Application of a Task Stalling Buffer in Distributed Hybrid Cloud Computing

Albertas Jurgelevicius[1, *], Leonidas Sakalauskas[2], Virginijus Marcinkevicius[1]
[1]*Institute of Data Science and Digital Technologies, Vilnius University, Akademijos St. 4, LT-08412 Vilnius, Lithuania*
[2]*Vilnius Gediminas Technical University, Sauletekio Aly. 11, LT-10223 Vilnius, Lithuania*
j.albertas@gmail.com

***Abstract*—The purpose of the research is to create a hybrid cloud platform that performs distributed computing tasks using high-performance servers and volunteer computing resources. The proposed platform uses a new task scheduling method, which is also presented in this paper. It uses a task stalling buffer to manage workload among the two grids without any additional information about the tasks. Since efficient task scheduling in these distributed systems is the actual problem, the system reliability issue is solved using a hybrid cloud architecture when both high-performance servers and volunteer computing resources are combined. The results of the experiment showed that the proposed solution solves the problem of balancing workload between two grids better than the standard scheduling algorithm. Computer study and experiments also showed that the proposed hybrid cloud tasks scheduling method with a task stalling buffer reduces up to 47.3 % of total task execution time. The outcome of this paper provides a background for future research on a task stalling buffer in hybrid cloud computing.**

***Index Terms*—Computer buffers; Distributed computing; Load flow control; Scheduling algorithms.**

I. INTRODUCTION

Data has become one of the most critical and valued assets in today's fast-paced business world. Organisations collect and use data to evaluate key performance indicators, make informed decisions, and establish goals. Useful data can help to find problems, increase business efficiency, find new opportunities, and stay ahead of competitors. Due to the ongoing transformation of industrial manufacturing through digitalisation (Industry 4.0 strategic initiative), data amounts tend to increase [1].

Large companies usually solve hardware capacity problems by upgrading existing or buying new servers and hiring additional staff to maintain the systems. Small and medium businesses typically do not have the financial ability to make such investments. In most cases, smaller companies purchase external grid computing services through various subscription or on-demand pricing schemes. Such services provide secure, scalable storage and compute capacity. Research shows that this makes a more affordable distributed volunteer computing model seem unreliable and too difficult to adopt [2].

The distributed volunteer computing model enables volunteers to donate their own computing resources to projects. Although this model can reduce service costs, it also lacks reliability. The required number of volunteers may not always be available or volunteers may not always complete the assigned tasks. Furthermore, protection of personal data can cause additional problems. Personal data privacy issues are especially relevant now since as of 25 May 2018 companies and organizations have had to comply with GDPR (General Data Protection Regulation) rules within the European Union.

As a result, we now encounter the concept of distributed cloud, which is one of Gartner's top 10 strategic technology trends for 2020 [3] and 2021 [4]. The distributed cloud is the distribution of public cloud services to different physical locations. Although such services are outside physical data centres, they are still controlled and supervised by the provider. This technology offers the benefits of a public cloud service alongside the benefits of a local private cloud.

Despite the benefits, a distributed hybrid cloud computing model presents various challenges. One such problem is task scheduling and execution. It is essential to maintain optimal workload between the grids. However, existing well-known hierarchical and non-hierarchical task scheduling algorithms, reviewed in Section IV, cannot balance the workload without any additional information about the tasks (such as task size, quantity, and incoming task rate). As reviewed in [5], existing hybrid distributed computing platforms ([6]–[18]) require preliminary data on the number of tasks to be performed, the execution time for each task, or the number of computing resources available. A task execution schedule is then created using these data. However, in heterogeneous distributed computing networks, these parameters are either constantly changing or no such information is available.

This paper presents a hybrid cloud platform that performs batch processing tasks using internal servers (or cloud computing services) and personal computers. Our proposed platform is different from the currently existing solutions ([6]–[18]), as it is designed to operate in a heterogeneous environment without simulation results or task replication. Our proposed platform combines public and private

Manuscript received 18 March, 2021; accepted 8 November, 2021.

computing grids into a distributed hybrid cloud. It uses our proposed task scheduling method to manage the workload between the two grids without any additional information about the tasks. We show that service reliability issues (caused by low-performance compute nodes) can be solved using an opportunistic task scheduling algorithm combined with a task stalling buffer. This method prioritises the private cloud for processing tasks and distributes tasks into the heterogeneous public cloud, only if the private cloud resources are exhausted. In a hybrid cloud environment, this approach is called "cloud bursting". In this way, our proposed platform allows companies to reduce service costs and still maintain service reliability.

The rest of the paper is structured as follows. In Section II, we overview and explain the technologies used for our proposed hybrid cloud platform. Section III presents the architecture for our proposed platform. Section IV explores task scheduling algorithms. Sections V and VI present the results of the simulation and platform experiment. In Section VII and Section VIII, we conclude this paper by summarizing the findings and presenting directions for the future.

II. TECHNOLOGIES

This section will introduce the technologies that we selected and used for our proposed platform architecture. We use these particular solutions because they are open source, widely used, and compatible (all support the same software virtualisation solution). However, it is essential to note that other compatible alternatives may also be used.

A. Public Distributed Cloud Computing

The public distributed computing model connects public computers to solve distributed tasks in parallel. This model aims to solve heterogeneous environment issues, allowing new external compute nodes to join the computations. It uses a client-server model, which enables the nodes to provide resources to the project server. This model allows the compute nodes to request the master server for new tasks and send back the results. Public distributed computing approaches can compete with existing cloud computing solutions [19].

There are various public distributed computing solutions: CharityEngine [20], GridMP [21], Xgrid [22], XtremWeb [23]. However, the most widely and actively used solution is called "Berkeley Open Infrastructure for Network Computing" (BOINC) [24]. BOINC is a platform for high-throughput computing on a large scale (thousands or millions of computers). It can run virtualised, in parallel, or for GPU-based applications. Furthermore, it can perform big data mining tasks using consumer devices or company servers [25]. BOINC performs computations only when the CPU is idle. This solution can allow organisations to use the computer resources available from employees of the company without disrupting any ongoing work. Since company employee computer CPUs are idle 99 % of the time [25], this solution may solve the computational resource demand problem.

B. Private Distributed Cloud Computing

The private cloud computing model uses the client-server model and is focused on achieving high internal resource utilisation and performance. Private distributed computing is the preferred model in companies and organisations, as it provides high-quality service, high performance, and ensures data security.

One of such cluster resource management platforms is called "Apache Mesos" [26]. It supports popular frameworks, such as Hadoop [27] and MPI [28]. It can scale up to 50,000 (emulated) nodes and have less than 4 % overhead. Small tasks should be preferred over large ones to minimise time costs caused by unexpected failures. Apache Mesos supports various job schedulers, such as Apache Chronos [29]. Apache Chronos is responsible for running schedule and dependency-based jobs. However, an increasing number of unprocessed tasks may cause the scheduler to crash. We solved this issue by limiting the number of unprocessed tasks to the number of available resources in our Apache Mesos cluster. Finally, it is essential to note that Apache Chronos and Apache Mesos require a trusted network environment, allowing direct interaction between systems without encryption.

C. Software Virtualisation

Software virtualisation is a technology that hides physical system resources from the operating system and helps solve various problems [30]. In heterogeneous environments, software virtualisation allows running the same tasks on multiple computer architectures and different operating systems.

There are many various software virtualisation technologies: Docker [31], Kubernetes [32], Oracle VM VirtualBox [33], QEMU [34], VMware [35], and many others. We will be using Docker and Oracle VM Virtual Box to maintain software compatibility with Apache Mesos, Apache Chronos, and BOINC.

Docker is a set of platform-as-a-service products. It uses OS-level virtualisation and provides means to bundle software into packages called "containers" (more lightweight than virtual machines). Docker allows software applications to run on various computer architectures and operating systems without requiring any changes to the application.

Oracle VM VirtualBox is an application to create, manage, and run virtual machines. It provides hardware-level virtualisation and has more security controls than Docker. However, virtual machines use more computer resources and take more time to start than containers.

Although software virtualisation can solve some security issues in cloud computing, it does not protect against all security threats.

D. Hybrid Distributed Computing

Hybrid distributed computing platforms combine private and public distributed computing clusters. Distributed computing tasks are distributed between private and public computing resources using various task scheduling algorithms. We selected to use BOINC for public distributed computing since it is the most popular and widely supported public computing platform. Even though the BOINC platform supports both Docker and Oracle VM VirtualBox, we used Docker since it requires less resources to operate

and is supported by Apache Mesos. Our proposed task scheduling algorithm will be presented in Section IV.

The next section presents the architecture of our proposed platform.

III. PROPOSED DISTRIBUTED HYBRID CLOUD PLATFORM ARCHITECTURE

As shown in Fig. 1, our proposed distributed hybrid cloud has a two-level hierarchy and contains physically distributed (hierarchical) cooperative schedulers. At the top level, there is a master scheduler that distributes tasks between the lower-level grids. This architecture provides a scalable and resilient core for task execution and gives more control over service quality. We propose using two grids: private (controlled by Apache Mesos) and public (controlled by BOINC) to distribute tasks between the company servers and employee computers. Each grid is managed by a scheduler specifically designed for each environment.

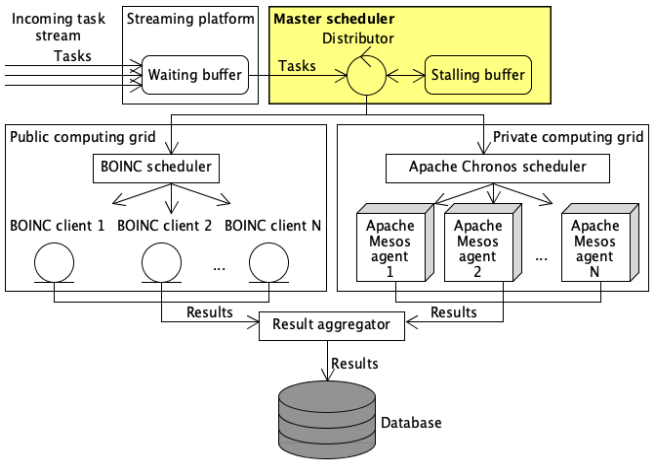

Fig. 1. Proposed distributed hybrid cloud platform architecture with proposed scheduling algorithm (highlighted in yellow).

Our design philosophy has been to push task scheduling to the lower-level grids by controlling which grid should receive the task. Our proposed architecture consists of the following main components: master scheduler, private computing grid, and public computing grid. We also added additional components that are not mandatory but help to illustrate the complete solution:

– *Streaming platform*: stores all new incoming tasks in the waiting buffer until the system accepts the tasks;

– *Result aggregator*: collects and aggregates results from the executed tasks;

– *Database*: used to store the aggregated results.

The master scheduler is the main focus of our research. Figure 1 contains our proposed scheduling method for this architecture and will be explained in Section IV-C. It is the top-level scheduler that distributes tasks to the lower-level schedulers. It consists of the following sub-components:

– *Stalling buffer*: the component that stalls tasks for later processing in the private computing grid (for more details, see Section IV-C);

– *Distributor*: the process that stores new incoming tasks in the stalling buffer and then distributes to the Apache Mesos scheduler whenever the private computing grid has available Apache Mesos agents. If the stalling buffer is full and the public computing grid has available BOINC clients that are idle, then the distributor forwards new incoming tasks to the BOINC scheduler.

Private and public computing grids consist of grid schedulers and clients (or agents) responsible for distributing and executing tasks in each grid. In Section IV, we will review scheduling algorithms to find a suitable algorithm for our top-level (master) scheduler.

It is important to note that our proposed platform may be required to deal with specific data privacy and availability issues in some cases, such as downloading or uploading large amounts of data and processing sensitive information. Data size and privacy issues are well-known and there are various solutions to these problems [36]–[39]. Such solutions could be considered for integration into our proposed distributed hybrid cloud architecture, improving its data security and availability. However, the analysis of these problems is not within the scope of this paper.

Our proposed platform requires incoming distributed computing tasks (Fig. 1) to be defined using the JSON (JavaScript Object Notation) format. Any preferred data format is suitable for defining tasks. However, we used the JSON format since it is well supported and easily readable. Here, the task definition is structured as follows

{"container": "**<task>**", "method": "**<method >**"},

where <*task*> is the name and parameters of the Docker container that contain task execution files (solution based on [40]). <*method*> can define the task scheduling method or specify a specific cluster to execute the task in. This can be used to execute urgent or sensitive personal data tasks using the private cluster. Examples:

– {"container": "ashael/pi 100000", "method": "FIFO"};

– {"container": "ashael/pi 200000", "method": "TSB-static(k=10)"};

– {"container": "-e \"INPUT_FILENAMES=1.json; 2.json\" -v /var/data/:/data -v /var/ out/:/out mrquad/map-reduce", "method ": "mesos"}.

Docker containers are stored in publicly or privately available repositories. This solution allows our platform to operate in a heterogeneous environment. Furthermore, it reduces network traffic load since Docker containers are downloaded only once by the compute nodes instead of distributed each time by the schedulers.

IV. SCHEDULING ALGORITHMS

In this section, we will review existing well-known hierarchical and non-hierarchical task scheduling algorithms that could be suitable for the top-level scheduler. The algorithm must be compatible with our proposed distributed hybrid cloud architecture; thus we will review existing hierarchical task scheduling algorithms that fall under the following classification [41]:

– *Global*. Tasks are executed on multiple compute nodes throughout multiple grids;

– *Dynamic*. Tasks come online dynamically, and task execution costs are unknown;

– *Physically distributed*. Scheduling is done using various distributed schedulers;

– *Cooperative*. Distributed schedulers cooperate to make better scheduling decisions.

Finally, in Section IV-C, we will explore the opportunistic load balancing approach using the proposed hybrid cloud task scheduling method with the task stalling buffer. Although this method is used to distribute tasks in queueing systems with two heterogeneous servers, we propose adopting and applying this method to grid computing. We will show that this method can be used to schedule tasks between two grids without requiring any additional information about the tasks, and thus improve workload balance between the two grids.

A. Hierarchical Scheduling Algorithms

The existing hierarchical scheduling algorithms [41] for task distribution among multiple grids can be summarized using the following four existing solutions, where:

– Tasks are moved from highly loaded clusters to less loaded neighbouring clusters, assuming that task arrival rates will not exceed service rates [42];

– Tasks are divided into subtasks and time estimations are made using simulation results, including possible resource allocation conflicts. Subtasks are assigned to available local grids that can complete executions the fastest [43];

– Tasks are sorted in descending order of their average execution times and assigned to workers. Execution times are estimated using simulation results together with historical data [44];

– Slow running tasks are replicated expecting quicker results from another resource [45].

Existing solutions assume that task arrival rates and service rates for the whole system always remain stable, operate on estimated task execution times, or use task replication. Such assumptions and requirements are also found in other methods such as QoGS [46], which selects the most suitable cloud in the intercloud for task execution. Such methods use a set of weighted coefficients, which are calculated either by user or by executing a set of test tasks to get simulation results. The simulation results are then used to calculate the weighted coefficients automatically. Although such methods are very efficient in their particular use cases, they all require specific information about the tasks or depend on simulation results. Our proposed solution is different from the currently existing solutions, as it is designed to operate in a heterogeneous environment without any simulation results or task replication.

B. Job Schedulers for Distributed Computing

In this section, we review widely adopted independent job schedulers. Some of them are also used for big data processing tasks by Facebook, Yahoo, and Hadoop. There are at least five well-known scheduling methods [47]–[50]:

– *Fair-share* [47], [49]: each job gets an equal amount of resources;

– *First In First Out (FIFO)* [47], [49] - the oldest tasks are executed on the first nodes to become available;

– *Capacity* [47], [49]: resources are allocated to job processing queues used to accept and process new tasks;

– *Longest Approximate Time to End (LATE)* [47]–[49]: replicates tasks that are stuck in slow compute nodes, using such replicated tasks as a backup (reliability is not guaranteed [47]);

– *Round-robin* [50]: runs all the applications from the first job on the first node, all the applications from the second job on the second node, etc.

The only algorithm here capable of distributing the incoming stream of dynamic tasks in a highly heterogeneous environment between two grids is FIFO. It does not require any information about the tasks and the available node capacity. Other well-known task scheduling algorithms, such as Min-min [41], [51], Min-max [41], [51], Minimum Completion Time (MCT) [41], [50], Suffrage algorithm [51], are not applicable since these algorithms require a list of all tasks and nodes in advance. Algorithms such as User Defined Assignment [51] are also not suitable since tasks are assigned in arbitrary order to machines with the best expected execution, regardless of the resource availability.

C. Opportunistic Load Balancing Using a Task Stalling Buffer

According to [52], a task stalling buffer (Fig. 2) improves task execution makespan in queueing systems with two heterogeneous servers. The task stalling buffer reduces slow server load by redirecting more tasks to the fast server. New tasks are added to the stalling buffer if the fast server is busy. If the buffer is full, the slow server receives the task.

We applied this method (Fig. 2) to improve the task distribution between two grids. The purpose of such an approach is similar to [46], as it aims to select the cloud most suitable for task execution. Since we can assume that the private grid will always perform better than the public grid, we can use a task stalling buffer to decrease the number of tasks distributed to the public grid. In this way, we can expect to reduce the execution makespan of tasks and improve the reliability of the service by reducing the number of tasks executed using heterogeneous servers.

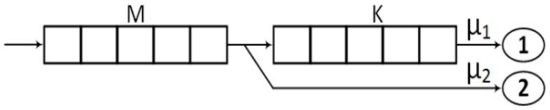

Fig. 2. Scheme of a queueing system with a stalling buffer [52].

Here, M is the buffer size to store new tasks, K is the buffer size for stalling tasks, μ_1 is the efficiency of the fast channel, μ_2 is the efficiency of the slow channel, *1* is the fast channel, and *2* is the slow channel. Then, according to [52], the length of the buffer for stalling the task K may be expressed as follows

$$K \approx \rho(1-q), \tag{1}$$

where ρ is the task execution efficiency ratio between fast and slow channels, and q is the task execution efficiency coefficient:

$$\rho = \frac{\mu_1}{\mu_2}, \tag{2}$$

$$q = \frac{c}{tm\mu_1}, \tag{3}$$

where c is the number of completed tasks, t is the total task execution makespan, m is the number of compute nodes in the fast channel and μ_1 is the fast channel efficiency:

$$\mu_1 = \frac{a_1}{b_1}, \tag{4}$$

$$\mu_2 = \frac{a_2}{b_2}, \tag{5}$$

where a_1 is the number of tasks completed using the fast channel, and b_1 is the time required to complete those tasks; a_2 is the number of tasks performed using the slow channel, and b_2 is the time required to complete those tasks.

It is important to note that according to [52], the task stalling buffer length should be calculated only once. However, due to the heterogeneous nature of our grid environment, we believe that the task stalling buffer could produce better results if re-calculated with each new task received. Thus, for our experiments (presented in Sections V and VI), we will include two variants of the Task Stalling Buffer (TSB) algorithm:

1. *TSB-static*: task stalling buffer length is calculated once;
2. *TSB-dynamic*: task stalling buffer length is re-calculated with each received new task.

V. COMPUTER STUDY

This computer study will test our hypothesis that the proposed hybrid cloud task scheduling method with task stalling buffer improves the task execution makespan compared to FIFO. We will use a virtual environment for our tests, programmed using PHP programming language. Task execution makespan will be estimated using iteration counts (instead of seconds) required to complete all tasks. A virtual environment (unlike the real platform experiments presented in Section VI) will allow us to simulate the infrastructure with more compute nodes and conduct large amounts of experiments in a reasonable time. However, our virtual environment will not simulate the behaviour of private and public cluster schedulers. Furthermore, it will not account for data transfer times and network load variations. This computer study aims to examine the task execution makespan between the following algorithms that would distribute tasks among the two grids using:

– Standard FIFO algorithm;

– *TSB-static(k)*: our proposed hybrid cloud tasks scheduling algorithm with a static length task stalling buffer, where k is the buffer length (buffer length is estimated only once after each grid has executed at least one task);

– *TSB-dynamic*: our proposed hybrid cloud tasks scheduling algorithm with a dynamic length task stalling buffer (buffer length is re-estimated with each new incoming task).

A. Simulation Scenarios

To evaluate the task execution makespan, we ran simulated tasks with established iteration counts for each task to complete. Scenarios were generated before experiments so that each algorithm would be tested using the same conditions. We will use the following annotations:

– *TS*: static size tasks. All tasks are of the same size and are equal to 200 iterations;

– *TD*: dynamic size tasks. The generated task sizes are distributed using the Poisson distribution (λ = 200);

– *STS*: static task stream. Delays between all tasks are equal to 8 iterations;

– *DTS*: dynamic task stream. The generated delays between tasks are distributed using the Poisson distribution (λ = 8).

The following scenarios were used to test each algorithm:

– *TS_STS*: static size tasks (TS), static incoming task stream (STS). The same tasks are supplied to the platform at a regular interval (or delays);

– *TS_DTS*: static size tasks (TS), dynamic incoming task stream (DTS). The same tasks are supplied to the platform at a changing interval (or delays);

– *TD_STS*: dynamic size tasks (TD), static incoming task stream (STS). Changing tasks are supplied to the platform at a regular interval (or delays);

– *TD_DTS*: dynamic size tasks (TD), dynamic incoming task stream (DTS). Changing tasks are supplied to the platform at a changing interval (or delays).

The number of iterations and the delays between tasks were adapted for the number of simulated compute nodes. For best results, the task stalling buffer should not always be empty or full. Otherwise, all tasks would be redirected into the private grid (the system would be underutilised), or our task distribution algorithm would behave exactly like the standard FIFO algorithm.

Each scenario will be executed using every possible task count, ranging from 40 to 400 tasks. Results will be aggregated. The slow channel will be serviced by 16 agents, while 8 agents will serve the fast channel. Slow channel agents will have a 1,000 iteration start penalty and will be set to perform 10 times slower than the agents servicing the fast channel.

B. Simulation Results

Aggregated simulation results are presented in Fig. 3 and Table I.

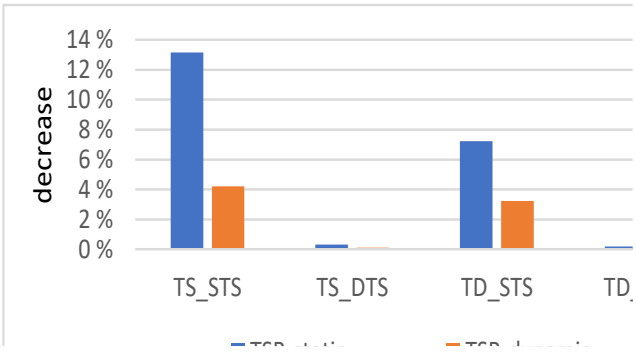

Fig. 3. Task execution makespan decrease compared to FIFO.

The results were obtained using 21,660 simulations employing different scenarios with various task counts. They show that TSB algorithm performs best in TS_STS scenario and achieves up to 13 % improvement compared to the standard FIFO algorithm.

TABLE I. SIMULATION RESULTS. MAKESPAN DECREASE COMPARED TO FIFO.

Algorithm	Scenario	Makespan (iterations)	Makespan decrease
FIFO	TS_STS	6,138	not applicable
	TS_DTS	236,231	not applicable
	TD_STS	5,661	not applicable
	TD_DTS	235,794	not applicable
TSB-static(10)	TS_STS	5,330	13.16 %
	TS_DTS	235,450	0.33 %
	TD_STS	5,252	7.22 %
	TD_DTS	235,371	0.18 %
TSB-dynamic	TS_STS	5,879	4.22 %
	TS_DTS	235,997	0.10 %
	TD_STS	5,478	3.23 %
	TD_DTS	235,583	0.09 %

This allows us to conclude that the task stalling buffer can be applied in hybrid clouds and can outperform the standard FIFO algorithm in all scenarios.

VI. REAL PLATFORM EXPERIMENT

This section will describe the experiment we conducted to test the proposed platform and compare our proposed hybrid cloud tasks scheduling method with static and dynamic length task stalling buffers to FIFO. This experiment aims to examine the task execution makespan between the following algorithms that would distribute tasks among the two grids using:

– Standard FIFO algorithm;

– *TSB-static(k)*: our proposed hybrid cloud tasks scheduling algorithm with a static length task stalling buffer, where *k* is buffer length (buffer length is estimated only once after each grid has executed at least one task);

– *TSB-dynamic*: our proposed hybrid cloud tasks scheduling algorithm with a dynamic length task stalling buffer (buffer length is re-estimated with each new incoming task).

This real platform experiment will further test our hypothesis that the proposed hybrid cloud tasks scheduling method with a task stalling buffer improves task execution makespan compared to FIFO.

A. Experimental Setup

We used two different setups (server setup A and B) to test different environments. Setup A had two separate servers running Docker containers (Fig. 4). The master server was used to control the grids and distribute tasks. The slave server was used to simulate the two grids by running multiple virtual machines representing separate compute nodes. Since both grids were on one single server, we added the upper limits for memory (RAM) and CPU usage using Docker container, VirtualBox image (required and run by BOINC clients), Mesos agent, and BOINC client settings. These limits allowed us to control resource usage and ensure equal resource distribution per task.

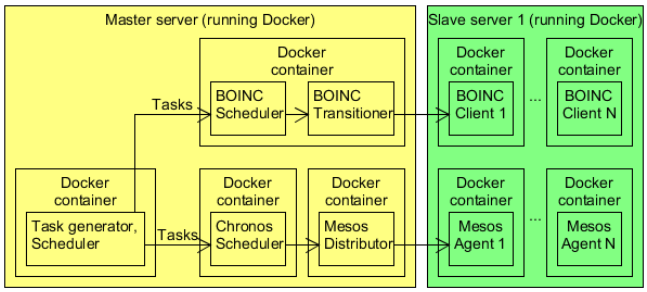

Fig. 4. Server setup A scheme. The master server and one slave server are running on different servers.

The server Setup B is very similar to the server Setup A, except that it uses two slave servers instead of one (Fig. 5). In this way, we separated the two grids and gained additional resources to simulate more compute nodes. In both setups, the number of simulated nodes was limited to the number of cores per server. Using the server Setup A, we simulated two Apache Mesos agents and two BOINC clients. Using the server Setup B, we simulated two Apache Mesos agents and four BOINC clients. These two server setups allowed us to test if adding more compute nodes to the public grid changes the results. In both configurations, we used a task generator to simulate the incoming task stream.

The server hardware specification was as follows:

– *Master*: Intel(R) Xeon(R) 5160 CPU, 2 cores, 3.00 GHz, 8 GB, 500 GB HDD, Ubuntu Linux 16.04.6 with 4.15.0-88-generic kernel (x86_64);

– *Slave 1*: Intel(R) Core(TM) i5-4460 CPU, 4 cores, 3.20 GHz, 24 GB, 450 GB SSD, Ubuntu Linux 18.04.4 with 4.15.0-91-generic kernel (x86_64);

– *Slave 2*: Intel(R) Core(TM) i7-6700HQ CPU, 4 cores, 2.60 GHz, 16 GB, 128 GB SSD, Linux Fedora 31 with 5.3.16-300.fc31.x86_64 kernel (x86_64).

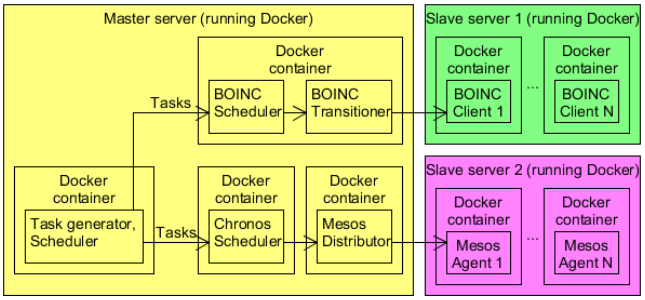

Fig. 5. Server setup B scheme. The master server and two slave servers are running on different servers.

B. Experiment Scenarios

To evaluate task execution makespan, we ran distributed tasks estimating the value of π using the Monte Carlo Method [53]. We selected this simple task to estimate task execution times depending only on CPU and eliminating other factors, such as networking and data storage. The same annotations are used as in Section V-A. However, due to the specifics of the experiment, there are some adjustments:

– *TS(i)*: static size tasks, where *i* is the number of iterations per task to calculate π;

– *TD(I)*: dynamic size tasks, where *I* is a set of task sizes.

The generated task sizes are distributed using a discrete uniform distribution (a = 0, b = 1);

– *STS(d)*: static task stream, where *d* is the delay in seconds between incoming new tasks;

– *DTS(D)*: dynamic task stream, where *D* is a set of possible delays between tasks. The generated delays are distributed using the Poisson distribution (λ = 30).

The following scenarios were used to test each algorithm:

– *TS_STS*: static size tasks TS(100000), static incoming task stream STS(60);

– *TS_DTS*: static size tasks TS(100000), dynamic incoming task stream DTS ({60, 120});

– *TD_STS*: dynamic size tasks TD({100000, 200000}), static incoming task stream STS(60);

– *TD_DTS*: dynamic size tasks TD({100000, 200000}), dynamic incoming task stream DTS({60, 120}).

The number of iterations and the delays between tasks were adapted for the hardware that we used for our experiments. For best results, the task stalling buffer should not always be empty or full. Otherwise, all tasks would get redirected into the private grid (system would be underutilised), or our task distribution algorithm would behave exactly like the standard FIFO algorithm.

1. Experimental results using Setup A

The first experiment was aimed at providing an overview of how the platform and the algorithms perform. Therefore, 100 tasks were executed using Setup A. The results show that the proposed hybrid cloud tasks scheduling method with task stalling buffer improves task execution makespan (see Fig. 6 and Table II).

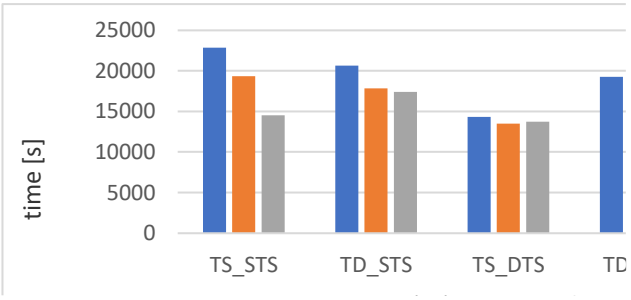

Fig. 6. Total makespan of 100 π value estimation tasks using the Monte Carlo method.

The static length task stalling buffer provides up to 15.45 % improvement over the standard FIFO algorithm, while the dynamic length task stalling buffer provides up to 36.48 % improvement.

TABLE II. TASK EXECUTION MAKESPAN DECREASE COMPARED TO FIFO (USING SETUP A).

Algorithm	Scenario	Makespan (seconds)	Makespan decrease
FIFO	TS_STS	22,863	not applicable
	TD_STS	20,627	not applicable
	TS_DTS	14,318	not applicable
	TD_DTS	19,251	not applicable
TSB-static(10)	TS_STS	19,331	15.45 %
	TD_STS	17,832	13.55 %
	TS_DTS	13,495	5.75 %
	TD_DTS	19,562	-1.62 %
TSB-dynamic	TS_STS	14,522	36.48 %
	TD_STS	17,394	15.67 %
	TS_DTS	13,727	4.13 %
	TD_DTS	17,830	7.38 %

Table II shows that the best results are achieved in the TS_STS and TD_STS scenarios. These results correspond to the simulation results presented in Fig. 3.

2. Experimental results using Setup B

We continued to run experiments using Setup B to test if adding more compute nodes to the public grid will produce similar results. We conducted multiple experiments using different numbers of tasks: 20, 40, and 60. Furthermore, we repeated each experiment five times to test the average time deviations. The aggregate results of 180 tests are presented in Tables III–V. We used the null hypothesis (two-tailed, α = 0.05) to prove that the alternate hypothesis is correct with at least 95 % probability. Here, the null hypothesis states that the task stalling buffer does not impact the task execution makespan. In this way, we tested whether the average task execution makespan using the FIFO scheduler is different compared to the average task execution makespan using our proposed scheduling algorithm:

1. TSB-static(10) scheduler.
2. TSB-dynamic scheduler.

TABLE III. TASK EXECUTION AVERAGE MAKESPAN USING THE FIFO ALGORITHM.

Scenario	Number of tasks	Average makespan (seconds)	Standard deviation (seconds)
TS_STS	20	3,598.4	189.20
	40	7,063.6	212.86
	60	9,995.4	223.01
TD_STS	20	5,841.8	654.40
	40	9,622.2	1,229.63
	60	14,905.4	1,382.52
TS_DTS	20	3,474.0	104.82
	40	6,576.0	145.25
	60	9,754.0	340.32
TD_DTS	20	5,933.4	455.89
	40	10,038.4	1,439.31
	60	14,620.4	1,255.28

TABLE IV. TASK EXECUTION AVERAGE MAKESPAN USING THE TSB-STATIC(10) ALGORITHM (COMPARED TO FIFO).

Scenario	Tasks	Average makespan (seconds)	Standard deviation (seconds)	Makespan decrease	p-value
TS_STS	20	3,503.8	394.22	2.63 %	0.6460
	40	**6,377.8**	**32.80**	**9.71 %**	**0.0020**
	60	**8,978.6**	**297.52**	**10.17 %**	**0.0005**
TD_STS	**20**	**3,967.4**	**959.76**	**32.09 %**	**0.0090**
	40	9,068.4	172.92	5.76 %	0.3750
	60	13,381.8	393.63	10.22 %	0.0640
TS_DTS	20	3,241.6	203.00	6.69 %	0.0630
	40	**6,168.8**	**256.39**	**6.19 %**	**0.0210**
	60	**9,254.0**	**240.38**	**5.13 %**	**0.0310**
TD_DTS	**20**	**3,127.0**	**824.78**	**47.30 %**	**0.0010**
	40	8,690.0	301.94	13.43 %	0.1100
	60	**12,293.0**	**1,566.73**	**15.92 %**	**0.0320**

The scenarios in which our proposed algorithms performed better than FIFO (with a significance level α = 0.05) are highlighted in Table IV and Table V. The results show up to 47.3 % improvement using static length task stalling buffer and up to 20.84 % improvement using dynamic length task stalling buffer compared to the standard FIFO algorithm. The static length task stalling buffer performed better because task stalling buffer capacity was never reached when executing only 20 tasks. It is inefficient to use the public grid to execute a small number of tasks,

since the private grid outperforms the public grid. The public grid did not receive any tasks. Thus, the platform underutilisation scenario occurred (as discussed in Section VI-B). Since the system was underutilised, we will not include the results from this particular test in our conclusions.

TABLE V. TASK EXECUTION AVERAGE MAKESPAN USING THE TSB-DYNAMIC ALGORITHM (COMPARED TO FIFO).

Scenario	Tasks	Average makespan (seconds)	Standard deviation (seconds)	Makespan decrease	p-value
TS_STS	20	3,493.0	133.24	2.93 %	0.3420
	40	**6,191.8**	**218.76**	**12.34 %**	**0.0002**
	60	**9,147.8**	**264.42**	**8.48 %**	**0.0010**
TD_STS	20	5,016.4	117.66	14.13 %	0.0500
	40	8,759.8	257.83	8.96 %	0.2000
	60	13,256.2	480.22	11.06 %	0.0530
TS_DTS	20	3,497.4	133.06	-0.67 %	0.7650
	40	**6,342.2**	**88.81**	**3.56 %**	**0.0180**
	60	**9,204.0**	**118.35**	**5.64 %**	**0.0190**
TD_DTS	**20**	**4,697.0**	**607.02**	**20.84 %**	**0.0080**
	40	9,018.8	193.90	10.16 %	0.1920
	60	13,262.2	461.70	9.29 %	0.0720

TABLE VI. TASK EXECUTION MAKESPAN DECREASE COMPARED TO FIFO (USING SETUP B).

Algorithm	Scenario	Makespan (seconds)	Makespan decrease
FIFO	TS_STS	31,361	not applicable
	TD_STS	47,502	not applicable
	TS_DTS	32,154	not applicable
	TD_DTS	45,159	not applicable
TSB-static(10)	TS_STS	30,504	2.73 %
	TD_STS	44,718	5.86 %
	TS_DTS	30,431	5.36 %
	TD_DTS	44,226	2.07 %
TSB-dynamic	TS_STS	30,700	2.11 %
	TD_STS	44,504	6.31 %
	TS_DTS	30,547	5.00 %
	TD_DTS	44,963	0.43 %

Finally, an extensive test was conducted by running 200 tasks. The results showed an improvement of up to 5.86 % using TSB-static(10) and an improvement of up to 6.31 % using TSB-dynamic (see Table VI).

VII. DISCUSSION

The proposed hybrid distributed computing platform can perform distributed computing tasks using cloud computing services and personal employee computers. Our proposed task scheduling method improves the efficiency of the platform, maintaining the same quality and reliability of the service. This innovation allows us to schedule tasks between two grids without requiring any additional information about the tasks.

The focus of ongoing research will be to test the capabilities of the proposed platform to solve big data mining tasks. Energy consumption and hardware usage cost minimisation could also be considered for future research.

VIII. CONCLUSIONS

Computer study and experiments show that the proposed hybrid cloud tasks scheduling method with static task stalling buffer reduces up to 47.3 % of the total task execution time. This allows us to conclude that a task stalling buffer can be applied for distributed hybrid cloud computing solutions and improve workload balance between two grids. The experiments showed that the most significant improvement is obtained when small batches of tasks are executed on a moderately loaded system. When the system is heavily loaded with large amounts of short tasks, the observed improvement is smaller.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] A. Rojko, "Industry 4.0 concept: Background and overview", *Int. J. of Interactive Mobile Technologies (iJIM)*, vol. 11, no. 5, pp. 77–90, 2017. DOI: 10.3991/ijim.v11i5.7072.

[2] A. Jurgelevičius and L. Sakalauskas, "BOINC from the view point of cloud computing", in *CEUR Workshop Proceedings*, 2017, pp. 61–66, vol. 1973. [Online]. Available: http://ceur-ws.org/Vol-1973/paper08.pdf

[3] Gartner Top 10 Strategic Technology Trends For 2020. [Online]. Available: https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2020/

[4] Gartner Top 10 Strategic Technology Trends For 2021. [Online]. Available: https://www.gartner.com/smarterwithgartner/gartner-top-strategic-technology-trends-for-2021/

[5] A. Jurgelevičius, L. Sakalauskas, and V. Marcinkevičius, "Task stalling for a batch of task makespan minimisation in heterogeneous multigrid computing", *Computational Science and Techniques*, vol. 8, pp. 631–638, 2021. DOI: 10.15181/csat.v8.2103.

[6] G. L. Stavrinides and H. D. Karatza, "Dynamic scheduling of bags-of-tasks with sensitive input data and end-to-end deadlines in a hybrid cloud", *Multimedia Tools and Applications*, vol. 80, pp. 16781–16803, 2021. DOI: 10.1007/s11042-020-08974-8.

[7] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002. DOI: 10.1109/71.993206.

[8] L. F. Bittencourt and E. R. M. Madeira, "HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds", *Journal of Internet Services and Applications*, vol. 2, pp. 207–227, 2011. DOI: 10.1007/s13174-011-0032-0.

[9] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. Da Fonseca, "Scheduling in hybrid Clouds", *IEEE Communications Magazine*, vol. 50, no. 9, pp. 42–47, 2012. DOI: 10.1109/MCOM.2012.6295710.

[10] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds", *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861–870, 2012. DOI: 10.1016/j.future.2011.07.005.

[11] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya, "Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka", *Future Generation Computer Systems*, vol. 28, no. 1, pp. 58–65, 2012. DOI: 10.1016/j.future.2011.05.008.

[12] R. Van Den Bossche, K. Vanmechelen, and J. Broeckhove, "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds", *Future Generation Computer Systems*, vol. 29, no. 4, pp. 973–985, 2013. DOI: 10.1016/j.future.2012.12.012.

[13] R. Duan, R. Prodan, and X. Li, "Multi-objective game theoretic scheduling of bag-of-tasks workflows on hybrid clouds", *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 29–42, 2014. DOI: 10.1109/TCC.2014.2303077.

[14] B. Wang, Y. Song, Y. Sun, and J. Liu, "Managing deadline-constrained bag-of-tasks jobs on hybrid clouds", in *Proc. of the 24th High Performance Computing Symposium*, 2016, article no. 22. DOI: 10.22360/SpringSim.2016.HPC.039.

[15] Y. Zhang and J. Sun, "Novel efficient particle swarm optimization algorithms for solving QoS-demanded bag-of-tasks scheduling problems with profit maximization on hybrid clouds", *Concurrency and Computation: Practice and Experience*, vol. 29, 2017. DOI: 10.1002/cpe.4249.

[16] S. Abdi, L. PourKarimi, M. Ahmadi, and F. Zargari, "Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds", *Future Generation Computer Systems*, vol. 71, no. C, pp. 113–128, 2017. DOI: 10.1016/j.future.2017.01.036.

[17] Y. Zhang, J. Zhou, L. Sun, J. Mao, and J. Sun, "A novel firefly algorithm for scheduling bag-of-tasks applications under budget constraints on hybrid clouds", *IEEE Access*, vol. 7, pp. 151888–151901, 2019. DOI: 10.1109/ACCESS.2019.2948468.
[18] Y. Zhang, J. Zhou, and J. Sun, "Scheduling bag-of-tasks applications on hybrid clouds under due date constraints", *Journal of Systems Architecture*, vol. 101, article ID 101654, 2019. DOI: 10.1016/j.sysarc.2019.101654.
[19] G. A. McGilvary, A. Barker, and M. Atkinson, "Ad hoc cloud computing", in *Proc. of IEEE 8th Int. Conf. (CLOUD 2015)*, 2015, pp. 1063–1068. DOI: 10.1109/CLOUD.2015.153.
[20] CharityEngine. [Online]. Available: http://charityengine.com
[21] GridMP. [Online]. Available: https://en.wikipedia.org/wiki/Grid_MP
[22] Xgrid. [Online]. Available: https://www.apple.com/server/docs/Xgrid_TB_v10.4.pdf
[23] XtremWeb. [Online]. Available: http://xtremweb.gforge.inria.fr
[24] Berkeley Open Infrastructure for Network Computing. [Online]. Available: https://boinc.berkeley.edu
[25] A. Jurgelevičius and L. Sakalauskas, "Big data mining using public distributed computing", *Information Technology and Control*, vol. 47, no. 2, pp. 236–248, 2018. DOI: 10.5755/j01.itc.47.2.19738.
[26] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center", *in Proc. of the 8th USENIX conference on Networked systems design and implementation*, 2011, pp. 295–308.
[27] Hadoop. [Online]. Available: https://github.com/mesos/hadoop
[28] MPI. [Online]. Available: https://github.com/mesosphere/mesos-hydra
[29] Apache Chronos. [Online]. Available: https://mesos.github.io/chronos/
[30] G. McGilvary, A. Barker, A. Lloyd, and M. Atkinson, "V-BOINC: The virtualization of BOINC", in *Proc. of 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, (CCGrid)*, 2013, pp. 285–293. DOI: 10.1109/CCGrid.2013.14.
[31] Docker. [Online]. Available: https://www.docker.com
[32] Kubernetes. [Online]. Available: https://kubernetes.io
[33] Oracle VM VirtualBox. [Online]. Available: https://www.virtualbox.org
[34] QEMU. [Online]. Available: https://www.qemu.org
[35] VMware. [Online]. Available: https://www.vmware.com
[36] P. Sun, "Security and privacy protection in cloud computing: Discussions and challenges", *Journal of Network and Computer Applications*, vol. 160, article ID 102642, 2020. DOI: 10.1016/j.jnca.2020.102642.
[37] A. Celesti, M. Fazio, M. Villari, and A. Puliafito, "Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems", *Journal of Network and Computer Applications*, vol. 59, pp. 208–218, 2016. DOI: 10.1016/j.jnca.2014.09.021.
[38] I. Sousa, M. P. Queluz, and A. Rodrigues, "A survey on QoE-oriented wireless resources scheduling", *Journal of Network and Computer Applications*, vol. 158, article ID 102594, 2020. DOI: 10.1016/j.jnca.2020.102594.
[39] D. Cidem Dogan and H. Altindis, "Storage and communication security in cloud computing using a homomorphic encryption scheme based Weil pairing", *Elektronika ir Elektrotechnika*, vol. 26, no. 1, pp. 78–83, 2020. DOI: 10.5755/j01.eie.26.1.25312.
[40] N. Schlitter and J. Lässig, "Distributed data analytics using RapidMiner and BOINC", in *Proc. of the 4th RapidMinder Community Meeting and Conference (RCOMM 2013)*, 2013, pp. 81–96.
[41] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems", School of Computing, Queen's University, Kingston, Ontario, Tech. Rep. 2006-504, Jan. 2006.
[42] M. Nandagopal and V. Uthariaraj, "Hierarchical load balancing approach in computational grid environment", *International J. of Recent Trends in Engineering and Technology*, vol. 3, no. 1, May 2010.
[43] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "GridFlow: Workflow management for grid computing", in *Proc. of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2003, pp. 198–205, DOI: 10.1109/CCGRID.2003.1199369.
[44] E. Heymann, M. A. Senar, E. Luque, and M. Livny, "Adaptive scheduling for master-worker applications on the computational grid", in *Grid Computing — GRID 2000. GRID 2000. Lecture Notes in Computer Science*, vol. 1971. Springer, Berlin, Heidelberg, 2000. DOI: 10.1007/3-540-44444-0_20.
[45] D. Paranhos, W. Cirne, and F. Brasileiro, "Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids", in *Euro-Par 2003 Parallel Processing. Euro-Par 2003. Lecture Notes in Computer Science*, vol. 2790. Springer, Berlin, Heidelberg, 2003. DOI: 10.1007/978-3-540-45209-6_26.
[46] G. Vilutis, R. Butkiene, I. Lagzdinyte-Budnike, D. Sandonavicius, and K. Paulikas, "The QoGS method application for selection of computing resources in intercloud", *Elektronika ir Elektrotechnika*, vol. 19, no. 7, pp. 98–103, 2013. DOI: 10.5755/j01.eee.19.7.2080.
[47] M. Usama, M. Liu, and M. Chen, "Job schedulers for Big data processing in Hadoop environment: Testing real-life schedulers using benchmark programs", *Digital Communications and Networks*, vol. 3, no. 4, pp. 260–273, Nov. 2017. DOI: 10.1016/j.dcan.2017.07.008.
[48] D. Yoo and K. M. Sim, "A comparative review of job scheduling for MapReduce", in *Proc. of IEEE Int. Conf. Cloud Computing and Intel. Syst. (CCIS)*, 2011, pp. 353–358. DOI: 10.1109/CCIS.2011.6045089.
[49] J. V. Gautam, H. B. Prajapati, V. K. Dabhi, and S. Chaudhary, "A survey on job scheduling algorithms in Big data processing", in *Proc. of 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2015, pp. 1–11. DOI: 10.1109/ICECCT.2015.7226035.
[50] R. Wisnesky, "Evaluating scheduling algorithms on distributed computational grids", 2010.
[51] M. Bhatia, "Task scheduling in grid computing: A review", *Advances in Computational Sciences and Technology*, vol. 10, no. 6, pp. 1707–1714, 2017.
[52] L. Kaklauskas, L. Sakalauskas, and V. Denisovas, "Stalling for solving slow server problem", *RAIRO - Operations Research*, vol. 53, no. 4, pp. 1097–1107, 2019. DOI: 10.1051/ro/2018056.
[53] Calculate Pi with Monte Carlo. [Online]. Available: https://hub.docker.com/r/ashael/pi

This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution 4.0 (CC BY 4.0) license (http://creativecommons.org/licenses/by/4.0/).