The Journal of Supercomputing (2025) 81:848
https://doi.org/10.1007/s11227-025-07295-7

Optimization of resource-aware parallel and distributed computing: a review

Paweł Czarnul[1] · Marcel Antal[2] · Hamza Baniata[3] · Dalvan Griebler[4] · Attila Kertesz[3] · Christoph W. Kessler[5] · Andreas Kouloumpris[6] · Salko Kovačić[7] · Andras Markus[3] · Maria K. Michael[6] · Panagiota Nikolaou[8] · Isil Öz[9] · Radu Prodan[10] · Gordana Rakić[11]

Accepted: 5 April 2025

Abstract
This paper presents a review of state-of-the-art solutions concerning the optimization of computing in the field of parallel and distributed systems. Firstly, we contribute by identifying resources and quality metrics in this context including servers, network interconnects, storage systems, computational devices as well as execution time/performance, energy, security, and error vulnerability, respectively. We subsequently identify commonly used problem formulations and algorithms for integer linear programming, greedy algorithms, dynamic programming, genetic algorithms, particle swarm optimization, ant colony optimization, game theory, and reinforcement learning. Afterward, we characterize frequently considered optimization problems by stating these terms in domains such as data centers, cloud, fog, blockchain, high performance, and volunteer computing. Based on the extensive analysis, we identify how particular resources and corresponding quality metrics are considered in these domains and which problem formulations are used for which system types, either parallel or distributed environments. This allows us to formulate open research problems and challenges in this field and analyze research interest in problem formulations/domains in recent years.

Keywords Review of resource-aware parallel and distributed computing · Parallel and distributed architectures · Parallel and distributed computing · Optimization

Abbreviations

ACO	Ant colony optimization
BC	Blockchain
CA	Consensus algorithm
CPU	Central processing unit
CP	Constraint programming
CUDA	Compute uniform device architecture

Extended author information available on the last page of the article

Published online: 09 May 2025

DAG	Directed acyclic graph
DC	Data center
DCT	Dynamic concurrency throttling
DRAM	Dynamic random access memory
DFS	Dynamic frequency scaling
DL	Distributed ledger
DP	Dynamic programming
DPN	Dataflow processing network
DSL	Domain-specific language/library
DSP	Digital signal processing/processor
DVFS	Dynamic voltage and frequency scaling
EPIC	Explicitly parallel instruction computing
FIFO	First in, first out
FL	Federated learning
GA	Genetic algorithm
GPU	Graphics processing unit
GrA	Greedy algorithm
GT	Game theory
HEFT	Heterogeneous earliest finishing time (First)
HPC	High-performance computing
ILP	Integer linear program(ming)
ILP	Instruction-level parallelism
IPM	Interior point method
IoT	Internet of things
MG	Matching game
MILP	Mixed integer linear program(ming)
ML	Machine learning
MPI	Message passing interface
pBFT	Practical byzantine fault tolerant
PGAS	Partitioned global address space
PoW	Proof of work
PSO	Particle swarm optimization
P2P	Peer to peer
RL	Reinforcement learning
SIMD	Single Instruction, Multiple Data
TDP	Thermal design power
TIG	Task interaction graph
TS	Tabu search
TX	Transaction
VC	Volunteer computing
VM	Virtual machine
VLIW	Very Long Instruction Word (computing)
VLSI	Very large-scale integration

1 Introduction

Parallel and distributed computing has already been deployed at practically all system levels, from instruction-level parallelism and massive multithreading in modern multi- and many-core central processing units (CPUs) and graphics processing units (GPUs). These can be integrated into clusters composed of powerful nodes with computational devices such as multi-core CPUs and GPUs as well as fast interconnects, e.g., Infiniband [1]. Such computational devices are suitable for a widening spectrum of application domains, ranging from high-performance computing (HPC) simulations to distributed processing. Those applications are commonly used on an everyday basis for communication over the Internet, as well as the Internet of Things (IoT) and edge solutions that may be integrated with clouds for data storage and processing. At all these levels, researchers deal with optimization problems to improve various computing resource usage.

This paper aims to describe the problem formulations and identify computing resources typically considered in parallel and distributed computing, including various contexts/subdomains. We further identify metrics associated with the resources that are used within optimization goals. Adopting the analysis of the field structured along problem formulations, resources, and metrics, the goal of this review is to investigate the usage of particular metrics and resources as well as the usage of specific problem formulations in contexts/subdomains and based on that identify concrete research gaps and topics for future research.

The methodology adopted in this paper assumes the identification of research works from the area of parallel and distributed computing that considers optimization problems expressed as one or more of the selected, assumed a priori formulations. These include integer linear programming (ILP) [2–7], greedy algorithms (GrA) [8, 9], dynamic programming (DP) [10], genetic algorithms (GA) [11–13], particle swarm optimization (PSO) [14–17], ant colony optimization (ACO) [18], game theory (GT) [19–23], and reinforcement learning (RL) [24–27]. The rationale of this approach is motivated by the fact that the aforementioned formulations would typically use certain variables referring to the usage of the resources, described in detail in Sect. 2. This, in turn, allows us to:

1. Identify problems with similar resources in various contexts/subdomains such as data centers, cloud computing, fog computing, blockchain, high-performance computing, and volunteer computing.
2. Upon consideration of these subdomains, identify resources as well as problem formulations frequently considered in specific subdomains.

The main structure of our manuscript is demonstrated in Fig. 1. In Sect. 2, we identify resources, associated metrics, along with exemplary trade-offs and optimization contexts. Additionally, we describe typical system and application modeling along with a frequently considered, generalized resource allocation problem definition. Subsequently, in Sect. 3 we follow with detailed categorization into the aforementioned problem formulations including ILP, GrA, DP, GA, PSO, ACO, game

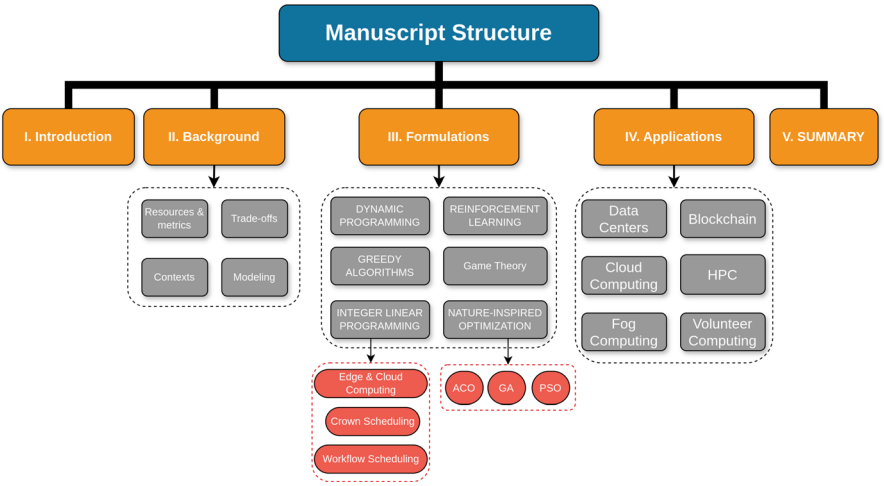

Fig. 1 Manuscript structure

theory, and RL. In Sect. 4, we describe real-world applications studied in the literature and their solutions. Specifically, we focus on the subdomains: data centers, cloud computing, fog computing, blockchain (BC), high-performance computing (HPC), and volunteer computing (VC). Finally, we provide conclusions in Sect. 5 regarding our observations, existing research gaps, as well as open problems.

2 Background

Before we start the analysis of problems considered in parallel and distributed computing, we identify resources and associated metrics that are the subject of optimization in the area of parallel and distributed computing. Product and process quality characteristics are measured and expressed by various metrics. These metrics can then be combined in order to express standard quality characteristics such as performance efficiency, reliability, security, safety, etc. Consequently, resource utilization is measured by resource utilization metrics that constitute specific optimization goals/functions. We further discuss frequently faced trade-offs between those metrics and characteristics and list exemplary optimization contexts in parallel and distributed computing. These serve as a starting point for further description of system and application models in this field as well as the statement of the resource allocation problem, fundamental regarding optimization in the considered domain.

2.1 Resources and metrics

Before we start the analysis of problems considered in parallel and distributed computing, we identify resources that are subject to utilization and optimization in the area of parallel and distributed computing. Consequently, resource

utilization is measured by metrics that reflect a single resource utilization level or different relations and ratios between multiple resources. These metrics reflect a single resource, e.g., execution time using concrete computational devices for a given data size.

In the field of parallel and distributed processing, problem-specific resources include nodes, network, memory/storage [1, 28], etc. In terms of metrics, for instance, computational devices such as CPUs and GPUs are described by actually obtained (single, double precision) performance (measured through execution time for a given workload) and power requirements (formally specified through Thermal Design Power (TDP) values by manufacturers [1]). In the case of memory, various implementations might result in various memory/storage levels, specifically of various levels such as DRAM, caches, disk, etc. Another relevant example is ingress traffic and network link capacity [29].

We shall note that the aforementioned (problem-specific) resource metrics allow assessing and are components of more complex optimization functions (specifically in scheduling problems [30–33]) typically considered in the context of application or workload execution. The typical resulting application-level quality metrics being either sole goals of optimization or components of optimization functions in the field of parallel and distributed computing include:

1. Execution time as a typical measure of the inverse of performance: We shall note, however, that the performance of computational devices (such as CPUs, GPUs) is associated with a given code/workload. Thus, performance ratios between two particular computational devices might differ for various workloads. In general, execution time of a parallel or distributed application results from consideration of the application flow graph and execution times of particular computational devices, transmission latency (considering data size of a message as well as network parameters including start-up time and bandwidth), and propagation throughout the network.
2. Energy consumption during application runtime: This can be measured at the node level, including computational devices, memory domains, storage devices affected by power supply characteristics, or/and the CPU+DRAM or GPU domains. In a broader sense, the consumption of resources could also be considered for the application development phase.

Additionally, the following quality characteristics are often of interest concerning the application:

1. Programming/development effort which can be measured in PMs, spent on the development/maintenance of a given application.
2. Application and system security, spanning consistency (concerning standards, design usage), availability (to a user), integrity (among components of a system), tolerance of cyber attacks and network vulnerabilities, etc.
3. Soft error vulnerability: Soft errors are a subset of transient faults caused by a single bit flip in the system [34]. If a soft error affects data in a register or memory

structure, data can be corrupted (SDC—silent data corruption), and the program may generate incorrect outcomes. Soft errors may lead to the unintended termination of the applications besides data corruption. The most common way to characterize the soft error vulnerability of an application is to perform fault injection experiments [35]. The experiments introduce deliberate faults as bit flips at hardware or software levels multiple times and measure the rate of the outcomes to have a statistically significant vulnerability value, where mostly the SDC rate represents the vulnerability of the target application. To overcome soft errors, the system implements redundancy techniques, where the hardware or software components are replicated. Specifically, redundant multithreading (RMT) utilizes parallelism in modern parallel systems and enables fault detection and correction [36]. RMT can correct faults by running identical program copies or code regions as separate threads or instructions in parallel execution units and comparing their outputs. While parallel executions can reduce redundancy costs by eliminating redundant execution latency, having multiple threads or instructions may incur additional costs due to contention/synchronization among many threads or instructions. Therefore, soft error precautions expose a trade-off between performance, cost, and reliability of the target execution [37, 38].

2.2 Trade-offs

In practical approaches, many trade-offs appear between and among the aforementioned metrics and resources. Examples in the context of various scopes of distribution include:

- Performance vs energy trade-off: Improving performance with more powerful processors, higher clock speeds, and parallel resources lead to high power consumption by high-capacity hardware components. On the other hand, if the execution time decreases for a target application, total energy consumption reduces for the same power consumption values [39–41]. While some studies target to perform dynamic configurations by power capping and considering performance effects simultaneously [42], there have been works trying to find out performance energy configurations for parallel applications to be run on heterogeneous multi-processing systems where a configuration space stems from different core types and voltage and frequency pairings [43].
- Performance vs security trade-off in the context of cloud storage systems in which data are replicated over several nodes of a cluster [44].
- Security, scalability, and efficiency trade-offs for data collection of IoT sensor systems considering various security settings such as encryption and authentication when using scalable cloud services [45].
- Other trade-offs can also be considered, such as the performance vs storage trade-off [46], performance vs reliability trade-off [47, 48], the DePriSS tetralemma [49], etc.

2.3 Exemplary optimization contexts

The aforementioned metrics are the subject of optimization in various contexts in the field of parallel and distributed computing, some of which are described below. Such problems are further categorized in terms of formulations in this section and described in more detail in respective domains in Sect. 4.

Various works target the reduction in the overall latency (considered as elapsed/wall time), maximizing, in that way, the application's performance in different market segments. Ghosh and Simmhan [50] minimize the end-to-end latency by using a brute force approach and genetic algorithm patterns to optimize the placement of complex event processing queries across a collection of edge and cloud resources.

Wang et al. [51] provide an automatic generator to find optimal partitioning for cross-end analytic engine architecture for wearable computing systems. Kolomvatsos and Anagnostopoulos [52] propose a two-step decision process to choose the tasks that will be executed locally at the edge nodes while the rest will be migrated to a group of peer nodes in the network or on a fog/cloud server to maximize performance. To maximize the user's quality of experience, Shah-Mansouri and Wong [53] allocate fog resources by formulating a computation offloading game.

Furthermore, Kouloumpris et al. [54] propose task allocation, taking into account reliability aspects while still targeting minimum latency. Additionally, Kouloumpris et al. [55] address the problem of task allocation within the edge–hub–cloud architecture to maximize the system's overall reliability while satisfying latency, energy, and memory constraints. Particularly, the works in [54, 56], and [55] are based on a mathematical programming (MP) scheme for achieving an optimal partitioning of an application across the edge/hub/cloud paradigm model by minimizing the overall latency of the system, taking into account the latency and energy consumption of both nodes and communication links, as well as the memory needed for the tasks to be executed.

Kouloumpris et al. [56] present a framework to provide the optimal task allocation and optimal scheduling [57] between the edge–hub–cloud architecture to satisfy the application constraints and simultaneously minimize the end-to-end latency.

On the other hand, several related works explore the problem of task allocation, targeting not only performance improvement but also reliability and energy, such as the works in [58] and [59] that focus on edge servers and mobile computing devices such as laptops, tablets, and mobile phones.

Several works try to allocate the tasks to minimize total cost which can be defined as a function of general quality metrics and metrics of the selected resources. Dinh et al. [60] introduced an optimization framework to allocate the processes from a single mobile device to multiple edge devices to minimize the total cost taking into account mobile device power consumption and end-to-end latency.

Nikolaou et al. [61] develop a Total Cost of Ownership (TCO) model to investigate the benefits of running an emerging security-focused IoT application at the edge vs. the cloud by also considering the application's requirements as well as the edge's constraints.

2.4 System, application, and resource allocation modeling

Resource-aware parallel and distributed computing is centered around the resource allocation problem which aims to map tasks onto resources in the system by considering specific constraints. Based on various (hardware) systems and (software) application models, the resource allocation problem can be defined using different (mathematical) formulations. In this section, we provide the formulations considering various resources. First, we identify target system models that host our applications. Secondly, we explain application models representing target workloads (tasks). Then, we provide a general definition of the resource allocation problem. Table 1 contains key symbols used within the paper, specifically throughout the remaining part of this section for the problem formulations.

2.4.1 System model

Modern computer architectures are complex systems that integrate multiple processing units, memory units, communication buses, and other kinds of hardware resources. They are usually organized in a hierarchical way, leveraging parallelism at multiple levels.

For example, a modern general-purpose processor core usually integrates Single Instruction, Multiple Data (SIMD) and instruction-level parallelism (ILP), in the form of pipelined instruction execution, multi-issue Very Long Instruction Word (VLIW) or superscalar architectures, and often combinations of these.

Modern CPUs can contain dozens of processor cores, and domain-specific accelerators such as GPUs contain even more. On-chip memory often comes in the form of hardware-managed caches for DSP (Digital Signal Processing), GPU, and embedded processors; often in the form of software-managed scratchpad memories, exposing memory allocation and data transfers to the programmer.

In a *homogeneous multi-core CPU*, all cores are of the same type. A *heterogeneous multi-core CPU* combines different types of cores with different micro-architectures, some of which may be optimized for single-thread performance while others are optimized for power efficiency, as in ARM big.LITTLE type architectures.

While low-end computers such as smartphones and laptops typically have only one general-purpose CPU (nowadays with multiple cores), servers may integrate several CPUs that share access to main memory in a uniform (UMA) or non-uniform (NUMA) way; in the latter case, main memory is organized in multiple memory units, some of which are located closer to one CPU (in terms of memory access latency and/or bandwidth) than to others.

Most computers today also comprise one or several GPUs and/or other accelerators, which usually have their own memory connected to main memory by communication buses such as PCI Express [1], with or without explicitly exposing memory management and data transfers to the programmer. We refer to computer systems combining one or several general-purpose CPUs with one or several accelerators as *heterogeneous computer systems*.

Finally, super-computing clusters and data centers are organized by aggregating nodes that are connected by special fast networks (e.g., Infiniband) [1]. Those

Table 1 Key symbols used for problem formulations

Symbol	Description
t_i	Task i of a workflow graph $G(T, E)$ with T – set of tasks and E – set of directed edges
S_i	Set of services $S_i = \{s_{i0}, s_{i1}, ...\}$ each of which is capable of executing t_i
t_{ij}	Candidate task, i.e., task t_i running on computational device j
$l_{ij}(d_i)$	Execution time/latency of service s_{ij} processing data of size d_i
l_i	Execution time of task t_i
l_{ij}	Execution time of task t_i (on resource j)/candidate task t_{ij} on computational device j
$pr_{ij}(d)$	Price (monetary) of s_{ij}'s execution
$pow_{ij}(d)$	Average power taken by service s_{ij}
L	Workflow execution time
$L_{\max}$	Upper bound on the workflow execution time
w_{ij} or w_e	Weight on edge connecting nodes i and j in a graph G
p	Number of cores
x, e.g., x_{ij}	(binary) Solution variables in an integer linear program, e.g., {0,1} denoting whether service s_{ij} is selected for execution of task t_i or binary decision variable corresponding to candidate task t_{ij}
Ψ	The upper bound fraction of faulty nodes tolerated by a named distributed system in order to maintain its security
R	Set of network shards
$Succ(t_i)$	Set of immediate successors of task t_i
$Pred(t_i)$	Set of immediate predecessors of task t_i
$\overline{c_{i,j}}$	Average communication cost of edge (i, j)
$\overline{l_i}$	Average execution time of task t_i
C_{exec}	Total execution cost of the application
C_{comm}	Total system communication cost
R_{exec}	Reliability of a processor
R_{comm}	Reliability of all communication paths
x_{ijk}	Binary decision variable corresponding to outgoing edge $j \rightarrow k$ from task i
e_{ij}	Computational energy consumption of candidate task t_{ij}
m_{ij}	Memory requirement of candidate task t_{ij}
st_{ij}	Storage requirement of candidate task t_{ij}
$l_{\text{comm}_{ijk}}$	Communication latency corresponding to outgoing edge $j \rightarrow k$ from task i
$e_{\text{comm}_{ijk}}$	Communication energy consumption corresponding to outgoing edge $j \rightarrow k$ from task i
$nd(i)$	Number of immediate descendants of task t_i
$na(i)$	Number of immediate ancestors of task t_i
M_j^{bgt}	Memory budget corresponding to device j
St_j^{bgt}	Storage budget corresponding to device j
E_j^{bgt}	Energy budget corresponding to device j

networks typically consist of multiple switches and lines as hardware resources. For a recent survey of computer architecture features, we refer to Hennessy and Patterson [62]. An up-to-date survey of recently installed supercomputer hardware and the

trends in node architecture and network technology is provided in regularly updated ranking lists such as the TOP500 and Green500 lists.[1]

Due to cost considerations, the available quantity of these hardware resources in a computer system is limited. This implies that there is usually (much) more computational work (e.g., tasks ready to execute) than could be possible to execute in parallel efficiently due to a limited number of computational devices. This leads to resource allocation and scheduling problems but also the need for consideration of resource (including network) contention, which we will consider in more detail in the following sections.

For distributed systems such as IoT, fog, cloud systems accessed from external clients, and volunteer systems, we distinguish physically distributed nodes that can vary in performance, power consumption, and reliability. Moreover, these are interconnected with relatively slow networks with much larger latency and smaller bandwidths compared to network solutions used in the aforementioned clusters and data centers. Additionally, the reliability of neither nodes nor networks is guaranteed.

Details of the aforementioned system models are outlined in relevant subsections of Sect. 4.

2.4.2 Application model

The target application with multiple tasks is represented in different ways by considering target tasks and their relationships. We distinguish between two main types of task-based processing scenarios: the DAG model for batch execution of partially ordered dependent tasks, where a task's execution is often not considered preemptable, and the TIG model for concurrent, possibly re-entrant and/or preemptable data processing tasks connected by edges modeling inter-task communication.

Accordingly, there exist at least two different types of task graphs:

1. *DAG Model:* The task graph modeling the application is a Directed Acyclic Graph (DAG) $G = (T, E)$, where the node set T is the set of tasks of the application, and E is a set of directed edges denoting precedence constraints among tasks due to dependencies. Such dependencies might be due to control flow or data dependence such as data flow. The DAG model assumes that a task can be executed only if all the predecessor tasks have completed their executions. Hence, DAG task graphs are dependence graphs, and the tasks are one-shot tasks. A task can execute only if all source operands are available as well as a processing resource for execution. The task graphs are thus generally acyclic (i.e., DAGs). A dependence (directed edge) between two tasks (nodes) usually means that the source task needs to finish before the destination task can start executing (however it may start even later if no resource is readily available for executing it. This is a matter of task scheduling, which in turn can happen statically or dynamically). Hence, tasks on the same path never execute concurrently. The longest path (in terms of accumulated execution time) in the task graph (also known as the *critical*

[1] TOP500, HPCG and Green500 lists: https://top500.org.

path) establishes a lower bound for the execution time, regardless of how many execution resources are available (Brent's Theorem [63]). Depending on the target system properties and on the mapping of the tasks to execution resources, data flow edges between tasks mapped to different resources with different memories may also imply a data transfer cost, which is usually linear in the size of the data packet sent. The cost for forwarding operand data between tasks mapped to resources within the same memory unit is typically much smaller but not necessarily zero.

Task graphs in the DAG model are a special case of workflow graphs. The task graph representation can be further extended by assigning fixed computation costs for the nodes (tasks) and, for systems where communication cost matters (primarily multi-node distributed memory systems), in communication volumes for the edges. In homogeneous systems with identical computation resources, each task $t_i \in T$ is associated with a *computation cost* l_i, representing the computation time needed to complete the execution of the task. In systems with symmetric communication costs, where the communication cost for an edge (t_i, t_j) only depends on the volume of data flowing from task t_i to t_j, an edge (t_i, t_j) can be labeled directly with a non-negative weight $e(t_i, t_j)$ modeling the communication delay time for the case that source and destination tasks are mapped to different resources; if t_i and t_j are mapped to the same resource, the communication cost will be 0.

In heterogeneous systems with different (types of) execution resources, the task computation costs will differ depending on where a task executes. For any task i and resource j, the *computation cost* l_{ij} models the latency of executing t_i on j. For tasks i that cannot execute on certain resources j, we have $l_{ij} = \infty$.

Where communication cost is asymmetric, i.e., differs depending on where t_i and t_j will execute, we could instead label the edge with the communication volume (the number of bytes to communicate) between (instances of) tasks t_i and t_j; the actual communication cost will depend on the task mapping and can be derived from the communication volume and the architectural parameters in a second step.

2. *TIG Model:* The TIG model describes communicating concurrent tasks.

   Formally, a *Task Interaction Graph* (TIG) is a graph $G = (T, E)$, where T is a set of tasks and E models the interaction (communication) between these tasks.

   TIG task graphs describe applications such as data flow processing networks (also known as *actor networks*) with continuously active tasks processing potentially infinite streams of data packets, with edges representing FIFO-buffered data flow channels between producer and consumer tasks, as formalized in established data flow models such as Kahn Process Networks (KPN) [64], Synchronous Data Flow (SDF), etc. A *task instance* (execution of a task for one input data packet) is triggered either by time as prescribed in a statically computed cyclic schedule or by data flow at runtime—whenever new operand data is available and executed when a resource has been allocated for its execution. Hence, (instances of) all tasks execute concurrently.

   While TIG tasks may occasionally interact with each other, which incurs communication overhead for tasks executing in different memory spaces, the model

does not necessarily contain information about precedence relations among the tasks, in contrast with the DAG model.

TIG tasks can be stateless or stateful. For a stateful task, its next instance is dependent on its predecessor instance. Stateful tasks can be modeled as stateless tasks if the state at the end of a task instance is passed back as an input of the task for its next instance.

In general, data flow in a TIG may be cyclic. However, virtually unfolding the execution of an entire TIG computation into the graph of all executed task instances and their dependencies would naturally result in an acyclic graph (DAG model) again.

Many real-world problems can be modeled as a TIG, such as signal/image data processing networks [64], iterative solutions of systems of equations, power system simulations, and VLSI simulation programs. In particular, *data processing networks* (DPN) such as *Kahn Process Networks* [64] are special cases of TIG. As in the DAG model, TIG tasks and edges may be assigned costs for computation and communication, respectively.

Internally, a task is, if not stated differently, usually executed sequentially and thus mapped to a single execution resource, such as a physical core, a virtual core, or an accelerator. Nevertheless, also *parallel tasks* can be defined, which require a statically known, fixed number (larger than one) of execution resources to be available in parallel to execute a parallel algorithm internally, usually sharing data; and the more general *parallelizable tasks*, which can run on any number of execution resources. Among the latter ones, we additionally distinguish between *moldable* tasks, which can use a fixed number of resources that needs to be known before task execution starts and remains constant throughout the execution of the task, and *malleable* tasks, which allow the number of workers to change *during* task execution.

If not stated otherwise, a task is associated with a block of code (e.g., a function call) to execute on a general-purpose CPU resource, as discussed previously. Considering heterogeneous systems, some tasks might, however, be specifically intended for execution on an accelerator resource, such as a GPU, or might be *multi-variant* tasks that come with multiple implementation variants for different execution platforms, e.g., an OpenMP [1] variant for multithreaded CPU execution together with a CUDA or OpenCL [1] variant for GPU execution, so that the selection of the variant (and thus, execution resource type) could be done at runtime. The code for such task implementation variants can either be explicitly specified by the programmer and a task API, e.g., in the task-based runtime system *StarPU* [65] for heterogeneous systems, or automatically compiled from a single high-level source code representation, as in recent OpenMP versions, in OneAPI,[2] or *SkePU*[3] [66].

[2] OneAPI documentation and download:https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html.

[3] SkePU documentation and download: https://skepu.github.io.

2.4.3 Resource allocation model

The resource allocation problem can be considered and formulated by referring to different resources as well as including relevant, associated metrics, mentioned in Sect. 2.1. As targeting multiple resources within different limitations, the resource allocation problem can be mainly represented by:

$$\min / \max f_m(\boldsymbol{x}), \quad m = 1, 2, \cdots, M \tag{1}$$

subject to

$$g_j(x) \geq 0, \quad j = 1, 2, \cdots, J \tag{2}$$

$$h_k(x) = 0, k = 1, 2, \cdots, K \tag{3}$$

where f_m represents objective functions to be minimized or maximized, g_j and h_k represent inequality and equality constraints, respectively.

When there are multiple objective functions, in many real-life problems, objectives under consideration conflict with each other. Hence, optimizing x with respect to a single objective often results in unacceptable results with respect to the other objectives. Therefore, a multi-objective solution that optimizes each objective function at the same time is almost impossible. An acceptable solution to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution [67]. If all objective functions are for minimization, a feasible solution x is defined to dominate another feasible solution y, if and only if, $z_i(x) \leq z_i(y)$ for all objective functions and $z_j(x) < z_j(y)$ for at least one objective function j. A solution is defined to be Pareto optimal if it is not dominated by any other solution in the solution space. A Pareto optimal solution cannot be improved with respect to any objective without making worse at least one other objective. The set of all feasible non-dominated solutions is defined as the Pareto optimal set, and for a given Pareto optimal set, the objective function values in the objective space are called the Pareto front. The goal of a multi-objective optimization algorithm is to find solutions in the Pareto optimal set. To solve the multi-objective problem, the objective function can be evaluated by different approaches like weighted sum, alteration, pareto-ranking.

We can have convexity or non-convexity depending of the problem specification. For instance, the resource allocation problem, where we have a fixed budget across multiple tasks, presents a convexity as the objective function is linear [68, 69]. On the other hand, the problems including integer variables, nonlinear functions, or interference constraints present non-convexity with multiple local optima points that make the problem harder to solve [70, 71].

Depending on the optimization objective such as latency, energy consumption, or reliability, and within the limit of the resource capacities such as computation units and memory size, the objective functions and constraints are defined as part of the formulations for the target execution.

2.5 Proposed contribution in the context of existing surveys

There are several survey papers concerning parallel and distributed computing, as well as algorithms, models, and tools used. In this section, we summarize these surveys and then outline the novelty of our approach, specifically by comparing and pointing out the areas and aspects addressed in our review but missing or described in a less comprehensive way in the existing works.

In paper [72], authors focus on load balancing algorithms in both distributed parallel computing and cloud computing. Task execution on traditionally considered physical resources is analyzed along with metrics including time, cost, throughput, usage of computing, and memory resources. Our study contains a larger set of metrics as well as considers more parallel and distributed processing domains, including cloud computing but also high-performance computing (HPC), fog computing, volunteer computing, blockchain, and computing in data centers.

In survey [73], the author focuses specifically on allocation of Virtual Machines in cloud data centers. As resources, they consider physical machines and cloud providers. They consider multi-objective optimization with consideration of possibly several objectives: monetary, performance related (including response time and makespan), energy related, technical (utilization, balance, traffic, temperature). Areas in need of further research have been identified including consideration of: hybrid clouds, power consumption at various levels: server, component including network etc., multi-core CPUs, networking, co-location interference. We, on the other hand, generalize the problem formulations to the formulations such as integer linear programming (ILP), greedy algorithms (GrA), dynamic programming (DP), genetic algorithms (GA), particle swarm optimization (PSO), ant colony optimization (ACO), game theory (GT), and reinforcement learning (RL), and consider more parallel and distributed system types, as mentioned above, including cloud computing. Our goal is to analyze these problem formulations as common across parallel and distributed computing in various domains and link actual optimization problems to these formulations. Consequently, our approach presents a more comprehensive view on parallel and distributed computing as a whole, analyzing resources and metrics used in all of these domains. Finally, we identify which and how resources and metrics, as well as problem formulations, are used in various contexts and domains, not present in study [73].

Some works investigate usage of particular problem formulations and specific domains for optimization of specific tasks. As an example, authors of paper [74] propose a master–slave parallel genetic algorithm (GA) that is used to solve the time–cost construction problem using high-performance computing and NSGA-II as the optimization engine. Such works are actually considered as use cases in our study providing specific use cases: problem formulation–resources/metrics–optimization algorithm–parallel domain/system type.

Authors of book [75] explore parallelism in constraint-based reasoning formalisms. They consider parallel implementations and taking advantage of computing resources—from single machine multi-core CPUs and GPUs to distributed systems. In the context of this paper, they explore parallel solvers for mixed integer linear programming and parallel local search algorithms. Instead, we study problem

formulations including MILP, as well as others, as a way of expressing optimization problems encountered in parallel and distributed systems.

Other review papers focus on specific aspects of the specific parallel and distributed computing domains discussed in our review. For instance, survey [33] focuses on high-performance computing and even more specifically on energy-aware scheduling methods used for HPC. The authors discuss resources and metrics involved in optimization including power, energy, and time. Selected algorithms are presented and classified by the programming method, such as machine learning or fuzzy logic. General scheduling of tasks of a parallel application onto a hybrid computing platform that consists of multi-cores and accelerators is discussed in detail in survey [76]. Greedy approaches are among the ones analyzed, along with incorporation of (integer) linear programming into algorithms.

Compared to those, in our study scheduling in an HPC system constitutes one of many optimization algorithms that can be represented by selected problem formulations and is subject to resource–problem formulation–domain analysis.

On the other hand, existing surveys also discuss programming models and paradigms. In paper [77], authors characterize multithreaded processing, message passing, partitioned global address space (PGAS), agent-based computing, MapReduce along with goals taking into consideration performance, energy, as well as programming APIs. Multithreaded processing, message passing, the actor model, and parallelism at various levels, including instruction level, vector parallelization, thread level, and request level, and future research concerning those are analyzed in study [78]. High-level programming models for multi-core systems are described in survey [79], including: C++-based, skeleton-based, STL-based, directive-based, and domain-specific ones. APIs and translators between them are detailed. Compared to those, our review assumes that the models are incorporated into the applications running in parallel and distributed systems and are not directly subject of our analysis. Rather than that, we target optimization problems—that typically refer to execution of parallel codes, problems' formulations, and applicability in parallel and distributed systems. Consequently, our and those reviews are complementary.

2.6 Summary and conclusion

In this section, we explore the fundamental challenges of resource-aware optimization in parallel and distributed computing. The discussion begins with an identification of essential resources, including computational devices, memory, and network bandwidth, alongside key performance metrics such as execution time, energy consumption, reliability, security, and error vulnerability. These metrics form the basis for assessing system performance and optimizing resource allocation.

We then explain how to balance trade-offs, such as performance versus energy efficiency, security versus scalability, and reliability versus cost, emphasizing the multidimensional nature of optimization in this domain. Several real-world optimization contexts are presented, including strategies for reducing latency in cloud and edge computing, optimizing task allocation to improve reliability and efficiency, and balancing cost, energy, and performance in mobile and edge devices.

This section also introduces modeling frameworks for systems, applications, and resource allocation. These include directed acyclic graph (DAG) and task interaction graph (TIG) models for representing application workflows, and mathematical formulations for solving resource allocation problems. The models are pivotal in addressing constraints like limited processing power or memory, facilitating the efficient mapping of tasks to available resources.

Additionally, this section contrasts the proposed approach with existing surveys in the literature, highlighting novel contributions and complementary aspects. Unlike previous reviews that focus on specific aspects such as virtual machine allocation or energy-aware scheduling, this study provides a holistic analysis of optimization problem formulations, linking them to various computing domains. This, in essence, demonstrates how our paper extends the state-of-the-art analyses in the field.

In conclusion, this section underscores the complexity of resource-aware optimization in distributed computing systems. It highlights the necessity of considering multiple interdependent metrics and trade-offs while modeling and optimizing tasks and resources. This foundational understanding sets the stage for exploring advanced optimization techniques and their practical applications in subsequent sections.

3 Problem formulations as a basis for resource-aware optimization in various domains

In this section, we present frequently used problem formulations utilized in target problems of parallel and distributed computing and considering the previously identified resources and metrics.

3.1 Integer linear programming

Integer linear programming (*ILP*) is a powerful generic technique for modeling and solving combinatorial optimization problems. An ILP model involves a finite set of integer-valued solution variables, a set of constraints in the form of inequalities that are linear in these variables, and an objective function to maximize or minimize that is likewise linear in these variables. For solving an ILP instance, a generic solver is used. A timeout is usually set because solving an ILP to optimality is NP-complete; if the solver hits the time limit, it might still be able to return an approximation or heuristic solution in some cases. Special cases of ILP include binary variables (0-1 ILP) or mixed ILPs involving both integer and continuous solution variables (MILP). ILP solver technology has made significant progress during the last three decades, and powerful solvers (e.g., CPLEX, Gurobi) exist [80].

3.1.1 ILP for workflow scheduling optimization

Traditionally, ILP has been used for workflow scheduling [7], which requires the selection of services performing tasks that constitute components of a complex

scenario. Specifically, the workflow scheduling problem can be defined as follows. Given are the following:

1. A directed acyclic graph, $G(T, E)$, representing a complex scenario, where nodes T of graph G correspond to tasks that need to be executed, and edges E denote time dependencies between the tasks.
2. For each task $t_i \in T$ there is a set of services $S_i = \{s_{i0}, s_{i1}, ...\}$ each of which is capable of executing t_i but possibly on various conditions. Specifically considered parameters for each service s_{ij} could include execution time $l_{ij}(d_i)$ of data processed of size d_i, price $pr_{ij}(d)$ but could also include, e.g., average power $pow_{ij}(d)$ taken by execution of the service (presumably a given service runs on a particular computational device) or any other relevant metric.

Workflow scheduling requires the assignment of one particular service s_{ij} to each task t_i such that specific metrics are optimized. For instance, typical optimization goals include:

1. Minimization of workflow execution time L (determined by times of selected services and graph G with a budget bound B on the total cost of selected services) $\sum_i \sum_j pr_{ij} \cdot x_{ij} \leq B : \forall i \sum_j x_{ij} = 1$,
2. Minimization of the total cost of selected services $\sum_i \sum_j pr_{ij} \cdot x_{ij} : \forall i \sum_j x_{ij} = 1$, with an upper bound on the workflow execution time $L \leq L_{\max}$,
3. Minimization of workflow execution time and cost product $L \cdot (\sum_i \sum_j pr_{ij} \cdot x_{ij}) : \forall i \sum_j x_{ij} = 1$.

Furthermore, workflow scheduling formulations can assume fixed/a priori known data sizes d_i, in particular workflow nodes t_i, or could be subject to partitioning. In the latter case, input data to a whole workflow may be partitioned for parallel execution of some parallel workflow paths while may be subject to processing by other tasks obligatory for the whole data. Even the problem of data partitioning, especially in hybrid environments, is NP-hard and requires heuristic solutions for middle-sized or larger scenarios [6].

The aforementioned and similar ILP-based approaches can be used in various contexts/subdomains of parallel and distributed processing such as clouds [2–5] and HPC [6, 7, 81] in the context of scientific and business but also involving mixed scientific and business services [7].

3.1.2 ILP-based co-optimization for energy efficiency: crown scheduling

ILP is a flexible and powerful tool for modeling and solving complex resource optimization problems that combine multiple interdependent subproblems. Nevertheless, the solution space might easily get too large for medium to large-sized problem instances. In some cases, introducing additional, artificial, domain-specific

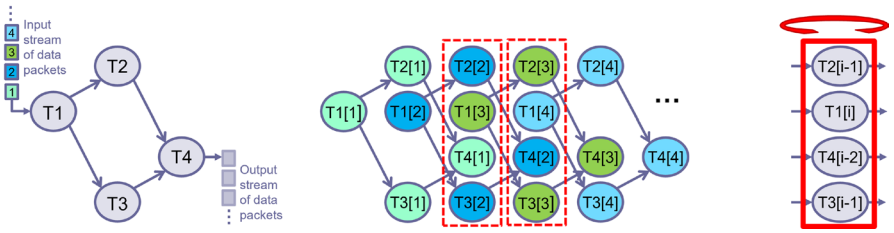

Fig. 2 A streaming task graph (left) with four moldable tasks and its software-pipelined execution (middle, time flowing from left to right) for a stream of input data packets, resulting in a repetitive steady-state pattern of independent streaming task instances (right). Adapted from previous work [32, 82]

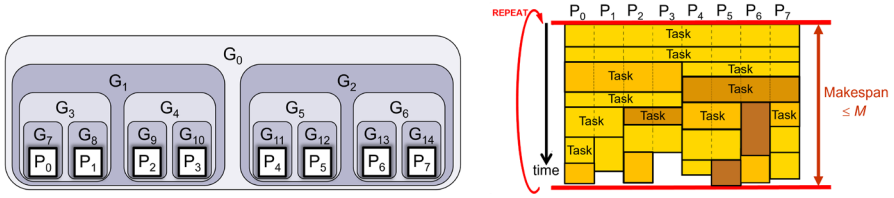

Fig. 3 Left: a crown (core group hierarchy) over eight cores. Right: a crown schedule of many tasks in a steady-state pattern for eight cores (horizontal axis); rectangles indicate tasks mapped to core groups and the colors indicate selected Dynamic Voltage and Frequency Scaling (DVFS) levels. Adapted from previous work [32, 82, 83]

constraints on the problem can significantly reduce the complexity of the solution space and make the use of ILP-based methods practical.

For example, in a series of recent works on *crown scheduling* [32, 82–87], ILP and MILP models have been used for energy-optimal, throughput-constrained software pipeline configuration of soft real-time streaming computations, modeled by a set or graph of moldable streaming tasks (see Fig. 2 (left)), for execution on homogeneous and heterogeneous multi-core CPUs with discrete dynamic voltage and frequency scaling (DVFS).

For input data packets arriving in regular time intervals, software-pipelined execution of the dataflow graph (Fig. 2 (middle)) results in the steady-state pattern of independent task instances shown in Fig. 2(right), for which we try to find a schedule such that a specific application-defined processing throughput (e.g., output packet data rate) is maintained and for which we try to minimize the overall energy cost.

Here we are faced with the complex combined problem of (1) allocating several CPU cores to each moldable task, (2) mapping each task to some set of that many cores for parallel execution, and (3) selecting the DVFS level for each task execution. Co-optimizing these problems together by a single ILP model is superior to a phase-ordered approach of deciding these highly interdependent problems one by one. In addition, (4) task (kernel) fusion for dependent tasks can be considered as a fourth problem for co-optimization [32].

The key to ILP complexity reduction is the introduction of a hierarchy of core groups to artificially limit the possible core allocations to (e.g.) powers of 2 only.

This is achieved by a binary decomposition of the group of all p cores into two disjoint subgroups containing one-half of the cores each. These are split in half again and so on until finally p singleton groups are formed that contain one core each and form the leaves of this group hierarchy (see Fig. 3 (left)).

This tree-shaped group hierarchy (i.e., the set of $2p-1$ applicable core groups) is referred to as the *crown* [83]; limiting mapping of moldable tasks to the crown's groups reduces the possible number of mapping targets (which impacts the number of ILP solution variables) from 2^p-1 to $2p-1$. Each core is thus a member of $\log_2 p$ core groups in the crown.

Then, a *crown schedule* for one round of the software pipeline's steady-state pattern is obtained by mapping tasks to the crown core groups only and by ordering the tasks on each core by non-increasing degree of parallelism (see Fig. 3 (right) for an 8-core crown schedule). Crown schedules avoid, by construction, any idle times on resources except at the end of the round's schedule and eliminate complex mapping and DVFS scaling interferences stemming from implicit barriers at the start of parallelized moldable tasks which are not uncommon in unconstrained schedules [85].

An ILP crown schedule is primarily described by the main binary solution variables which are usually of the form $x_{i,j,k}$, indicating that task j is mapped to core group i of the crown and executed at DVFS level k. Constraints such as $\forall j: \sum_{i,k} x_{i,j,k} = 1$ make sure that each task j is mapped exactly once and to one DVFS level. Further constraints are required, e.g., to bound the makespan of the crown schedule by the maximum time M permitted for one round to guarantee the required real-time data rate, while the accumulated energy costs of all tasks in the schedule form the objective function to be minimized. The tasks' power or time coefficients for different group sizes and DVFS levels can be obtained by microbenchmarking on the target architecture or by a theoretical model. In contrast with some other schedulers for moldable tasks in the literature, crown schedulers need not make any assumptions about the absence of speed-up anomalies for moldable tasks.

While also fast heuristics have been developed for crown scheduling [84], ILP crown schedulers have practical solution times for realistic problem sizes and outperform competing approaches [84]. While the additional crown restrictions might, in theory, introduce significant penalties on (energy) optimality in contrived corner cases, they are shown to have negligible impact in practice [85]. Generalizations of crown scheduling have been developed for considering DVFS islands [88] and heterogeneous multi-core CPUs such as ARM big.LITTLE-based architectures [89]. The generalization of crown scheduling to moldable stream computations on distributed systems with multi-core nodes (specifically, the device–edge–cloud continuum) is presented in [87]. The restricted flexibility of a few discrete DVFS levels can be relaxed by dynamic extensions [90]. An extension of crown scheduling for a more dynamic scenario of varying task workloads depending on input data packet types is described in [86], where multiple different optimized schedules for the steady-state pattern with different task workloads are computed off-line by ILP and switched at runtime depending on the current input packet type sequence. Crown schedules have also been shown to have robustness against unforeseen delays in individual tasks [82].

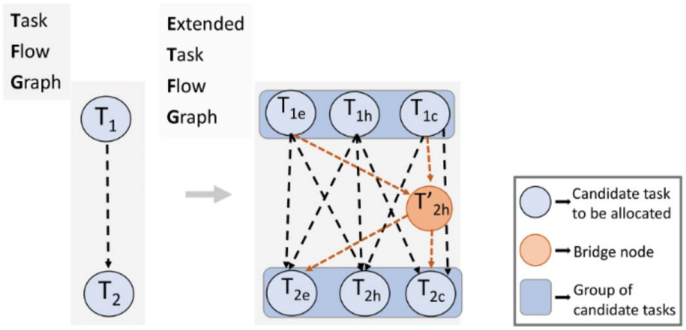

Fig. 4 Transformation of a TFG to its corresponding ETFG

Such problem-specific restrictions of the solution space of ILP models that trade better exact solvability for a marginal loss in solution quality (here, energy) are not uncommon. For example, for mapping moldable tasks to homogeneous parallel systems, Xu et al. [91] use a level-packing ILP model to solve a similar problem (DVFS not considered). A systematic experimental study of the effect of such restrictions in each of the four problem dimensions: resource allocation, mapping, DVFS selection, and, where applicable, task ordering (on each core), which also includes Crown scheduling and Xu scheduling as special cases, is given by Keller and Litzinger [92].

3.1.3 ILP for edge and cloud computing

Some other works in the literature optimize the overall latency [54, 56, 93], and the overall reliability [55] of the system. These works investigate the task allocation problem between edge, hub, and cloud infrastructures. To deliver an optimal task allocation while targeting performance and energy constraints, a mathematical programming-based framework was developed.

The framework takes as input an extended task flow graph (ETFG) which encapsulates the operating conditions and constraints for different devices and communication channels, and using mathematical models based on mixed integer linear programming (MILP) estimates the optimal task allocation for a given application.

Initially, the designer has to determine which tasks can run on which computational devices of the system, i.e., in this case, the edge, hub, and cloud. Then, the original task flow graph (TFG) of the application is transformed into a set of up to n nodes, referred to as candidate tasks, forming a group of candidate tasks in the ETFG, each representing the specific original task, running on the specific computational device. Each candidate task in the group encapsulates several parameters such as execution time, energy consumption, memory footprint, storage requirements, and the amount of data in case of offloading to another task.

Furthermore, parameters such as task dependency and communication cost (communication latency and communication energy) are needed to establish a direct connection between two candidate tasks. Besides that, a bridge node is added to the ETFG to represent the extra communication cost (both latency

and energy) if there is no direct communication between a pair of computational units, e.g., from the edge to the cloud and vice versa.

Figure 4 illustrates the TFG and its corresponding ETFG. It is important to mention that the difference between the TFG and the ETFG is that in TFG each task is required to be executed in order for the algorithm to be completed correctly, in contrast with the ETFG, where only one candidate task per group is executed.

The mathematical model uses a binary variable x_{ij} to indicate whether a task i will be allocated to be executed on the computational unit j ($x_{ij} = 1$). More specifically, this variable is responsible for selecting the best candidate task to be executed from a specific group of candidate tasks. In addition, a binary variable x_{ijk} is defined, which shows whether the outgoing edge $j \rightarrow k$ from a task i is selected ($x_{ijk} = 1$). The sets I and J represent the candidate tasks in the ETFG and different computational units (edge/hub/cloud) of the system, respectively.

The objective function is to minimize the overall latency of the system by taking into account both computation (l_{ij}) and communication latency ($l_{comm_{ijk}}$) as shown in Eq. 4.

$$\min \left(\sum_{i \in I} \sum_{j \in J} l_{ij} x_{ij} + \sum_{i \in I} \sum_{j \in J} \sum_{k \in J} l_{comm_{ijk}} x_{ijk} \right) \tag{4}$$

Equation 5 enforces that each task i must be allocated to be executed on one of the computational devices:

$$x_{ij} \in \{0, 1\}, \forall i, j \text{ and } \sum_{j \in J} x_{ij} = 1, \forall i \tag{5}$$

Equations 6 and 7 ensure that all paths of the graph derived by the optimal task allocation are connected, that is, if candidate task i and task $i + 1$ were selected in the optimal solution, then the corresponding edge in the ETFG will be selected as well.

$$x_{ijk} \in \{0, 1\}, \forall\, i, j, k \text{ and } \sum_{j \in J} \sum_{k \in J} x_{ijk} = nd(i), \forall\, i \tag{6}$$

where $nd(i)$ is the number of immediate descendants of task i.

$$\sum_{k:(ij \rightarrow k)} x_{ijk} - \sum_{k:(k \rightarrow ij)} x_{kij} = (nd(i) - 1) - na(i), \forall\, i, j \tag{7}$$

where $na(i)$ is the number of immediate ancestors of task i.

Finally, the rest of the equations model the resource constraints. In particular, the total main memory usage (Eq. 8) and the secondary memory usage (Eq. 9) should not exceed the specified memory budgets. Similarly, the total energy consumption, that is, the total computation (e_{ij}) and communication ($e_{comm_{ijk}}$) energy added together (Eq. 10), should not exceed the given energy budget of each computational device.

$$\sum_{i \in I} m_{ij} x_{ij} \leq M_j^{\text{bgt}}, \forall j \tag{8}$$

$$\sum_{i \in I} st_{ij} x_{ij} \leq St_j^{\text{bgt}}, \forall j \tag{9}$$

$$\sum_{i \in I} \sum_{k \in J} e_{\text{comm}_{ijk}} x_{ijk} + e_{ij} x_{ij} \leq E_j^{\text{bgt}}, \forall j \tag{10}$$

In the same manner, the aforementioned formulation can be easily extended to support different objectives, such as optimizing the overall energy or the overall reliability of the system. The framework has been evaluated using a real use case example. The use case uses an autonomous drone to inspect the power lines in a power grid system.

The evaluation results, presented in Fig. 5, illustrate the normalized execution time of the application across different task allocation strategies. The proposed optimization framework (denoted as F) was compared against scenarios where tasks, when feasible, were executed entirely on the edge (E), hub (H), or cloud (C) across four distinct configurations.

Each configuration (C1–C4) represents a unique combination of edge and hub devices, while the cloud remains the same across all configurations, as detailed in Table 2. These configurations were carefully selected to assess the framework's effectiveness in diverse deployment scenarios, considering that real-world

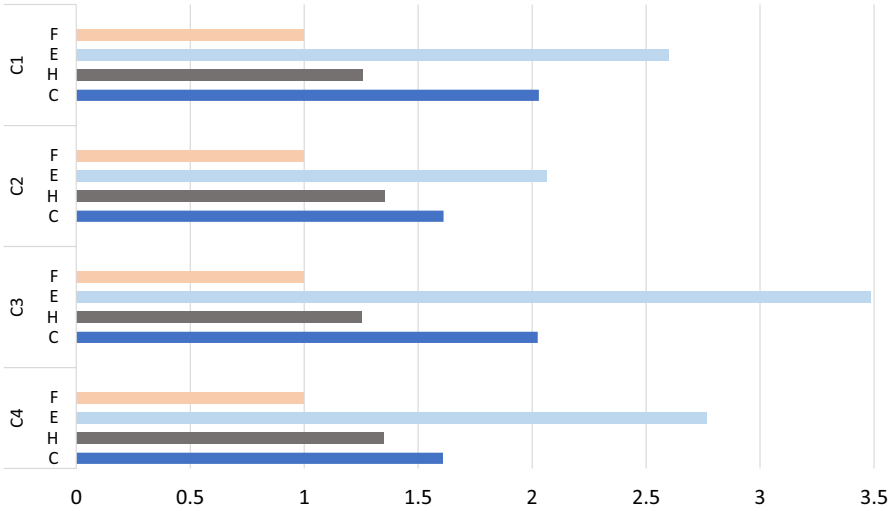

Fig. 5 Comparison of the optimization framework (F) solution, in terms of the execution time, with scenarios where tasks are executed entirely on the edge (E), hub (H), or cloud (C) across four configurations (C1–C4)

Table 2 Configurations of computational devices

Device	e/h/c	Configuration			
		C1	C2	C3	C4
Raspberry Pi 3 Model B	e			✓	✓
Odroid XU4	e	✓	✓		
Samsung Tab S2	h		✓		✓
Mi Notebook Pro	h	✓		✓	
HPE ProLiant DL580 Gen10	c	✓	✓	✓	✓

systems often involve heterogeneous devices with varying hardware and software capabilities.

Before conducting the evaluation, it was necessary to first profile the execution of each task on every available device. This step was essential for obtaining accurate measurements, such as execution time, energy consumption, and memory usage. By executing each task individually on the edge, hub, and cloud, detailed insights into their computational behavior were obtained. These precise measurements enabled a realistic and reliable assessment of the optimization framework's effectiveness.

As shown in Fig. 5, the proposed framework consistently outperformed fixed allocation strategies in terms of execution time. Across all configurations, it achieved substantial performance gains. For instance, in Configuration 3 (C3), it reduced execution time by up to $\sim 3.5\times$ compared to a fully edge-based allocation. Similar improvements were observed across other configurations, demonstrating the framework's adaptability and efficiency in minimizing latency by effectively leveraging edge, hub, and cloud resources.

These findings suggest that task allocation decisions were not intuitive, as execution time was influenced by multiple factors, including task-specific constraints, device capabilities, and network characteristics. Beyond identifying the optimal task allocation, the proposed framework enables extensive design space exploration with respect to different device configurations and their connectivity.

3.2 Greedy algorithms

Greedy algorithms (GrAs) construct solutions to combinatorial optimization problems bottom up in an iterative process, where a partial solution is extended step by step. In each step, one component of the not-yet-considered input is selected based on some (often heuristic) global ranking of these components at this step. GrAs are usually fast, but for most problems, their solutions can be suboptimal, as making a locally best decision at each step does not necessarily lead to an optimal overall solution. Only for a few simpler problems, greedy algorithms can guarantee the optimality of the derived overall solution, such as Dijkstra's shortest path algorithm [94] or Kruskal's minimum spanning tree algorithm [95].

A common use of greedy algorithms is scheduling a directed acyclic graph of dependent tasks for multiprocessors, or similarly, of a graph of dependent instructions for instruction-parallel processor architectures. The most common heuristic

technique for such problems is *greedy list scheduling*, a heuristic technique originally introduced by Graham [96], which imitates the limited-scope decision behavior of an online scheduler also for off-line scheduling problems, using a fixed priority list.

Greedy scheduling starts from an empty schedule and iteratively selects and schedules one task/instruction at a time from the task graph in topological order of dependencies. A task/instruction that has no unscheduled predecessors is considered ready for scheduling. It is placed as early as possible (as allowed by resource availability and pending data dependencies from predecessor tasks) on some free time slot on some execution resource at the current end of the partial schedule. In each step, the selection among all schedulable tasks is based on a global priority ordering, which might be based on the length of the longest accumulated delay along any path from that task/instruction toward a sink (result) of the task graph.

List scheduling is an approximation algorithm that guarantees that the resulting makespan is by at most a factor $(2 - 1/p)$ longer than the optimal makespan for a parallel target system with p equal processors. This holds both for independent tasks and dependencies between the tasks [97].

Greedy algorithms were among the earliest scheduling heuristics considered for instruction-level parallel processor architectures [98]. A survey of algorithmic techniques for instruction scheduling for use in compilers for Very Long Instruction Word processor architectures is given by Faraboschi et al. [99].

Heterogeneous earliest-finish-time (HEFT) algorithm [100], based on a DAG representation and targeting a parallel system with fully connected heterogeneous processors, executes in two phases: task prioritization and processor selection. The algorithm orders the tasks according to their priorities that are computed with upward ($rank_u$) and downward ($rank_d$) ranking, which are defined as follows:

$$\mathrm{rank}_u\left(t_i\right) = \overline{l_i} + \max_{t_j \in \mathrm{Succ}\left(t_i\right)} \left(\overline{c_{i,j}} + \mathrm{rank}_u\left(t_j\right)\right), \tag{11}$$

$$\mathrm{rank}_d\left(t_i\right) = \max_{t_j \in \mathrm{Pred}\left(t_i\right)} \left\{\mathrm{rank}_d\left(t_j\right) + \overline{l_j} + \overline{c_{j,i}}\right\}, \tag{12}$$

where $\mathrm{Succ}(t_i)$ is the set of immediate successors of task t_i, $\mathrm{Pred}(t_i)$ is the set of immediate predecessors of task t_i, $\overline{c_{i,j}}$ is the average communication cost of edge (i, j), and $\overline{l_i}$ is the average computation cost of task t_i. In the task prioritization phase, the task list is generated by sorting the tasks by decreasing order of rank_u. In the processor selection phase, the algorithm aims to insert a task in the earliest idle time slot between two already scheduled tasks on a processor by checking the time difference between the start time and finish time of two tasks to be larger than the computation time of the task to be scheduled.

LeTS [8] proposes a greedy heuristic algorithm for task scheduling problems on homogeneous multi-core systems. The algorithm considers both locality and load balancing to reduce the execution time of target applications. While it utilizes a DAG as its application model, the locality considerations target optimizing the executions on multi-core systems with no inter-core communication overhead.

In [9], authors use greedy algorithms to optimize virtual machine placement across data centers considering the optimization of a function taking energy and prices of distributing VMs. They evaluate random allocation (RA), next fit allocation (NFA), first fit allocation (FFA), best fit allocation (BFA), and worst fit allocation (WFA) and demonstrate that BFA performs the best.

A different use case—the buffer management problem using a greedy technique for packet buffering—is studied in [101]. Specifically, a switch can buffer (space permitting) packets incoming from input ports on their way to output ports, minimizing packet loss. Studies of resource availability versus performance were performed. The authors show that a greedy algorithm cannot be better than 2-competitive and propose a semi-greedy approach with a competitive ratio of 17/9.

In paper [102], authors propose the greedy firefly algorithm (GFA), where the greedy approach is used for local searching improving convergence and efficiency of the firefly method for scheduling jobs in an IoT grid environment, showing its benefits in obtaining makespans better and execution times lower than tabu search (TS), genetic algorithm (GA), and the standard firefly algorithm.

Greedy approaches are used for scheduling in high-performance computing systems. For instance, in paper [103] authors present an energy-aware greedy algorithm with particle swarm optimized parameters for scheduling malleable jobs (the number of servers/processors allocated can change over time). Job stretch times as well as average server power consumption are reduced (power offs are considered).

In paper [104], the minimization of makespan of workflow DAGs stemming from multifrontal factorizations of sparse matrices is considered. The authors propose a novel GreedyFilling 2-approximation heuristic and demonstrate (using synthetic and application workloads) that optimized variants and GreedyFilling outperform traditional algorithms. In work [105], the authors minimize parallel application execution time with an upper bound on the total power consumption of computational devices used in a heterogeneous cluster with multi-core CPUs and GPUs. For that, the problem of selection of the computational devices is considered, which can be generalized to the 0/1 knapsack problem for which, in real tests, a greedy approximation algorithm selection of successive elements for packing is used. Increasing speed-ups vs ideal and decreasing execution times for increasing power bounds are shown for a system with five distinct GPU models.

3.3 Dynamic programming

Dynamic programming (DP) is an important algorithmic paradigm for solving combinatorial optimization problems to optimality. It requires a method for decomposing a problem instance into *independent* sub-problem instances, storing and retrieving solutions to sub-problems, and building solutions to larger problem instances from solutions to smaller problem instances. The space of partial solutions is constructed bottom-up so that each partial solution is computed only once. Some problems, where dynamic programming is applicable, include the knapsack problem, the matrix chain product problem, single-source shortest paths, the optimal triangulation problem, and many others. The solution space is usually organized as

a multidimensional table. In contrast with exhaustive search or branch-and-bound approaches, dynamic programming algorithms determine a solution bottom-up, by building optimal solutions to larger sub-problems from already computed and stored optimal solutions to smaller sub-problems. The space of sub-problems and their solutions is typically organized as a multidimensional table.[4] In contrast with an exhaustive search, DP may speed up the search for an optimal solution by orders of magnitude, at the expense of increased space requirements.

3.3.1 Optimal code generation for instruction-level parallel processors

Dynamic programming-based techniques have been used extensively in compilers for generating optimal target code (instruction selection, resource allocation, and instruction scheduling) for instruction-level parallel processor architectures (pipelined, VLIW, EPIC, DSP), where the time and space requirements are accepted in cases where high code-quality (time, energy, size) is important, such as compiling performance-critical DSP code. For example, DP-based instruction scheduling methods for instruction-level parallel architectures have been developed for acyclic code (basic blocks, extended basic blocks, traces) [106] and for loops, i.e., modulo scheduling [107].

More recently, however, the compiler back-end research community has mostly switched to integer linear programming [108–112] and constraint programming [113, 114] as the main technology for optimal code generation, in particular, due to their solvers becoming much more powerful during the last two decades (so that also realistic problem sizes can be solved to optimality) and their greater flexibility in problem formulation (DSL and tool chain support). For certain types of problems and cases where the solution space to be constructed by DP is not excessively large or can be folded by exploiting symmetries, DP might compete well with ILP/CP-based techniques performance-wise [111]. Recent surveys of such approaches in target code generation can be found in [115] and [116].

3.3.2 Dynamic programming for distributed processing

The dynamic programming approach is used for decision-making in fog environments where data should be processed (fog, cloud) [117], including the introduction of periodic randomization into offloading (fog, cloud) decisions of a dynamic programming offloading algorithm [118]. In [119], it was used in the context of optimization of data encryption with consideration of timing constraints and energy costs. In the cloud resource and provider selection context, dynamic programming has been selected for the maximization of the total computing capacity of the selected

[4] Note that DP goes beyond a simple hashtable-based automated software cache memoizing results of the search function in an exhaustive search, in that the latter is constrained by the quality of the problem-unaware hash function and that the usually depth-first construction order of exhaustive search is generally different from the usually breadth-first construction order applied in DP algorithms, which more likely avoids capacity misses.

instance types subject to budget as well as instance types proportional constraints (specifying the required proportion of specific cloud instance types) [120].

3.4 Nature-inspired optimization algorithms

3.4.1 Genetic algorithms (GA)

In paper [121], several algorithms were investigated for workflow scheduling assuming potential failures of services. Specifically, before a workflow is to be executed, services are selected optimizing a global criterion involving specific resources, as mentioned in Sect. 3.2. However, when the workflow is executed, some of the previously selected services might have become unavailable or might fail, which results in the need for workflow rescheduling, taking into account the already executed services and their contribution to the workflow execution time, cost, etc. Rerunning a scheduling algorithm can be costly. The paper compares approaches such as integer linear programming (ILP) formulation, divide-and-conquer approach partitioning the workflow into subgraphs and using, e.g., ILP, genetic algorithm (GA), and heuristic GAIN starting with a valid solution substituting services for optimized schedules. Out of the proposed solutions, GA appeared to be the slowest one. On the other hand, it has advantages such as no need for configuration (optimizing the starting solution can speed it up considerably) and the possibility to use nonlinear constraints and optimization goals.

Generally, GAs are often used for scheduling such as for HPC systems [11], specifically under power constraints which is of interest to the HPC community, data center optimization and maintenance [12], also cloud systems in which HPC workloads might be run [122], scheduling workflows in the cloud systems [13]. An interesting use case is using GAs for performance tuning of an HPC benchmark [123].

3.4.2 Particle swarm optimization (PSO)

Particle swarm optimization (PSO) mimics the movement of organisms in a bird flock or fish school to solve an optimization problem by having a population of candidate solutions, i.e., particles, and moving these particles in the search space based on a mathematical formula formed by the particle's position (x_k) and velocity (v_k) [14]:

$$x_{k+1}^i = x_k^i + v_{k+1}^i. \tag{13}$$

$$v_k^{(i+1)} = w_k v_k^i + c_1 r_1 \left(\text{pbest } t_k^i - x_k^i \right) + c_2 r_2 \left(\text{gbest } - x_i^k \right). \tag{14}$$

where c_1 and c_2 are constants, r_1 and r_2 are normal distributions within the range of [0,1], w_k is called an inertia weight to adjust the search ability of the solution space, pbest represents the best position obtained by the ith particle since the beginning (particle best), and gbest is the best position of all the particles (global best).

Pandey et al. [15] present a PSO-based heuristic algorithm to schedule tasks of an application workflow to cloud resources by considering computation and data transmission cost. The particles represent tasks in the application workflow (represented by a directed acyclic graph) and compute nodes (processors) assigned for the execution of those tasks. The algorithm utilizes three main data structures: task processor execution cost (TP), cost of communication between processor pairs (PP), and input/output size of each task (DS), and solves the scheduling problem by a basic PSO method.

Gill et al. [16] extend the resource scheduling by taking into account execution cost, time, and energy consumption as well as quality of service (QoS) parameters including availability, reliability, and resource utilization. The authors present a set of test cases that perform pair-wise comparisons to evaluate the impact on the constraints.

Potu et al. [17] propose an extended particle swarm optimization (EPSO) algorithm with an extra gradient method to optimize the task scheduling problem in cloud–fog environments by considering resource utilization, response time, and latency constraints.

There have been other swarm intelligence algorithms inspired by different species to tackle optimization problems [124, 125]. While they present valid metaphors based on population behaviors and evaluate mathematical models, most of them lack novelty and can be seen as a form of PSO [126]. Therefore, we prefer not to include them in our paper.

3.4.3 Ant colony optimization (ACO)

Ant colony optimization (ACO) is a metaheuristic algorithm inspired by the pheromone tracking of the ants to reach the desired destination [18, 127].

Kumar and Venkatesan [128] present a hybrid genetic algorithm–ant colony optimization (HGA-ACO) for multi-objective task allocation problems in a cloud environment. The authors propose a utility-based scheduler that identifies the task order and appropriate cloud resources by considering response time, completion time, and throughput objectives formulated as follows:

$$RT_{ir} = \sum_{i=1}^{M} subt_i + wt_i, \quad 1 \le i \le M, \quad 1 \le r \le k \tag{15}$$

$$ET_{ir} = \sum_{i=1}^{M} ft_i - st_i, \quad 1 \le i \le M, \quad 1 \le r \le k \tag{16}$$

$$Th_{ir} = \sum_{i=1}^{M} sucs_i/T, \quad 1 \le i \le M, \quad 1 \le r \le k \tag{17}$$

where T is total time for execution, $subt_i$ is submission time, wt_i is waiting time, st_i is starting time of task execution on resource r, ft_i is finishing time of task execution on resource r, $sucs_i$ are completed tasks, RT_{ir} is response time of task i on resource r,

ET_{ir} is completion time of task i on resource r while Th_{ir} denotes throughput of the resource r.

Jia et al. [129] propose a scheduling system for cloud workflows based on an adaptive ACO algorithm. The system consists of four main components including an estimation module, scheduling module, reservation module, and execution module. The *estimation module* presents a task execution time estimation model by considering both CPU computation and memory access of parts of the tasks. For the CPU computation part, the degree of parallelism and the speed-up of a task are utilized, while the memory access part is estimated by considering the relationship between memory size and the upper bound of memory demand. Overall execution time is calculated with the following formula:

$$EXE_i^{j,k} = \left(\frac{ts_i \cdot \left(1 - pt_i\right) + \frac{ts_i \cdot pt_i}{sc \cdot \min\left(ub_i, ms_j\right)}}{\min\left(cn_j, dp_i\right) \cdot su_i} \right) / \left(1 - deg_k\right) \tag{18}$$

where ts_i is the task size, pt_i is the percentage of data processing time of the task, sc is the scale of memory size to execution time, ub_i is the upper bound of memory demand, ms_j is the memory size of VM, cn_j is the computing capacity of virtual CPUs in VM, dp_i is the max degree of parallelism of the task, su_i is the speed-up of the task, deg_k is the performance degradation of the VM instance.

With the estimated execution times, the *scheduling module* finds a scheduling for the tasks based on the following problem definition:

$$\begin{aligned} &\text{minimize} \quad \mathrm{T\,C} = \sum_{k=0}^{l-1} up_j \cdot \left\lceil \left\lceil \frac{\mathrm{L\,E\,T}_k - \mathrm{L\,S\,T}_k}{\tau} \right\rceil \right. \\ &\text{subject to} \quad \mathrm{T\,T} = \max\left\{ET_1, ET_2, \ldots, ET_n\right\} \leq D \end{aligned} \tag{19}$$

where TC is the total cost, TT is the total execution time, *LST* is the lease start time of a VM instance (when it first receives a task), LET is the lease end time (when it finishes the last task scheduled onto it), τ is the unit time to lease a VM, *up* is the unit price of VM.

After getting a near-optimal schedule, the scheduling module sends the VM requirements to the *reservation module* and gives the execution scenario to the *execution module*.

3.5 Game theory

Game theory (GT) dates from nineteenth century, while the modern game theory is over 70 years old. Still, it is very popular and widely applied in a real life. It observes real-life situations as games [130].

Players make decisions in a game taking into account the situation, rules, and consequences. In this setup, a player may be any entity with the interest of the result of the game. Each player (there will be at least two of them) aims for an optimal decision in the observed situation, independently of other players, taking into

account payoffs. In this sense, game theory may be observed as a discipline that explores optimization strategies. Furthermore, game theory is based on mathematical modeling theory and provides a theoretical and formal problem and solution description.

Observing the optimization in a game, a significant situation of interest for the game theory is when players made a decision (move) and the outcome is clear. This situation is called Equilibrium. At the equilibrium point, there is a tendency to minimization of the players' regrets. The happiest scenario for all players in a game is called Nash equilibrium or no-regret situation [131].

There are various categorization of games, while the primary one is to cooperative and non-cooperative games. In a cooperative game, groups of entities may appear as players where group modeling may play an additional role (e.g., Shapley Value defines a fair distribution among group or coalition members [132]), but each group is observed as a single player. Non-cooperative games involve only individual players.

Another categorization is to zero-sum and non-zero-sum games. In zero-sum games, resources are exchanged between players without changing its value. This means that if a player wins resources of a particular value, the other players will loose in total the same value of the resources. In simultaneous move game, a player will adapt decisions simultaneously based on the situation that changes between moves, while in a sequential game, a player can plan in advance own moves and moves of other players and make strategic path of moves in the game. Finally, a game may be played as one-shot game without a possibility to repeat some moves or may be repeated while the outcome changes due to in-process learning by players.

There exist also different game strategies in Game Theory: a strategy that aims for maximizing the payoff risking to loose everything (maximax strategy), a strategy that aims to lower the risk by taking lower payoffs and trying to maximize the pay of from so-selected moves (maximin), etc. However, there are even hybrid strategies that combine two or more basic ones.

Game theory is often a natural choice for dealing with trade-offs, optimization, and management in resource utilization in a broadest sense [133, 134], as well as in system engineering [135, 136] and large distributed systems in the most general sense [137, 138], while dedicated contribution focuses on more specific domains such as cloud computing [139], Internet of Things (IoT) [140], or cellular [141] and wireless networks [142]. Number and diversity of problem and solution models constitute a large and still open research and application area.

While above referred surveys and reviews provide a big picture about the space game theory covers, we can illustrate its application by some illustrative cases. In [20], authors consider renewable energy for utilization by cloud data centers. In a smart grid cloud architecture, authors investigate the grid power dispatching to cloud data centers. The main grid power dispatching to cloud data centers is modeled as a non-cooperative game and used for calculation of optimal level of power to be delivered to each data center.

In paper [19], authors employ game theory-based virtual machine migration with PSO for energy sustainability in cloud data centers, minimizing energy consumption while using renewable energy. Another cloud-related game-based solution [143] is

focused on task scheduling for energy management in cloud computing. Authors provide scheduling algorithm based on game theory model.

Another usage of non-cooperative games is applied for computing unloading strategy of massive IoT devices in mobile edge computing [144]. A cooperative game model is applied in [145] for selecting energy-efficient access points for sensors in the Internet of Things. Another cooperative approach is applied in [146] for coalition creation based intelligent three-level structure toward the maximizing the profit from cooperation with other resources. In paper [147], authors incorporated a game theory approach for detecting vulnerable data centers in cloud computing network. In the context of high-performance computing, the authors of paper [23] used a game theory-based solution to minimization of the energy consumption and the makespan of computationally intensive workloads in power-aware scheduling and mapping of tasks onto heterogeneous and homogeneous multi-core systems. The problem was formulated as a cooperate game.

In paper [148], authors use game theory to study the behavior of agents in a volunteer computing system. They assume that providers of computations arrive and depart and offer power in return for rewards. The authors model and analyze a stochastic game and conclude that in case of a large number of volunteers and a homogeneous system the system is stable and the total power received by the central server is proportional to the offered reward. Studying the arrival and departure rates and system parameters, they observe that players maximize their utilities when the average number of players present in the system is typically between 1 and 2 which corresponds to smallest competition.

A game model that is often applied in resource management is Stackelberg game. This is a sequential model where there are two types of players: leaders and followers. Leaders are trying to maximize profit first. Followers are following the leader and trying to maximize their own profit. Stackelberg model is used in [149] to deal with resource allocation and pricing issues in a mobile edge computing system. The solution decomposes multiple resource allocation and pricing problem into a set of sub-problems, each of which only considers a single resource type. In this model, mobile edge cloud is modeled as a leaders and end users are followers. Stackelberg games are used also to manage task scheduling in edge computing [150], but in this case as a repeated game. In the Stackelberg model, users are modeled as short-term leaders and edge service provider are long-term followers. Two-layer resource allocation Stackelberg model aimed to provide high quality of services while balancing the benefit requirements of all participants in the cloud network convergence service system. Stackelberg game on the second level includes a cloud-edge sub-game and an edge-end sub-game [151].

Another very applicable game theory is evolutionary game theory (see the survey for the applications [152]). Behavioral processes involve interaction of multiple players, and the profit (payoff) of any one of these players depends on how its behavior (strategy) interacts the of others. So it is impossible to observe players individually but rather as a part of population in which they live. Specifically, population members do not necessarily reason or even make explicit decisions.

Matching theory is a model among mutually related entities in a beneficial interaction [153]. The matching theory assigns a set of entities to one another while

satisfying the preferences of entities [154]. The resource allocation problem is represented in a cloud computing distributed system as a matching game (MG) using two sets of players: workflow tasks and resources in the cloud or fog. The game aims to match each resource to a task. The result is a bilateral resource allocation agreement representing the players' preferences over each other. A valid matching is stable if it satisfies the properties of a one-to-one MG [155, 156]: Each task is allocated to exactly one resource, a resource can execute just one task. The matching does not contain blocking pairs of tasks and resources that prefer matching to each other rather than their current assignments [29].

Additionally, Sharghivand et al. [21, 157] utilized the MG and designed a scheduler for data analytics tasks execution at the edge based on the device and application preferences toward lower response time and cost. This method is just proposed for the edge and fog computing environment.

In addition, Mehran et al. [22] proposed an MG-based method, named C3-Match, for scheduling asynchronous data processing applications in the computing environment. Thereafter, C3-Match is evaluated and compared with other workflow scheduling methods in terms of application execution and data transmission times on the C^3 cloud and fog computing environments [158].

The complex nature of game theory allows extending its theory and application by involving other disciplines and approaches such as reinforcement learning [137], see also Sect. 3.6.

3.6 Reinforcement learning

Reinforcement learning (RL) has been identified as an approach that, on the one hand, is a modern machine learning paradigm, and on the other, it has already been proposed to optimize processing in all the considered contexts/domains, as will be shown next.

Acting along supervised and unsupervised learning, in the RL approach an agent performs actions to maximize a cumulative reward. In essence, the process is to find a trade-off/balance between already gained knowledge and exploration of an unknown space.

As an example, this approach has been used for the problem of workload optimization among data centers in which incoming workloads are managed among servers considering the limited variation in energy that can be supplied from fuel cells [24]. In this approach, deep Q-learning is used and evaluated positively using real-time traces. Deep reinforcement learning (DRL) combining deep neural networks (DNNs) and RL has been reviewed as a modern approach to scheduling in cloud environments [159]. Several RL-based algorithms have been reviewed with objectives: response time, cost, energy consumption, service latency, load balancing, makespan, etc. It has been argued that DRL-based algorithms have been mainly tested in laboratory environments and several challenges need to be faced for those to be used in large-scale cloud environments including the large computational power required to train for multi-cluster large systems, difficult-to-assess worst-case

results and additionally the problem of unexplainability, specifics of solutions leading to local rather than global optimization.

DRL is also used for the placement of application components in a distributed fog environment [25]. Specifically, IoT, edge, and cloud layers are distinguished, the last two forming the fog computing environment. On the application (input) side, there are several IoT applications, each of which is assumed to be a DAG with processing requirements on the graph node and data sizes assumed for internode transfers. It is further assumed that each DAG's node, i.e., a task, can be executed locally (on an IoT device) or relegated for execution on the fog side. Various execution and optimization models can be distinguished, specifically a weighted cost one in which there is minimization of weighted execution time and energy consumption of tasks of IoT applications. The energy of components executed on the IoT side only is considered. X-DDRL (based on an actor-critic framework) is used to solve the problem with a pre-scheduling phase (tasks are ranked and sorted for execution) and placement phase. DNNs are used by both the actor and the critic as the function approximator means. It has been shown through simulations and testbed experiments that the solution outperforms greedy, Double-DQN, PPO-RNN, and PPO-No-RNN for execution time, energy, and weighted cost optimization.

In the context of high-performance computing, software for job cluster scheduling using reinforcement learning has been proposed. In [26], the authors presented DRAS-CQSim—a solution with deep reinforcement learning for scheduling that allows automatic learning of the customized scheduling policies. Deep reinforcement agent for scheduling (DRAS) cooperates with the CQSim simulator that reads job arrival events from a job log. Deep q-learning and policy gradient can be used.

Learning the dynamism of events such as gaining back control of computers in a volunteer computing environment can also be dealt with in RL, as proposed in [160]. The authors considered an RL approach aiming at learning which resource to schedule a job, or for the job to remain in the queue considering energy and overhead. Several levels of resources can be considered including computer (hours), cluster (several clusters, including taking into consideration weekdays), and system (either running or staying in the queue). The authors tested the impact of exploration vs exploitation and concluded that the cluster approach with $\epsilon = 0.1$ and $\sigma = 0.8$ performs the best saving between 30% and 53% of energy, depending on overheads.

Tu et al. [27] designed a method based on DRL to predict task dynamic information in real time based on the observed fog network status and the server load.

Additionally, it shall be noted that very recently federated learning (FL) and RL were used together for resource allocation. For instance, in paper [161], authors aimed maximization of the network spectrum energy efficiency and combined federated learning and deep reinforcement learning. They proposed an asynchronous federated learning framework with local model training and the client–server architecture. They demonstrated usefulness of the approach for an Internet of vehicles environment. A meta-federated reinforcement learning framework was proposed in paper [162] for the problem of distributed resource allocation aiming at maximization of energy efficiency while ensuring quality of service for users. Local users optimize transmit power and assign channels using neural network models trained locally. Authors demonstrated that the approach outperforms the traditional

decentralized reinforcement learning in terms of the optimization goal and convergence speed.

3.7 Summary and conclusion

This section explores various problem formulations used in resource-aware optimization for parallel and distributed computing. In particular, we describe integer linear programming (ILP), a robust method for modeling and solving combinatorial optimization problems. Applications of ILP, such as workflow scheduling and energy-efficient crown scheduling, demonstrate its versatility in optimizing performance, energy, and cost in diverse contexts like cloud and edge computing. Subsequently, we also discuss other techniques, including greedy algorithms, which are efficient for certain scheduling problems but may yield suboptimal results in complex scenarios. Dynamic programming is presented as a powerful approach for solving problems to optimality by building solutions incrementally, especially in resource allocation and task scheduling. Nature-inspired algorithms like genetic algorithms, particle swarm optimization, and ant colony optimization are emphasized for their adaptability in dynamic and multi-objective optimization problems. The section concludes with methods like game theory and reinforcement learning, which provide innovative solutions for resource allocation and workload optimization in distributed systems.

In conclusion, the section provides a comprehensive overview of optimization techniques, illustrating their strengths and trade-offs. These methods, tailored to specific scenarios, enable efficient resource management in distributed computing environments. The diversity of approaches ensures adaptability to various challenges, paving the way for more refined and scalable solutions in the field.

4 Resource-aware applications in various domains

Parallel and distributed computing is required by several applications and is present in different computing system domains. Figure 6 illustrates the architectures of the computing domains. The *Data Center* (DC) is the lowest level computing layer and is present in all other distributed architectures. It refers to centralizing the computing resources in a single room/building with several levels of availability, reliability, security, energy supply, cooling, and other important infrastructure aspects to deliver working guarantees, close to being 100% online 24 h per day. The optimization challenges are concentrated on hardware management and power systems.

The *cloud computing* architectures build upon data center infrastructures that can be, or are not, geographically distributed. The virtualization technology enables abstract hardware and allows hardware sharing to multi-tenants. However, it has performance degradation due to the software abstraction layer. Optimizations are crucial to avoid SLA violations and to achieve QoS. The challenges in this architecture domain are many, which are from virtual machine (VM) placement to resource consolidation and isolation.

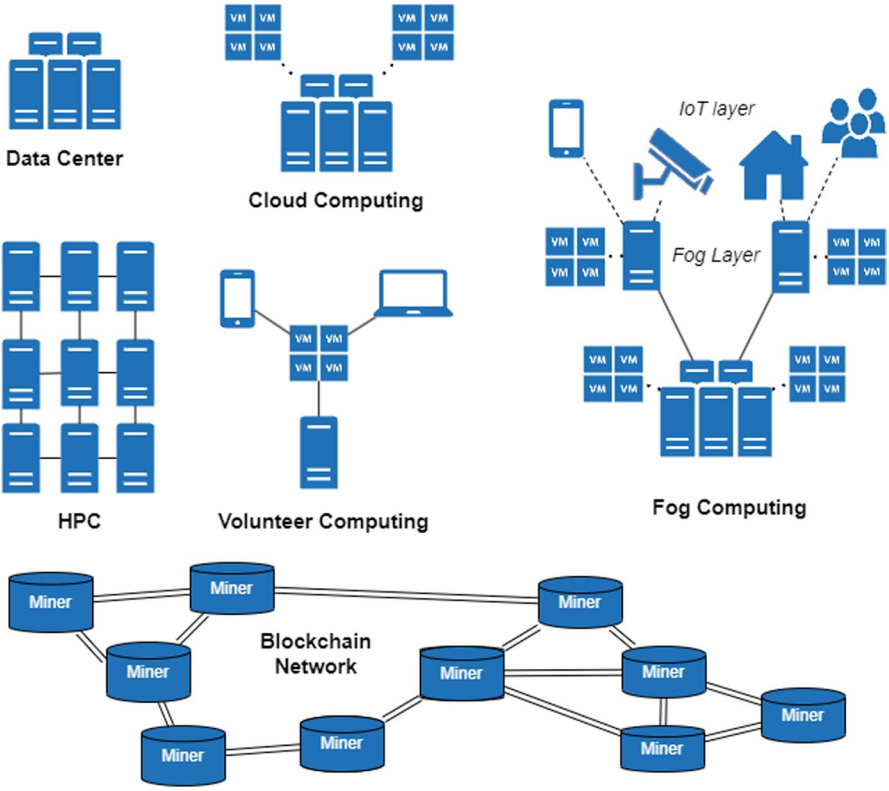

Fig. 6 Distributed computing architecture domains

The *HPC* architectures also build upon a data center infrastructure, usually with few hardware optimizations (e.g., memory, storage, network, and computing). Unlike cloud architectures, they are designed for a specific set of applications and specialized users that interact with a job management system (e.g., Torque and Slurm). Jobs are not expected to share the same hardware since it is expected that the application is optimized to fully utilize the computing resources. The optimization challenges are broad, transcending the job management systems and going beyond the inherent parallelism exploitation challenges.

The *volunteer* computing architecture has decentralized and heterogeneous computer resources interconnected via the Internet, which usually has less security, network bandwidth, and reliability than data center networks. Implementing optimizations in this environment is important and challenging at the same time. There are several middlewares used to manage these computing resources for applications that must be designed for such kinds of architectures. The optimizations in the middleware are most related to resource scheduling while application optimizations focus on parallelism exploitation on the volunteer hardware.

The *fog* architecture can be layered in three levels: an IoT layer, where small computing resources are placed, a fog layer, where more robust computer resources are placed close to the edge, and a data center layer, where complex computation and big data storage are performed. Fog computing was introduced to avoid unnecessary data being transferred via the Internet and to improve latency in several applications. The resource optimization challenges may be applied in the three layers since the architecture and its applications are built with this layered infrastructure in mind. For instance, it is a challenge to improve communication and resource provisioning.

The *blockchain* is one of the most recently adopted distributed ledger technologies within the ICT domain. The balance point among scalability, security, and decentralization has been researched for decades. The main challenge in Blockchain-based distributed systems is the lack of a central entity (e.g., TTP, PKI, Leader, etc.) that controls data flow and information security within the system. Among tens of reported design decisions, concerned about achieving suitable blockchain-based solutions for different applications, is the level of decentralization a given application can tolerate to maintain its security and meet scalability requirements.

4.1 Data centers

To integrate the DC with the smart grid, all the energy-consuming subsystems are represented and abstracted for dynamic control. Thus, besides the workload scheduling on servers, which is the heart of cloud computing, cooling system operation control and energy storage devices are considered [163–165] to define methodologies to allow the DC integrating energy marketplaces on day-ahead and intraday sessions, as well as to provide ancillary services, and which are based on predictions at various granularities that estimate the DC operation over a time window equivalent of the marketplace session. Thus, the optimization problem becomes very complex due to the introduction of the time component, which increases the number of variables. Approaches such as [166] use a genetic heuristic to compute an approximate scheduling plan of the problem, while other methods use mathematical solvers to determine a solution to the scheduling problem, such as [167] used in [168].

The DC operation scheduling problems for electrical energy efficiency contain models for discrete and continuous systems whose operation has to be scheduled over a time window, thus increasing the complexity of the models and leading to problems of class MINLP that sometimes can be relaxed to NLP by eliminating integer variables or PINLP, by discretizing the entire system.

Thermal-aware workload scheduling adds a new constraint to the optimization problem defined for server consolidation and dynamic server allocation problem by introducing the thermal map of the server room and aiming to minimize the hot spot temperatures. One of the first approaches for thermal efficiency in a DC is presented in [169]. The authors describe a workload scheduling algorithm that considers the temperatures of the servers when scheduling workload and promises a drop of 25% in cooling costs. Moreover, a complex thermodynamic model of the server room that defines the connection between the server's exhaust, inlets, and cooling system is proposed in [170]. The model is less computationally expensive

than a CFD simulation for determining temperatures in case of a thermal-aware workload scheduling policy. In [171] and [172], the DC model is defined in a complex manner by adding an energy storage device for both electrical and thermal energy and proposing a set of different policies and algorithms for shaving power peaks and reducing overall DC energy costs that are presented. Other approaches for thermal-aware scheduling are presented in [173–175], and [176], where the authors present various algorithms and policies for VM placement considering thermal constraints. [177] describes a VM placing algorithm that considers the server room temperatures, while [178] proposes an optimization problem based on a thermodynamic model that merges energy footprint reduction with thermal exchanges. Finally, [179] proposes a thermal-aware VM consolidation mechanism that uses the heat recirculation matrix defined in [170] and a set of bio-inspired algorithms to compute the optimum assignment of tasks that minimizes overall DC energy consumption.

The DC operation scheduling problems for thermal energy efficiency add an extra layer of complexity from a mathematical point of view due to many variables and nonlinear equations due to the representation of the thermodynamics processes within the server room. Furthermore, some approaches leverage simulators, such as Computational Fluid Dynamics (CFD) engines, that define a function for temperature estimation but cannot be included in traditional solvers. Even if there is MINLP, some problems of this class have equations defined as Black Boxes that make them extremely difficult to solve.

Finally, the latest trend in the DC industry is to transform DCs into thermal energy producers by harvesting the thermal energy generated by the IT equipment, raising its temperature using heat pumps, and injecting it into the district heating network. Dharkar et al. [180] propose such a system for the data center of the mathematics department at the Purdue University, a large data center that has an hourly consumption of about 1MWh. Houbak-Jensen et al. [181] propose a model of a heat reuse system composed of a heat pump and thermal tanks as energy buffers to increase the system inertia. A set of feasibility studies for data center heat reuse in Norden Europe is performed by [182] and [183]. Finally, Antal et al. [184] propose a workload placement strategy to maximize the heat reuse potential of the DC, by slowly raising the temperature in the IT room while keeping it in operable thresholds. The thermodynamic model is based on a neural network trained using CFD simulations, leading to a high-complexity optimization problem solved using genetic algorithms, since conventional solvers could not model the problem and determine a solution.

Approaches for resource allocation problems for DC heat reuse can be characterized by the complexity due to the thermal models used to estimate the heat reuse capabilities and temperatures in the server room. These models must be simple enough to allow fast calculation and complex enough to provide enough accuracy to avoid equipment malfunction due to high temperatures. Thus, many approaches rely on machine learning techniques to estimate temperatures due to workload placement, leading to complex mathematical problems with equations defined as black boxes. Thus, classical algorithms that rely on differentiation and gradient cannot be applied, and relaxations or heuristics are used to solve the scheduling problems.

In the context of data centers, the dynamic programming approach has been found to be a viable approach for the optimization of several resources, for instance, energy cost, delay cost (reflecting service quality), and switching cost (in the context of servers' active/idle mode) in [185] considering off-line and online approaches; reliability, availability, cost [186]; communication latency, fault-tolerance, and CPU usage in the context of routing in large-scale data centers [187]; saving energy and cutting emmissions [30].

4.2 Cloud computing

Cloud computing is a heterogeneous environment that abstracts an enormous hardware and software ecosystem and provides users a potentially infinite pool of resources. It is divided into several service models (e.g., IaaS, PaaS, SaaS, and others) and deployment models (private, public, hybrid, and community) [188, 189]. We classify resource allocation techniques on two levels: (i) physical resource management and (ii) workflow management–related to software resource planning and allocation. While physical resource allocation is managed by IaaS tools [190], workflow management is managed by PaaS or SaaS tools [31]. Moreover, different deployment models may also impact how resource allocation strategies and algorithms are designed.

To begin with, we classify the physical resource allocation in server consolidation techniques, VM allocation and Dynamic Server Allocation Problem. The server consolidation aims to minimize the DC energy demand by determining an optimum allocation of tasks or VMs such that the number of active servers is minimized. Other constraints of the problem aim to minimize the number of migrations and to maintain the SLA and QoS of the services [191]. The selection of the servers that remain active is done based on various resource constraints, such as CPU and RAM [192], SLA [193], timing constraints [194], maximum bandwidth [191], and last but not least network related constraints presented in [195].

It can be shown that the problem of allocation of VMs to the servers by the constraints presented above (CPU, RAM, bandwidth, etc.) is an NP problem. This can be proved by performing a problem reduction to the bin-packing problem [196]. Consequently, the deterministic algorithms cannot run for large configurations of DCs. The research was focused on enhancing VM consolidation by developing heuristics and approximation algorithms to replace the first fit decreasing approach used by the first consolidation systems [197–199] and [200].

The dynamic server allocation problem (DSAP) [201] represents a well-known proactive optimization technique that aims to plan the DC server operation over a time period in the future by planning the workload deployment carefully, so that when a set of tasks is finished, they free a set of servers that can be turned off without performing further migrations. One of the initial solutions to the problem is presented in [201]. Similar approaches that use workload forecasting techniques to enforce proactive scheduling are presented in [202, 203], and [204]. Furthermore, the authors of [203] represent the optimization problem as a mixed integer nonlinear problem for scheduling server workload over a time interval. This DSAP class of

scheduling problems is more complex than the server consolidation technique, most of the approaches being classified as MINLP since a continuous dimension of time is added to the classical resource allocation problem. However, by discretizing the time and considering linear models for servers, a relaxed version of the problem can be represented as ILP. Secondly, an important part of cloud management is represented by workflow management. Workflows are coarse-grained parallel applications that consist of a series of computational tasks logically connected by data and control flow dependencies. Workflow applications are commonly represented as a directed acyclic graph. There are several classes of workflow scheduling techniques, in the state of the art, for cloud resources. The first class of workflow scheduling is based on the PSO algorithm to determine an optimal task allocation of resources. A PSO-based heuristic that considers both computation cost and data transmission cost is used to schedule applications to cloud resources in [15]. Another workflow scheduling for cloud resources is presented in [205], where the authors use a PSO algorithm to assist cloud users in selecting the optimal operating frequency of the VM CPUs considering task makespan, data dependencies, and prices. Sellami et al. [206] use a variant of the PSO, aiming to implement a workflow scheduling technique using a combined chaotic PSO that integrates a model for the dynamic voltage scaling (DVS) technique. The proposed solution is validated using a complex workflow application, showing promising results. A more complex approach is presented in [207], where the authors tackle a multi-objective optimization problem for scientific workflow scheduling by proposing an improved multi-objective discrete particle swarm optimization (IMODPSO) algorithm. Their approach has several novelties, such as a strategy for velocity limitation for particles, discrete particle positioning, Gaussian-based mutation for position update, and Pareto optimal convergence. The authors validate their approach by comparing it with three state-of-the-art algorithms. Furthermore, Awad et al. [208] propose the load balancing mutation particle swarm optimization (LBMPSO) technique for cloud computing workflow scheduling. Their approach is derived from PSO, adding constraints for task reliability, execution time, transmission time, round trip time, and makespan of tasks. Another class of workflow scheduling techniques is based on the Partitioned balanced time scheduling (PBTS) heuristic [209], which can compute an approximate resource pool for executing a workflow in a given deadline. Furthermore, the PBTS heuristic can handle elastic resource provisioning and be used with Amazon EC2 and MapReduce. Furthermore, a novel technique aiming to minimize the workflow cost while meeting a deadline for execution uses partial critical paths (PCP). Abrishami et al. [210] propose a PCP-based approach for grids, using a two-phase algorithm. The first phase adds sub-deadlines recursively to the tasks from the partial critical paths, while the second phase assigns a service with the lowest cost to each task. The authors' work is extended in [211], where they propose a Budget-PCP, aiming to create a workflow scheduling within a given budget. Both approaches are evaluated on various simulations showing promising results. Abrishami and Naghibzadeh [212] propose a QoS-aware workflow scheduling using PCP-based techniques. Their solution uses a recursive strategy to schedule the critical path while minimizing the workflow cost with the constraint of meeting the deadline. Another critical path-based workflow scheduling is proposed by [213], where the authors propose a

set of heuristics based on greedy techniques to determine optimal and valid workflows with critical paths. Finally, unsupervised machine learning techniques, such as K-Means clustering, can be used for workflow scheduling. This approach is presented in [214], where the authors propose a methodology for efficient resource allocation using VM clustering with the K-Means algorithm. Using this machine learning technique, the VMs are clustered and classified according to their similarity, leading to a more efficient resource allocation. Moreover, Sharma and Bala [215] proposed a modified K-means clustering to classify cloudlets and VMs in classes, considering parameters such as task length, priority, deadline, and cost. Using the classification, a scheduling algorithm is proposed and tested using CloudSim.

4.3 Data center sustainability and green computing

Data centers (DCs) and implicitly cloud computing are becoming a constant presence in modern society, being irreplaceable for a set of services they offer for customers, both individuals and companies. Their services extend due to technological progress in wireless and mobile networking as well as portable devices and IoT, leading to new types of computing, such as fog computing or edge computing, that bring the services closer to the user to enhance latency, scalability or user experience [216]. All this comes with a great cost, both from energy perspective and computing equipment waste.

To begin with, from energy consumption perspective, DCs are accountable for approximately 1.3% of global electricity demand [217], an estimated 240 and 340 terawatt-hours (TWh). This figure excludes energy used for cryptocurrency mining, which was estimated to be around 110 TWh in 2022 [217]. These numbers are expected to rise up to 3000 TWh in 2030 [218], especially considering new advances in AI and the need for more powerful DCs equipped with GPUs to train new AI models. Furthermore, the big tech companies own huge DCs, Microsoft alone owning over 160 DCs in the world, while one of its largest DCs, located in Chicago, has more than 300.000 servers that consume 23.5% of electric power generated by coal in the USA [219].

To tackle these problems, green computing aims to reduce the energy footprint of data centers by applying various techniques both on hardware and on software [216, 220] that are further classified in VM consolidation, power-aware techniques, cooling optimization techniques, etc. [219]. While VM consolidation is strictly connected to cloud computing, other techniques can be applied to any type of DC, even if it does not have virtualization. Modern DCs are designed to improve energy consumption by an efficient design [221], considering special equipment for data center air management, cooling and electrical systems, and heat recovery. As cooling systems are the second largest energy consumer in DCs, and the largest waste energy consumer, optimization techniques are employed for smart control and management. Latest optimization techniques are based on AI data-driven models that can obtain better results than classical simulation-based methods, such as computational fluid dynamics [222]. Thus, we can mention AI-based optimization approaches for air-cooled system management [223] based

on SVM to predict and regulate cooling needs more efficiently. The proposed model achieves an accuracy of 82% in predicting a real system evolution. Another more complex approach is presented in [224], where authors use TCN-BiGRU-Attention-based thermal prediction models to optimize cooling in hybrid-cooled data centers. The method developed by them balances energy consumption and cooling effectiveness by predicting temperature changes and adjusting cooling power dynamically. Finally, the authors present a set of numerical experiments using real data traces, showing that the proposed multi-objective cooling control optimization reduces cooling energy consumption in summer and winter while maintaining the rack cooling index above 95%. Furthermore, AI based is also employed on server and workload for sustainability goals, the authors of [225] investigating advanced strategies for resource allocation in cloud environments aiming energy efficient. Their study emphasizes predictive analytics, AI-driven algorithms, and dynamic scaling techniques, showing that large cloud providers, such as Microsoft, Google and Amazon, mainly use techniques such as AI-managed resource allocation, AI-based cooling system optimization, and integration with photovoltaic power plants. Among the algorithms used for energy efficiency, they mention that Google uses reinforcement learning for workload scheduling, Microsoft uses Deep Neural Networks for workload prediction, and Amazon uses decision trees for resource provisioning and planned maintenance.

Secondly, considering waste recovery, there are several directions to improve DC sustainability: Circular Economy Practices [226], Reusing and Refurbishing IT Equipment [227], and Innovative Waste Management Solutions for water and residual heat [228]. To begin with, DCs are adopting circular economy principles, which emphasize reusing, refurbishing, and recycling IT equipment to extend their lifecycle, reduce e-waste, conserve resources, and lower operational costs. Companies often partner with specialized recycling firms to ensure responsible disposal of obsolete equipment. A relevant example from industry are Microsoft DCs Circular Centers [229] that process 12.000 decommissioned servers each month, aiming to extract useful components so that more than 90% of the spare parts to be re-used, either inside the company or sold to other beneficiaries. To increase sustainability, Microsoft also increased the lifetime of IT equipment within DCs from 4 to 6 years. Besides physical equipment waste, or e-waste, DCs are also responsible for wasting water used for cooling and wasting residual heat from the cooling system. Thus, DCs are exploring the use of treated wastewater for cooling purposes, which reduces the strain on local water resources and supports broader water conservation efforts [228]. Furthermore, as DCs generate substantial amounts of heat as a byproduct of their operations, innovative solutions aim to redirect it to support local agriculture and heating systems. For instance, several data centers in Northern Europe have successfully implemented district heating solutions, channeling waste heat to nearby residential and commercial buildings, using heat pumps, absorption chillers, or heat exchangers [230]. Cities such as Stockholm and Helsinki use DC heat to warm nearby homes and businesses, providing a model that can be replicated in other regions. Moreover, waste heat is used to warm greenhouses, reducing the need for external energy sources to have year-round crop production. [231]. A recent example of heat reuse is Paris Olympics 2024, when heat from an Equinix

data center was repurposed to warm swimming facilities, showcasing the practical application of heat reuse in large-scale events [232].

Last but not least, a critical feature regarding DC energy efficiency and sustainability is the actual geographical location [233]. This has implications for several sustainability aspects. To begin with, energy costs are influenced by regions with deregulated energy markets which may offer lower energy costs compared to those with regulated markets. Secondly, access to renewable energy sources is crucial for data centers aiming to reduce their carbon footprint, thus favoring geographic location with plenty of renewable energy options such as hydro power, solar, and wind energy. Finally, the climate has also a great impact on energy efficiency, especially from cooling system perspective, thus regions with colder climate being more suited for DC construction. Besides these, transmission lines also have impact on DC energy efficiency and reliability [234], line length influencing directly energy losses, thus DC construction near power plants reducing energy losses. Furthermore, DCs usually need at least two transmission lines for redundancy and access to renewable power plants for lowering carbon emissions. However, these geographical criteria are trade-offs when considering DC service quality, SLAs, and latency, especially because favorable regions for energy-efficient DC construction are isolated and far from human-dense populated regions that need DC services. Thus, big tech companies build their own networking infrastructure [235] and expand their hardware infrastructure closer to clients by building edge and fog sites [236].

4.4 Fog computing

The appearance of smart devices and sensor systems gave birth to the Internet of Things paradigm [237], where sensors, actuators, and smart devices cooperate with each other to ensure semi- or fully automated consumer products [238], such as smart cars or smart home appliances. Through the use of so-called unlimited computing and storing capacity of cloud resources, this might be a solution theoretically for processing and preserving the vast amount of IoT data, real-time applications utilized in healthcare, and the automotive industry represents needs and wants that the cloud services can hardly provide. Therefore, the network of complex systems was extended closer to end users and their devices, known as fog computing. It is intended to extend cloud computing by providing a new service layer for improving the QoS, affecting throughput, transmission delay, and availability [239].

IoT–fog–cloud systems are often described with a layered topology: the lowest layer contains the IoT sensor and actuator network, which continuously monitor and sample the environment (i.e., by using temperature and humidity sensors) or react to the instructions and the messages from other system entities. The intermediate layer contains the fog nodes, which typically utilize less computing power than the cloud resources placed in the top layer. Due to the structure of the system, such layering avoids the bottleneck effect caused by overloaded communication channels of clouds, decreases the delay and the response time, and increases transparency and security.

The complexity of cooperating smart devices and fog–cloud resources to provide reliable services created various challenges in the field. The following emerging issues can be considered [240], respectively:

- Connectivity problems often come together with delay and capacity-related questions. It involves data partitioning for fog clusters to cover a certain area effectively. Entering vast amounts of IoT entities might have a negative influence on the throughput and latency as well, which is critically important for latency-sensitive services such as IoT healthcare or streaming applications. Palattella et al. [241] present a handover mechanism for vehicular services as well as an architecture for latency-critical applications at the edge of networks considering time, safety, and security.
- Computation offloading aims to unload and outsource task processing to the computing nodes, which in this case reflects fog resources and mobile devices introducing mobile edge computing. The goal of the offloading decision is to adapt dynamic changes in the systems, and it has an effect on mobile batteries, storage, and resource-constrained fog nodes as well. Numerous studies aim to resolve the offloading issue of fog computing and find the balance among energy consumption, payment cost, and service delay. For instance, Liu et al. [242] present it as a multi-objective optimization problem, and it is extended with the interior point method (IPM) to improve the effectiveness of the proposed offloading algorithm. Bio-inspired algorithms such as ant colony optimization (ACO) can be used to offload IoT sensor applications tasks in a fog environment as well. In [243], an improved ACO is proposed, called smart-ACO, to meet the specified QoS limitations considering latency time, network characteristics, and loads of fog nodes.
- Billing and operation costs are substantial components of IoT–fog–cloud systems, since different types of pricing schemes are usually applied for resource utilization, and they should be adapted dynamically for application needs. Cloud-side pricing typically follows the pay-as-you-go manner, while the cost calculation for the IoT side varies, stakeholders must pay after the data are exchanged or message size, and limits can be considered for distinct time intervals. Kalmar and Kertesz [244] performed a cost-based analysis of four major cloud service providers and compared on-demand service costs through existing IoT use cases.
- Resource provisioning is more important than ever, especially for device mobility. The moveable devices require the elasticity of the IoT services deployed on fog nodes; therefore, latency, unallocated computing power, and storing capacity can dynamically change. Bittencourt et al. [245] discuss the resource allocation problem for mobile users, and they presented three scheduling algorithms, which consider user mobility and computing capacities as well to ensure a seamless service. Yadav et al. [246] present a service allocation strategy using a genetic algorithm combined with particle swarm optimization (GA-PSO) to find the appropriate nodes and VMs to allocate the IoT application requests. In this approach, the authors focus on minimizing the energy consumption and thereby to maximizing the resource utilization of the VMs.

The detailed analysis of the IoT-to-cloud continuum cannot be done without significant investments, since setting up various configurations of resources and networks is costly. Design and implementation goals may require more sophisticated solutions; therefore, simulators have become more and more accepted by the scientific community over time. These simulation tools can mimic such scenarios in a realistic way; however, one cannot cover each challenge mentioned in the previous section. The effectiveness of the simulators can be measured by the execution time, the number of simultaneously simulated entities, their accuracy, and available functionalities. Next, we present the most representative simulators dedicated to model IoT–fog–Cloud systems. It is impossible to present an exhaustive overview of the representative tools; therefore, in this study, we focus only on the most well-known and popular simulators, which are usable for the design, development, and operational phases of real systems as well [247].

iFogSim/iFogSim2 [248] is a recently upgraded simulation toolkit built upon CloudSim [249], which was originally designed for evaluating different resource management policies focusing on latency, energy consumption, network load, and operational costs. The simulator follows the sense–process–actuate model involving sensors, applications, and actuators as well. The recent update aims to model mobility, service migration, microservice orchestration, and clustering.

DISSECT–CF–Fog [250] is dedicated to analyzing the offloading mechanism of IoT–fog–cloud systems, where the loads (i.e., IoT data) are generated by simulated IoT devices until it is processed by IoT applications installed on computing resources. It can also consider various greedy or optimization algorithms to offload tasks. For making the best possible decision, the simulator may take into consideration the energy consumption of the entities, the position and mobility of the devices, pricing patterns, IaaS utilization, the characteristics of the network, and so on.

FogTorchPI [251] was created to investigate the deployment problem of IoT application components on fog resources. It applies the Monte Carlo algorithm to satisfy Quality of Service requirements, involving latency, bandwidth, operational costs, and processing needs. Nevertheless, the simulator lacks a detailed model of such a complex system, which restricts its usability.

YAFS [252] is a simulation library for edge–fog–cloud ecosystems, which supports application module placement with resource allocation policies, billing, and network planning as well. It also introduces message routing and control policies in order to execute any kind of action.

The FogNetSim++ [253] toolkit can be used to simulate a distributed fog computing environment with its primary goal being to model device mobility and the closely related handover process. The tool puts a great emphasis on realistic network operations since it deals with various communication protocols (e.g., HTTP, MQTT).

Plenty of other extensions and simulations are built upon iFogSim, or directly on CloudSim. The interoperation of these tool components represents a significant drawback, since they were introduced at different moments of the development phase of the core simulators, and the developers were not paying enough attention to compatibility issues. Besides this, a few of them are definitely worth mentioning: MobFogSim [254] aims at supporting mobility and migration functions for

Fog Computing, while EdgeCloudSim [255] and IoTSim-Edge [256] rather focus on the simulation of Edge Computing.

In essence, simulation solutions gain ground progressively, because their capability makes them a distinguished choice for analyzing the aforementioned challenges of IoT–fog–cloud systems. Nevertheless, it could still require suitable programming knowledge as well to implement various scheduling algorithms.

In addition to fog computing, edge computing also aims to achieve decentralized data processing. Fog computing includes a wider network hierarchy, but edge computing focuses on device-level data processing directly at the edge of the network. This means that computation tasks are carried out closer to the data source, such as smartphones, wearables, and routers. This approach enables faster decision-making and reduces the dependence on centralized cloud services [257].

The evolution of edge computing to edge intelligence includes the incorporation of AI and ML capabilities into edge devices. These devices can then handle tasks such as traffic optimization in smart cities, real-time anomaly detection in critical systems, and predictive maintenance in industrial applications. This progress is particularly relevant in industrial environments such as Industry 4.0 and emerging Industry 5.0, where edge computing and artificial intelligence are combined to optimize processes and improve operational efficiency [258].

Data processing and decision-making closer to the data source reduce latency and bandwidth consumption compared to centralized solutions, allowing the application-centric operation of edge devices. In order to effectively use limited computational resources, techniques such as model compression, quantification, and pruning are usually used to ensure performance and accuracy. Edge devices use a range of artificial intelligence algorithms that are adapted to specific applications, including decision trees and support vectors for classification tasks, as well as convolutionary neural networks (CNN), long-term short-term memory (LSTM) for time series analysis and recurrent neural networks (RNN) for image recognition. Federated optimization might also be involved in this process by allowing multiple devices to collaboratively train models without sharing raw data, preserving privacy and reducing the need for large-scale data transfers. However, the selection of algorithms depends on the computing capacity of the device and the application requirements, such as real-time processing capabilities and operation under limited conditions [259].

In the context of edge intelligence, several key concepts also appear. Edge caching refers to the temporary storage of data at the edge of the network for faster access, reducing dependence on requesting information from various cloud services and therefore improving performance. Edge training involves training AI models directly on edge devices, enabling localization and adaptation, while ensuring data privacy by retaining sensitive information on the site. Edge inference concerns the application of trained AI models on edge devices to make predictions or decisions based on incoming data, which is essential for applications that require immediate responses, such as autonomous vehicles or intelligent surveillance systems. Edge offloading involves transferring computation tasks from edge devices to more powerful clouds or fog resources, if necessary, optimizing resource use and balancing workloads [260].

4.5 Blockchain

Blockchain (BC) technology was proposed in 2009 [261] as the basis of a TTP-free fully distributed peer-to-peer (P2P) system. The combination of solutions and methods deployed within has furnished the road toward a distributed infrastructure of the next-generation Internet. BC-based systems are typically characterized by their infrastructure, data structures, a networking model, and consensus algorithms (CA). The infrastructure can be formally described by a set of nodes V, usually termed as miners, where $V = \{v_1, v_2,..v_N\}$. Data shared between elements of the set V are described according to the application of the system. For example, transactions (TXs) are submitted by the end users to the BC network so that they are processed and added to its distributed ledger (DL). Usually, TXs are shared with all miners triggering them to generate new blocks of data. A block usually consists of a header and a body. The header may consist of data such as the type of block, the type of CA, the timestamp, the hash of the body, and, most importantly, the proof of block validity. The body, on the other hand, usually includes a group of TXs and the hash of the previous block body.

As BC nodes form a distributed system, those nodes exchange data through a P2P network and communicate by message passing via directly connected lines. BC nodes connect to their peers once they are granted access to the network, making them demonstrable as a graph $G = (V, E, w)$ of a connected giant component, where E is the set of edges in G, representing the communication lines between the miners. Each $e_{i,j} \in E$ connects exactly two nodes $i, j \in V$ and can be traveled in both directions. Each $e \in E$ is associated with a distinct non-negative value, namely weight ($w_{i,j}$ or w_e), which represents the transmission time needed to deliver 1 bit of data from node i to node j or vice versa, computed in ms. A sub-graph of G is any graph $G' = (V', E', w')$, such that $V' \subseteq V$ and $E' \subseteq E$. G' is also connected, undirected, and weighted as it inherits the properties of the original graph.

Every BC-based system must operate a CA in order to maintain the consistency of its DL. As tens of CAs were proposed in the literature, a CA is usually considered *valid* if it was proven secure under specific formalized circumstances. One of the main benchmarks used to describe the security level of a given CA is its tolerance for K faulty/adversarial nodes where $K < N$. For example, the most famous CA, known as the proof of work (PoW) algorithm, was proven secure as long as $K < N/2$. Similarly, the practical Byzantine fault-tolerant (pBFT) algorithm was proven secure as long as $K <= \frac{N-1}{3}$. The upper bound fraction Ψ, tolerated by a given CA in order to maintain the system secure, is formalized in Eq. 20.

$$\Psi = \frac{K}{N} \tag{20}$$

Sharding is, in its generality, a type of database partitioning technique that separates a very large database into much smaller, faster, more easily managed parts called data shards [262]. Technically, sharding is a synonym for horizontal partitioning, which makes a large database more manageable and efficient in terms of scalability and energy. The key idea of BC sharding is to partition V (which are formed initially

as one giant shard) into a set $R = \{r_1, r_2, ..r_d\}$ of d smaller shards. As demonstrated in Fig. 7, each shard r_i, $i <= d$ consists of $n_{r_i} < N$ nodes until condition 21 is satisfied. Each shard processes a disjoint set of TXs, yet all shards utilize the same CA, leading to increased overall system throughput. As described in [263], d grows linearly with both the total computational power of the network and with N. A sharded BC network is usually declared to be more efficient, than its non-sharded version, because it uses the same available computational and storage resources with much-enhanced throughput and total system latency.

$$\sum_{i=1}^{d} n_{r_i} = N \tag{21}$$

Assuming all $v_i \in V$ have the same computational power and a fraction Ψ of which is controlled by a Byzantine adversary, all nodes have access to an externally specified constraint function(s) $C \rightarrow \{0, 1\}$ to determine the validity of each TX submitted to, or confirmed by, the network. A sharding protocol outputs a set R where each shard $r_i \in R$ contains a subset of $k_{r_i} <= K$ adversary nodes leading to state 22.

$$\sum_{i=1}^{d} k_{r_i} = K \tag{22}$$

The scalability of a BC-based solution is usually benchmarked by the overall throughput of the system while increasing N. On the other hand, the security-related *agreement* property is benchmarked with reference to a security parameter that is satisfied in a sharded BC, if the following condition holds:

$$\forall\, r_i \in R : \frac{k_{r_i}}{n_{r_i}} <= \Psi \tag{23}$$

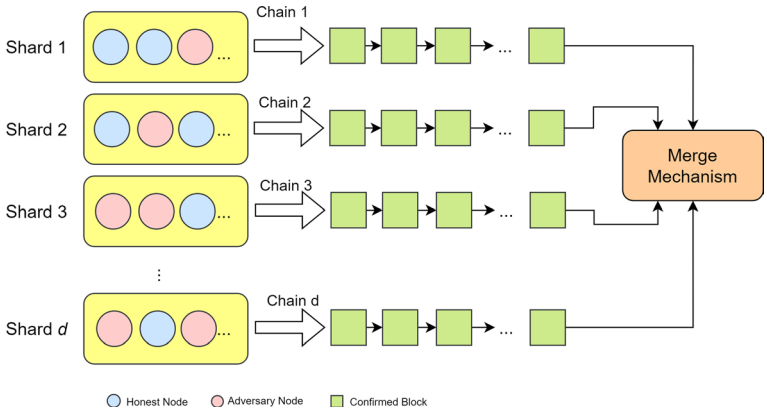

Fig. 7 Shards in a blockchain network, where each shard may consist of adversary nodes. Typically, each shard maintains its own local chain, and a merging mechanism is deployed for global consensus

That is, each shard consists of a fraction of at most Ψ faulty/adversary nodes out of all nodes contained in that shard. Consequently, each shard provides a level of agreement that is equivalent to the agreement level of the same BC if it were not sharded. Each shard maintains its local chain while the system merges all local chains into one global chain according to predefined criteria (Fig. 7). Merge approaches of local chains are out of the scope of this work but technical details can be found in [264].

In most sharding frameworks, randomized sharding is regularly run in order to maintain a high probability that Condition 23 holds. However, such a sharding approach waves away the optimal data propagation within, and among, shards. Specifically, this approach results in relatively low throughput in exchange for a high level of security.

The trade-off between scalability and security in sharded BCs is an open optimization problem [49]. The goal of a sharding optimization approach is to propose a sharded scheme of the BC that is both secure and optimized. That is, the approach shall be able to regularly and efficiently find the optimal distribution of V among R in terms of scalability, such that Condition 23 is satisfied. Specifically, an optimally sharded BC would provide higher efficiency in its resource utilization, compared to a non-sharded and/or non-optimally sharded BC.

BC sharding problem matches the well-known graph partitioning problem with an additional constraint as described in Condition 23. This problem can be optimally solved using LPs, but the complexity of such an approach is very high as the problem is considered NP-complete. Additionally, the selection of the optimization objectives is arguable, depending on the application of the developed LP. For instance, Henzinger et al. [265] targeted the minimization of the weights on edges between the partitions, regardless of the total weights at the intra-shard level. On the other hand, Cordero et al. [266] minimized the total weights only at the intra-shard level, regardless of the weights at the inter-shard level.

The BC sharding problem can also be solved using one of the previously discussed metaheuristic approaches which are much more efficient in terms of complexity. Meanwhile, metaheuristic approaches satisfy a "good" level of scalability optimization compared to the currently used randomized sharding [267]. The security condition can then be added as an extra parameter to validate a proposed solution for a given network. However, several previous works deployed digital signatures to guarantee a secure sharding while optimizing for scalability. As this indeed solves the security requirement, more analysis needs to be conducted regarding the effect of digital signatures utilization on the overall system throughput. In particular, increased throughput indicates higher efficiency in using the network's resources to perform the same tasks using the same resources.

4.6 High-performance computing

Parallel programming for use with high-performance computing systems has been traditionally possible mainly through several APIs dedicated to particular system types, to be used with selected programming languages, mainly C/C++ and Fortran. These include [1]:

1. Message passing interface (MPI) for distributed memory/cluster-based systems with a multi-process model allowing communication and synchronization through point-to-point and collective operations, among others.
2. OpenMP for multithreaded shared memory systems through the use of directives and library calls including offloading computations to accelerators such as GPUs.
3. OpenACC for multithreaded shared memory systems through the use of directives for GPU systems. CUDA for shared memory multithreaded GPU programming with an application model represented by a grid of thread blocks that consist of up to 1024 threads each.
4. OpenCL for shared memory CPU+GPU systems with an application model represented by an NDRange of workgroups that consist of up to 1024 work items, the model very similar to that of CUDA, generalized to CPU+GPU systems.

These APIs have been traditionally oriented on performance/execution time at the cost of flexibility, security, and reliability features. Nevertheless, in recent years the parallel programming field has been receiving attention with respect to some of the other resources considered in this work, as follows:

- Energy – especially in the context of performance energy optimization [268, 269]. In particular, optimization involves trade-offs such as minimization of energy of an application run, minimization of energy-delay product, etc. [270–272]. These has been possible by applying several techniques such as scheduling, DVFS/DFS/DCT, power capping, and application optimizations [273, 274]. Tools for individual compute devices [272, 275] and whole HPC centers have been proposed [276].
- Memory/storage in the context of persistent memory/non-volatile memory (NVRAM) that is byte-addressable, persistent storage presumably larger than typical RAM. For instance, [277] considers a large parallel persistent memory model with P processors, each with fast and small ephemeral memory and access to a large shared persistent memory. Usage of NVRAMs in cluster nodes for MPI applications through the incorporation of the well-adopted MPI file I/O interface has been shown, e.g., multi-agent simulations [278] and image processing [279].
- Security such as encryption/decryption of messages sent between nodes used by an MPI application [280] and among distributed clusters using MPI [281].
- Reliability extending the possibility of a successful application run in the case of partial system failures, important, especially in the context of long-running compute-intensive codes running on large HPC systems. Existing research works focus on proposals of resilience constructs for MPI [282, 283] as well as multithreaded OpenMP [284] applications.

In the context of optimization/problem formulations, most of the relevant approaches use: ILP for data partitioning, scheduling at the level of APIs for parallel programming such as OpenCL and MPI [6] but also service integration frameworks [1, 81]; greedy approaches, e.g., energy-aware scheduling [103]; dynamic programming for optimization of code generation [106]; genetic algorithms for scheduling in HPC environments, including energy-related formulations [12, 122];

reinforcement learning similarly for scheduling in HPC systems [26] including batch job schedulers [285].

4.7 Volunteer computing

Volunteer computing is a type of distributed computing that allows public participants to share their computing resources that they do not need in a given period, and thus help launch computer-aided projects. Existing voluntary platforms have up to a million users, thus providing vast amounts of memory, storage, and processing power.

Volunteer hosts (desktop PC, notebook, mobile phone) connected together form the equivalent of a super powerful virtual machine. The reason for connecting is that computers usually use up to 15% of the total capacity, so many potential resources can be connected. The calculations, which would take us several thousand years to process on a single desktop computer, can be processed in this way in just a few months.

There are currently almost a million dedicated volunteers, and their hardware has a significant impact on science, enabling projects that are impossible without computing power. In this mode, the problem is divided into a large number of tasks that solve one or more computers at once.

Several systems were created for running this paradigm, most notable being BOINC [286]. It should be noted that running a volunteer system client can be a deterrent factor for users, although it can be mitigated in several ways, e.g., running using a separate account in a system. Some systems such as Comcute [287] rely on running client codes within a web browser, which sacrifices performance (code is executed using Javascript) for a potentially more secure sandbox as well as ease of use (no installation required). Other systems of this type, using Web workers, are those described in [288] and Weevilscout [289]—for which the authors demonstrated a bioinformatics use case completed within 5 h, while in total the Web thus provided 135 CPU hours for computing. In [290], the authors presented CrowdCL which is an open-source, web-based, cross-platform volunteer computing framework that provides KernelContext—an abstraction layer for WebCL allowing development and execution of OpenCL programs in an online Javascript setting. For the Thomson problem and sufficiently large N (N-body computations), the performance of WebCL is visibly larger than those of optimized Java and Javascript implementations, run on a laptop with Nvidia 320 M GPU and a desktop with an NVIDIA K20 GPU. Pando [291] provides a volunteer solution using failure-prone personal devices made available by volunteers whose number can change, parallelizing a function on a stream of values.

In terms of browser-based voluntary systems, the authors of [292] distinguish three generations: the first one is based on Java applets, no REST support, and communication protocols: HTTP, UDP, TCP, and Java RMI; the second one uses JavaScript, occasional REST support, and protocols HTTP, AJAX, and finally, the third generation uses JavaScript for tasks, WebWorkers for threading, and HTTP, AJAX, and WebSockets.

This means that, in the context of volunteer-based systems, several trade-offs between resources are present: performance vs security but also the traditional performance vs reliability (the need for more volunteer clients to make sure results are correct decreases performance/number of volunteers).

LHC@Home[5] is a volunteer computing platform that uses the donated idle time of your computer, thus helping physicists compare theory with an experiment in search of new fundamental particles, as well as questions about the Universe. All LHC@home projects are run using the BOINC-established platform used by most volunteer computer projects worldwide. There are several LHC@home projects, all focusing on the Large Hadron Collider (LHC) engineer and physics at CERN in Switzerland. Some of the most significant projects are mentioned below.

ATLAS@home[6]: ATLAS is a project that researches the physical particle experiment, and searches for new particles and processes using direct proton collisions of very high energy. In this project, petabytes of data were handled, which were recorded, processed, and analyzed during the first three years, and this led to the discovery of the Higgs boson in 2012.

CMS@home[7]: The Compact Muon Solenoid (CMS) is a general-purpose detector that takes place at the LHC. This project has a broad physics program, from studying the Standard Model (which includes the Higgs boson) to searching for additional dimensions and particles. The CMS has the same scientific goals as the ATLAS experiment but uses different technical solutions and magnetic system design.

SixTrack[8]: This application simulates 60 particles traveling around the LHC ring at the same time, and pores the simulation for 100,000 loops, and even up to 1 million loops around the ring. In this way, it is tested whether the beam will remain in a stable orbit or will happen to lose control and fly off course into the walls of the vacuum tube. Beam instability can cause a serious problem if it happens in real life, which can lead to machine shutdowns due to repairs. This project helps physicists and LHC beam engineers make necessary corrections to create cleaner, more stable, and safer beams.

Test4Theory[9]: The project allows volunteers to run high-energy particle collision simulations. These simulations can be run on their home computers, use theoretical models based on the Standard Particle Physics Model, and are calculated using the Monte Carlo method. Theoretical models use adjustable parameters, and the goal is for a given set of parameters (called "tuna") to correspond to the widest possible range of experimental results. The results are sent to a database containing a very wide set of experimental data from many experiments, collected from accelerators around the world, including the experiments on the Large Hadron Collider at CERN. The database and the theoretical fit process are part of the MCPLots project, located in the theory department at CERN.

[5] https://lhcathome.web.cern.ch.
[6] https://atlas.cern/Resources/Atlasathome.
[7] https://lhcathome.web.cern.ch/projects/cms.
[8] https://lhcathome.web.cern.ch/projects/sixtrack.
[9] https://lhcathome.web.cern.ch/projects/test4theory.

4.8 Summary and conclusion

This section discusses resource-aware applications in various computing domains, emphasizing their architectures, optimization challenges, and practical applications. It also identifies seven primary architectures: data centers, cloud computing, high-performance computing (HPC), edge computing, fog computing, blockchain, and volunteer computing. Each architecture introduces unique optimization needs, challenges, and opportunities for resource management.

Data centers are the backbone of many computing architectures, facing challenges in workload scheduling, energy efficiency, and thermal management. Strategies such as integrating data centers with smart grids, dynamic power management, and thermal-aware workload scheduling are explored. Emerging trends include heat reuse for sustainability and AI-driven cooling system optimizations.

Cloud computing relies on virtualization to provide scalable resources, but optimizations are necessary to avoid performance degradation. The chapter highlights techniques for virtual machine (VM) allocation, dynamic server allocation, and workload scheduling using heuristic and AI-based methods such as PSO and genetic algorithms (GA) to solve NP-hard allocation problems and ensure QoS.

HPC Systems focus on specialized workloads that demand efficient scheduling, parallel execution, and resource allocation. Traditional APIs such as MPI and OpenMP are widely used, with growing emphasis on energy efficiency, security, and reliability.

Fog computing and edge computing bring computation closer to end users, reducing latency and improving efficiency. This chapter discusses offloading strategies, cost-aware resource provisioning, and AI-driven optimizations. Simulation tools such as iFogSim, FogTorchPI, and YAFS play a crucial role in evaluating fog and edge computing strategies.

Blockchain technology addresses the trade-off between scalability and security, particularly through consensus mechanisms and sharding, which improve throughput while maintaining decentralization.

Volunteer computing leverages distributed, publicly available computing resources, such as platforms like BOINC. The chapter reviews frameworks like BOINC, browser-based volunteer computing approaches, and the trade-offs between security, performance, and reliability, enabling large-scale scientific computation.

In conclusion, resource-aware computing spans multiple architectures, each with unique optimization needs. Addressing energy efficiency, security, scalability, and cost constraints is crucial for improving system performance. The insights presented in this chapter serve as a foundation for future research and advancements in resource-aware optimization across distributed and parallel computing environments.

5 Summary—coverage analysis, open problems, and conclusions

We start the summary of our review by observing that specific algorithms are applied in particular contexts, as indicated in Sect. 4. We further note that potential ranking or preference of algorithms requires formulation of well-defined

criteria against which the algorithms could be compared. We have come to the conclusion that the one that could be used in this context is the number of workers, processes or threads as these are common in actual implementations in parallel and distributed systems. This way, we can assess potential sizes of problems that could realistically be handled by a particular algorithm using a particular number of workers etc. For instance, in paper [6] authors proposed optimization of data assignment for parallel processing in a heterogeneous hybrid CPU+GPU environment using ILP. For a system with 10 processing units (CPUs and accelerators) interconnected with a 1Gbit/s Ethernet network, having considered profiling of start-up times, bandwidths, and units' performance metrics, ILP was used to assign data for solving systems of equations using the Jacobi method. For the numbers of problems exceeding 256 (problem sizes 512-2048 tested) lp_solve's execution time had to be limited, but the ILP-based approach could handle large problems efficiently, and best results were obtained with a timeout of 30 s for problem sizes of 512-1024 and approx. 60 s for 2048. Assessment of ILP in the context of edge and cloud computing is provided in more detail in Sect. 3.1.3. Practical comparisons of algorithms can be realistically provided for a particular problem. For instance, for the problem of scheduling workflow applications with dynamically changing service availability, that requires online rescheduling, in paper [121] it was concluded that ILP for workflows with more than 6 nodes and a timeout of 20 s (optimal solution not feasible due to the problem size) returns much better solutions (scheduling and workflow execution times combined) than GA and is only followed closely by a heuristic GAIN algorithm. Workflows with up to more than 100 nodes and 400 services in total were handled by the algorithms. Typically, other arguments used as input to the algorithms could be very specific that makes a direct comparison of algorithms rather unfeasible.

In particular, optimization algorithm complexity is also significantly impacted by application parameters such as the number of tasks and by additional hardware configuration parameters such as discrete DVFS levels, as these manyfold the number of ILP variables in the model (and ILP solving complexity can be exponential in the number of variables). For moldable-parallel tasks where the number of workers to use for each task is part of the optimization solution, large CPU core counts can quickly lead to prohibitively long ILP optimization times. Simple ILP models that do not model these additional application and hardware properties [91] lead to faster solution time but worse optimization quality [85]. However, by introducing additional constraints such as crown scheduling (Sect. 3.1.2), the combined optimization problem for streaming task graphs with up to 80 tasks on up to 32 cores can still be solved to (crown-)optimality within three minutes and vastly outperforms unrestricted ILP models for the same problem in optimization time [84, 85], which in turn still perform better than nonlinear models [85].

In terms of characteristics, advantages and disadvantages of the particular problem formulations, we can note the following:

- ILP has the benefit of encoding decision variables as integers and possibly problem values as real variables in optimization problems. On the other hand, problem formulation is limited by the linear formulation and many integer variables

result in very high complexity requiring heuristic algorithms and usage of time-outs in ILP solvers.
- Greedy approaches are typically fast and can be formulated in a natural way but, on the other hand, result in sub-optimal solutions.
- Dynamic programming is very useful for combinatorial problems but requires a method for decomposing a problem instance into independent subproblem instances and the optimal substructure property must be met. Solutions can be fast at the cost of space required.
- Nature-inspired algorithms allow to encode an optimization problem in a natural way and do not have formulation requirements like linearity, but convergence can be very slow and dependent on the input configurations. Might result in lower quality solutions than dedicated algorithms with domain knowledge.
- Matching game allows consideration of entities and their preferences. Not all problems can be naturally expressed using this approach.
- Reinforcement learning is useful for problems where an agent performing actions, either optimization space exploration or knowledge exploitation can be naturally considered, especially considering long-term effects. Consequently, it is useful for elaboration of strategies or policies finding trade-off/balance between the two for a particular optimization goal.

We further discuss which and how various quality metrics/resources are considered in the context of domains such as data centers, cloud, fog, volunteer computing, blockchain, and high-performance computing, included in Table 3. We shall note that we consider some frequently occurring objectives including execution/response time, energy consumption, security, (soft) error reliability. We can say, based on the analysis, that for the full picture, these emerged to be important cross-domain objectives in resource management that shall be considered in any context as they are addressed by works in all of the domains, obviously typically a selection of those at a time. Additionally, we provide information on problem-specific resources that are related to the particular domain. Even though these objectives are present across all the domains, their meaning depends on the context. For instance, requests might be handled in real time, in batch mode, in a regular reactive manner. Energy, depending on the domain, refers to the servers and systems managed by a provider, a client or both types. Security and reliability measures differ primarily depending on whether the system is owned and managed by a dedicated provider or is an open distributed system. Finally, all domains use computer nodes linked with interconnects, albeit with different latencies and bandwidths.

Table 4 outlines which problem formulations and corresponding algorithms have been proposed in the context of various parallelization levels of non-distributed parallel systems, i.e., instruction-level and either homogeneous or heterogeneous (e.g., CPU+GPU) environments. Specifically, for reinforcement learning techniques, approaches have been proposed for parallelization within or among clusters, which can essentially be either homogeneous or heterogeneous and RL can handle such cases due to the knowledge using/exploration and reward approach. Heterogeneity can be considered both in the context of usage of various computational devices such as CPUs+GPUs [7] but also in the context of consideration of CPUs as well as

Table 3 Objectives (quality metrics/resources) in various contexts/domains

Formulation/domain	Execution/response time	Energy	Security	(soft) Error vulnerability	Problem specific resources
Data centers	DC Control systems that perform resource scheduling can be executed in real time for specific subsystems (e.g., cooling system control) or on longer time spans (e.g., day-ahead plans [166])	Green data centers aim to provide energy efficiency, so energy consumption is part of the optimization goal. More complex approaches also consider the thermal energy of the data center. [178, 179, 184]	Security and data privacy are maintained by the fact that management systems that perform resource allocation are executed within the closed physical environment of the data center, being protected by Network function virtualization (NFV) and software-defined networking (SDN) [300]	DC management systems use redundancy to ensure command execution in due time	Servers, cooling system, UPS system, smart switches, thermal energy storage systems, heat pumps [171, 172]
Cloud computing	Cloud scheduling systems work in either a reactive manner, aiming to schedule workload as it arrives [191], or in a proactive manner, allocating resources before the workload arrives [201]	Cloud systems perform workload scheduling and allocation considering the energy efficiency of the underlying hardware infrastructure [301, 302]	Cloud infrastructures use both physical and cyber-security layers to ensure client privacy, using firewalls, integrity checks, and data encryption. Modern approaches rely on homomorphic encryption for performing computation on encrypted data [303, 304]	Cloud management systems use replication to assure data recovery in case of software errors [305]	virtual machines, clusters, software-defined networks [195]

Table 3 (continued)

Formulation/domain	Execution/response time	Energy	Security	(soft) Error vulnerability	Problem specific resources
Fog computing	fog topology is primarily dedicated to real-time IoT applications because it is able to maintain QoS for latency-critical solutions [241]	besides execution time and cost, energy is a frequent goal of algorithms aiming for optimal trade-offs, as the consumption of low power resources located at the edge of the network, as well as computing nodes with high capacity, is measured [250]	due to the API usage of fog services and the presence of IoT devices providing security by encryption/decryption, integrity checks are pivotal [306]	privacy risks (sensitive data, location, and usage information collected from IoT services) and misconfiguration of fog services [306]	physical machines, virtual machines, smart devices, and microcontrollers
Blockchain	Sharding the Blockchain network usually proceeds in epochs. For each epoch, new and existing nodes execute the shard allocation protocol to obtain a new shard membership w.r.t. the current system state [307]	One of the major goals of sharding BCs is to decrease total energy consumption per confirmed TX – that is, since a TX is processed by nodes within the shard, which trivially count less than the total number of nodes in the network, then a TX confirmed in a sharded BC costs much less than the same TX confirmed by the same BC non-sharded [308]	BC sharding protocols must prove their security with reference to Condition 23. This condition is currently only guaranteed in permissioned BCs [309] with the usage of digital signatures [310]. However, security in sharded permissionless BCs is still an open issue [311]	BC technology utilizes different consensus algorithms to maintain the consistency of the distributed ledger. For permissionless BCs, costly algorithms (e.g., PoW [261]) are usually needed, while soft errors (termed forks) in permissioned BCs can be mitigated using more centralized algorithms (e.g., PoS [312], and PoA [313])	Miners (CPUs, GPUs, FPGAs, and ASICs) [314, 315]

Table 3 (continued)

Formulation/domain	Execution/response time	Energy	Security	(soft) Error vulnerability	Problem specific resources
High-performance computing	traditionally the main objective subject to optimization for HPC systems, associated with both scheduling, load balancing (cluster, computational device, core level) as well as optimizations such as caching, overlapping communication and computations, false sharing minimization, etc. [1]	one of the key metrics considered in the field now, often in the context of execution time + energy [268, 269]/power optimizations [105], using techniques such as device selection/scheduling, power capping/DVFS, etc. [103, 273]	while MPI was designed especially for performance, solutions such as ES-MPICH2 exist for encryption/decryption of communication among nodes [280, 281]	need for solutions, especially for large-scale petascale + systems [316, 317]; User Level Failure Mitigation (ULFM) interface has been proposed for implementation of resilient MPI applications, as well as MPI Chameleon [282, 316]	computing devices (CPUs, GPUs), interconnects (network, buses—PCI/NVLink, etc.)
Volunteer computing	Many distributed computing applications require real-time responses; on volunteer resources, it is possible to get a low execution time on a platform like BOINC. The system gives low worst-case execution time for task management operations, such as task scheduling, state transitioning, and validation	It is still not known whether volunteer computing is more or less energy efficient than computer centers; however, this does not affect the cost of volunteer computers for scientists, because volunteers pay for the energy	There are several security threats when it comes to the Volunteer computing approach, so projects must be designed and focused on security threats	Volunteers in VC systems are anonymous and are not linked to a real-world identity. That is why volunteers do not respond to projects but use trust based on project policy. In case the credibility is lower than the threshold, it is re-checked with other volunteers, and it can be guaranteed that the error rate will not exceed the given and acceptable value	Volunteers' computers, server middleware (1+ server, 1+ server layer)

Table 4 Problem formulations and solving techniques for various types of (non-distributed) parallel computer architectures

Formulation / Architecture	Instruction-level parallel	Homogeneous parallel	Heterogeneous parallel
Integer linear programming	[109–112]	[83–85, 91]	[6, 7, 89]
Greedy algorithms	[98, 99]	[8, 96, 97]	[100, 105]
Dynamic programming	[106]		
Nature-inspired algorithms		[318] (GA)	[7, 11] (GA)
Reinforcement learning		[24, 26, 285, 319, 320]	[24, 26, 285, 319, 320]

burst buffers when scheduling [11]. We can see that some of the approaches like ILP or greedy algorithms are present for all the contexts while dynamic programming is proposed for instruction-level parallelism and nature-inspired and more recently RL at a higher level of abstraction. We can also note that cited papers target either higher homogeneous/heterogeneous levels or intstruction-level parallelism but not both.

Table 5 outlines the works that address and use particular problem formulations in various previously described domains and contexts. The domains of the research works can be either centralized in terms of location, like data centers and typically high-performance computing, or geographically distributed, i.e., cloud, fog, volunteer computing, or BC. For nature-inspired solutions, specific algorithms are given and we can see many variants within the latter group. Additionally, we can see that relatively few formulation-domain pairs have not been considered which points out potential future research areas.

Based on the analysis performed, we can formulate the following topics for future research in the field:

1. More thorough comparison of performance and energy efficiency, such as extending computational efficiency in CPU Marks/W analyzed in [293], of volunteer computing versus data/high-performance computing centers but also considering other factors such as reliability of computations for both naturally large-scale volunteer systems and large-scale HPC systems (currently millions of cores). More specifically, this also requires elaboration of metrics to be evaluated in HPC and volunteer systems that would incorporate not only execution times and energy consumption, as, e.g., EDP or EDS [294], but also reliability and availability. We advise the reliability to be considered in the assessment of execution time and energy consumption of a specific application, given the probability of component failure of very large-scale Exascale systems, necessary to process massive data in various fields of science and economics [295].
2. Investigation of applicability of energy control mechanisms such as power capping (currently available for mobile, desktop, and server CPUs and GPUs) for the distributed type of systems, apart from high-performance computing machines and clusters; this is motivated by the need for more research toward energy savings and energy efficiency of computing, especially not fully loaded ones when

Table 5 Problem formulations in various contexts/domains

Formulation/domain	Data centers	Cloud computing	Fog computing	Blockchain	High-performance computing	Volunteer computing
ILP	[54–56]	[54–56]	[54–56]	[321–323]	[6, 7, 81]	–
Greedy algorithms	[9]	[9]	[250]	[324, 325]	[103–105]	[287]
Dynamic programming	[30, 185–187]	[117–120]	[117–119]	[326]	[106]	–
Nature-inspired algorithms	[12] (GA)	[13, 122] (GA), [327] (MBO), [128] (GA-ACO), [129] (ACO), [15, 16] (PSO)	[243] (SACO), [246] (GA-PSO), [17] (PSO), [328] (GA)	[267, 329–332]	[11, 12, 122, 123, 333] (GA)	[334–336] (GA)
Game theory	[19, 20, 22, 29]	[19, 20, 22, 29, 143, 147]	[21, 22, 29, 157, 337]	[338–340]	[23]	[148]
Reinforcement learning	[24, 320, 341–344]	[159, 342]	[25, 27, 345, 346]	[347–349]	[26, 285, 319]	[160]

energy can be saved and a small performance penalty can be accepted. Such research has been performed in the field of HPC [33] and can be investigated in distributed types of systems.

3. More thorough security solutions for high-performance computing systems (e.g., MPI-based solutions quite dated now) as well as volunteer solutions, considering not only the performance penalty but also the increased energy cost. While there exist solutions like ES-MPICH2 for encryption/decryption of communication among nodes [280, 281] or VAN-MPICH2 that integrates security measures to ensure data confidentiality using One-Time Pad (OTP) encryption [296], detailed assessment of trade-offs taking into account performance, security, and energy costs is advised. Furthermore, the same consideration is of importance for future volunteer-based systems in terms of performance, security, and energy costs of the technologies allowing to run client code on the volunteers' computers.
4. Consideration of the energy and other resources in approaches using reinforcement learning, apart from performance/execution times. More specifically, energy consumption and metrics describing the other resources can be incorporated into the reward function used by RL to be naturally used in its operation.
5. Consideration of using reinforcement learning for instruction-level parallelism in modern parallel architectures.
6. As energy costs are growing very considerably, some computing models might benefit from incorporation of a (even partly) more global view on optimization involving energy. This seems to be specifically applicable to volunteer computing that, as can be seen from Table 5, might then benefit from, e.g., ILP formulations that consider energy consumption or current power draw of geographical groups of volunteers. This, in essence, leads to mixing architectures such as partitioning of large-scale volunteer models into smaller ones with servers located, e.g., in several cloud systems. Additionally, energy consumption consideration of various parties, i.e., project owners and volunteers from various locations, shall be considered based on specific factors such as various carbon footprint in various locations, sources of energy in various locations resulting in various carbon footprint, etc.
7. While some automated approaches for optimization of performance energy exist, such as DEPO or Zeus, these typically make some assumptions regarding application model. For instance, DEPO assumes that the load of an application, after the tuning phase and setting an optimized power cap, would remain stable or would otherwise require rerunning the tuning. Zeus optimizes batch size and power caps for DNN training jobs. Ideally, a fully automated software for a hybrid CPU+GPU system would be desired that would perform dynamic, online performance energy optimization, with a low overhead, being able to do that for potentially changing and any workload type, based on, e.g., low-overhead profiling of a system through measuring of a set of metrics and responding by setting parameters like power caps, frequency/voltage based on, e.g., a pretrained DNN model. Already DRLCAP achieves a part of such a goal using GPU frequency capping for energy-aware reinforcement learning-based optimization of, e.g., EDP [275]. In paper [297], authors presented a model-free online energy efficiency optimization framework MF-GPOEO. It uses kernel activity information for finding optimized

GPU clock. In the future, application and system agnostic automated dynamic performance energy optimization tools for CPU+GPU systems will be needed, also considering scaling down GPUs [298] or using multiple GPUs [299] at the same time, considered as individual cases so far.

8. Typically existing works consider optimizing functions incorporating quality metrics and usage of resources from the point of view of a specific actor such as one user. Optimization functions shall also consider several views or functions involving, e.g., weighted resource optimization from user(s) and provider(s). Such views are especially valuable for environments such as clouds (user, provider), data and high-performance computing centers (user, provider center), volunteer computing (user versus global view).
9. Consideration of resources and associated metrics in optimization spanning many levels of parallel and distributed systems, i.e., clusters, nodes, computational devices, possibly with more detailed models for the resources of particular interest.

Another factor that we analyzed was the papers' publication dates for particular problem formulations × domains indicated in Table 5 indicating which contexts have been receiving the most attention recently. From the point of view of algorithm formulations, the analyzed articles listed for ILP range from 2008, 2011 up to present, for greedy algorithms from 2013, 2014 up to present, for dynamic programming from 2006 up to present, nature-inspired algorithms 2008 up to present, for game theory papers from 2008 to present (all but the HPC one considered are from 2019 onwards). We can see articles from 2014 onwards with increased intensity in 2020–present for RL. Based on that, we can see that all of those are of interest now, but clearly, the recent interest is associated with domains in various degrees. We can identify several sections depending on years of publications. Specifically, we can see that the research interest in volunteer computing has not been intense in recent years, in contrast with both blockchain and fog computing using all formulations; RL for HPC, data centers, fog, and cloud computing; dynamic programming for data centers and cloud, as well as nature-inspired algorithms for clouds.

Additionally, we believe that this paper can open a space for a unified resource model, inclusion of alternative emerging resources of interest in trade-offs, and optimization formulas as well as identification of links to involve surrounding disciplines that consider the same resources and metrics.

Author contributions The authors contributed to particular sections, as follows: Introduction (PC, HB, DG), background (PC, HB, CK, AKo, IO, MM, PN, GR), problem formulations (PC, HB, CK, AKo, IO, MM, PN, RP, GR), applications (PC, MA, HB, DG, AKe, AM, SK), open problems and conclusions (PC). PC was leading preparation of the article.

Funding This work is supported by CERCIRAS COST Action CA19135 funded by the COST Association.

Data availability No datasets were generated or analyzed during the current study.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

1. Czarnul P (2018) Parallel programming for modern high performance computing systems, 1st edn. Taylor & Francis, Chapman and Hall/CRC (**9781138305953**)
2. Mohammadi S, Pedram H, PourKarimi L (2018) Integer linear programming-based cost optimization for scheduling scientific workflows in multi-cloud environments. J Supercomput 74:4717–4745. https://doi.org/10.1007/s11227-018-2465-8
3. Mohammadi S, PourKarimi L, Pedram H (2019) Integer linear programming-based multi-objective scheduling for scientific workflows in multi-cloud environments. J Supercomput 75:6683–6709. https://doi.org/10.1007/s11227-019-02877-8
4. Bharathan S, Rajendran C, Sundarraj RP (2017) Penalty based mathematical models for web service composition in a geo-distributed cloud environment. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 886–889. https://doi.org/10.1109/ICWS.2017.113
5. Genez TAL, Bittencourt LF, Madeira ERM (2020) Time-discretization for speeding-up scheduling of deadline-constrained workflows in clouds. Future Gener Comput Syst 107(C):1116–1129
6. Boiński T, Czarnul P (2021) Optimization of Data Assignment for Parallel Processing in a Hybrid Heterogeneous Environment Using Integer Linear Programming. Comput J bxaa187
7. Czarnul P (2019) Integration of services into workflow applications. Taylor & Francis
8. Bhatti MK, Öz I, Amin S, Mushtaq M, Farooq U, Popov K, Brorsson M (2018) Locality-aware task scheduling for homogeneous parallel computing systems. Computing 100(6):557–595. https://doi.org/10.1007/s00607-017-0581-6
9. Paul AK, Addya SK, Sahoo B, Turuk AK (2014) Application of greedy algorithms to virtual machine distribution across data centers. In: 2014 Annual IEEE India Conference (INDICON), pp. 1–6. https://doi.org/10.1109/INDICON.2014.7030633
10. Bellman R (1957) Dynamic programming. Princeton University Press, Princeton
11. Fan Y, Lan Z, Rich P, Allcock WE, Papka ME, Austin B, Paul D (2019) Scheduling beyond CPUs for HPC. In: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing. ACM. https://doi.org/10.1145%2F3307681.3325401
12. Kassab A, Nicod J-M, Philippe L, Rehn-Sonigo V (2018) Assessing the use of genetic algorithms to schedule independent tasks under power constraints. In: 2018 International Conference on High Performance Computing Simulation (HPCS), pp. 252–259. https://doi.org/10.1109/HPCS.2018.00052
13. Sardaraz M, Tahir M (2020) A parallel multi-objective genetic algorithm for scheduling scientific workflows in cloud computing. Int J Distrib Sens Netw 16(8):1550147720949142. https://doi.org/10.1177/1550147720949142
14. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, pp. 1942–19484. https://doi.org/10.1109/ICNN.1995.488968

15. Pandey S, Wu L, Guru SM, Buyya R (2010) 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, pp. 400–407. https://doi.org/10.1109/AINA.2010.31
16. Gill SS, Buyya R, Chana I, Singh M, Abraham A (2018) Bullet: particle swarm optimization based scheduling technique for provisioned cloud resources. J Netw Syst Manag 26(2):361–400. https://doi.org/10.1007/s10922-017-9419-y
17. Potu N, Jatoth C, Parvataneni P (2021) Optimizing resource scheduling based on extended particle swarm optimization in fog computing environments. Concurr Comput Pract Exp 33(23):e6163. https://doi.org/10.1002/cpe.6163
18. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intell Mag 1(4):28–39. https://doi.org/10.1109/MCI.2006.329691
19. Maldonado-Carrascosa FJ, García-Galán S, Valverde-Ibáñez M, Marciniak T, Szczerska M, Ruiz-Reyes N (2024) Game theory-based virtual machine migration for energy sustainability in cloud data centers. Appl Energy 372:123798. https://doi.org/10.1016/j.apenergy.2024.123798
20. Benblidia MA, Brik B, Esseghir M, Merghem-Boulahia L (2021) A renewable energy-aware power allocation for cloud data centers: A game theory approach. Comput Commun 179:102–111
21. Sharghivand N, Derakhshan F, Mashayekhy L, Mohammad Khanli L (2020) An edge computing matching framework with guaranteed quality of service. IEEE Trans Cloud Comput. https://doi.org/10.1109/TCC.2020.3005539
22. Mehran N, Samani ZN, Kimovski D, Prodan R (2022) Matching-based scheduling of asynchronous data processing workflows on the computing continuum. In: 2022 IEEE International Conference on Cluster Computing (CLUSTER), pp. 58–70
23. Ahmad I, Ranka S, Khan SU (2008) Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In: 2008 IEEE International Symposium on Parallel and Distributed Processing, pp. 1–6. https://doi.org/10.1109/IPDPS.2008.4536420
24. Hu X, Sun Y (2020) A deep reinforcement learning-based power resource management for fuel cell powered data centers. Electronics 9(12):2054
25. Goudarzi M, Palaniswami M, Buyya R (2023) A distributed deep reinforcement learning technique for application placement in edge and fog computing environments. IEEE Trans Mob Comput 22(5):2491–2505. https://doi.org/10.1109/TMC.2021.3123165
26. Fan Y, Lan Z (2021) Dras-cqsim: a reinforcement learning based framework for hpc cluster scheduling. Software Impacts 8:100077
27. Tu Y, Chen H, Yan L, Zhou X (2022) Task offloading based on lstm prediction and deep reinforcement learning for efficient edge computing in iot. Future Internet 14(2):30
28. Nikolow D, Slota R, Polak S, Pogoda M, Kitowski J (2018) Policy-based SLA storage management model for distributed data storage services. Comput Sci 19(4). https://doi.org/10.7494/csci.2018.19.4.2878
29. Mehran N, Kimovski D, Prodan R (2021) A two-sided matching model for data stream processing in the cloud - fog continuum. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 514–524. IEEE
30. Li X, Nie L, Chen S (2014) Approximate dynamic programming based data center resource dynamic scheduling for energy optimization. In: 2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom), pp. 494–501. https://doi.org/10.1109/iThings.2014.87
31. Genez TAL, Bittencourt LF, Madeira ERM (2012) Workflow scheduling for saas / paas cloud providers considering two sla levels. In: 2012 IEEE Network Operations and Management Symposium, pp. 906–912. https://doi.org/10.1109/NOMS.2012.6212007
32. Kessler C, Litzinger S, Keller J (2020) Static scheduling of moldable streaming tasks with task fusion for parallel systems with DVFS. IEEE Trans Comput-Aided Des Integrated Circuits Syst (TCAD) 39(11):4166–4178
33. Kocot B, Czarnul P, Proficz J (2023) Energy-aware scheduling for high-performance computing systems: a survey. Energies 16(2):890. https://doi.org/10.3390/en16020890
34. Mukherjee S (2008) Architecture design for soft errors. Morgan Kaufmann Publishers Inc., San Francisco
35. Hsueh M-C, Tsai TK, Iyer RK (1997) Fault injection techniques and tools. Computer 30(4):75–82

36. Oz I, Arslan S (2019) A survey on multithreading alternatives for soft error fault tolerance. ACM Comput Surv 52(2):1–38
37. Veronesi A, Nazzari A, Passarello D, Krstic M, Favalli M, Cassano L, Miele A, Bertozzi D, Bolchini C (2024) Cross-layer reliability analysis of nvdla accelerators: Exploring the configuration space. In: 2024 IEEE European Test Symposium (ETS), pp. 1–6. https://doi.org/10.1109/ETS61313.2024.10568018
38. Sezgin Y, Oz I (2024) Performance-reliability tradeoff analysis for safety-critical embedded systems with gpus. In: Yüksek Başarımlı Hesaplama Konferansı (BAŞARIM). https://indico.truba.gov.tr/event/140/attachments/310/642/BASARIM2024-BildiriKitabi.pdf
39. Katal A, Dahiya S, Choudhury T (2023) Energy efficiency in cloud computing data centers: a survey on software technologies. Clust Comput 26(3):1845–1875
40. Mastelic T, Oleksiak A, Claussen H, Brandic I, Pierson J-M, Vasilakos AV (2014) Cloud computing: survey on energy efficiency. ACM Comput Surv 47(2):1–36. https://doi.org/10.1145/2656204
41. Diouani S, Medromi H (2019) Trade-off between performance and energy management in autonomic and green data centers. In: Proceedings of the 2nd International Conference on Networking, Information Systems & Security. NISS19. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3320326.3320332
42. Krzywaniak A, Czarnul P, Proficz J (2023) Dynamic GPU power capping with online performance tracing for energy efficient GPU computing using DEPO tool. Future Gener Comput Syst 145:396–414. https://doi.org/10.1016/J.FUTURE.2023.03.041
43. Coutinho Demetrios AM, De Sensi D, Lorenzon AF, Georgiou K, Nunez-Yanez J, Eder K, Xavier-de-Souza S (2020) Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems. Energies 13(9):2409. https://doi.org/10.3390/en13092409
44. Müller S (2017) Security trade-offs in cloud storage systems. PhD thesis, Technischen Universität Berlin, Berlin, Germany
45. Mishra A, Reichherzer T, Kalaimannan E, Wilde N, Ramirez R (2020) Trade-offs involved in the choice of cloud service configurations when building secure, scalable, and efficient internet-of-things networks. Int J Distrib Sens Netw 16(2):1550147720908199. https://doi.org/10.1177/1550147720908199
46. Pandurangan G, Robinson P, Scquizzato M (2019) A time-and message-optimal distributed algorithm for minimum spanning trees. ACM Trans Algorithms (TALG) 16(1):1–27
47. Gupta M, Roberts D, Meswani M, Sridharan V, Tullsen D, Gupta R (2016) Reliability and performance trade-off study of heterogeneous memories. In: Proceedings of the Second International Symposium on Memory Systems. MEMSYS '16, pp. 395–401. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2989081.2989113
48. Oz I, Topcuoglu HR, Kandemir M, Tosun O (2012) Performance-reliability tradeoff analysis for multithreaded applications. In: 2012 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 893–898. https://doi.org/10.1109/DATE.2012.6176624
49. Baniata H, Kertesz A (2022) Bitcoin revisited: formalization, benchmarking, and open security issues
50. Ghosh R, Simmhan Y (2018) Distributed scheduling of event analytics across edge and cloud. ACM Trans -Phys Syst 2(4):1–28
51. Wang A, Chen L, Xu W (2017) Xpro: a cross-end processing architecture for data analytics in wearables. ACM SIGARCH Comput Architecture News 45(2):69–80
52. Kolomvatsos K, Anagnostopoulos C (2019) Multi-criteria optimal task allocation at the edge. Futur Gener Comput Syst 93:358–372
53. Shah-Mansouri H, Wong VW (2018) Hierarchical fog-cloud computing for iot systems: a computation offloading game. IEEE Internet Things J 5(4):3246–3257
54. Kouloumpris A, Michael MK, Theocharides T (2019) Reliability-aware task allocation latency optimization in edge computing. In: 2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS), pp. 200–203. IEEE
55. Kouloumpris A, Theocharides T, Michael MK (2020) Cost-effective time-redundancy based optimal task allocation for the edge-hub-cloud systems. In: 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 368–373. IEEE
56. Kouloumpris A, Theocharides T, Michael MK (2019) Metis: optimal task allocation framework for the edge/hub/cloud paradigm. In: Proceedings of the International Conference on Omni-Layer Intelligent Systems, pp. 128–133

57. Kouloumpris A, Stavrinides GL, Michael MK, Theocharides T (2024) Optimal multi-constrained workflow scheduling for cyber-physical systems in the edge-cloud continuum. In: Proceedings of the IEEE Annual International Computer Software and Applications Conference (COMPSAC), pp. 483–492. https://doi.org/10.1109/COMPSAC61105.2024.00072. in press
58. Dong L, Wu W, Guo Q, Satpute MN, Znati T, Du DZ (2019) Reliability-aware offloading and allocation in multilevel edge computing system. IEEE Trans Reliab 70(1):200–211
59. Liu C-F, Bennis M, Poor HV (2017) Latency and reliability-aware task offloading and resource allocation for mobile edge computing. In: 2017 IEEE Globecom Workshops (GC Wkshps), pp. 1–7. IEEE
60. Dinh TQ, Tang J, La QD, Quek TQ (2017) Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans Commun 65(8):3571–3584
61. Nikolaou P, Sazeides Y, Lampropoulos A, Guilhot D, Bartoli A, Papadimitriou G, Chatzidimitriou A, Gizopoulos D, Tovletoglou K, Mukhanov L et al (2019) On the evaluation of the total-cost-of-ownership trade-offs in edge vs cloud deployments: a wireless-denial-of-service case study. IEEE Transactions on Sustainable Computing
62. Hennessy JL, Patterson DA (2012) Computer architecture - a quantitative approach, 5th Edition. Morgan Kaufmann
63. Brent RP (1974) The parallel evaluation of general arithmetic expressions. J ACM 12(2):201–206
64. Kahn G (1974) The semantics of a simple language for parallel programming. In: Proceedings IFIP Congress on Information Processing, pp. 471–475. North-Holland
65. Augonnet C, Thibault S, Namyst R, Wacrenier P (2011) StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurr Comput Pract Exp 23(2):187–198. https://doi.org/10.1002/cpe.1631
66. Ernstsson A, Ahlqvist J, Zouzoula S, Kessler C (2021) SkePU 3: portable high-level programming of heterogeneous systems and HPC clusters. Int J Parallel Prog 49:846–866. https://doi.org/10.1007/s10766-021-00704-3
67. Konak A, Coit DW, Smith AE (2006) Multi-objective optimization using genetic algorithms: a tutorial. Reliab Eng Syst Saf 91(9):992–1007. https://doi.org/10.1016/j.ress.2005.11.018. (**Special Issue - Genetic Algorithms and Reliability**)
68. Katoh N, Ibaraki T (1998) Resource allocation problems. Handbook of Combinatorial Optimization: Vol. 1–3, 905–1006
69. Shi C, Zhang H, Qin C (2015) A faster algorithm for the resource allocation problem with convex cost functions. J Discrete Algorithms 34:137–146
70. Li S, Liu H, Li W, Sun W (2024) Non-convex optimization of resource allocation in fog computing using successive approximation. J Syst Sci Complexity 37(2):805–840
71. Tatarenko T, Touri B (2017) Non-convex distributed optimization. IEEE Trans Autom Control 62(8):3744–3757
72. Rashid ZN, Zebari SRM, Sharif KH, Jacksi K (2018) Distributed cloud computing and distributed parallel computing: a review. In: 2018 International Conference on Advanced Science and Engineering (ICOASE), pp. 167–172. https://doi.org/10.1109/ICOASE.2018.8548937
73. Mann ZA (2015) Allocation of virtual machines in cloud data centers-a survey of problem models and optimization algorithms. ACM Comput Surv 48(1):1–34. https://doi.org/10.1145/2797211
74. Salimi S, Mawlana M, Hammad A (2018) Performance analysis of simulation-based optimization of construction projects using high performance computing. Autom Constr 87:158–172. https://doi.org/10.1016/j.autcon.2017.12.003
75. Hamadi Y, Sais L (2018) Handbook of parallel constraint reasoning, 1st edn. Springer
76. Beaumont O, Canon L-C, Eyraud-Dubois L, Lucarelli G, Marchal L, Mommessin C, Simon B, Trystram D (2020) Scheduling on two types of resources: a survey. ACM Comput Surv 53(3):1–36. https://doi.org/10.1145/3387110
77. Czarnul P, Proficz J, Drypczewski K (2020) Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems. Sci Program 2020(1):4176794. https://doi.org/10.1155/2020/4176794
78. Pervan B, Knezović J (2020) A survey on parallel architectures and programming models. In: 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 999–1005. https://doi.org/10.23919/MIPRO48935.2020.9245341
79. Fang J, Huang C, Tang T, Wang Z (2020) Parallel programming models for heterogeneous many-cores: a comprehensive survey. CCF Trans High Perform Comput 2(4):382–400. https://doi.org/10.1007/s42514-020-00039-4

80. Anand R, Aggarwal D, Kumar V (2017) A comparative analysis of optimization solvers. J Stat Manag Syst 20(4):623–635. https://doi.org/10.1080/09720510.2017.1395182
81. Czarnul P (2011) Parallelization of compute intensive applications into workflows based on services in beesycluster. Scalable Comput Pract Exp 12(2)
82. Kessler CW, Litzinger S, Keller J (2021) Crown-scheduling of sets of parallelizable tasks for robustness and energy-elasticity on many-core systems with discrete dynamic voltage and frequency scaling. J Syst Archit 115:101999
83. Kessler CW, Melot N, Eitschberger P, Keller J (2013) Crown scheduling: energy-efficient resource allocation, mapping and discrete frequency scaling for collections of malleable streaming tasks. In: 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation, pp. 215–222
84. Melot N, Kessler C, Keller J, Eitschberger P (2015) Fast Crown scheduling heuristics for energy-efficient mapping and scaling of moldable streaming tasks on manycore systems. ACM Trans Archit Code Optim 11(4):1–24
85. Melot N, Kessler C, Eitschberger P, Keller J (2019) Co-optimizing core allocation, mapping and DVFS in streaming programs with moldable tasks for energy efficient execution on manycore architectures. In: Proceeding 19th International Conference on Application of Concurrency to System Design (ACSD 2019). IEEE. https://doi.org/10.1109/ACSD.2019.00011
86. Boulasikis M, Kessler C, Gruian F, Keller J, Litzinger S (2024) Packet-type aware scheduling of moldable streaming tasks on multicore systems with DVFS. In: Proceedings 39th ACM/SIGAPP Symposium on Applied Computing. SAC '24, pp. 449–451. ACM. https://doi.org/10.1145/3605098.3636081
87. Khosravi S, Kessler C, Litzinger S, Keller J (2024) Energy-efficient scheduling of moldable streaming computations for the edge-cloud continuum. In: 9th International Conference on Fog and Mobile Edge Computing (FMEC), pp. 268–276. https://doi.org/10.1109/FMEC62297.2024.10710310
88. Melot N, Kessler C, Keller J (2020) Voltage island-aware energy-efficient scheduling of parallel streaming tasks on many-core CPUs. In: Proceedings 28th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP'20), Västerås, Sweden, March 2020, pp. 157–161. IEEE. https://doi.org/10.1109/PDP50117.2020.00030
89. Litzinger S, Keller J, Kessler C (2019) Scheduling moldable parallel streaming tasks on heterogeneous platforms with frequency scaling. In: Proceedings 27th European Signal Processing Conference (EUSIPCO 2019)
90. Kessler C, Litzinger S, Keller J (2020) Adaptive crown scheduling for streaming tasks on many-core systems with discrete DVFS. In: Euro-Par 2019: Parallel Processing Workshops, pp. 17–29. Springer,
91. Xu H, Kong F, Deng Q (2012) Energy minimizing for parallel real-time tasks based on level-packing. In: 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'12), pp. 98–103. https://doi.org/10.1109/RTCSA.2012.10
92. Keller J, Litzinger S (2022) Systematic search space design for energy-efficient static scheduling of moldable tasks. J Parallel Distributed Comput 162:44–58. https://doi.org/10.1016/j.jpdc.2022.01.004
93. Kouloumpris A, Stavrinides GL, Michael MK, Theocharides T (2024) An optimization framework for task allocation in the edge/hub/cloud paradigm. Futur Gener Comput Syst 155:354–366. https://doi.org/10.1016/j.future.2024.02.005
94. Ortega-Arranz H, Llanos DR, Gonzalez-Escribano A (2022) The shortest-path problem. Analysis and comparison of methods. Springer
95. Sedgewick R, Wayne K (2011) Algorithms, 4th edn. Addison-Wesley Professional
96. Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM J. Appl Math 17:416–429
97. Garey MR, Graham RL (1975) Bounds for multiprocessor scheduling with resource constraints. SIAM J Comput 4(2):187–200
98. Ellis JR (1985) Bulldog: a compiler for VLIW architectures. PhD thesis, Yale University
99. Faraboschi P, Fisher JA, Young C (2001) Instruction scheduling for instruction level parallel processors. Proc IEEE 89(11):1638–1659
100. Topcuoglu H, Hariri S, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274. https://doi.org/10.1109/71.993206

101. Albers S, Schmidt M (2005) On the performance of greedy algorithms in packet buffering. SIAM J Comput 35(2):278–304. https://doi.org/10.1137/S0097539704446268
102. Yousif A, Alqhtani SM, Bashir MB, Ali A, Hamza R, Hassan A, Tawfeeg TM (2022) Greedy firefly algorithm for optimizing job scheduling in iot grid computing. Sensors 22(3):850
103. Dupont B, Mejri N, Da Costa G (2020) Energy-aware scheduling of malleable hpc applications using a particle swarm optimised greedy algorithm. SustainComput: Inform Syst 28:100447
104. Marchal L, Simon B, Sinnen O, Vivien F (2018) Malleable task-graph scheduling with a practical speed-up model. IEEE Trans Parallel Distrib Syst 29(6):1357–1370. https://doi.org/10.1109/TPDS.2018.2793886
105. Czarnul P, Rościszewski P (2014) Optimization of execution time under power consumption constraints in a heterogeneous parallel system with gpus and cpus. In: Chatterjee M, Cao J-n, Kothapalli K, Rajsbaum S (eds.) Distributed Computing and Networking, pp. 66–80. Springer, Berlin, Heidelberg
106. Keßler CW, Bednarski A (2006) Optimal integrated code generation for VLIW architectures. Concurr Comput Pract Exp 18(11):1353–1390
107. Vegdahl SR (1992) A dynamic-programming technique for compacting loops. In: Hwu, W.W. (ed.) Proceedings 25th Annual International Symposium on Microarchitecture, pp. 180–188. ACM / IEEE Computer Society. https://doi.org/10.1109/MICRO.1992.697014
108. Leupers R, Marwedel P (1997) Time-constrained code compaction for DSPs. IEEE Trans Very Large Scale Integ syst 5(1):112–122
109. Wilken KD, Liu J, Heffernan M (2000) Optimal instruction scheduling using integer programming. In: Lam, M.S. (ed.) Proceedings ACM SIGPLAN Conferece on Programming Language Design and Implementation (PLDI), pp. 121–133. ACM. https://doi.org/10.1145/349299.349318
110. Winkel S (2004) Exploring the performance potential of Itanium® processors with ILP-based scheduling. In: 2nd IEEE / ACM International Symposium on Code Generation and Optimization (CGO 2004), pp. 189–200. IEEE Computer Society. https://doi.org/10.1109/CGO.2004.1281674
111. Bednarski A, Kessler CW (2006) Optimal integrated VLIW code generation with integer linear programming. In: Nagel WE, Walter WV, Lehner W (eds.) Proceedings of Euro-Par 2006 Conference, Dresden, Germany. Lecture Notes in Computer Science, vol. 4128, pp. 461–472. Springer. https://doi.org/10.1007/11823285_48
112. Eriksson MV, Kessler CW (2012) Integrated code generation for loops. ACM Trans Embed Comput Syst 11(S1):19
113. Bashford S, Leupers R (1999) Phase-coupled mapping of data flow graphs to irregular data paths. Des Autom Embed Syst 4(2–3):119–165
114. Lozano RC, Blindell GH, Carlsson M, Drejhammar F, Schulte C (2013) Constraint-based code generation. In: Corporaal H, Stuijk S (eds.) International Workshop on Software and Compilers for Embedded Systems, M-SCOPES'13, Sankt Goar, Germany, pp. 93–95. ACM. https://doi.org/10.1145/2463596.2486155
115. Kessler CW (2019) Compiling for VLIW DSPs. In: Bhattacharyya SS, Deprettere EF, Leupers R, Takala J (eds.) Handbook of Signal Processing Systems, pp. 1177–1214. Springer. https://doi.org/10.1007/978-3-319-91734-4_27
116. Lozano RC, Schulte C (2019) Survey on combinatorial register allocation and instruction scheduling. ACM Comput Surv 52(3):62–16250
117. Sharkh MA, Kalil M (2020) A dynamic algorithm for fog computing data processing decision optimization. In: 2020 IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1–6. https://doi.org/10.1109/ICCWorkshops49005.2020.9145296
118. Bai W, Yang Z, Zhang J, Kumar R (2021) Randomization-based dynamic programming offloading algorithm for mobile fog computing. Secur Commun Netw 2021:4348511–143485119
119. Gai K, Qiu M, Liu M (2018) Privacy-preserving access control using dynamic programming in fog computing. In: 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), pp. 126–132. https://doi.org/10.1109/BDS/HPSC/IDS18.2018.00037
120. Wang P, Zhou W, Zhao C, Lei Y, Zhang Z (2020) A dynamic programming-based approach for cloud instance type selection and optimisation. Int J Inf Technol Manag 19(4):358–375. https://doi.org/10.1504/ijitm.2020.110240
121. Czarnul P (2014) Comparison of selected algorithms for scheduling workflow applications with dynamically changing service availability. J Zhejiang Univ Sci C 15(6):401–422

122. Quang-Hung N, Tan LT, Phat CT, Thoai N (2014) A gpu-based enhanced genetic algorithm for power-aware task scheduling problem in HPC cloud. In: Linawati, Mahendra MS, Neuhold EJ, Tjoa AM, You I (eds.) Information and Communication Technology - Second IFIP TC5/8 International Conference, ICT-EurAsia 2014, Bali, Indonesia, April 14-17, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8407, pp. 159–169. Springer. https://doi.org/10.1007/978-3-642-55032-4_16
123. Dunlop D, Varrette S, Bouvry P (2008) On the use of a genetic algorithm in high performance computer benchmark tuning. In: 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, pp. 105–113
124. Duman E, Uysal M, Alkaya AF (2012) Migrating birds optimization: a new metaheuristic approach and its performance on quadratic assignment problem. Inf Sci 217:65–77
125. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
126. Camacho-Villalón CL, Dorigo M, Stützle T (2023) Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors. Int Trans Oper Res 30(6):2945–2971
127. Fidanova S, Luque G, Roeva O, Paprzycki M, Gepner P (2018) Hybrid ant colony optimization algorithm for workforce planning. In: 2018 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 233–236
128. Kumar AMS, Venkatesan M (2019) Multi-objective task scheduling using hybrid genetic-ant colony optimization algorithm in cloud environment. Wirel Pers Commun 107(4):1835–1848
129. Jia Y-H, Chen W-N, Yuan H, Gu T, Zhang H, Gao Y, Zhang J (2021) An intelligent cloud workflow scheduling system with time estimation and adaptive ant colony optimization. IEEE Trans Syst, Man, Cyber: Syst 51(1):634–649. https://doi.org/10.1109/TSMC.2018.2881018
130. Neumann Jv, Morgenstern O (1953) Theory of games and economic behavior
131. Kreps DM (1989) Nash equilibrium. In: Game Theory, pp. 167–177. Springer
132. Shapley LS (1969) Utility comparison and the theory of games. The Shapley Value. Essays in Honor of Lloyd S. Shapley, 307–319
133. Chang S-L, Lee K-C, Huang R-R, Liao Y-H (2021) Resource-allocation mechanism: Game-theory analysis. Symmetry 13(5):799
134. Narbaev T, Hazır Ö, Agi M (2022) A review of the use of game theory in project management. J Manag Eng 38(6):03122002
135. Grigoryan G, Collins AJ (2021) Game theory for systems engineering: a survey. Int J Syst Syst Eng 11(2):121–158
136. Marousi A, Charitopoulos VM (2023) Game theoretic optimisation in process and energy systems engineering: a review. Front Chem Eng 5:1130568
137. Riahi S, Riahi A (2019) Game theory for resource sharing in large distributed systems. Int J Electr & Comput Eng (2088-8708) 9(2)
138. Shamshirband S, Joloudari JH, Shirkharkolaie SK, Mojrian S, Rahmani F, Mostafavi S, Mansor Z (2021) Game theory and evolutionary optimization approaches applied to resource allocation problems in computing environments: A survey. Math Biosci Eng 18(6):9190–9232
139. Agbaje M, Ohwo O, Ayanwola T, Olufunmilola O (2022) A survey of game-theoretic approach for resource management in cloud computing. J Comput Netw Commun 2022(1):9323818
140. Chi C, Wang Y, Tong X, Siddula M, Cai Z (2021) Game theory in internet of things: a survey. IEEE Internet Things J 9(14):12125–12146
141. Ogidiaka E, Nonyelum OF, Irhebhude ME (2021) Game-theoretic resource allocation algorithms for device-to-device communications in fifth generation cellular networks: a review. Int J Inf Eng Electr Bus (IJIEEB) 13(1):44–51
142. Wang Q, Zhou Y, Ni Y, Zhao H, Zhu H (2019) A review of game theoretical resource allocation methods in wireless communications. In: 2019 IEEE 19th International Conference on Communication Technology (ICCT), pp. 881–887. IEEE
143. Yang J, Jiang B, Lv Z, Choo K-KR (2020) A task scheduling algorithm considering game theory designed for energy management in cloud computing. Futur Gener Comput Syst 105:985–992
144. Ding X, Zhang W (2021) Computing unloading strategy of massive internet of things devices based on game theory in mobile edge computing. Math Probl Eng 2021(1):2163965
145. Zeng X (2022) Game theory-based energy efficiency optimization model for the internet of things. Comput Commun 183:171–180
146. Moafi M, Ardeshiri RR, Mudiyanselage MW, Marzband M, Abusorrah A, Rawa M, Guerrero JM (2023) Optimal coalition formation and maximum profit allocation for distributed energy

resources in smart grids based on cooperative game theory. Int J Electr Power & Energy Syst 144:108492
147. Hosseini S, Vakili R (2019) Game theory approach for detecting vulnerable data centers in cloud computing network. Int J Commun Syst 32(8):3938. https://doi.org/10.1002/dac.3938. (**e3938 IJCS-18-0436.R2**)
148. Dhamal S, Ben-Ameur W, Chahed T, Altman E, Sunny A, Poojary S (2024) A game theoretic framework for distributed computing with dynamic set of agents. Ann Oper Res 336(3):1871–1904. https://doi.org/10.1007/s10479-023-05231-
149. Chen Y, Li Z, Yang B, Nai K, Li K (2020) A stackelberg game approach to multiple resources allocation and pricing in mobile edge computing. Futur Gener Comput Syst 108:273–287
150. Jie Y, Tang X, Choo K-KR, Su S, Li M, Guo C (2018) Online task scheduling for edge computing based on repeated stackelberg game. J Parallel Distributed Comput 122:159–172. https://doi.org/10.1016/j.jpdc.2018.07.019
151. Lyu T, Xu H, Liu F, Li M, Li L, Han Z (2024) Two layer stackelberg game-based resource allocation in cloud-network convergence service computing. IEEE Transactions on Cognitive Communications and Networking
152. Shamshirband S, Joloudari JH, Shirkharkolaie SK, Mojrian S, Rahmani F, Mostafavi S, Mansor Z (2021) Game theory and evolutionary optimization approaches applied to resource allocation problems in computing environments: A survey. Math Biosci Eng 18(6):9190–9232
153. Bayat S, Li Y, Song L, Han Z (2016) Matching theory: applications in wireless communications. IEEE Signal Process Mag 33(6):103–122
154. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69(1):9–15
155. Diebold F, Aziz H, Bichler M, Matthes F, Schneider A (2014) Course allocation via stable matching. Bus & Inf Syst Eng 6(2):97–110
156. Wilde H, Knight V, Gillard J (2020) Matching: a python library for solving matching games. J Open Source Softw 5(48):2169
157. Sharghivand N, Derakhshan F, Mashayekhy L (2018) Qos-aware matching of edge computing services to internet of things. In: 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC), pp. 1–8. IEEE
158. Kimovski D, Mathá R, Hammer J, Mehran N, Hellwagner H, Prodan R (2021) Cloud, fog or edge: Where to compute? IEEE Internet Computing
159. Zhou G, Tian W, Buyya R, Xue R, Song L (2024) Deep reinforcement learning-based methods for resource scheduling in cloud computing: a review and future directions. Artif Intell Rev 57:124
160. McGough AS, Forshaw M (2014) Reduction of wasted energy in a volunteer computing system through reinforcement learning. Sustain Comput: Inform Syst 4(4):262–275
161. Song X, Hua Y, Yang Y, Xing G, Liu F, Xu L, Song T (2024) Distributed resource allocation with federated learning for delay-sensitive iov services. IEEE Trans Veh Technol 73(3):4326–4336. https://doi.org/10.1109/TVT.2023.3328988
162. Ji Z, Qin Z, Tao X (2024) Meta federated reinforcement learning for distributed resource allocation. IEEE Trans Wireless Commun 23(7):7865–7876. https://doi.org/10.1109/TWC.2023.3345363
163. Goiri Í, Katsak W, Le K, Nguyen T, Bianchini R (2013) Parasol and greenswitch: Managing datacenters powered by renewable energy. In: ASPLOS 2013 - 18th International Conference on Architectural Support for Programming Languages and Operating Systems–ASPLOS, pp. 51–63. https://doi.org/10.1145/2451116.2451123
164. Cioara T, Anghel I, Antal M, Crisan S, Salomie I (2015) Data center optimization methodology to maximize the usage of locally produced renewable energy. Sustain Internet ICT for Sustain (SustainIT) 2015:1–8
165. Baniata H, Mahmood S, Kertesz A (2021) Assessing anthropogenic heat flux of public cloud data centers: current and future trends. PeerJ Comput Sci 7:478
166. Antal M, Pop C, Cioara T, Anghel I, Tamas I, Salomie I (2017) Proactive day-ahead data center operation scheduling for energy efficiency: Solving a miocp using a multi-gene genetic algorithm. In: 2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), pp. 527–534. https://doi.org/10.1109/ICCP.2017.8117058
167. Knitro (2022) Knitro Software. https://www.artelys.com/solvers/knitro/. [Online; accessed 21-march-2022]

168. Parolini L, Sinopoli B, Krogh BH, Wang Z (2012) A cyber-physical systems approach to data center modeling and control for energy efficiency. Proc IEEE 100(1):254–268. https://doi.org/10.1109/JPROC.2011.2161244
169. Moore J, Chase J, Ranganathan P, Sharma R (2005) Making scheduling "cool": Temperature-Aware workload placement in data centers. In: 2005 USENIX Annual Technical Conference (USENIX ATC 05). USENIX Association, Anaheim, CA. https://www.usenix.org/conference/2005-usenix-annual-technical-conference/making-scheduling-cool-temperature-aware-workload
170. Tang Q, Mukherjee T, Gupta SKS, Cayton P (2006) Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters. In: 2006 4th International Conference on Intelligent Sensing and Information Processing, pp. 203–208. https://doi.org/10.1109/ICISIP.2006.4286097
171. Zhang Y, Wang Y, Wang X (2012) Testore: Exploiting thermal and energy storage to cut the electricity bill for datacenter cooling. In: 2012 8th International Conference on Network and Service Management (cnsm) and 2012 Workshop on Systems Virtualiztion Management (svm), pp. 19–27
172. Zheng W, Ma K, Wang X (2014) Exploiting thermal energy storage to reduce data center capital and operating expenses. In: 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 132–141. https://doi.org/10.1109/HPCA.2014.6835924
173. Das R, Yarlanki S, Hamann H, Kephart JO, Lopez V (2011) A unified approach to coordinated energy-management in data centers. In: 2011 7th International Conference on Network and Service Management, pp. 1–5
174. Wang L, Khan SU, Dayal J (2011) Thermal aware workload placement with task-temperature profiles in a data center. J Supercomput 61:780–803
175. Wang L, Laszewski G, Dayal J, Furlani TR (2009) Thermal aware workload scheduling with backfilling for green data centers. In: 2009 IEEE 28th International Performance Computing and Communications Conference, pp. 289–296. https://doi.org/10.1109/PCCC.2009.5403821
176. Wang L, Laszewski G, Dayal J, He X, Younge AJ, Furlani TR (2009) Towards thermal aware workload scheduling in a data center. In: 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks, pp. 116–122. https://doi.org/10.1109/I-SPAN.2009.22
177. Kaur A, Kinger S (2013) Temperature aware resource scheduling in green clouds. In: 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1919–1923. https://doi.org/10.1109/ICACCI.2013.6637475
178. Van Damme T, De Persis C, Tesi P (2019) Optimized thermal-aware job scheduling and control of data centers. IEEE Trans Control Syst Technol 27(2):760–771. https://doi.org/10.1109/TCST.2017.2783366
179. Marcel A, Cristian P, Eugen P, Claudia P, Cioara T, Anghel I, Ioan S (2016) Thermal aware workload consolidation in cloud data centers. In: 2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP), pp. 377–384. https://doi.org/10.1109/ICCP.2016.7737177
180. Dharkar S, Kurtulus O, Groll EA, Yazawa K (2014) Analysis of a data center using liquid-liquid co2 heat pump for simultaneous cooling and heating
181. Houbak-Jensen L, Holten A, Blarke MB, Groll EA, Shakouri A, Yazawa K (2013) Dynamic analysis of a dual-mode CO2 heat pump with both hot and cold thermal storage. ASME International Mechanical Engineering Congress and Exposition, vol. Volume 8B: Heat Transfer and Thermal Engineering. https://doi.org/10.1115/IMECE2013-62894. V08BT09A039
182. Wahlroos M, Pärssinen M, Manner J, Syri S (2017) Utilizing data center waste heat in district heating - impacts on energy efficiency and prospects for low-temperature district heating networks. Energy 140:1228–1238. https://doi.org/10.1016/j.energy.2017.08.078
183. Wahlroos M, Pärssinen M, Rinne S, Syri S, Manner J (2018) Future views on waste heat utilization - case of data centers in northern Europe. Renew Sustain Energy Rev 82:1749–1764. https://doi.org/10.1016/j.rser.2017.10.058
184. Antal M, Cioara T, Anghel I, Pop C, Salomie I (2018) Transforming data centers in active thermal energy players in nearby neighborhoods. Sustainability 10(4):939. https://doi.org/10.3390/su10040939
185. Shi L, Shi Y, Wei X, Ding X, Wei Z (2017) Cost minimization algorithms for data center management. IEEE Trans Parallel Distrib Syst 28(1):60–71. https://doi.org/10.1109/TPDS.2016.2549016

186. Abadi MF, Haghighat F, Nasiri F (2022) Application of dynamic programming in developing availability-based maintenance prioritization model for data centers. In: INFORMS Conference on Service Science, Shenzen, China
187. Xie J, Lyu L, Deng Y, Yang LT (2015) Improving routing performance via dynamic programming in large-scale data centers. IEEE Internet Things J 2(4):321–328. https://doi.org/10.1109/JIOT.2014.2386326
188. Buyya R, Vecchiola C, Selvi T (2013) Mastering cloud computing. McGraw Hill, San Francisco
189. Mell P, Grance T (2011) The NIST definition of cloud computing
190. Vogel A, Griebler D, Maron CAF, Schepke C, Fernandes LG (2016) Private iaas clouds: A comparative analysis of opennebula, cloudstack and openstack. In: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), pp. 672–679. https://doi.org/10.1109/PDP.2016.75
191. Alboaneen DA, Pranggono B, Tianfield H (2014) Energy-aware virtual machine consolidation for cloud data centers. In: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp. 1010–1015. https://doi.org/10.1109/UCC.2014.166
192. Kayed A, Akijian T (2015) Resource allocation technique to obtain energy efficient cloud. In: Proceedings of the The International Conference on Engineering & MIS 2015. ICEMIS '15. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2832987.2833028
193. Soltanshahi M, Asemi R, Shafiei N (2019) Energy-aware virtual machines allocation by krill herd algorithm in cloud data centers. Heliyon 5(7):02066. https://doi.org/10.1016/j.heliyon.2019.e02066
194. Berral JL, Goiri In, Nou R, Julià F, Guitart J, Gavaldà R, Torres J (2010) Towards energy-aware scheduling in data centers using machine learning. In: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. e-Energy '10, pp. 215–224. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1791314.1791349
195. Cao B, Gao X, Chen G, Jin Y (2014) Nice: Network-aware vm consolidation scheme for energy conservation in data centers. In: 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 166–173. https://doi.org/10.1109/PADSW.2014.7097805
196. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. John Wiley & Sons Inc, USA
197. Haque Monil MA, Qasim R, Rahman RM (2014) Energy-aware vm consolidation approach using combination of heuristics and migration control. In: 9th International Conference on Digital Information Management (ICDIM 2014), pp. 74–79. https://doi.org/10.1109/ICDIM.2014.6991413
198. Joshi S, Kaur S (2015) Cuckoo search approach for virtual machine consolidation in cloud data centre. In: International Conference on Computing, Communication Automation, pp. 683–686. https://doi.org/10.1109/CCAA.2015.7148461
199. Lee S, Panigrahy R, Prabhakaran V, Ramasubramanian V, Talwar K, Uyeda LK, Wieder U (2011) Validating heuristics for virtual machines consolidation
200. Roytman A, Kansal A, Govindan S, Liu J, Nath S (2013) Algorithm design for performance aware vm consolidation
201. Bichler M, Setzer T, Speitkamp B (2006) Capacity planning for virtualized servers. Information Technology & Systems
202. Lin H, Qi X, Yang S, Midkiff S (2015) Workload-driven vm consolidation in cloud data centers. In: 2015 IEEE International Parallel and Distributed Processing Symposium, pp. 207–216. https://doi.org/10.1109/IPDPS.2015.90
203. Goudarzi H, Ghasemazar M, Pedram M (2012) Sla-based optimization of power and migration cost in cloud computing. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pp. 172–179. https://doi.org/10.1109/CCGrid.2012.112
204. Hoyer M, Schröder K, Schlitt D, Nebel W (2011) Proactive dynamic resource management in virtualized data centers. In: Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking. e-Energy '11, pp. 11–20. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2318716.2318719
205. Genez TAL, Pietri I, Sakellariou R, Bittencourt LF, Madeira ERM (2015) A particle swarm optimization approach for workflow scheduling on cloud resources priced by cpu frequency. In: Proceedings of the 8th International Conference on Utility and Cloud Computing. UCC '15, pp. 237–241. IEEE Press,
206. Sellami K, Tiako PF, Sellami L, Kassa R (2020) Energy efficient workflow scheduling of cloud services using chaotic particle swarm optimization. In: 2020 IEEE Green Technologies Conference(GreenTech), pp. 74–79. https://doi.org/10.1109/GreenTech46478.2020.9289818

207. Peng G, Wolter K (2019) Efficient task scheduling in cloud computing using an improved particle swarm optimization algorithm. In: CLOSER
208. Awad AI, El-Hefnawy NA, Abdel_kader, H.M. (2015) Enhanced particle swarm optimization for task scheduling in cloud computing environments. Procedia Comput Sci 65:920–929. https://doi.org/10.1016/j.procs.2015.09.064
209. Byun E-K, Kee Y-S, Kim J-S, Maeng S (2011) Cost optimized provisioning of elastic resources for application workflows. Futur Gener Comput Syst 27(8):1011–1026. https://doi.org/10.1016/j.future.2011.05.001
210. Abrishami S, Naghibzadeh M, Epema DHJ (2012) Cost-driven scheduling of grid workflows using partial critical paths. IEEE Trans Parallel Distrib Syst 23(8):1400–1414. https://doi.org/10.1109/TPDS.2011.303
211. Abrishami S, Naghibzadeh M (2011) Budget constrained scheduling of grid workflows using partial critical paths. https://api.semanticscholar.org/CorpusID:18484532
212. Abrishami S, Naghibzadeh M (2012) Deadline-constrained workflow scheduling in software as a service cloud. Scientia Iranica 19(3):680–689. https://doi.org/10.1016/j.scient.2011.11.047
213. Taal A, Wang J, de Laat C, Zhao Z (2019) Profiling the scheduling decisions for handling critical paths in deadline-constrained cloud workflows. Futur Gener Comput Syst 100:237–249. https://doi.org/10.1016/j.future.2019.05.002
214. Indhira M, Divya P, AshfaqHaris A, Keerthika P (2018) Efficient resource allocation in cloud environment. Int J Eng Res Technol 6
215. Sharma V, Bala M (2020) An improved task allocation strategy in cloud using modified k-means clustering technique. Egypt Inform J 21(4):201–208. https://doi.org/10.1016/j.eij.2020.02.001
216. Nagarathinam A, Chellasamy A, Antonysamy K, Saravanan V, Gopalakrishnan M (2024) In: El Khoury R (ed.) Green data centers: a review of current trends and practices, pp. 251–264. Springer, Cham. https://doi.org/10.1007/978-3-031-51997-0_21
217. International Energy Agency (2023) Data centres and data transmission networks. Accessed: 2025-02-28. https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks
218. Katal A, Dahiya S, Choudhury T (2023) Energy efficiency in cloud computing data centers: a survey on software technologies. Clust Comput 26:1845–1875. https://doi.org/10.1007/s10586-022-03713-0
219. Bharany S, Sharma S, Khalaf OI, Abdulsahib GM, Al Humaimeedy AS, Aldhyani THH, Maashi M, Alkahtani H (2022) A systematic survey on energy-efficient techniques in sustainable cloud computing. Sustainability 14(10):6256. https://doi.org/10.3390/su14106256
220. Name A (2023) Green cloud computing: a sustainable energy-efficiency approach for business rapidity and the environment. Int J Green Computi 12:67–89. https://doi.org/10.1109/IJGC.2023.1234567
221. Van Geet O, Sickinger D (July 2024) Best practices guide for energy-efficient data center design. Technical report, National Renewable Energy Laboratory (NREL), Golden, CO (United States). https://doi.org/10.2172/2417618. https://www.osti.gov/biblio/2417618
222. Azarifar M, Arik M, Chang J-Y (2024) Liquid cooling of data centers: a necessity facing challenges. Appl Therm Eng 247:123112. https://doi.org/10.1016/j.applthermaleng.2024.123112
223. Zaman SN, Sharme NH, Sumona RN, Islam MJ, Reza AW, Arefin MS (2023) Ai-based air cooling system in data center. In: intelligent computing and optimization. Lecture notes in networks and systems, vol. 852, pp. 53–65. Springer. https://doi.org/10.1007/978-3-031-50330-6_6
224. Lin J, Lin W, Lin W, Liu T, Wang J, Jiang H (2024) Multi-objective cooling control optimization for air-liquid cooled data centers using tcn-bigru-attention-based thermal prediction models. Build Simul 17:2145–2161. https://doi.org/10.1007/s12273-024-1185-7
225. Ganesh N, Rao TS (2025) Advancing sustainability in cloud computing: energy-efficient resource allocation and green infrastructure strategies. SSRN Electron J. https://doi.org/10.2139/ssrn.5138669
226. Swinhoe D (2022) Re-use, refurb, recycle: circular economy thinking and data center it assets. DatacenterDynamics
227. Cooper L (2024) Data center refurbishment: an alternative solution to recycling. Human-I-T
228. Jarnagin S (2024) Circular economy in data center operations: turning waste into opportunity. Data Center Knowledge
229. Circular Center (2025) Microsoft news
230. Engineering A (2024) From waste to resource: the future of data center heat reuse. Ardebili Engineering Blog

231. From byproduct to resource (2024) How data centers are turning waste heat into valuable energy. Datacenters.com
232. Tozzi C (2024) Top 10 data center sustainability stories of 2024. Data Center Knowledge
233. Impact of geographic location on data center energy costs (2025) DataCenters.com
234. U.S. Department of Energy (2024) Best Practices Guide for Energy-Efficient Data Center Design. U.S. Department of Energy. https://www.energy.gov/sites/default/files/2024-07/best-practice-guide-data-center-design.pdf
235. Tech giants are building their own undersea fibre-optic networks (2017) The Economist
236. Top 10: Edge computing companies and solutions (2023) Data Centre Magazine
237. Atzori L, Iera A, Morabito G (2010) The internet of things: a survey. In: computer networks, pp. 54–15. https://doi.org/10.1016/j.comnet.2010.05.010
238. Asghari P, Rahmani AM, Javadi HHS (2019) Internet of things applications: a systematic review. In: computer networks, pp. 148–241261. https://doi.org/10.1016/j.comnet.2018.12.008
239. Mahmud R, Ramamohanarao R, Buyya R (2018) Fog computing: a taxonomy, survey and future directions. In: internet of everything: algorithms, methodologies, technologies and perspectives, pp. 103–13. https://doi.org/10.1007/978-981-10-5861-5_5
240. Yi S, Li C, Li Q (2015) A survey of fog computing: concepts, applications and issues. In: In Proceedings of the Workshop on Mobile Big Data, pp. 37–42. https://doi.org/10.1145/2757384.2757397
241. Palattella MR, Soua R, Khelil A, Engel T (2019) Fog computing as the key for seamless connectivity handover in future vehicular networks. In: In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, pp. 1996–2000. https://doi.org/10.1145/3297280.3297475
242. Liu L, Chang Z, Guo X, Mao S, Ristaniemi T (2018) Multiobjective optimization for computation offloading in fog computing. IEEE Internet Things J 5(1):283–294. https://doi.org/10.1109/JIOT.2017.2780236
243. Kishor A (2021) Task offloading in fog computing for using smart ant colony optimization. In: wireless personal communications, pp. 1572–834. https://doi.org/10.1007/s11277-021-08714-7
244. Kalmar E, Kertesz A (2017) What does i(o)t cost? In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, pp. 19–24. https://doi.org/10.1145/3053600.3053601
245. Bittencourt LF, Diaz-Montes J, Buyya R, Rana OF, Parashar M (2017) Mobility-aware application scheduling in fog computing. In: IEEE Cloud Computing, pp. 4–2635. https://doi.org/10.1109/MCC.2017.27
246. Yadav V, Natesha BV, Guddeti RMR (2019) Ga-pso: service allocation in fog computing environment using hybrid bio-inspired algorithm. In: TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), pp. 1280–1285. https://doi.org/10.1109/TENCON.2019.8929234
247. Markus A, Kertesz A (2020) A survey and taxonomy of simulation environments modelling fog computing. Simul Model Practice Theory 101:102042. https://doi.org/10.1016/j.simpat.2019.102042
248. Mahmud R, Pallewatta S, Goudarzi M, Buyya R (2022) ifogsim2: an extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments. J Syst Softw 190:111351. https://doi.org/10.1016/j.jss.2022.111351
249. Buyya R, Ranjan R, Calheiros RN (2009) Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In: 2009 International Conference on High Performance Computing & Simulation, pp. 1–11. https://doi.org/10.1109/HPCSIM.2009.5192685
250. Markus A, Biro M, Kecskemeti G, Kertesz A (2021) Actuator behaviour modelling in iot-fog-cloud simulation. PeerJ Comput Sci 7:e651. https://doi.org/10.7717/peerj-cs.651
251. Brogi A, Forti S, Ibrahim A (2017) How to best deploy your fog applications, probably. In: 1st International Conference on Fog and Edge Computing, pp. 105–114. https://doi.org/10.1109/ICFEC.2017.8
252. Lera I, Guerrero C, Juiz C (2019) Yafs: a simulator for iot scenarios in fog computing. IEEE Access 7:91745. https://doi.org/10.1109/ACCESS.2019.2927895
253. Qayyum T, Malik AW, Khattak MAK, Khalid O, Khan SU (2018) Fognetsim++: a toolkit for modeling and simulation of distributed fog environment. IEEE Access 6:63570. https://doi.org/10.1109/ACCESS.2018.2877696

254. Puliafito C, Gonçalves DM, Lopes MM, Martins LL, Madeira E, Mingozzi E, Rana O, Bittencourt LF (2020) Mobfogsim: simulation of mobility and migration for fog computing. Simul Model Practice Theory 101:102062. https://doi.org/10.1016/j.simpat.2019.102062
255. Sonmez C, Ozgovde A, Ersoy C (2018) Edgecloudsim: an environment for performance evaluation of edge computing systems. Trans Emerg Telecommun Technol 29:e3493. https://doi.org/10.1002/ett.3493
256. Jha DN, Alwasel K, Alshoshan A, Huang X, Naha R, Battula S, Garg S, Puthal D, James P, Zomaya A, Dustdar S, Ranjan R (2020) Iotsim-edge: a simulation framework for modeling the behavior of internet of things and edge computing environments. Softw: Practice Exp 50(6):844–867. https://doi.org/10.1002/spe.2787
257. Varghese B, Wang N, Barbhuiya S, Kilpatrick P, Nikolopoulos DS (2016) Challenges and opportunities in edge computing. In: 2016 IEEE International Conference on Smart Cloud (SmartCloud), pp. 20–26. https://doi.org/10.1109/SmartCloud.2016.18
258. Dmitrieva, E., Thakur, G., Prabhakar, P.K., Prakash, A., Vyas, A., Prasanna, Y.L.: Edge computing and ai: Advancements in industry 5.0- an experimental assessment. BIO Web of Conferences 86, 01096 (2024) https://doi.org/10.1051/bioconf/20248601096
259. Deng S, Zhao H, Fang W, Yin J, Dustdar S, Zomaya AY (2020) Edge intelligence: the confluence of edge computing and artificial intelligence. IEEE Internet Things J 7(8):7457–7469. https://doi.org/10.1109/JIOT.2020.2984887
260. Xu D, Li T, Li Y, Su X, Tarkoma S, Jiang T, Crowcroft J, Hui P (2021) Edge intelligence: empowering intelligence to the edge of network. Proc IEEE 109(11):1778–1837. https://doi.org/10.1109/JPROC.2021.3119950
261. Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. Decentralized Business Review, 21260
262. Liu Y, Wang Y, Jin Y (2012) Research on the improvement of mongodb auto-sharding in cloud environment. In: 2012 7th International Conference on Computer Science & Education (ICCSE), pp. 851–854. IEEE
263. Wang G, Shi ZJ, Nixon M, Han S (2019) Sok: Sharding on blockchain. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, pp. 41–61
264. Yu G, Wang X, Yu K, Ni W, Zhang JA, Liu RP (2020) Survey: sharding in blockchains. IEEE Access 8:14155–14181
265. Henzinger A, Noe A, Schulz C (2020) Ilp-based local search for graph partitioning. J Exp Algorithmics (JEA) 25:1–26
266. Cordero M, Miniguano-Trujillo A, Recalde D, Torres R, Vaca P (2022) Graph partitioning in connected components with minimum size constraints via mixed integer programming https://doi.org/10.48550/ARXIV.2202.11254
267. Baniata H, Anaqreh A, Kertesz A (2023) Distributed scalability tuning for evolutionary sharding optimization with random-equivalent security in permissionless blockchain. Internet Things 24:100955
268. Manumachu RR, Lastovetsky A (2018) Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy. IEEE Trans Comput 67(2):160–177. https://doi.org/10.1109/TC.2017.2742513
269. Khaleghzadeh H, Fahad M, Shahid A, Manumachu RR, Lastovetsky A (2021) Bi-objective optimization of data-parallel applications on heterogeneous hpc platforms for performance and energy through workload distribution. IEEE Trans Parallel Distrib Syst 32(3):543–560. https://doi.org/10.1109/TPDS.2020.3027338
270. Krzywaniak A, Proficz J, Czarnul P (2018) Analyzing energy/performance trade-offs with power capping for parallel applications on modern multi and many core processors. In: 2018 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 339–346
271. Krzywaniak A, Czarnul P (2020) Performance/energy aware optimization of parallel applications on gpus under power capping. In: Processing Parallel, Mathematics Applied (eds) Wyrzykowski R, Deelman E, Dongarra J, Karczewski K. Springer, Cham, pp 123–133
272. Krzywaniak A, Czarnul P, Proficz J (2022) Depo: a dynamic energy-performance optimizer tool for automatic power capping for energy efficient high-performance computing. Softw: Practice Exp 52(12):2598–2634. https://doi.org/10.1002/spe.3139
273. Czarnul P, Proficz J, Krzywaniak A (2019) Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments. Sci Program 2019:8348791–1834879119

274. Cunha Sa, Silva L, Saraiva Ja, Fernandes JaP (2024) Trading runtime for energy efficiency: leveraging power caps to save energy across programming languages. In: Proceedings of the 17th ACM SIGPLAN International Conference on Software Language Engineering. SLE '24, pp. 130–142. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3687997.3695638
275. Wang Y, Hao M, He H, Zhang W, Tang Q, Sun X, Wang Z (2024) Drlcap: runtime gpu frequency capping with deep reinforcement learning. IEEE Trans Sustain Comput 9(5):712–726. https://doi.org/10.1109/TSUSC.2024.3362697
276. Simmendinger C, Marquardt M, Mäder J, Schneider R (2024) Powersched - managing power consumption in overprovisioned systems. In: 2024 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops), pp. 1–8. https://doi.org/10.1109/CLUSTERWorkshops61563.2024.00012
277. Blelloch GE, Gibbons PB, Gu Y, McGuffey C, Shun J (2018) The parallel persistent memory model. In: Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures. SPAA '18, pp. 247–258. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3210377.3210381
278. Malinowski A, Czarnul P (2019) Multi-agent large-scale parallel crowd simulation with nvram-based distributed cache. J Comput Sci 33:83–94
279. Malinowski A, Czarnul P (2018) A solution to image processing with parallel MPI I/O and distributed NVRAM cache. Scalable Comput Pract Exp 19(1):1–14
280. Ruan X, Yang Q, Mohammed IA, Yin S, Ding Z, Xie J, Lewis J, Qin X (2010) Es-mpich2: A message passing interface with enhanced security. In: International Performance Computing and Communications Conference, pp. 161–168. https://doi.org/10.1109/PCCC.2010.5682312
281. Maffina MA, RamPriya RS (2013) An improved and efficient message passing interface for secure communication on distributed clusters. In: 2013 International Conference on Recent Trends in Information Technology (ICRTIT), pp. 329–334. https://doi.org/10.1109/ICRTIT.2013.6844225
282. Losada N, González P, Martín MJ, Bosilca G, Bouteiller A, Teranishi K (2020) Fault tolerance of mpi applications in exascale systems: the ulfm solution. Futur Gener Comput Syst 106:467–481
283. Lion R, Thibault S, (2020) From tasks graphs to asynchronous distributed checkpointing with local restart. In: 2020 IEEE/ACM 10th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS). Atlanta, USA https://doi.org/10.1109/FTXS51974.2020.00009. https://hal.archives-ouvertes.fr/hal-02970529
284. Tahan O, Shawky M (2012) Using dynamic task level redundancy for openmp fault tolerance. In: Proceedings of the 25th International Conference on Architecture of Computing Systems. ARCS'12, pp. 25–36. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-28293-5_3
285. Zhang D, Dai D, He Y, Bao FS, Xie B (2020) Rlscheduler: an automated hpc batch job scheduler using reinforcement learning. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '20. IEEE Press,
286. Anderson DP (2020) Boinc: a platform for volunteer computing. J Grid Comput 18:99–122
287. Czarnul P, Kuchta J, Matuszek MR (2013) Parallel computations in the volunteer-based comcute system. In: Wyrzykowski R, Dongarra JJ, Karczewski K, Wasniewski J (eds.) Parallel Processing and Applied Mathematics - 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part I. Lecture Notes in Computer Science, vol. 8384, pp. 261–271. Springer. https://doi.org/10.1007/978-3-642-55224-3_25
288. Chorazyk P, Godzik M, Pietak K, Turek W, Kisiel-Dorohinicki M, Byrski A (2017) Lightweight volunteer computing platform using web workers. Procedia Comput Sci 108:948–957
289. Cushing R, Putra GHH, Koulouzis S, Belloum A, Bubak M, Laat C (2013) Distributed computing on an ensemble of browsers. IEEE Internet Comput 17(5):54–61. https://doi.org/10.1109/MIC.2013.3
290. MacWilliam T, Cecka C (2013) Crowdcl: Web-based volunteer computing with webcl. In: 2013 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–6. https://doi.org/10.1109/HPEC.2013.6670348
291. Lavoie E, Hendren L, Desprez F (2017) Pando: A volunteer computing platform for the web. In: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 387–388. https://doi.org/10.1109/FAS-W.2017.184
292. Fabisiak T, Danilecki A (2017) Browser-based harnessing of voluntary computational power. Foundations Comput Decision Sci 42(1):3–42. https://doi.org/10.1515/fcds-2017-0001

293. Czarnul P, Matuszek M (2016) Considerations of computational efficiency in volunteer and cluster computing. In: Wyrzykowski R, Deelman E, Dongarra J, Karczewski K, Kitowski J, Wiatr K (eds) Processing Parallel, Mathematics Applied Springer, Cham, pp 66–74
294. Roberts SI, Wright SA, Fahmy SA, Jarvis SA (2017) Metrics for energy-aware software optimisation. In: Kunkel JM, Yokota R, Balaji P, Keyes D (eds) High performance computing. Springer, Cham, pp 413–430
295. Bouras M, Idrissi A (2023) In: Idrissi A (ed.) A Survey of Parallel Computing: Challenges, Methods and Directions, pp. 67–81. Springer, Cham. https://doi.org/10.1007/978-3-031-33309-5_6
296. Rekhate V, Tale A, Sambhus N, Joshi A (2016) Secure and efficient message passing in distributed systems using one-time pad. In: 2016 International Conference on Computing, Analytics and Security Trends (CAST), pp. 393–397. https://doi.org/10.1109/CAST.2016.7915001
297. Wang F, Hao M, Zhang W, Wang Z (2024) Model-free gpu online energy optimization. IEEE Trans Sustain Comput 9(2):141–154. https://doi.org/10.1109/TSUSC.2023.3314916
298. Espenshade C, Peng R, Hong E, Calman M, Zhu Y, Parida P, Lee EK, Kim MA (2024) Characterizing training performance and energy for foundation models and image classifiers on multi-instance gpus. In: Proceedings of the 4th Workshop on Machine Learning and Systems. EuroMLSys '24, pp. 47–55. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3642970.3655830
299. Koszczał G, Dobrosolski J, Matuszek M, Czarnul P (2024) Performance and energy aware training of a deep neural network in a multi-gpu environment with power capping. In: Zeinalipour D, Blanco Heras D, Pallis G, Herodotou H, Trihinas D, Balouek D, Diehl P, Cojean T, Fürlinger K, Kirkeby MH, Nardelli M, Di Sanzo P (eds) Euro-Par 2023: parallel processing workshops. Springer, Cham, pp 5–16
300. Pattaranantakul M, He R, Song Q, Zhang Z, Meddahi A (2018) Nfv security survey: from use case driven threat analysis to state-of-the-art countermeasures. IEEE Commun Surv & Tutorials 20(4):3330–3368. https://doi.org/10.1109/COMST.2018.2859449
301. Patel YS, Mehrotra N, Soner S (2015) Green cloud computing: A review on green it areas for cloud computing environment. In: 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), pp. 327–332. https://doi.org/10.1109/ABLAZE.2015.7155006
302. Tudosoiu M-F, Pop F (2021) Bin packing scheduling algorithm with energy constraints in cloud computing. In: 2021 IEEE 17th International Conference on Intelligent Computer Communication and Processing (ICCP), pp. 77–84. https://doi.org/10.1109/ICCP53602.2021.9733463
303. Tebaa M, Hajji SE, Ghazi AE (2012) Homomorphic encryption method applied to cloud computing. In: 2012 National Days of Network Security and Systems, pp. 86–89. https://doi.org/10.1109/JNS2.2012.6249248
304. Popa RA, Redfield CMS, Zeldovich N, Balakrishnan H (2011) Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. SOSP '11, pp. 85–100. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2043556.2043566
305. Milani BA, Navimipour NJ (2017) A systematic literature review of the data replication techniques in the cloud environments. Big Data Res 10:1–7. https://doi.org/10.1016/j.bdr.2017.06.003
306. Khan S, Parkinson S, Qin Y (2017) Fog computing security: a review of current applications and security solutions. J Cloud Comput. https://doi.org/10.1186/s13677-017-0090-3
307. Han R, Yu J, Zhang R (2020) Analysing and improving shard allocation protocols for sharded blockchains. Cryptology ePrint Archive
308. Khan D, Jung LT, Hashmani MA (2021) Systematic literature review of challenges in blockchain scalability. Appl Sci 11(20):9372
309. Luu L, Narayanan V, Zheng C, Baweja K, Gilbert S, Saxena P (2016) A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 17–30
310. Huang C, Wang Z, Chen H, Hu Q, Zhang Q, Wang W, Guan X (2020) Repchain: a reputation-based secure, fast, and high incentive blockchain system via sharding. IEEE Internet Things J 8(6):4291–4304
311. Baniata H, Kertesz A (2023) Approaches to overpower proof-of-work blockchains despite minority. IEEE Access 11:2952–2967
312. Barhanpure A, Belandor P, Das B (2018) Proof of stack consensus for blockchain networks. In: International Symposium on Security in Computing and Communication, pp. 104–116. Springer

313. A Avasthi A, Saxena A (2018) Two hop blockchain model: resonating between proof of work (pow) and proof of authority (poa). Int J Inform Syst & Manag Sci 1(1)
314. Pandya SB, Sanghvi HA, Patel RH, Pandya AS (2022) Gpu and fpga based deployment of blockchain for cryptocurrency–a systematic review. In: 2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES), pp. 18–25. IEEE
315. Ferraz O, Subramaniyan S, Chinthalaa R, Andrade J, Cavallaro JR, Nandy SK, Silva V, Zhang X, Purnaprajna M, Falcao G (2021) A survey on high-throughput non-binary ldpc decoders: Asic, fpga, and gpu architectures. IEEE Commun Surv & Tutorials 24(1):524–556
316. Hübner L, Kozlov AM, Hespe D, Sanders P, Stamatakis A (2021) Exploring parallel MPI fault tolerance mechanisms for phylogenetic inference with RAxML-NG. Bioinformatics 37(22):4056–4063
317. Czarnul P, Kuchta J, Matuszek M, Proficz J, Rościszewski P, Wójcik M, Szymański J (2017) Merpsys: an environment for simulation of parallel application execution on large scale hpc systems. Simul Model Pract Theory 77:124–140
318. Kwok Y-K, Ahmad I (1997) Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. J Parallel Distributed Comput 47(1):58–77
319. Fan Y, Lan Z, Childers T, Rich P, Allcock W, Papka ME (2021) Deep reinforcement agent for scheduling in hpc. In: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 807–816. https://doi.org/10.1109/IPDPS49936.2021.00090
320. Lin X, Wang Y, Pedram M (2016) A reinforcement learning-based power management framework for green computing data centers. In: 2016 IEEE International Conference on Cloud Engineering (IC2E), pp. 135–138. https://doi.org/10.1109/IC2E.2016.33
321. Özdemir MB (2019) Evaluation of multi-objective closed loop supply chain network integrated with blockchain for used lubrication oils obtained from vehicles in military forces by the linear programming. PhD thesis, Ankara Yıldırım Beyazıt Üniversitesi Fen Bilimleri Enstitüsü
322. Li F, Huang J, Lippman A (2008) A linear integer programming approach to analyze p2p media streaming. In: 2008 42nd Annual Conference on Information Sciences and Systems, pp. 1125–1130. IEEE
323. Sasabe M (2021) Analysis of minimum distribution time of tit-for-tat-based p2p file distribution: linear programming based approach. Peer-to-Peer Netw Appl 14(4):2127–2138
324. Li D, Dai J, Jiang R, Wang X, Xu Y (2020) Gapg: a heuristic greedy algorithm for grouping storage scheme in blockchain. In: 2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops), pp. 91–95. IEEE
325. Chang S-Y Greedy networking in cryptocurrency blockchain. In: ICT Systems Security and Privacy Protection: 37th IFIP TC 11 International Conference, SEC 2022, Copenhagen, Denmark, June 13–15, 2022, Proceedings, p. 343. Springer
326. Kalysh I, Alimkhan A, Temirtayev I, Nunna HK, Doolla S, Vipin K (2019) Dynamic programming based peer-to-peer energy trading framework for smart microgrids. In: 2019 IEEE 13th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG), pp. 1–6. IEEE
327. Yuan H, Zhou M (2021) Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems. IEEE Trans Autom Sci Eng 18(3):1277–1287. https://doi.org/10.1109/TASE.2020.3000946
328. Kimovski D, Ijaz H, Saurabh N, Prodan R (2018) Adaptive nature-inspired fog architecture. In: 2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), pp. 1–8. IEEE
329. Bizzaro F, Conti M, Pini MS (2020) Proof of evolution: leveraging blockchain mining for a cooperative execution of genetic algorithms. In: 2020 IEEE International Conference on Blockchain (Blockchain), pp. 450–455. IEEE
330. Mureddu M, Ghiani E, Pilo F (2020) Smart grid optimization with blockchain based decentralized genetic algorithm. In: 2020 IEEE Power & Energy Society General Meeting (PESGM), pp. 1–5. IEEE
331. Chao S (2022) Construction model of e-commerce agricultural product online marketing system based on blockchain and improved genetic algorithm. Secur Commun Netw 2022:4055698
332. Baniata H, Anaqreh A, Kertesz A (2021) Pf-bts: a privacy-aware fog-enhanced blockchain-assisted task scheduling. Inf Process & Manag 58(1):102393
333. Omara FA, Arafa MM (2010) Genetic algorithms for task scheduling problem. J Parallel Distributed Comput 70(1):13–22

334. Balicki J, Korłub W, Szymanski J, Zakidalski M (2014) Big data paradigm developed in volunteer grid system with genetic programming scheduler. In: Intelligence Artificial, Computing Soft (eds) Rutkowski L, Korytkowski M, Scherer R, Tadeusiewicz R, Zadeh LA, Zurada JM. Springer, Cham, pp 771–782
335. Qu B, Lei Y, Zhao Y (2010) A new genetic algorithm based scheduling for volunteer computing. In: 2010 International Conference on Computer and Communication Technologies in Agriculture Engineering, vol. 3, pp. 228–231. https://doi.org/10.1109/CCTAE.2010.5544240
336. Estrada T, Fuentes O, Taufer M (2008) A distributed evolutionary method to design scheduling policies for volunteer computing. SIGMETRICS Perform Eval Rev 36(3):40–49
337. Abedin SF, Alam MGR, Kazmi SA, Tran NH, Niyato D, Hong CS (2018) Resource allocation for ultra-reliable and enhanced mobile broadband iot applications in fog network. IEEE Trans Commun 67(1):489–502
338. Wu Y, Tang S, Zhao B, Peng Z (2019) Bptm: blockchain-based privacy-preserving task matching in crowdsourcing. IEEE access 7:45605–45617
339. Liu Z, Luong NC, Wang W, Niyato D, Wang P, Liang Y-C, Kim DI (2019) A survey on blockchain: a game theoretical perspective. IEEE Access 7:47615–47643. https://doi.org/10.1109/ACCESS.2019.2909924
340. Fan Y, Shen G, Jin Z, Hu D, Shi L, Yuan X (2020) Stackelberg game based edge computing resource management for mobile blockchain. In: Proceedings of the ACM Turing Celebration Conference - China. ACM TURC '20, pp. 225–229. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3393527.3393565
341. Wei T, Ren S, Zhu Q (2021) Deep reinforcement learning for joint datacenter and hvac load control in distributed mixed-use buildings. IEEE Trans Sustain Comput 6(03):370–384. https://doi.org/10.1109/TSUSC.2019.2910533
342. Shaw R, Howley E, Barrett E (2022) Applying reinforcement learning towards automating energy efficient virtual machine consolidation in cloud data centers. Inf Syst 107:101722
343. Chen L, Lingys J, Chen K, Liao X (2021) Datacenter Traffic Optimization with Deep Reinforcement Learning, pp. 223–259. John Wiley & Sons, Ltd. Chap. 10. https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119675525.ch10
344. Tessler C, Shpigelman Y, Dalal G, Mandelbaum A, Haritan Kazakov D, Fuhrer B, Chechik G, Mannor S (2022) Reinforcement learning for datacenter congestion control. SIGMETRICS Perform Eval Rev 49(2):43–46. https://doi.org/10.1145/3512798.3512815
345. Baek J-y, Kaddoum G, Garg S, Kaur K, Gravel V (2019) Managing fog networks using reinforcement learning based load balancing algorithm. In: 2019 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–7. https://doi.org/10.1109/WCNC.2019.8885745
346. Poltronieri F, Tortonesi M, Stefanelli C, Suri N (2021) Reinforcement learning for value-based placement of fog services. In: 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 466–472
347. Dai Y, Xu D, Maharjan S, Chen Z, He Q, Zhang Y (2019) Blockchain and deep reinforcement learning empowered intelligent 5g beyond. IEEE Network 33(3):10–17
348. Xiao L, Ding Y, Jiang D, Huang J, Wang D, Li J, Poor HV (2020) A reinforcement learning and blockchain-based trust mechanism for edge networks. IEEE Trans Commun 68(9):5460–5470
349. Ning Z, Sun S, Wang X, Guo L, Wang G, Gao X, Kwok RY (2021) Intelligent resource allocation in mobile blockchain for privacy and security transactions: a deep reinforcement learning based approach. Sci China Inf Sci 64(6):1–16

Authors and Affiliations

Paweł Czarnul[1] · Marcel Antal[2] · Hamza Baniata[3] · Dalvan Griebler[4] · Attila Kertesz[3] · Christoph W. Kessler[5] · Andreas Kouloumpris[6] · Salko Kovačić[7] · Andras Markus[3] · Maria K. Michael[6] · Panagiota Nikolaou[8] · Isil Öz[9] · Radu Prodan[10] · Gordana Rakić[11]

✉ Paweł Czarnul
pczarnul@eti.pg.edu.pl

Marcel Antal
marcel.antal@cs.utcluj.ro

Hamza Baniata
baniatah@inf.u-szeged.hu

Dalvan Griebler
dalvan.griebler@pucrs.br

Attila Kertesz
keratt@inf.u-szeged.hu

Christoph W. Kessler
christoph.kessler@liu.se

Andreas Kouloumpris
kouloumpris.andreas@ucy.ac.cy

Salko Kovačić
salko@unmo.ba

Andras Markus
markusa@inf.u-szeged.hu

Maria K. Michael
mmichael@ucy.ac.cy

Panagiota Nikolaou
pnikolaou1@uclan.ac.uk

Isil Öz
isiloz@iyte.edu.tr

Radu Prodan
radu.prodan@aau.at

Gordana Rakić
gordana.rakic@dmi.uns.ac.rs

[1] Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, 11/12 Narutowicza, 80-233 Gdańsk, Poland

[2] Computer Science Department, Technical University of Cluj-Napoca, Cluj-Napoca, Romania

[3] Department of Software Engineering, University of Szeged, Szeged, Hungary

[4] School of Technology, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil

[5] Department of Computer and Information Science, Linköping University, Linköping, Sweden

[6] Department of Electrical and Computer Engineering, KIOS Research and Innovation Center of Excellence, University of Cyprus, Nicosia, Cyprus

[7] Džemal Bijedić University of Mostar, Mostar, Bosnia and Herzegovina

[8] University of Central Lancashire Cyprus, Larnaka, Cyprus

[9] Computer Engineering Department, Izmir Institute of Technology, Izmir, Turkey

[10] Institute of Information Technology, Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria

[11] Faculty of Sciences, University of Novi Sad, Novi Sad, Serbia