the assumptions being a core of this thesis statement. It was particularly evident in the Wilcoxon tests carried out for the two most important experiments. The role of the proposed scheduler is to return the correct and very secure schedule; monitoring and execution of the tasks are not considered. The limitation of the applied solution is that it can be used only in the case of independent batch static scheduling, which means that the availability of virtual machines cannot change during the preparation of the schedule. The configuration of virtual machines specified in the request by the Task Manager must remain unchanged, otherwise, the prepared schedule will be useless. In the provided model, only the independent tasks are considered, but nothing stands in the way of expanding it with the additional information on the relationships between tasks and to handle such cases.

Chapter 8

Conclusions

This chapter summarizes the dissertation. First, the thesis research hypothesis is verified, then the final conclusions and future research plans are presented.

8.1. Research hypothesis verification

The main aim of the research presented in this dissertation was to propose and implement a new method for finding the optimal schedule to execute tasks in distributed computational clouds. The proposed method takes into account performing tasks in accordance with the requirements of end-users; the attempt to increase the security of the prepared schedules was also made. The research hypothesis was formulated in Section 1.4. Chapter 2 comprised the research on the current achievements in the field of task scheduling. The most popular approaches currently existing in the literature were presented and compared. A discussion was conducted to discover that although task scheduling is not a new topic, optimization and safety of schedules still need many improvements. In Section 3.3, a new criterion, i. e. security level of the schedule, had been adopted as necessary to define the problem. In Section 3.1 the considered problem was formulated and metrics for model evaluation were given. The most important development in this dissertation is a system model based on blockchain technology. In order to determine the consensus in the implemented BC network, it was defined the new *Proof of Schedule* algorithm based on Stackelberg game. Chapter 6 presented the proposed system model with details on establishing consensus in the network and adding subsequent blocks. It included also a description of the implemented Blockchain Secure Cloud Scheduler Simulator, which was used to conduct the experiments. Chapter 7 contained a description of the conducted experiments and results of two-tailed Wilcoxon signed-ranks tests, which positively verified the research hypothesis. The use of the BC technology proved to force competition between different scheduling modules, which led to the optimal result for the client. In addition, the use of BC network increased the security of the prepared schedule, greatly hindering its falsification or nefarious modification unauthorized entities. The experiments carried out in Section 7.3 have confirmed that the proposed Secure Blockchain Scheduler, in most cases, returns the optimal schedules (taking into account different criteria for evaluation). Optimal schedules are usually those meeting the expectations of end-users, who primarily want to bear the lowest execution costs. However, this is not the only requirement from customers. Sometimes, for legal reasons, they want their tasks to be executed on machines in a specific geographical location. It also happens that due to sensitive data they want their tasks to be processed on machines with a high level of security. In the proposed model, presented in Section 6.1, the Task Manager ensures that such requirements are taken into account. First, it collects them from users. Then, allocating the appropriate resources, it describes them and places in the pool of requests together with the tasks, waiting for preparation of the schedule in accordance with the expected level of security. The SL of the schedule was checked in subsequent experiments and they proved that the use of blockchain technology significantly reduced the probability of receiving a false or manipulated schedule.

The conducted research significantly extends the possibilities of using different scheduling modules by users who need services offered by cloud service providers. There are many service providers on the market, but the planning algorithms they use are kept secret. The proposed approach allows them to get benefits from pure preparation of schedules, without disclosing the algorithms used for this and their execution.

8.2. Critical remarks and future work

The research lasted about 4 years. If the dissertation was written today, slightly different assumptions would be made regarding the implementation of the simulator used in the experiment. During the research period, blockchain technology has significantly developed and many blockchain-based solutions facilitating system construction entered the market. Perhaps the most promising of them is Ethereum [49], a platform based on smart contracts [35] for which a dedicated Solidity language [115] was created. Thus, one thing is sure: the implementation of the algorithms described in this dissertation would be much simpler and faster with a system using such a platform.

In the dissertation, only scheduling of independent tasks was considered; scheduling of dependent tasks was not discussed. In further research, the proposed model will be expanded to cover this type of tasks. It will require substantial changes with regard to storing dependencies between tasks in the chain of blocks and their inclusion by the scheduling modules. In the proposed Stackelberg game used to establish consensus in the BC network, one metric, usually a makespan, is taken as the criterion for evaluating the schedule. This game can be extended to take into account several criteria of schedule evaluation at once, for instance, both makespan and flowtime or makespan and economic costs. Further research can also pertain to placing tasks that are to be executed in the blockchain, and not only to their characteristics. This would allow scheduling modules to compete not only in terms of preparation of the schedule but also when it comes to the way they provide the results of their execution. This type of data is sensitive, but blockchain technology provides encryption mechanisms able to successfully protect it against explicit and unwanted publication.

List of Figures

List of Tables

List of Listings

List of Abbreviations

ICT Information and Communication Technology. 31, 32

IT Information Technology. 2, 43

JSON JavaScript Object Notation. 61

LP linear programming. 48

MFLO milion floating point operations. 66, 67, 69, 70, 74, 78

MFLOPS milion floating point operations performed in one second. 69, 77

MTC minimum number of transaction confirmation. 57, 70, 73, 78, 86, 89–91

MTRhv Merkle tree root hash value. 54, 72

NFV Network Function Virtualization. 14, 15, 97

NP-hard nondeterministic polynomial time. 4, 48

P2P peer-to-peer. 35

PaaS Platform as a Service. 2

PBhv previous block hash value. 54

PKN public key of the sender node. 53, 72

PKTM public key of the Trust Manager. 53, 55, 72

PoS Proof of Stake. 39, 40

PoSch Proof of Schedule. 55, 59

PoW Proof of Work. 36, 39–41

PSO Particle Swarm Optimization. 5, 12, 17

QoS Quality of Service. 3, 12–14, 22

RR Round Robin. 76, 85, 86, 88, 90

SaaS Software as a Service. 2

SC scheduling criterion. 53, 65, 72, 79, 84, 85, 89

SCAS security and cost aware scheduling. 17

sd security demand. 16, 18, 50, 78

SJF Shortest Job First. 19, 49, 76, 84–86, 88, 90

SL security level. 26, 28, 29, 50, 52, 54, 76, 79, 85, 86, 88–91, 94, 99

SLA Service Level Agreements. 3

SOA Service-Oriented Architecture. 2

SSE Strong Stackelberg Equilibrium. 47

Tim Timestamp. 54

tl trust level. 16, 18, 50, 77

TM Task Manager. 50, 52, 54, 55, 90

VM virtual machine. 19, 50, 77–79

VNF Virtualized Network Function. 14

W Test statistic of Wilcoxon test. 84, 85, 89

W+ positive sum of Wilcoxon test. 84, 85, 89

W- negative sum of Wilcoxon test. 84, 85, 89

wl workload. 50, 78

Wn number of samples in Wilcoxon test where difference was not equal to 0. 84, 85, 89

WSE Weak Stackelberg Equilibrium. 47

Bibliography

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”. In: *FUTURE GENERATION COMPUTER SYSTEMS-THE INTERNATIONAL JOURNAL OF ESCIENCE* 25.6 (2009), 599–616. ISSN: 0167-739X. DOI: *{10.1016/j.future.2008.12.001}*.

[2] R. Guharoy, S. Sur, S. Rakshit, S. Kumar, A. Ahmed, S. Chakborty, S. Dutta, and M. Srivastava. “A theoretical and detail approach on grid computing a review on grid computing applications”. In: *2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON)*. 2017, pp. 142–146. DOI: *10.1109/IEMECON.2017.8079578*.

[3] M. G. Gagniuc. “Virtual Machines Technologies”. In: *Proc. of 9th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS*. Suceava, Romania, 2008, pp. 200–205.

[4] D. Grzonka, M. Szczygiel, Bernasiewicz A., Wilczyński A., and Liszka M. “Short Analysis Of Implementation And Resource Utilization For The Openstack Cloud Computing Platform”. In: *ECMS 2015 Proceedings edited by: Valeri M. Mladenov, Petia Georgieva, Grisha Spasov, Galidiya Petrova European Council for Modeling and Simulation*. Albena (Varna), Bulgaria, 2015. DOI: *{10.7148/2015}*.

[5] Andrzej Wilczyński and Joanna Kołodziej. “Virtualization Model for Processing of the Sensitive Mobile Data”. In: *Modeling and Simulation in HPC and Cloud Systems*. Ed. by Joanna Kołodziej, Florin Pop, and Ciprian Dobre. Cham: Springer International Publishing, 2018, pp. 121–133. ISBN: 978-3-319-73767-6. DOI: *10.1007/978-3-319-73767-6_7*.

[6] Inderveer Chana and Tarandeep Kaur. “Delivering IT as A Utility- A Systematic Review”. In: *CoRR* abs/1306.1639 (2013). arXiv: *1306.1639*.

[7] R. Perrey and M. Lycett. “Service-oriented architecture”. In: *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.* 2003, pp. 116–119. DOI: *10.1109/SAINTW.2003.1210138*.

[8] *Global Infrastructure*. 2020. URL: *https : / / aws . amazon . com / about - aws / global - infrastructure/* (visited on 2020-02-10).

[9] *Discover our data center locations*. 2020. URL: *https://www.google.com/about/datacenters/locations/* (visited on 2020-02-10).

[10] Anupama Prasanth. "Cloud Computing Services: A Survey". In: *International Journal of Computer Applications* 46 (May 2012), pp. 975–8887.

[11] Damian Serrano, Sara Bouchenak, Yousri Kouki, Frederic Alvares de Oliveira Jr., Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, and Pierre Sens. "SLA guarantees for cloud services". In: *FUTURE GENERATION COMPUTER SYSTEMS-THE INTERNATIONAL JOURNAL OF ESCIENCE* 54 (2016), 233–246. ISSN: 0167-739X. DOI: *{10.1016/j.future.2015.03.018}*.

[12] Hamid Madni, Abd Latiff Shafie, Mohammed Abdullahi, Shafi'i Abdulhamid, and Mohammed Usman. "Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment". In: *PLoS ONE* 12 (May 2017), e0176321. DOI: *10.1371/journal.pone.0176321*.

[13] Amid Khatibi Bardsiri and Seyyed Mohsen Hashemi. "QoS Metrics for Cloud Computing Services Evaluation". In: *International Journal of Intelligent Systems and Applications* 6 (Nov. 2014), pp. 27–33. DOI: *10.5815/ijisa.2014.12.04*.

[14] Helen D. Karatza. *Performance Evaluation and Analysis of Large Scale Distributed Systems: Issues, Trends, Problems and Solutions*. URL: *http://chipset-cost.eu/wp-content/uploads/2016/05/EK-SummerSchool-Bucharest-Sep2016.pdf*. Sept. 2016.

[15] Michela Taufer and Arnold Rosenberg. "Scheduling DAG-based workflows on single cloud instances: High-performance and cost effectiveness with a static scheduler". In: *International Journal of High Performance Computing Applications* 31 (July 2015). DOI: *10.1177/1094342015594518*.

[16] Ruby Annette, Aisha Banu, and Shriram Shriram. "A Taxonomy and Survey of Scheduling Algorithms in Cloud: Based on task dependency". In: *International Journal of Computer Applications* 82 (Nov. 2013), pp. 20–26. DOI: *10.5120/14240-2389*.

[17] Mala Kalra and Sarbjeet Singh. "A review of metaheuristic scheduling techniques in cloud computing". In: *Egyptian Informatics Journal* 16.3 (2015), pp. 275 –295. ISSN: 1110-8665. DOI: *https://doi.org/10.1016/j.eij.2015.07.001*.

[18] Kashif Hussain, Mohd Salleh, Shi Cheng, and Yuhui Shi. "Metaheuristic research: a comprehensive survey". In: *Artificial Intelligence Review* (Jan. 2018). DOI: *10.1007/s10462-017-9605-z*.

[19] AR. Arunarani, D. Manjula, and Vijayan Sugumaran. “Task scheduling techniques in cloud computing: A literature survey”. In: *Future Generation Computer Systems* 91 (2019), pp. 407 –415. ISSN: 0167-739X. DOI: *https://doi.org/10.1016/j.future.2018.09.014*.

[20] Andrzej Wilczyński and Adrian Widłak. “Blockchain Networks – Security Aspects and Consensus Models”. In: *Journal of Telecommunications and Information Technology* 2 (July 2019), pp. 46–52. DOI: *10.26636/jtit.2019.132019*.

[21] Andrzej Wilczyński, Agnieszka Jakóbik, and Joanna Kołodziej. “Stackelberg Security Games: Models, Applications and Computational Aspects”. In: *Journal of Telecommunications and Information Technology* 3 (Sept. 2016), pp. 70–79.

[22] Maria Alejandra Rodriguez and Rajkumar Buyya. “A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments”. In: *Concurrency and Computation: Practice and Experience* 29 (2017).

[23] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. “A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments”. In: *2010 24TH IEEE INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS (AINA)*. International Conference on Advanced Information Networking and Applications. 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), Perth, AUSTRALIA, APR 20-23, 2010. IEEE; IEEE Comp Soc; IEEE Tech Comm Distributed Proc (TCDP); IEEE Ind Elect Soc Tech Comm Ind Informat (IES TCII). 2010, 400–407. ISBN: 978-0-7695-4018-4. DOI: *{10.1109/AINA.2010.31}*.

[24] M. Zhang, Y. Yang, Z. Mi, and 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*.

[25] B. Kezia Rani, B. Padmaja Rani, and A. Vinaya Babu. “Cloud Computing and Inter-Clouds – Types, Topologies and Research Issues”. In: *Procedia Computer Science* 50 (2015). Big Data, Cloud and Computing Challenges, pp. 24 –29. ISSN: 1877-0509. DOI: *https://doi.org/10.1016/j.procs.2015.04.006*.

[26] Virajith Jalaparti, Giang D. Nguyen, Indranil Gupta, and Matthew Caesar. “Cloud Resource Allocation Games”. In: (July 2019).

[27] Agnieszka Jakóbik and Andrzej Wilczyński. “Using Polymatrix Extensive Stackelberg Games in Security – Aware Resource Allocation and Task Scheduling in Computational Clouds”. In: *Journal of Telecommunications and Information Technology* 2017 (Mar. 2017).

[28] Saurabh Kumar Garg, Srikumar Venugopal, James Broberg, and Rajkumar Buyya. “Double auction-inspired meta-scheduling of parallel applications on global grids”. In: *Journal of Parallel and Distributed Computing* 73.4 (2013), pp. 450 –464. ISSN: 0743-7315. DOI: *https://doi.org/10.1016/j.jpdc.2012.09.012*.

[29] W. Borjigin, K. Ota, and M. Dong. “In Broker We Trust: A Double-Auction Approach for Resource Allocation in NFV Markets”. In: *IEEE Transactions on Network and Service Management* 15.4 (2018), pp. 1322–1333. ISSN: 1932-4537. DOI: *10.1109/TNSM.2018.2882535*.

[30] Joanna Kołodziej and Fatos Xhafa. “Integration of task abortion and security requirements in GA-based meta-heuristics for independent batch grid scheduling”. In: *Computers & Mathematics with Applications* 63.2 (2012). Advances in context, cognitive, and secure computing, pp. 350 –364. ISSN: 0898-1221. DOI: *https://doi.org/10.1016/j.camwa.2011.07.038*.

[31] Joanna Kołodziej. *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems"*. Sept. 2012. ISBN: 978-3-642-28970-5.

[32] Zhongjin Li, Jidong Ge, Hongji Yang, Liguo Huang, Haiyang Hu, Hao Hu, and Bin Luo. “A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds”. In: *Future Generation Computer Systems* 65 (2016). Special Issue on Big Data in the Cloud, pp. 140 –152. ISSN: 0167-739X. DOI: *https://doi.org/10.1016/j.future.2015.12.014*.

[33] Agnieszka Jakóbik, Daniel Grzonka, and Francesco Palmieri. “Non-deterministic security driven meta scheduler for distributed cloud organizations”. In: *Simulation Modelling Practice and Theory* 76 (2017). High-Performance Modelling and Simulation for Big Data Applications, pp. 67 –81. ISSN: 1569-190X. DOI: *https://doi.org/10.1016/j.simpat.2016.10.011*.

[34] Fatema Akbar Lokhandwala. “A Heuristic Approach to Improve Task Scheduling in Cloud Computing using Blockchain technology”. MA thesis. Dublin, National College of Ireland.

[35] Shuai Wang, Liwei Ouyang, Yong Yuan, Xiaochun Ni, Xuan Han, and Fei-Yue Wang. “Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49 (Feb. 2019), pp. 2266–2277. DOI: *10.1109/TSMC.2019.2895123*.

[36] Zhen Hong, Zehua Wang, Wei Cai, and Victor C. M. Leung. “Blockchain-Empowered Fair Computational Resource Sharing System in the D2D Network”. In: *Future Internet* 9 (2017), p. 85.

[37] Udit Narayana Kar and Debarshi Kumar Sanyal. “An overview of device-to-device communication in cellular networks”. In: *ICT Express* 4.4 (2018), pp. 203 –208. ISSN: 2405-9595. DOI: *https://doi.org/10.1016/j.icte.2017.08.002*.

[38] Georgios L. Stavrinides and Helen D. Karatza. “Scheduling real-time bag-of-tasks applications with approximate computations in SaaS clouds”. In: *Concurrency and Computation: Practice and Experience* 0.0 (). e4208 cpe.4208, e4208. DOI: *10.1002/cpe.4208*. eprint: *https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4208*.

[39] Steven Hotovy. “Workload evolution on the Cornell Theory Center IBM SP2”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror G. Feitelson and Larry Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 27–40.

[40] Kaivalya M. Dixit. “The SPEC benchmarks”. In: *Parallel Computing* 17 (Dec. 1991), pp. 1195–1209. DOI: *10.1016/S0167-8191(05)80033-X*.

[41] Dirk Eddelbuettel and Debian. “Benchmarking Single- and Multi-Core BLAS Implementations and GPUs for use with R”. In: 2010.

[42] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. “The LINPACK Benchmark: past, present and future”. In: *Concurrency and Computation: Practice and Experience* 15 (2003), pp. 803–820.

[43] Andrzej Wilczyński, Joanna Kołodziej, and Paweł Szynkiewicz. “Security aspects in blockchain-based scheduling in cloud computing”. In: *Sensors* (in review).

[44] Joanna Kołodziej and Andrzej Wilczyński. “Blockchain Secure Cloud: a New Generation Integrated Cloud and Blockchain Platforms – General Concepts and Challenges”. In: *EUROPEAN CYBERSECURITY JOURNAL* 4.2 (July 2018).

[45] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009).

[46] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang. “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends”. In: *2017 IEEE International Congress on Big Data (BigData Congress)*. 2017, pp. 557–564. DOI: *10.1109/BigDataCongress.2017.85*.

[47] Paolo Tasca and Claudio Tessone. “A Taxonomy of Blockchain Technologies: Principles of Identification and Classification”. In: *Ledger* 4.0 (2019). ISSN: 2379-5980. DOI: *10.5195/ledger.2019.140*.

[48] Iuon-Chang Lin and Tzu-Chun Liao. “A Survey of Blockchain Security Issues and Challenges”. In: *I. J. Network Security* 19 (2017), pp. 653–659.

[49] D. Vujičić, D. Jagodić, and S. Ranđić. “Blockchain technology, bitcoin, and Ethereum: A brief overview”. In: *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*. 2018, pp. 1–6. DOI: *10.1109/INFOTEH.2018.8345547*.

[50] Alan Cohn Michael J.W. Rennock and Jared Butcher. “Blockchain technology and regulatory investigations”. In: *Practical Law* (2018).

[51] Garrick Hileman and Michel Rauchs. *Global Cryptocurrency Benchmarking Study*. Cambridge Centre for Alternative Finance, Cambridge Judge Business School, University of Cambridge, 2017.

[52] *State of the DApps — Ranking the Best Ethereum DApps*. *https://www.stateofthedapps.com/rankings/platform/ethereum*. Oct. 2019.

[53] Choon Hoong Ding, Sarana Nutanong, and Rajkumar Buyya. “P2P Networks for Content Sharing”. In: *CoRR* cs.DC/0402018 (2004).

[54] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. “Blockchain Technology Overview”. In: *NIST Interagency/Internal Report (NISTIR) - 8202* (2018). DOI: *https://doi.org/10.6028/NIST.IR.8202*.

[55] David Pointcheval. “Asymmetric cryptography and practical security”. In: 2002.

[56] L. Bošnjak, J. Sreš, and B. Brumen. “Brute-force and dictionary attack on hashed real-world passwords”. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, pp. 1161–1166. DOI: *10.23919/MIPRO.2018.8400211*.

[57] Dian Rachmawati, Jos Tarigan, and A B C Ginting. “A comparative study of Message Digest 5(MD5) and SHA256 algorithm”. In: *Journal of Physics: Conference Series* 978 (Mar. 2018), p. 012116. DOI: *10.1088/1742-6596/978/1/012116*.

[58] BENNETT GARNER. *What Is Decred (DCR)? | A Guide on Decentralized Blockchain Governance*. 2019. URL: *https://coincentral.com/decred-lowdown-decentralized-blockchain-governance/* (visited on 2019-06-20).

[59] Daria Dorovskaya. *Review of Peercoin Cryptocurrency: History of Creation and Predictions*. 2018. URL: *https://peercoin.net/* (visited on 2019-06-20).

[60] Archana Prashanth Joshi, Meng Han, and Yan Wang. “A survey on security and privacy issues of blockchain technology”. In: *Mathematical Foundations of Computing* 1 (2018), p. 121. ISSN: A0000-0001. DOI: *10.3934/mfc.2018007*.

[61] J. Kolodziej and F. Xhafa. “A Game-Theoretic and Hybrid Genetic Meta-Heuristics Model for Security-Assured Scheduling of Independent Jobs in Computational Grids”. In: *2010 International Conference on Complex, Intelligent and Software Intensive Systems*. Feb. 2010, pp. 93–100. DOI: *10.1109/CISIS.2010.74*.

[62] Steven Tadelis. *Game Theory: An Introduction*. Economics Books 10001. Princeton University Press, 2012.

[63] Rajiv Sethi and Jörgen Weibull. “WHAT IS...Nash Equilibrium?” In: *Notices of the American Mathematical Society* 63 (May 2016), pp. 526–528. DOI: *10.1090/noti1375*.

[64] Zhou Feng. “On the Nash Equilibrium in Game Theory- A Learning Report for Optimization Method Class”. In: (Nov. 2018).

[65] John F. Nash. “Equilibrium Points in N-Person Games”. In: *Proceedings of the National Academy of Sciences of the United States of America* 36 (Feb. 1950), pp. 48–9.

[66] Bellal Bhuiyan. “An Overview of Game Theory and Some Applications”. In: *Philosophy and Progress* 59.1-2 (2018), pp. 111–128. DOI: *10.3329/pp.v59i1-2.36683*.

[67] Eb’rahim DUROSIMI. *Game Theory a study of strategic decision maki*. 2019. URL: *https://ujuzi.pressbooks.com/chapter/chapter-3-types-of-games/* (visited on 2019-06-20).

[68] Zaki Wahhaj Jean-Philippe Platteau. *Handbook of the Economics of Art and Culture*. 2014.

[69] Christos Papadimitriou. *The Complexity of Computing Equilibria in Handbook of Game Theory with Economic Applications*. 2015.

[70] *Extensive Form Games*. 2019. URL: *http://economics.fundamentalfinance.com/game-theory/extensive-form-game-theory.php* (visited on 2019-07-10).

[71] Hasan Adeeb Jafri. *Distinguish between simultaneous and sequential games*. 2018. URL: *https://www.slideshare.net/HasanAdeebJafri/distinguish-between-simultaneous-and-sequential-games* (visited on 2019-07-12).

[72] John Nash. “Two-Person Cooperative Games”. In: *Econometrica* 21.1 (1953), pp. 128–140. ISSN: 00129682, 14680262.

[73] John Nash. “Non-Cooperative Games”. In: *Annals of Mathematics* 54.2 (1951), pp. 286–295. ISSN: 0003486X.

[74] Yasuhito Tanaka. “On zero-sum game formulation of non zero-sum game”. In: *arXiv e-prints*, arXiv:1809.02984 (2018), arXiv:1809.02984. arXiv: *1809.02984* `[math.OC]`.

[75] Sergiu Hart. “Nonzero-Sum Two-Person Repeated Games with Incomplete Information”. In: *Mathematics of Operations Research* 10.1 (1985), pp. 117–153. ISSN: 0364765X, 15265471.

[76] Mark Fey. "Symmetric games with only asymmetric equilibria". In: *Games and Economic Behavior* 75.1 (2012), pp. 424 –427. ISSN: 0899-8256. DOI: *https://doi.org/10.1016/j.geb.2011.09.008*.

[77] Bernhard von Stengel. "Leadership with Commitment to Mixed Strategies". In: (Mar. 2004).

[78] Anton Frantsev, Ankur Sinha, and Pekka Malo. "Finding Optimal Strategies in Multi-period Stackelberg Games Using an Evolutionary Framework". In: *IFAC Proceedings Volumes* 45.25 (2012). 15th IFAC Workshop on Control Applications of Optimization, pp. 33 –38. ISSN: 1474-6670. DOI: *https://doi.org/10.3182/20120913-4-IT-4027.00038*.

[79] Jiarui Gan. "Minimum Support Size of the Defender ' s Strong Stackelberg Equilibrium Strategies in Security Games". In: 2013.

[80] Valeriu Ungureanu. *Pareto-Nash-Stackelberg Game and Control Theory*. Mar. 2018. ISBN: 978-3-319-75150-4. DOI: *10.1007/978-3-319-75151-1*.

[81] Vincent Conitzer and Tuomas Sandholm. "Computing the Optimal Strategy to Commit to". In: *Proceedings of the 7th ACM Conference on Electronic Commerce*. EC '06. ACM, 2006, pp. 82–90. ISBN: 1-59593-236-4. DOI: *10.1145/1134707.1134717*.

[82] Irina Sokolinskaya and Leonid Sokolinsky. "On the Solution of Linear Programming Problems in the Age of Big Data". In: June 2017, pp. 86–100. DOI: *10.1007/978-3-319-67035-5_7*.

[83] George B. Dantzig. "A History of Scientific Computing". In: ed. by Stephen G. Nash. New York, NY, USA: ACM, 1990. Chap. Origins of the Simplex Method, pp. 141–151. ISBN: 0-201-50814-1. DOI: *10.1145/87252.88081*.

[84] Aleksandar Velinov and Vlado Gicev. "Practical application of simplex method for solving linear programming problems". In: (Aug. 2018).

[85] *Scheduling Amazon ECS Tasks*. 2019. URL: *https://docs.aws.amazon.com/AmazonECS/latest/developerguide/scheduling_tasks.html* (visited on 2019-12-20).

[86] Nevila Xoxa, Marjo Zotaj, Igli Tafa, and Julian Fejzaj. "Simulation of First Come First Served (FCFS) and Shortest Job First (SJF) Algorithms". In: 2014.

[87] Subrata Chowdhury. "Survey on various Scheduling Algorithms". In: *"Imperial Journal of Interdisciplinary Research (IJIR)* 3 (Dec. 2018), p. 4.

[88] *Amazon Web Services (AWS) - Cloud Computing Services*. 2019. URL: *https://aws.amazon.com/* (visited on 2019-12-18).

[89] *Cloud Computing Services - Microsoft Azure*. 2019. URL: *https://azure.microsoft.com/* (visited on 2019-12-17).

[90] Andrzej Wilczyński and Joanna Kołodziej. “Modelling and Simulation of Security-aware Task Scheduling in Cloud Computing Based on Blockchain Technology”. In: *Simulation Modelling Practice and Theory* (2019), p. 102038. ISSN: 1569-190X. DOI: *https://doi.org/10.1016/j.simpat.2019.102038*.

[91] James E. Reeb and Scott Allen Leavengood. “Using the simplex method to solve linear programming maximization problems”. In: 1998.

[92] *What are Blockchain Confirmations? https://www.ethos.io/what-are-blockchain-confirmations*. Nov. 2019.

[93] *51% Attack. https://www.investopedia.com/terms/1/51-attack.asp*. Oct. 2019.

[94] Daniel Davis Wood. “ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER”. In: 2014.

[95] Tim Lindholm and Frank Yellin. “The Java Virtual Machine Specification”. In: 1996.

[96] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. “Apache Maven”. In: 2010.

[97] Jan Kotek. *MapDB Release 2.0*. 2016. URL: *http://www.mapdb.org/down/mapdb-manual-20.pdf* (visited on 2019-08-20).

[98] Limi Kalita. “Socket Programming”. In: *International Journal of Computer Science and Information Technologies* 5 (2014), pp. 4802–4807.

[99] Google. *Gson. https://github.com/google/gson*. 2018.

[100] The Apache Software Foundation. *Commons Math: The Apache Commons Mathematics Library. https://commons.apache.org/proper/commons-math/*. 2016.

[101] *Centralization in Proof of Stake. https://cryptocurrencyhub.io/centralization-in-proof-of-stake-96f6605c0c13*. Mar. 2018.

[102] Don Johnson, Alfred Menezes, and Scott Vanstone. “The Elliptic Curve Digital Signature Algorithm (ECDSA)”. In: *International Journal of Information Security* 1.1 (2001), pp. 36–63. ISSN: 1615-5262. DOI: *10.1007/s102070100002*.

[103] Abhijit A. Rajguru and S. S. Apte. “A Performance Analysis of Task Scheduling Algorithms using Qualitative Parameters”. In: 2013.

[104] Arash Ghorbannia Delavar and Yalda Aryan. “HSGA: a hybrid heuristic algorithm for workflow scheduling in cloud systems”. In: *Cluster Computing* 17.1 (2014), pp. 129–137. ISSN: 1573-7543. DOI: *10.1007/s10586-013-0275-6*.

[105] Michael Fahmy. *CloudSim Task Allocation and Scheduling. https://github.com/michaelfahmy/cloudsim-task-scheduling*. 2017.

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

[107] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". In: *Softw., Pract. Exper.* 41 (2011), pp. 23–50.

[108] *Geekbench 5 - Cross-Platform Benchmark*. 2019. URL: *https://www.geekbench.com/* (visited on 2019-09-20).

[109] *PassMark - CPU Benchmarks - List of Benchmarked CPUs*. 2019. URL: *https://www.cpubenchmark.net/cpu_list.php* (visited on 2019-09-20).

[110] *CPU performance*. 2019. URL: *https://setiathome.berkeley.edu/cpu_list.php* (visited on 2019-09-20).

[111] *EC2 Instance Pricing – Amazon Web Services (AWS)*. 2019. URL: *https://aws.amazon.com/ec2/pricing/on-demand/* (visited on 2019-10-05).

[112] Andrew C. Leon. "3.12 - Descriptive and Inferential Statistics". In: *Comprehensive Clinical Psychology*. Ed. by Alan S. Bellack and Michel Hersen. Oxford: Pergamon, 1998, pp. 243 –285. ISBN: 978-0-08-042707-2. DOI: *https://doi.org/10.1016/B0080-4270(73)00264-9*.

[113] *Wilcoxon Signed-Ranks Table | Real Statistics Using Excel. http://www.real-statistics.com/statistics-tables/wilcoxon-signed-ranks-table/.* 2019.

[114] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles A. Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. "Exploring the Attack Surface of Blockchain: A Systematic Overview". In: *ArXiv* abs/1904.03487 (2019).

[115] Chris Dannen. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. 1st. Berkely, CA, USA: Apress, 2017. ISBN: 1484225341, 9781484225349.

Appendix A

Scientific Curriculum

Academic degrees obtained

- M.Eng., 2015, dissertation title: *Multimedia database management system in combination with music player*, supervisor: Stanisława Plichta, Ph.D.
- B.Eng., 2013, dissertation title: *Web application with database supporting the functioning of a computer service*, supervisor: Lech Jamroż, Ph.D.

Publications

- A. Wilczyński, J. Kołodziej, P. Szynkiewicz "Security aspects in blockchain-based scheduling in cloud computing", *Sensors*, In review.
  IF=3.031, MNiSW=100
- A. Wilczyński, J. Kołodziej, "Modelling and Simulation of Security-aware Task Scheduling in Cloud Computing Based on Blockchain Technology", *Simulation Modelling Practice and Theory*, Volume 99, 2020.
  IF=2.426, MNiSW=100
- A. Wilczyński, A. Widłak, "Blockchain Networks – Security Aspects and Consensus Models", *Journal of Telecommunications and Information Technology*, Volume 2, 2019, Pages 46-52.
  MNiSW=40
- J. Kołodziej, A. Wilczyński, D. Fernandez-Cerero, A. Fernandez-Montes, "Blockchain Secure Cloud: a New Generation Integrated Cloud and Blockchain Platforms – General Concepts and Challenges", *European Cybersecurity Journal*, Volume 4, Issue 2, 2018, Pages 28-35.
  MNiSW=0

- A. Wilczyński, J. Kołodziej, "Virtualization Model for Processing of the Sensitive Mobile Data", *In: Kołodziej J., Pop F., Dobre C. (eds) Modeling and Simulation in HPC and Cloud Systems. Studies in Big Data*, vol 36. Springer, Cham, 2018, Pages 121-133.
  MNiSW=5

- A. Jakóbik, A. Wilczyński, "Using Polymatrix Extensive Stackelberg Games in Security – Aware Resource Allocation and Task Scheduling in Computational Clouds", *Journal of Telecommunications and Information Technology*, Volume 1, 2017, Pages 71-80.
  MNiSW=12

- A. Wilczyński, A. Jakóbik, J. Kołodziej, "Stackelberg Security Games: Models, Applications and Computational Aspects", *Journal of Telecommunications and Information Technology*, Volume 3, 2016, Pages 70-79.
  MNiSW=12

- A. Wilczyński, J. Kołodziej, "Cloud Implementation of Agent-Based Simulation Model in Evacuation Scenarios", *30th European Conference on Modelling and Simulation (ECMS)*, May 31st - June 03rd, Regensburg, Germany, 2016, Pages 641-647.
  MNiSW=15

- D. Grzonka, M. Szczygieł, A. Bernasiewicz, A. Wilczyński, M. Liszka, "Short Analysis of Implementation and Resource Utilization for the Openstack Cloud Computing Platform", *29th European Conference on Modelling and Simulation (ECMS)*, May 26th – 29th, Albena (Varna), Bulgaria, 2015, Pages 608-614.
  MNiSW=12

Participation in research projects

- ICT COST Action IC1406 High-Performance Modelling and Simulation for Big Data Applications (cHiPSet), COST Action, coordinator: Professor Joanna Kołodziej, Ph.D., D.Sc., 2016-2019.

Reviewer activity

- The 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, 2020.

- IEEE Transactions on Services Computing, 2018-2020.
  IF=5.707

- European Conference on Modelling and Simulation (ECMS), 2016-2017.

Prizes and awards

- Team award of rector for organizational achievements in 2018, 2019.
- Nomination for Special Student Award for paper "Cloud Implementation of Agent-Based Simulation Model in Evacuation Scenarios" presented at the *30th European Conference on Modelling and Simulation*, 2016.

Participation in conferences

- BlockchainFiesta, November 16th, 2018, Cracow, Poland.
- 32nd European Conference on Modelling and Simulation, May 22nd - May 25th, 2018, Wilhelmshaven, Germany.
- ICT Proposers' Day 2017, November 9th-10th, 2017, Budapest, Hungary.
- Rola technik cyfrowych w prognozowaniu gwałtownych zjawisk atmosferycznych, August 26th, 2017, Cracow, Poland.
- 31st European Conference on Modelling and Simulation, May 23rd - May 26, 2017, Budapest, Hungary.
- 30th European Conference on Modelling and Simulation, May 31st - June 03rd, 2016, Regensburg, Germany, presentation of the paper "Cloud Implementation of Agent-Based Simulation Model in Evacuation Scenarios".

Organizational activity

- IPC member of *30th, 31st, 32nd, 33rd, 34th European Conference on Modelling and Simulation (ECMS)*, 2016-2020.
- Organizing Committee Member and Scientific Committee Member of conference "Rola technik cyfrowych w prognozowaniu gwałtownych zjawisk atmosferycznych", Cracow University of Technology, 2017.
- Organizing visits of students in commercial companies as part of the POWER project, Cracow University of Technology, 2018-2020.

Teaching activity

- Mobile Technologies And Programming, Cracow University of Technology, 2015–2020.
- IT Project Management, Cracow University of Technology, 2018–2020.
- Object-oriented Technology 2, AGH University of Science and Technology, 2017–2018.
- Object And Component-oriented Systems, AGH University of Science and Technology, 2017.
- Programming in Java, Cracow University of Technology, 2016–2017.
- Object-oriented Technologies, Cracow University of Technology, 2015–2017.