Minimum Dependencies Energy-Efficient Scheduling in Data Centers

Mateusz Żotkiewicz, Mateusz Guzek, *Member, IEEE*, Dzmitry Kliazovich, *Senior Member, IEEE*, and Pascal Bouvry, *Member, IEEE*

Abstract—This work presents an on-line, energy- and communication-aware scheduling strategy for SaaS applications in data centers. The applications are composed of various services and represented as workflows. Each workflow consists of tasks related to each other by precedence constraints and represented by Directed Acyclic Graphs (DAGs). The proposed scheduling strategy combines advantages of state-of-the-art workflow scheduling strategies with energy-aware independent task scheduling approaches. The process of scheduling consists of two phases. In the first phase, virtual deadlines of individual tasks are set in the central scheduler. These deadlines are determined using a novel strategy that favors tasks which are less dependent on other tasks. During the second phase, tasks are dynamically assigned to computing servers based on the current load of network links and servers in a data center. The proposed approach, called Minimum Dependencies Energy-efficient DAG (MinD+ED) scheduling, has been implemented in the GreenCloud simulator. It outperforms other approaches in terms of energy efficiency, while keeping a satisfiable level of tardiness.

Index Terms—Workflow scheduling, DAG scheduling, energy-efficient, dynamic scheduling

1 Introduction

Parallel and distributed systems became standard in industrial data processing. They provide massive-scale systems on demand, using abstraction mechanisms, such as virtualization, and design methodologies such as service-oriented architectures. On-demand service provisioning requires elasticity, a value of responsiveness of the system facing variable patterns of users' requests. Data centers process user requests and provide storage. In 2013, worldwide power consumption of data centers accounted for 38.9 GW, and recently grew 7 percent [6]. This value is going to grow together with the size of the cloud computing market, estimated to be $174.2 billion in 2014, with predicted steady yearly growth rate around 20 percent until 2017 [18]. Due to those trends, it is necessary to find techniques to reduce IT systems energy consumption [3]. Recent surveys on energy-saving techniques can be found in [26], [33].

One of the most widely used models to represent distributed applications is based on DAGs. DAGs are simple and expressive at the same time. However, DAG scheduling problem is NP-hard [39], meaning that there are no exact, deterministic, and fast algorithms solving the problem. Accounting for additional requirements, such as energy consumption, makes the problem even harder. Exact algorithms can solve only small instances, while most of existing solutions for large-scale systems have shortcomings. Main difficulty is to handle a surge in the number of services and tasks of a workflow, as the amount of communication between servers increases. The performance of existing techniques remains limited in balancing energy efficiency and communication-awareness in the large-scale environment [1], [9], [11], [13], [43].

To address the aforementioned shortcomings, we present a novel methodology named Minimum Dependencies Energy-efficient DAG scheduling (MinD+ED), which operates in data centers and dynamically schedules batch workflows consisting of interdependent tasks targeting energy-efficiency while taking into account both computing and communication requirements. MinD +ED outperforms state-of-the-art approaches that deal with similar problems and is of a similar computational complexity. It can be easily used on already deployed systems and be complementary to other energy-savings techniques.

The main novelty of the presented research is an efficient workflow scheduling strategy inspired by state-of-the-art approaches based on decomposition and virtual deadlines, and energy efficient independent task scheduling algorithms. The proposed strategy has been evaluated with simulations focusing on communication processes between tasks and task interdependencies.

The rest of the paper is organized as follows. Section 2 addresses the problem of workload scheduling. It is followed by the literature review in Section 3, which summarizes background of our research and refers the reader to related works. Section 4 presents the proposed scheduling methodology. In Section 5, experimental results are presented. The paper ends with conclusions in Section 6.

- *M. Żotkiewicz is with the Institute of Telecommunications, Warsaw University of Technology, Nowowiejska 15/19, Warszawa 00-665, Poland, and University of Luxembourg, Luxembourg. E-mail: mzotkiew@tele.pw.edu.pl.*
- *M. Guzek, D. Kliazovich, and P. Bouvry are with the University of Luxembourg, Luxembourg. E-mail: {mateusz.guzek, pascal.bouvry}@uni.lu, Dzmitry.Kliazovich@gmail.com.*

Manuscript received 1 July 2015; revised 24 Feb. 2016; accepted 1 Mar. 2016. Date of publication 16 Mar. 2016; date of current version 16 Nov. 2016. Recommended for acceptance by C. Morin. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2016.2542817

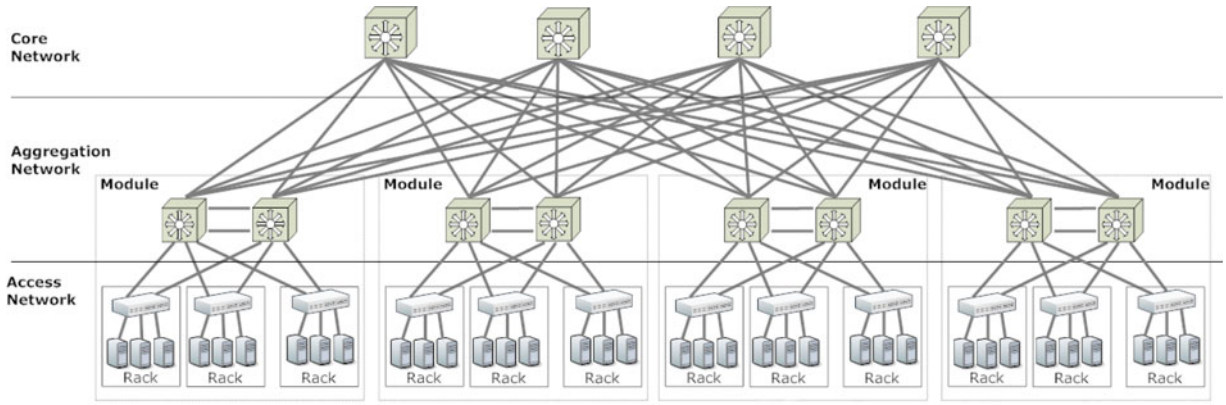

Fig. 1. Fat-tree data center topology.

2 Scheduling Problem

The description of the considered scheduling problem is divided into two parts. First, we present the problem in general. Then, we describe all assumptions making the problem specific. In most cases, the assumptions are not critical—the methods can be easily adjusted if the assumptions are not met.

2.1 Problem Statement

We target an on-line scheduling problem of workflows consisting of interrelated tasks in a data center. The future workflows are unknown; however, at the moment of arrival, we learn the structure of the arriving workflow, so this subproblem becomes static. Still, an execution of a workflow may be interrupted by other arriving workflow.

The data center is constantly fed with workflows consisting of smaller indivisible elements called tasks; thus, they can be decomposed and the tasks can run in parallel. Directed Acyclic Graphs (DAGs) are used to model this phenomenon. Each workflow is represented by a single DAG. Nodes of a DAG represent indivisible tasks, while arcs represent precedence relations between tasks, i.e., if there is an arc between node A and B, the task represented by node B cannot be executed before the task represented by node A is successfully completed.

Efficient scheduling of the aforementioned tasks is a challenge. Minimize makespan is the most common objective. As workflows dynamically arrive at the gateway, instead of considering the global makespan of all workflows, we focus on the average makespan of all submitted workflows. The second considered objective is to minimize energy consumption. Unfortunately, minimizing consumed energy usually requires extending execution time of some or all the workflows. Therefore, to keep the research practical, constraints on durations of workflows (deadlines) are introduced.

Following [10], we can present our problem in the three-field standard notation as $R|prec, r_i, d_i|\sum Energy_i, \sum T_i$, where R stands for unrelated machines, i.e., there are no relations between execution time of a task by different machines, and $prec$ means that there are precedence constraints between tasks. Each task belongs to a workflow, and there are multiple workflows in the system. Each workflow has its release time r_i (moment of arrival) and deadline d_i. Objectives are calculated for workflows rather than tasks. Our objective is to simultaneously minimize total energy consumption ($\sum Energy_i$) and tardiness ($\sum T_i$). The important aspect of the considered system not described in the 3-field notation is the communication, which introduces uncertainties due to the network resource sharing and topology.

We want to stress here that the presented approach works for unrelated machines meaning that an execution time for each machine-task pair is independent. Still, to facilitate explanations, especially in Section 4.1, we temporary assume that the execution time affinely depends on the size of the task and the computational power of the machine, which corresponds to Q in the 3-field standard notation.

2.2 Assumptions

In this subsection, we discuss all specific assumptions concerning the considered problem. Most of the assumptions are not strict in the sense that the methods presented in this paper can be easily adjusted to work with opposite assumptions.

The first non-strict assumption is that data centers consist of a number of heterogeneous servers. Heterogeneity of data centers is important nowadays, as the majority of modern data centers tend to heterogeneity. Even if they were originally homogeneous systems, a continuous process of improvements gradually transforms them into heterogeneous data centers.

We assume that a data center contains a gateway, and servers are connected to the gateway using Ethernet links and switches forming a fat-tree topology (see Fig. 1). Three-tier fat-trees of servers and switches are currently the most widely used data center architecture [35]. The topology consists of the core tier at the root of the tree, which is connected to a gateway, the aggregation tier, which is responsible for routing, and the access tier, which interconnects the pool of computing servers. It is assumed that the topology and characteristics of all links, switches, and servers are known. The workflows appear in the gateway, are sent to servers of the access tier for processing, while the results are sent back to the gateway. The fat-tree assumption is not critical and the proposed methods can be successfully used in other setting—the scheduling methods of Section 4.2 have to be accordingly adjusted.

We assume that each server is characterized by its processing speed, while each task is characterized by a number of instructions that have to be executed to finish the task. This assumption is crucial for computing some of the facilitating constants (see Sections 4.1.1 and 4.1.1). However, it is irrelevant from the viewpoint of the main method.

Each arc is characterized by the amount of bytes that have to be communicated between tasks. In case when the involved tasks are executed at the same server, inter-task communication is performed locally via memory; thus, the tasks can be executed one after another without any delay. However, if the task represented by a sink node of the edge is to be executed on a different server than the task represented by a source node of the edge, a network transmission has to take place between those servers. It is assumed that the transmission is performed using the TCP/IP protocol and communication time cannot be predicted accurately. Each workflow is additionally characterized by the amount of bytes that have to be sent from the gateway to a server before executing the first task of the workflow (referred to as the source task), and the amount of bytes that have to be sent from a server executing the last task of the workflow (referred to as the sink task) to the gateway after the workflow is successfully completed. It is assumed that each workflow has a single source task and a single sink task. In other words, each DAG representing a workflow contains only one node that has no incoming arcs (source node) and only one node that has no outgoing arcs (sink node). Finally, each workflow is characterized by its deadline, i.e., a time limit for sending the last byte of the output data from the server executing the sink task to the gateway.

We assume that server processing speeds, task sizes, and the amounts of inter-task communication are known. The assumption is critical from the viewpoint of our methods. However, the numerical experiments presented in Section 5.2 indicate that the proposed methods are also efficient when provided with only rough estimates of the mentioned values. In practical systems, administrators can provide information about processing speeds, while task sizes and the amount of inter-task communication should be either specified by the users explicitly or be already known to the system through advance profiling.

In addition, it is assumed that several tasks can be assigned to a server at a time. The server executes them simultaneously or one after another depending on task priorities. Neither running, nor assigned tasks can be migrated between servers. Such specifics can be motivated for instance by low tasks granularity, when migration may cause too much overhead in comparison with possible gains. Additionally, migration overheads can reduce energy-efficiency. However, the execution of one task can be suspended, when another task of higher priority arrives at the server. Still, the proposed algorithm would work in a data center that allows task migration, as it can be treated as an independent process which does not influence the scheduling. More precisely, if tasks are scheduled on VMs, the migration process can be transparent to scheduling decisions.

Finally, for energy consumption, our assumptions are that Dynamic Voltage and Frequency Scaling (DVFS) [34], which adjusts server power consumption according to the applied computing load, is not used. Instead, Dynamic Power Management (DPM) [4] is utilized, which achieves most of energy savings by powering down unused devices during runtime. Therefore, it is assumed that a server consumes constant base energy while being awaken, increased by energy resulting from computational effort, which is proportional to a number of executed instructions. Such assumption is consistent with full-system power models [36], including models of virtualized systems [16], which report linear relationship between server load and its power. The assumption implies that DVFS does not improve results, as the fastest and most energy efficient strategy is to take advantage of DPM and consolidate workloads on a minimum number of computing servers. For switches, we assume that all of them consume constant amount of energy and they cannot be switched off, to support network connectivity. Summarizing, a server is awaken whenever any of its cores processes a task or a server is involved in an inter-task communication. Servers left idle are put into sleep. Both putting into sleep and waking a server up cause overheads. However, the impact of overheads in the considered setup is small and negligible. Therefore, for the sake of clearness and simplicity DPM overheads are not taken into account by the presented methods in current work and left for future study.

3 Background and Related Works

The general scheduling problem is relatively old. Over the last half-century, a significant amount of articles dealing with the problem have been published. Still, a version of the problem addressed in this work is new, as it is derived from a recent trend, which is distributed computing. The abundance of literature covering the scheduling problem in distributed systems has convinced us to focus mainly on papers that are related to the three distinctive aspects of our research, i.e., inter-task communication aspects, energy efficiency, and dynamic workflow scheduling.

3.1 Inter-Task Communication

Majority of works on task scheduling ignore the inter-task communication, and when noticed, the required communication is considered as another resource consumed by a workflow and is modeled like computing power or memory. Such an approach can be seen in [39], [40], in which communication between servers, resulting from precedence constraints between tasks, completely blocks the possibility of another communication on utilized links. Such approach does not reflect the specifics of the TCP/IP protocol used for communication between servers clearly presented in [23], [24]. In our research, we go a step further and address the problem of communication between tasks using the GreenCloud simulator [22], [32]. The approach guards us against pitfalls modeling such a complex issue as the TCP/IP communication can cause.

It is important to restate the difference between various meanings of the word communication in the context of resource allocation in data centers. This research considers the inter-task communication, and not the communication understood as one of many resources needed to complete a task. Examples of the second understanding of communication in the context under consideration can be found for instance in [25], in which the aspect is noticed and modeled using a simple non-linear program. The resulting problem is solved by a heuristic method called BATS.

3.2 Energy Efficiency and Workflow Scheduling

The problem of energy efficiency in data centers is more widely tackled than the inter-task communication problem.

Therefore, in this case, the literature review is limited solely to documents that do not significantly deviate from the assumptions adopted in our research. First, we assume that a task cannot be transferred between servers during the execution, and frequency used by central processing units (CPUs) cannot be dynamically adjusted (still, unused servers can be temporarily put into sleep to save energy). The former assumption prevents us from using the energy saving methods based on migration of virtual machines [17], [45]. The latter excludes all methods based on delaying tasks that are not on a critical path and saving energy utilizing DVFS technology [20], [30], [31]. Additionally, in our research we do not use duplication-based scheduling performing the same tasks on different servers in order to minimize inter-task communication delays [28], [29].

A comprehensive survey of basic heuristic methods for workflow scheduling can be found in [5]. This standard collection is usually supplemented with Heterogeneous Earliest-Finish-Time (HEFT) algorithm [43]. It is a greedy approach which ranks all tasks according to the lengths of their critical paths and then at each step selects the best server for the highest ranked task that has not been assigned yet. The approach uses a simplified model of inter-task communication; thus, it can be unconscious of communication delays resulting from congested links. The sole optimization goal of all the methods from the mentioned standard collection is to minimize the total makespan. In our research, two optimization goals are taken into account. First, as in the collection, we minimize the total makespan. Second, consumed energy is minimized, which usually contradicts the first goal. One of the most popular algorithm minimizing the cost (defined for instance as consumed energy), while satisfying the requested deadlines is Partial Critical Path (PCP) [1], [2]. Its main idea is to calculate virtual deadlines for each task, and then to allocate resources to the tasks, in a way that minimizes the cost while satisfying the calculated virtual deadlines. We present a novel method that is also based on virtual deadlines. However, our approach is built on a concept of minimum dependencies between tasks instead of the concept of partial critical paths, which makes it more efficient in vast spectrum of considered test cases.

Other heuristic algorithms solving the problem of resource allocation taking into account two independent criteria for assessing the solutions can be found in [7], [37], [46], while in [8] an algorithm is described, which solves the general problem of multi-criteria optimization of scheduling in the cloud. One of the few algorithms that meets the vast majority of our assumptions is PASTA presented in [38]. Still, it insufficiently models inter-task communication aspects and it only considers two possible CPU energy states. What is more, it does not support dynamic management of servers, thus it cannot be directly applied to the version of the problem approached in this paper.

Multiple researchers who focused on detailed system modeling dealt with the resulting complexity of optimization using heuristics [11]. Such soft computing tools can be successfully used to optimize energy-aware workflow scheduling by specialized operators [15], hybridization with heuristics [30], or learning mechanisms [41]. A detailed, bi-objective study of the problem suggests that the scale (in terms of the number of tasks and machines) and communication intensity are the factors that make the problem harder to solve [14]. The common disadvantage of methods based on computational intelligence is their high computational cost, which makes them usable mostly for small scale, preferably static problems, such as batch scheduling.

4 MinD+ED Scheduling

The proposed Minimum Dependencies Energy-efficient DAG scheduling is a novel task scheduling algorithm designed to optimize energy efficiency. It forms scheduling decisions based on the utilization level of network links and servers in two phases. During the first phase, virtual deadlines for each task of a received workflow are calculated. The calculations are based on the idea of Minimum Dependencies (MinD) described in detail in Section 4.1.4. During the second phase, tasks are dynamically assigned to servers that can satisfy previously computed virtual deadlines. Task assignment is done in an energy-efficient way; thus, the acronym ED stands for Energy-efficient DAG assignment. Although the mentioned decomposition of the problem is not new, we managed to provide substantial novelties to both phases of the decomposed scheduling process.

Notice that by dividing the process into two phases it is much easier to handle uncertainty in inter-task communication and task execution times. Although virtual deadlines are not subject to changes after setting, the assignment algorithm can react to drifts from expected communication and execution times.

As the scheduling process is decomposed into two phases, this section is also divided into two subsections. In the first subsection, the issue of setting virtual deadlines is covered, while in the second subsection the task assignment problem is discussed. Both subsections constitute the complete task scheduling algorithm called MinD+ED.

We again stress that the presented approach works for unrelated machines. However, to facilitate explanations, we temporary assume that the execution time is a quotient of the size of the task and the computational power of the machine.

4.1 Setting Deadlines

The first phase of the scheduling process is entirely executed in a gateway of a data center right after the arrival of a workflow. Therefore, the process cannot be complex, as the available resources are scarce and results are expected immediately. In this section, we present a novel way of setting virtual deadlines based on Minimum Dependencies. However, before presenting MinD, two different approaches, and their variations, are presented that are improved versions of methods available in the literature. Those two approaches are presented for two reasons:

- To allow for better understanding of MinD by referencing and building our method on state-of-the-art algorithms.
- To explain benchmark approaches that are used to experimentally grade MinD.

We focus on computational power of cores and inter-task communication. However, the proposed approach, when appropriately adjusted, can also take into account other resources (RAM, hard disk, etc.). Still, for the purpose of

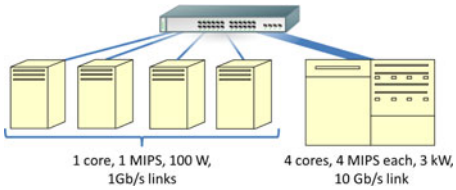

Fig. 2. Simplified data center topology.

this work we only focus on computational power of cores, and leave extensions of the approach for further research.

4.1.1 Notation

The first reference approach is based on [43]. Therefore, our notation is similar to the notation presented there.

- $\mathcal{N}$ set of tasks of a workflow;
- d workflow deadline;
- $\overline{w_i}$ approximate time needed to execute task i;
- $\overline{c_{i,k}}$ approximate time needed to send data from task i to task k.

In the approach, parameters $\overline{w_i}$ and $\overline{c_{i,k}}$ have to be provided. It is possible to either use a simple average over all possibilities, as in [43], or suggest something more sophisticated. In our approach, we utilize the following data.

- $P(l)$ probability density function (PDF) of a data center load defined for segment $(0, 1)$, where 0 represents an empty data center, while 1 represents a fully occupied data center.
- $\mathcal{C}$ set of all available computational cores; assume that the cores are ordered with respect to their power efficiency—the most energy-efficient cores come first.
- f_i number of instructions core i can execute in a second; total computational power of the data center equals $F = \sum_{i \in \mathcal{C}} f_i$.
- W_i number of instructions of task i.
- g_i link capacity in access tier available to a server accommodating core i.
- q_i fraction of an access tier link capacity available to core i; the parameter is set to a value between 1 and $\frac{1}{K}$, where K is the number of cores in a server accommodating the considered core.
- $C_{i,k}$ number of bytes sent during the inter-task communication between task i and task k.
- L constant average communication overhead.

Approximate computation time: We introduce a function of a data center load that expresses the computational power of a core that will accommodate a new task temporary assuming that tasks are assigned to the most energy-efficient available cores. For calculation of this function, it is temporary assumed that tasks are infinitely short (their execution time approaches zero, i.e., it is infinitesimal), evenly distributed in time, and those that are currently being executed are instantly migrated to a more energy-efficient core, whenever such a core is available (Please, note that in the base problem tasks cannot migrate). The function looks as follows:

$$F(l) = f_i : \sum_{j \in \mathcal{C}: j < i} f_j < lF \wedge \sum_{j \in \mathcal{C}: j \leq i} f_j \geq lF. \tag{1}$$

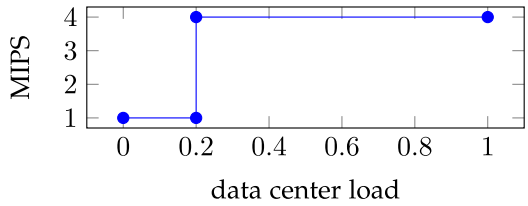

Fig. 3. Computational power of a core that will accommodate a new task temporary assuming possible migrations.

Notice that more computationally powerful cores are privileged by the function, as they can serve more tasks than weaker cores during the same amount of time. Therefore, chances a new task will be assigned to them are greater. In the base considered problem, tasks cannot be migrated between cores. Therefore, a function of a data center load that expresses the expected computational power of a core that will accommodate a new task is not simply (1), but

$$U(l) = \int_0^l F(l')\mathrm{d}l' \cdot l^{-1}. \tag{2}$$

The expected time of executing task i depends on its size W_i and the expected computational efficiency of a core executing it, which is computed as a weighted average of $U(l)$, where the weights are $P(l)$. The formula is

$$\overline{w_i} = W_i \left(\int_0^1 P(l)U(l)\mathrm{d}l \right)^{-1}. \tag{3}$$

In the formula, the number of instructions of task W_i is divided by the aforementioned weighted average, which is formally defined in the brackets of the formula.

Consider a simplified data center topology presented in Fig. 2 consisting of five servers. Four servers are one-core machines that can process one million instructions per second (1 MIPS). The last server is equipped with four cores, which are more powerful (4 MIPS each). However, the last server is less energy-efficient needing 187.5 W/MIPS, while the other servers need only 100 W/MIPS. In such a data center, function $F(l)$ that expresses the computational power of a core that will accommodate a new task temporary assuming possible immediate migrations of tasks is presented in Fig. 3.

Function $U(l)$ expressing the expected computational power of a core that will accommodate a new task assuming that migrations are not possible is depicted in Fig. 4.

Assuming a probability density function of a data center load as in Fig. 5, the expected time of executing task i consisting of one million instructions will be $w_i \approx 1 \cdot (2.35)^{-1}$, which is approximately 0.43 second. Notice that the obtained time is larger than the time that would be obtained considering an average computational power of cores (0.4 s)

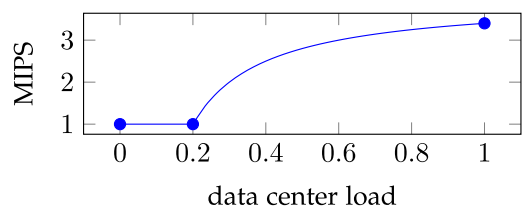

Fig. 4. Expected computational power of a core that will accommodate a new task assuming no migrations.

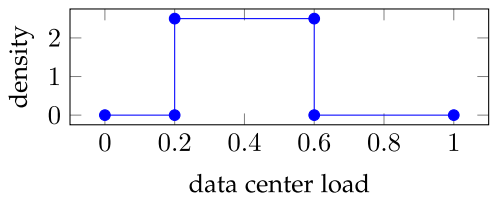

Fig. 5. Probability density function of a data center load.

or their weighted average (0.29 s). The reason is that in the considered data center weaker, but more energy-efficient, cores are much more frequently selected for computations than other cores.

Approximate communication time: Similar approach can be used to define an approximate inter-task communication time. We introduce a function of a current data center load that expresses the available access link capacity of the most probable server that accommodates the most energy-efficient vacant core. As in the previous case, also here we temporary assume unrestricted migrations of infinitely small tasks. In the formula, link capacity g_i is multiplied by q_i expressing the expected fraction of the link capacity available to core i. The fraction is a parameter. It should be set to a value between 1 and $\frac{1}{K}$, where K is the number of cores in a server accommodating the core, depending on the proportion between computations and communications in expected workflows. The function looks as follows:

$$G(l) = q_i g_i : \sum_{j \in \mathcal{C}: j < i} f_j < lF \wedge \sum_{j \in \mathcal{C}: j \leq i} f_j \geq lF. \tag{4}$$

The expected inter-task communication time between tasks i and j is:

$$\overline{c_{i,k}} = C_{i,k} \left(\int_0^1 P(l) \frac{\int_0^l G(l')\mathrm{d}l'}{l} \mathrm{d}l \right)^{-1} + L. \tag{5}$$

where L is a constant average communication overhead.

In the considered data center, assuming that parameter q_i for the four-core server equals $\frac{1}{4}$ and an average communication overhead is 1 ms, sending 1 MB takes 5.7 ms.

4.1.2 Critical Path Deadlines (CP-P and CP-E)

The first reference approach is based on HEFT algorithm [43]. The algorithm can be directly applied, as it is also based on virtual deadlines. However, its objective is to minimize a makespan. Therefore, its results should be appropriately adjusted to account for different optimization goals. Two adjustments are presented in this section.

The defined parameters $\overline{w_i}$ and $\overline{c_{i,k}}$ allow us, similarly to [43], to compute an *upward rank*, denoted by $r(i)$ for task i, using formula:

$$r(i) = \max_{j \in \delta^+(i)} (\overline{c_{i,j}} + r(j) + \overline{w_j}), \tag{6}$$

where $\delta^+(i)$ is a set of immediate successors of task i. The rank should be computed recursively by traversing the task graph upwards. For the sink task the upward rank equals 0. The rank is simply the length of the critical path for the task. Consider a workflow consisting of tasks $\mathcal{N} = \{A, B, \ldots, Z\}$, such that task A is the source task, submitted to the data center in time t. In the straightforward approach, the deadline of task i, denoted by $d(i)$, is set to:

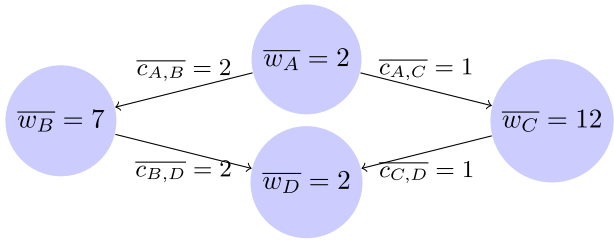

Fig. 6. Considered directed acyclic graph (DAG).

$$d(i) = t + r(A) + \overline{w_A} - r(i), \tag{7}$$

where $r(A) + \overline{w_A}$ is the minimum total time to execute the whole workflow.

Those virtual deadlines are obviously useless from the viewpoint of our studies, as they solely account for the minimization of the makespan, i.e., they do not allow for extending the makespan to improve energy efficiency. The simplest way to tackle it is to set virtual deadlines to the highest possible values that allow for completing workflows on time. Assuming that the workflow deadline is denoted by d, virtual deadlines can be set to:

$$d(i) = d - r(i). \tag{8}$$

The main drawback of such an approach is the unfair division of the available slack time. Although setting virtual deadlines does not directly imply how the available slack time should be divided, the energy-aware scheduling strategies, described in Section 4.2 and used in this research for the actual task assignment, tend to distribute tasks between servers in a way their execution times are close to their deadlines. Therefore, the maximum deadline approach clearly favors earlier tasks to late tasks, as relative deadlines are longer for them. The drawback can be mitigated in various ways of which two are presented in following subsections.

Proportional approach (CP-P): One of the possible approaches is to set a virtual deadline of each task proportionally to ranks $r(i)$ computed using (6). Consider a workflow consisting of tasks $\mathcal{N} = \{A, B, \ldots, Z\}$, such that task A is the source task. Consider that the workflow is submitted to the data center in time t and its deadline is d. Then, the deadline of task i equals:

$$d_{CP-P}(i) = t + (d - t) \frac{r(A) + \overline{w_A} - r(i)}{r(A) + \overline{w_A}}. \tag{9}$$

The value in the bracket, i.e., $d - t$, is the available time, the denominator is the length of the workflow's critical path, while the numerator is the length of the considered task's critical path.

Consider DAG presented in Fig. 6 submitted to a data center at time $t = 0$ with a given workflow deadline $d = 25$. The values of $r(A) + \overline{w_A} - r(i)$ equal: 18 for D, 15 for C, 14 for B, and 2 for A. As the total fastest time of executing the whole workflow, i.e., $r(A) + \overline{w_A}$, equals 18, virtual deadlines should be set to:

- $d_{CP-P}(D) = 25 \cdot \frac{18}{18} = 25$.
- $d_{CP-P}(C) = 25 \cdot \frac{15}{18} \approx 20.8$.

- $d_{CP-P}(B) = 25 \cdot \frac{14}{18} \approx 19.4$.
- $d_{CP-P}(A) = 25 \cdot \frac{2}{18} \approx 2.8$.

Equal slack approach (CP-E): In the previous approach, the available time was divided proportionally to the latest finish time of the tasks. Unfortunately, such an approach can also lead to unfair division of the slack time, as it favors tasks that cannot be parallelized. Consider a fork-join DAG of identical tasks with the source node A, the sink node Z, and 10 independent tasks in-between that can be executed in parallel. In such a situation, the previous approach gives a third of the time to task A, a third of the time to task Z, and also a third of the time to the remaining tasks. Such an approach is reasonable when unlimited number of servers are available and minimizing a makespan is the main objective. However, when power efficiency is the factor, the approach seems not to be the best option.

The drawback can be mitigated by dividing the available time equally between tasks. The idea is simple but very efficient. Assume that $n(i)$ defines an order of tasks with respect to the inverse of $r(i)$, i.e., $n(i) \in \{1, 2, \ldots, |\mathcal{N}|\}$, $n(i) \neq n(j) \Leftrightarrow i \neq j$, and $r(i) > r(j) \Rightarrow n(i) < n(j)$. Then, the deadlines can be defined as follows:

$$d(i) = t + (d - t)\frac{n(i)}{|\mathcal{N}|}. \tag{10}$$

Finally, the approach can incorporate the impact of the execution time mixing it with the fair distribution of the slack time. The resulting approach is called the equal slack approach and is expressed by the formula:

$$\begin{aligned} d_{CP-E}(i) = t + r(A) + \overline{w_A} - r(i) + \\ + (d - t - r(A) - \overline{w_A})\frac{n(i)}{|\mathcal{N}|}. \end{aligned} \tag{11}$$

In the example of Fig. 6, the virtual deadlines are:

- $d_{CP-E}(D) = 18 + 7 \cdot \frac{4}{4} = 25$.
- $d_{CP-E}(C) = 15 + 7 \cdot \frac{3}{4} = 20.25$.
- $d_{CP-E}(B) = 14 + 7 \cdot \frac{2}{4} = 17.5$.
- $d_{CP-E}(A) = 2 + 7 \cdot \frac{1}{4} = 3.75$

4.1.3 Partial Critical Path Deadlines (PCP)

The second benchmark approach is based on Partial Critical Path approach published in [1]. The detailed explanation of the approach can be found in the cited paper. In this section, we concentrate on a brief introduction to the approach and on defining concepts that are shared by the approach and our novel algorithm presented in the following section.

PCP is based on the latest finish time rank for each task, which is similar to rank $r(i)$ introduced for CP approach. The only difference lay in the way expected execution time $\overline{w_i}$ are defined. In CP, they were computed as a weighted average over execution time of the task on all possible servers. On the other hand, in [1], they are set to the execution time obtained on the fastest available machines. Unfortunately, in practice, such an approach is not efficient, especially when the fastest machines are scarce resources, which was the case in our experiments. Therefore, we use parameters $\overline{w_i}$ defined in previous sections. Formally, the rank is denoted by $LFT(i)$ and is the same as the virtual deadlines defined in (8):

$$LFT(i) = d - r(i), \tag{12}$$

where d is the deadline of the workflow.

Another rank used in the approach is the earliest start time rank. The rank, denoted by $EST(i)$, expresses the earliest moment an execution of task i can start. As for $LFT(i)$, also in this case we use parameters $\overline{w_i}$ instead of the execution time on the fastest machine. The parameters, together with parameters $\overline{c_{i,k}}$ defining the average inter-task communication time (also defined earlier), allow us to compute the rank using formula:

$$EST(i) = \max_{j \in \delta^-(i)} (\overline{c_{j,i}} + EST(j) + \overline{w_j}), \tag{13}$$

where $\delta^-(i)$ is a set of immediate predecessors of task i. The rank should be computed recursively by traversing the task graph downwards. For the source task the upward rank equals t, which is the time the workflow is submitted to the data center.

In the considered example, the ranks are as follows:

- $EST(D) = 16$, $LFT(D) = 25$.
- $EST(C) = 3$, $LFT(C) = 22$.
- $EST(B) = 4$, $LFT(B) = 21$.
- $EST(A) = 0$, $LFT(A) = 8$.

The main idea of the algorithm is to iteratively find partial critical paths and try to assign slower, but more energy-efficient, servers to tasks of the path. The partial critical path finishing at node i, which in the first iteration is the sink node, is a path that reaches node i from node j, which has not been assigned yet and its $EST(j) + \overline{w_j} + \overline{c_{j,i}}$ is maximal. If there are no unassigned nodes that can precede node i, then the partial critical path is found, all its nodes are marked assigned, and more energy-efficient servers are trying to be assigned to its nodes. Then the process repeats from another node that does not have any unassigned nodes that succeed it.

In [1], three ways of assigning more cost-efficient servers to tasks of a partial critical path are presented. We decided to use Fair Policy in our study, as according to [1], it not only gives satisfactory results, but it is also computationally affordable. The policy for each task orders all Pareto optimal servers (not dominated by other servers both in computational power and energy efficiency) with respect to their computational power. Then, in the first iteration it tries to replace the fastest server with the second fastest server for each task. In the next iteration, the third fastest server is taken into account, etc. The process ends when all Pareto optimal servers for all tasks are checked. After each successful replacement of servers ranks LFT and EST are appropriately adjusted limiting possibilities of further replacements. This observation led us to the crucial novelty of our research, which is a deadline setting algorithm described in the following section.

But first consider the DAG of Fig. 6 executed in a data center of Fig. 2. In Section 4.1.1, we saw that the expected computational efficiency is 2.35 times greater than the computational power of the most energy-efficient cores. The first critical

partial path in the considered example consists of tasks A, C, and D. Time constraints let only two of them, namely A and D, be assigned to more energy-efficient machines. The second critical partial path consists solely of task B with updated $EST(B) = 6.7$ and $LFT(B) = 18.3$. The new time constraints do not allow assigning task B to a more energy-efficient core. Therefore, the final virtual deadlines are:

- $d_{PCP}(D) = 25$.
- $d_{PCP}(C) = 19.3$, as task D needs 4.7 for execution.
- $d_{PCP}(B) = 18.3$, as task D needs 4.7 for execution.
- $d_{PCP}(A) = 6.3$.

4.1.4 Minimum Dependencies Deadlines (MinD)

In this section, we present a novel energy-aware approach to setting virtual deadlines. The approach is based on the idea of prioritizing tasks with smaller dependencies on other tasks while extending their deadlines. We say that task i depends on task j if in the DAG representing the workflow there is a path from i to j or from j to i. By $\mathcal{D}_i$ we denote a set of all tasks that are dependent on task i. Then, a priority of task i, denoted by $p(i)$, is expressed as follows:

$$p(i) = -\sum_{j \in \mathcal{D}_i} \overline{w_j}. \tag{14}$$

The tasks are ordered with respect to their priorities, and tasks of higher priorities are considered earlier. Notice that, as the priorities are negative, the highest priority is the one closest to zero. For each task (taking the previously described order into account), similarly to PCP, slower, but more energy-efficient, servers are being tried, and if the replacement is possible, ranks LFT and EST are appropriately adjusted. As the approach starts with the least dependable tasks, first server replacements have relatively smaller impact on LFT and EST in comparison to PCP; thus, they allow for more server replacements in the future, making the approach more efficient, which is clearly indicated in Section 5.2.

In the example, tasks B and C have the highest priority, but only task B can be assigned to a more energy-efficient core. After modifying the time constraints, tasks A and D cannot be assigned to other servers; thus, the final virtual deadlines are:

- $d_{MinD}(D) = 25$.
- $d_{MinD}(C) = 22$.
- $d_{MinD}(B) = 21$.
- $d_{MinD}(A) = 2.55$.

Finally, it is important to notice that MinD, similarly to PCP, still experiences the slack time problem. In other words, the available time can be more than enough to execute all tasks on the most energy-efficient servers. We understand the slack time as the amount of time a whole workflow can be idle and still finish before its deadline, which formally can be understood as the latest start time of source task A minus the current time. Assuming that in the results of MinD task i is assigned to core $e(i)$ let us define the slack time for MinD as follows:

$$s = LFT(A) - \frac{W_A}{f_{e(A)}} - t \tag{15}$$

In the described situation, the whole available slack time would be assigned to the source task, as deadlines are set to LFT. We mitigate the issue by setting virtual deadlines to:

$$d_{MinD}(i) = LFT(i) - s\frac{|\mathcal{N}| - n(i)}{|\mathcal{N}|}, \tag{16}$$

where $n(i)$ is a one-based order of a task calculated with respect to LFT (lower LFTs come first).

In our example, the slack time is only 0.55. Therefore, the adjustments are not critical, i.e., $d_{MinD}(C) \approx 21.9$, $d_{MinD}(B) \approx 20.7$, $d_{MinD}(A) \approx 2.1$. However, when workflow deadlines are relatively high, the approach becomes similar to CP-E of Section 4.1.2

Let us now comment on the computational complexity of MinD. To compute a priority of a single task, one execution of DFS is needed; thus, priorities of all tasks can be computed in $O(V \times E)$, where V is the number of tasks of a workflow, while E is the number of edges in a graph representing a workflow. When the priorities are set, different core assignments are being tried. There are V tasks, each of them can be executed on one of C cores, and in the worst case each change in core assignment can lead to V updates of LFT and EST. Therefore, the total computational complexity of MinD is $O(V \times E + V^2 \times C)$.

4.2 Task Assignment

In the first phase of the scheduling process, virtual deadlines for individual tasks were set. The proposed virtual deadline setting algorithm (MinD) is not computationally expensive. Therefore, the process can be centralized and all virtual deadlines can be set in a gateway of a data center. The second phase of the scheduling process deals with assigning tasks to particular servers. We propose a way of assigning tasks to servers named Energy-efficient DAG (ED) assignment. It is a novel approach to interrelated task assignment built on the state-of-the-art approaches to independent task assignment that takes advantage of available current link and server usage data. As not only much greater computational effort is needed here, but also required data are much larger, the proposed method is partially decentralized.

Before presenting the method in detail, let us state some general rules applied in this phase of the scheduling process.

- Each workflow has its master server knowing locations of all other tasks of the same workflow; master server roles can move between physical machines.
- Each task is scheduled when all data required by the task are available, i.e., a source task is scheduled immediately, while the remaining tasks are scheduled when all tasks, which are its predecessor, are finished.
- Scheduling decisions are irreversible.
- Tasks are executed by a server according to their priorities—the nearer the virtual deadline, the higher the priority of the task.
- Execution of a task can be suspended when a higher-priority task arrives at a server.

Let us now explain the concept of *master server* and *master path*. Consider a workflow represented by a DAG. One of

the shortest paths in terms of hops from the source task of the DAG to its sink task is called a master path. We assume that a server executing a task of a master path is at that time called a master server of the workflow. The master server accumulates all data concerning locations of all other tasks of the workflow. When the master server changes its location, i.e., when two consecutive tasks of the master path are executed on two different servers, the information is propagated to all servers executing tasks of the workflow. This way the master server can easily manage the whole workflow dynamically assigning tasks to servers.

4.2.1 Consolidation-Based DAG Assignment (CD)

Before presenting ED, we introduced a benchmark algorithm, called Consolidation-based DAG (CD) assignment, mainly to accustom the reader with basic notions of the issue, and to show advantages of more advanced task assignment policies.

The consolidation-based DAG assignment is supposed to minimize the total energy used by the data center. It orders all servers with respect to their energy efficiency, and assigns a task to the first server from the list that can accommodate it without violating the virtual deadlines of the scheduled task and all other previously assigned tasks. When the virtual deadlines cannot be satisfied, the task is assigned to the server that minimizes the sum of tardiness for all the tasks assigned to this server.

4.2.2 Energy-Efficient DAG Assignment (ED)

The energy-efficient DAG assignment is inspired by the HEROS methodology—recently published way of assigning independent tasks to heterogeneous servers [13]. HEROS is based on DENS [23] and e-STAB [21], as it relies on similar approaches to establishing server selection and communication potential functions.

ED uses four features while selecting a server. They are listed below in order of their importance:

1) Maximum Performance per Watt (MaxPpW) of a server.
2) Server physical location with respect to servers executing other tasks of the workflow.
3) Current utilization of a server.
4) Current communication potential of a server.

The final decision is based on the introduced features in the mentioned order, but it considers only servers that can meet all virtual deadline requirements. When servers are tied on the first feature, the second feature is considered taking into account only servers that are the best on the first feature. When the tie persists, the third feature is taken into account. Finally, if also the third feature does not resolve the problem, the fourth feature decides.

The most important feature is MaxPpW. It is based on Performance per Watt (PpW), which underlines energy efficiency and can be directly used to select the currently most energy-efficient server. A practical drawback of the straightforward usage of PpW is the fact that servers become the most energy-efficient when fully loaded; thus, we would like to fully load the majority of working servers. Therefore, we are not interested in the current PpW of a server, as the server will immediately either reach its peak utilization or be put into sleep. That is why, in ED, we select MaxPpW as the most crucial feature, scheduling a task on a server characterized by the highest value of MaxPpW.

It is important to emphasize that we do not consider exact MaxPpW, but rather classes of MaxPpW. There is a finite (and relatively small) group of different MaxPpW classes and each server is assigned to one of them. The way the number of classes and criteria that assign servers to classes are defined depends on actual needs of a user. In our experiments, we define one MaxPpW class for each server type. However, users may define fewer classes, if different types of servers resemble similar MaxPpW, or more, if for instance some servers are characterized by different MaxPpW, as a result of their location in a rack.

The second most important feature is location-based. It is defined as a minimum hop distance of a server to any of servers that execute tasks directly preceding the currently processed task. The value equals zero when at least one preceding task is being executed at the server, equals two when at least one preceding task is being executed at the same rack, etc. The maximum value of the feature depends on the number of tiers in a data center. A task is scheduled on the nearest server. Obviously, for the source task the feature is irrelevant as all servers are equally distant from the gateway. The goal of the feature is to limit excess communication between servers.

The third feature depends on the utilization of a server. When DVFS is not allowed, cores are considered either working or idle. Therefore, a single-core server can have only two possible states, while a 4-core server has five possible states. Such definition of the utilization does not fulfill the main goals of the feature, which are to consolidate tasks and prevent servers from not using all their available computing power. In our research, we define the utilization of a core as a fraction of the nearest time the core will be occupied, where the nearest time is the time needed by the core to execute the considered task assuming the core is vacant. In this way, only servers with all cores occupied during the nearest time are considered fully occupied. Higher utilizations are preferred.

The last feature is the actual server link load. Servers with smaller link loads are prioritized.

As features are considered in order, it may seem that the less important features are rarely taken into account; thus, their importance is questionable. However, taking a closer look dispels doubts concerning the issue. There are usually few possible values of the first feature (usually as many as distinct server types), and there are only few possible values of the second feature; thus, the third feature is often taken into account. The importance of the fourth feature is also significant, as it is used to select among fully loaded servers that still can execute the task currently being assigned.

In Figs. 7 and 8, the behavior of ED is presented with different virtual deadlines. We assume that four independent workflows of Fig. 6 almost simultaneously appear at a data center of Fig. 2. As shown in the figures, PCP deadlines allow for faster execution of all workflows taking 22.56 s overall, while MinD deadlines result in 24.33 s execution time. However, MinD deadlines are more energy-efficient

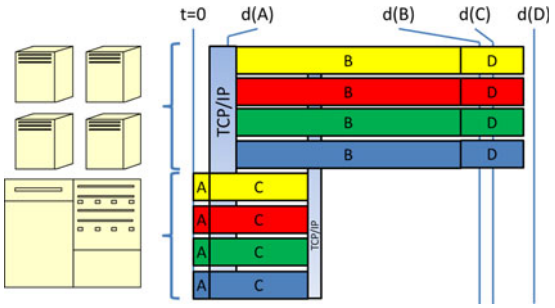

Fig. 7. ED task assignment with MinD deadlines.

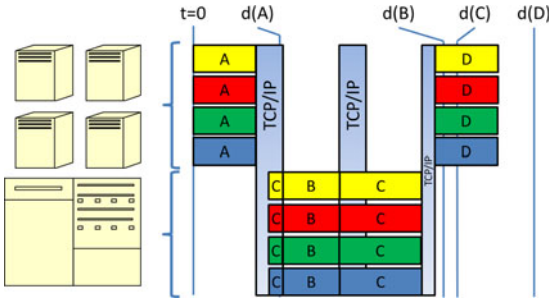

Fig. 8. ED task assignment with PCP deadlines.

TABLE 1
Considered Three-Tier Configurations

Configuration	Basic	Homogeneous	Large
Core Switches	1	1	4
Aggregation Switches	2	2	8
Access Switches	3	3	64
Servers in a Rack	48	48	8
Total Servers	144	144	512
Commodity Servers	18	144	48
HPC Servers	6	0	16
Micro Servers	120	0	448
Total MIPS	216,000	576,000	652,800

needing only 36.95 kJ, while PCP deadlines need 45.25 kJ. This general tendency concerning energy efficiency of PCP and MinD deadlines has been confirmed experimentally in the following section.

5 Experiments

Simulation results obtained while testing MinD+ED are presented in this section. The method is compared to benchmark algorithms introduced in the paper. The focus is put on differences the deadline setting methods imply and how the task assignment algorithms influence the results.

The experiments have been performed using GreenCloud—a popular simulation tool which offers fine-grained simulation of modern cloud computing environments focusing on data center communications and energy efficiency [32]. GreenCloud is based on the ns-2 [27] simulation platform, which enables simulation of the TCP/IP communications.

5.1 Test Cases

Table 1 presents three data center configurations that were used in the experiments. These configurations are: *Basic*, *Homogeneous*, and *Large*. *Basic* denotes a 144-server heterogeneous configuration with High Performance Computing (HPC) servers and highly energy-efficient yet computationally weak micro servers, *Large* denotes a similar 512-server configuration, while *Homogeneous* configuration is composed of 144 commodity servers only. The server specifications are presented in Table 2.

TABLE 2
Server Specifications

Server	Commodity	HPC	Micro
Core#	4	8	4
MIPS/Core	1,000	1,500	150
Total MIPS	4,000	12,000	600
Max Power	200 W	300 W	6 W
Min Power	100 W	100 W	3 W

TABLE 3
Workflow Characteristics

Feature	Montage	Epigenomics	Layer-by-layer
Avg. tasks	77.4	16.9	20.0
Instructions	$582 \cdot 10^9$	$753 \cdot 10^9$	$300 \cdot 10^9$
Avg. arcs	184.8	26.6	70.9
Input comm.	2.0 MB	242.0 MB	0.1 MB
inter-task comm.	688.6 MB	3.6 MB	71.3 MB
Output comm.	1.0 MB	83.0 MB	2.5 MB

Links connecting the access tier to the aggregation tier and all links inside the access tier are 1 Gb/s Ethernet links, while all other links are 10 Gb/s Ethernet links.

A mixture of different workflows has been used in our experiment consisting, in equal proportions, of the following workflows: *Montage*, *Epigenomics*, and *Layer-by-layer*. *Montage* and *Epigenomics* are real-world workflows [19], while *Layer-by-layer* is an artificial workflow commonly used to assess scheduling algorithms [42]. Characteristics of the workflows are briefly summarized in Table 3.

Let us now calculate parameters $\overline{w_i}$ and $\overline{c_{i,k}}$, i.e., a mean execution time of task i and a mean communication time between tasks i and k, respectively. The parameters are defined in (3) and (5). In the calculations, the capacity of access network links, the computational power and energy efficiency of available servers, and finally the PDF of data center load is needed. We conduct the majority of our experiment with 70 percent data center load. Therefore, we assume a simplified modeling of the current data center load using a normal distribution $\mathcal{N}(0.7, 0.03)$. The order of the servers with respect to their energy efficiency is as follows: Micro servers, HPC servers, and Commodity servers. For Basic data center configuration results are $\overline{w_i} = \frac{W_i}{817{,}048{,}000}$, which for a representative Montage workflow of $582 \cdot 10^9$ instructions returns an execution time of 712.3 s. As for the mean communication time, assuming the average communication overhead of 1 ms, we obtain $\overline{c_{i,k}} = \frac{C_{i,k}}{23{,}529{,}500} + 0.001$ for Basic data center configuration, which for a representative Montage inter-task communication of 688.6 MB divided into 185 communication session returns a total inter-task communication time of 29.5 s. The obtained parameters for other configurations and data center loads can be found in Table 4.

TABLE 4
Average Computing and Communication Time of a Representative Montage Workflow for Different Configurations and Data Center Loads

Configuration	Basic	Homogeneous	Large	
Load	30%	any	30%	70%
Computation [s]	2119.3	582.0	2910.6	878.7
Communication [s]	23.3	22.2	22.6	27.4

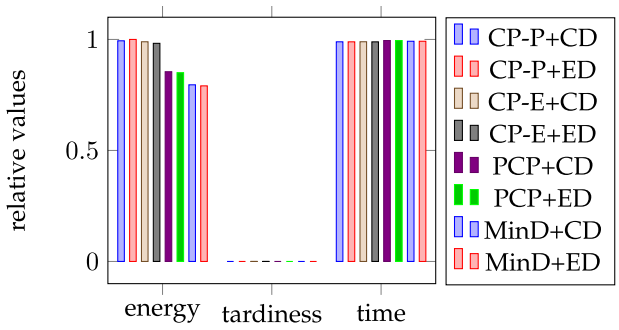

Fig. 9. Results for Basic configuration.

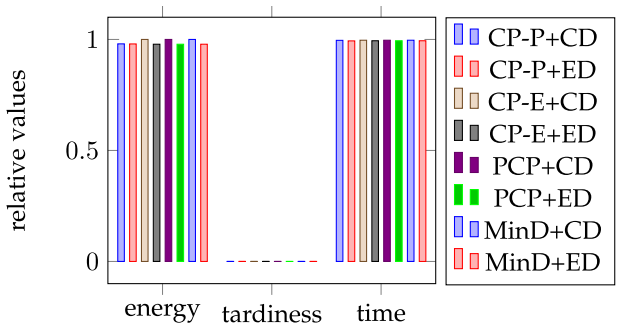

Fig. 10. Results for Homogeneous configuration.

5.2 Numerical Results

In this section, we present numerical results obtained for a number of combinations of deadline setting approaches, task scheduling algorithms, and data center configurations. As for deadline setting approaches, we have tested: Proportional (CP-P) and Equal slack (CP-E) approaches to Critical Path deadlines. We have also tested Partial Critical Path deadlines (PCP) and Minimum Dependencies deadlines (MinD). Concerning task scheduling algorithms, we have tested Consolidation-based DAG assignment (CD) and Energy-efficient DAG assignment (ED). This way we have ended with eight possible scheduling strategies.

For each combination six independent simulations have been performed each constituting one hour of data center operation. During one hour, assuming 70 percent of data center load, over 2 k workflows have been served by Basic configuration, Homogeneous configuration has served more than 5 k workflows on average, while Large configuration has served more than 6 k workflows. The results are presented in Fig. 9 for Basic configuration, in Fig. 10 for Homogeneous configuration, and in Fig. 11 for Large configuration. Energy is scaled to the highest value obtained by any scheduling strategy, while tardiness and time are scaled to the deadline. The figures clearly indicate that all the strategies tend to use all available time for workflows, seldom

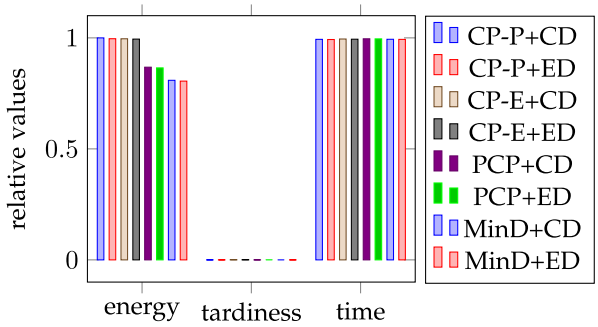

Fig. 11. Results for Large configuration.

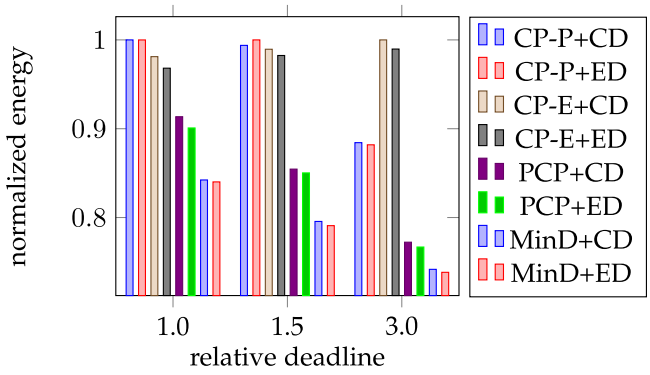

Fig. 12. Energy consumption for different deadlines.

exceeding deadlines. Still, MinD+ED does it in the most energy-efficient way.

The numerical experiments indicate that MinD+ED outperforms other strategies in terms of energy efficiency, beating them in wide range of tested scenarios. In general, it:

- acts like state-of-the-art deadline setting strategies in case of homogeneous data centers;
- saves more energy in heterogeneous data centers;
- exhibits similar workflow running times to other strategies.

The results do not display significant differences between particular assignment strategies, i.e., CD and ED. In fact, the difference is not substantial, but visible and stable—ED slightly outperforms CD in great majority of test cases.

In Fig. 12, energy consumption for all strategies is presented for different relative deadlines, where value 1.0 relates to a situation in which the deadline is set to the time needed to execute the whole workflow using a single core of a Commodity server. The results exhibit the key advantage of MinD over PCP, which is effectiveness in finding tasks that can be prolonged and run on more energy-efficient machines. When deadlines increase, it becomes easier; thus, a relative difference between PCP and MinD decreases.

In Fig. 13, an impact of load on energy-efficiency is presented. The superiority of MinD deadlines can be observed for greater loads. For smaller loads (less than 0.3 in our case), significant energy savings can be obtained through consolidating tasks on single machines. MinD uses energy-efficient machines more often that other scheduling strategies. Still when the load is small, the strategy can lead to underutilization of fast but non-economic machines.

In the next step, a number of experiments comparing behavior of the proposed strategies depending on served

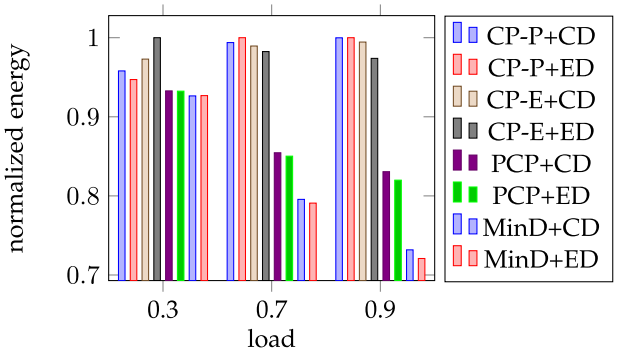

Fig. 13. Energy consumption for different loads.

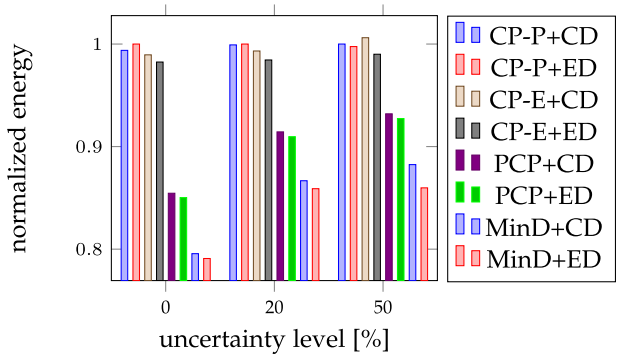

Fig. 15. Energy consumption for different evaluation precisions.

workflows have been performed. The results are displayed in Fig. 14. The experiments indicate that MinD+ED performs better than other strategies independently of the workflow structure.

Notice that all values describing tasks and inter-task communications that are provided to the gateway are estimates and may be not precise. Fig. 15 indicates relation between the level of uncertainty of those estimates and the consumed energy. The results confirm that all the evaluated methods suffer from incorrect estimates. Although suffering the most, MinD+ED still performs better than other methods for the considered levels of uncertainty.

Finally, computational complexity of MinD+ED has been evaluated concentrating mostly on the complexity of MinD deadline setting strategy, as this part of the scheduling process is centralized. When workflows of 20 tasks on average are considered, the mean time of executing MinD on a single core of a modern commodity CPU is below 10 ms, a workflow of 200 tasks is processed in less than 0.2 s, while a 2,000-task workflow needs less than 5 s on average. The results allow us to conclude that the computational complexity is not an issue in practical implementations of the proposed strategy.

Summarizing, MinD+ED proved to be more energy-efficient than other state-of-the-art approaches characterized by a similar computational complexity. MinD+ED exhibits its superiority in heterogeneous environments. In some cases, the better energy efficiency results in higher tardiness of submitted workflows. However, it is not a rule.

6 Conclusion

We presented a novel methodology named Minimum Dependencies Energy-efficient DAG scheduling that operates in data centers and dynamically schedules workflows consisting of interrelated tasks in an energy-efficient and communication-aware fashion. MinD+ED clearly outperforms state-of-the-art approaches that deal with similar problems and are of similar computational complexity. It can reduce the server energy consumption by almost 30% in comparison to other approaches.

In MinD+ED, the scheduling process is decomposed into two phases. In the first phase, virtual deadlines of individual tasks are set in the gateway of the data center. In the second phase, tasks are dynamically assigned to servers taking into account: characteristics of servers, physical locations of servers, and current loads of servers and links. Additionally, in MinD+ED, a novel way of setting virtual deadlines is used, which is based on minimizing dependencies between tasks. The proposed approach has been implemented in the GreenCloud simulator. It was shown that MinD+ED outperforms other strategies in terms of energy efficiency.

We propose multiple research directions that can follow this research. For instance, data flows with more complex communication models could be investigated, e.g., intermediary tasks (which are neither sink nor source tasks) that communicate with a database or considering distribution of input data among servers. Moreover, interactions in a single system between workflows and tasks without dependencies can be analyzed. The impact of virtualization and multitenancy on the proposed scheduling approach is another research opportunity. In this work, we statically define the time when tasks are scheduled and virtual deadlines are set. However, multiple choices are available, as it could be done a priori, incrementally, or a posteriori. The comparison of these approaches and designing an adaptive policy is an interesting topic. Finally, adapting and testing the scheduling algorithm for the future systems with more efficient implementations of DVFS, or developing a server distance metric which includes not only the number of hops but also the usage of links and their latency, sound like promising directions of extending the research.

The study could be also extended and adapted to multiple data centers or mobile applications scenarios. In this context, we consider the aspect of cloud brokering [12] as a way to share the benefits of efficient scheduling with end-user, which could create an additional incentive to share the descriptions of submitted workflows and tasks.

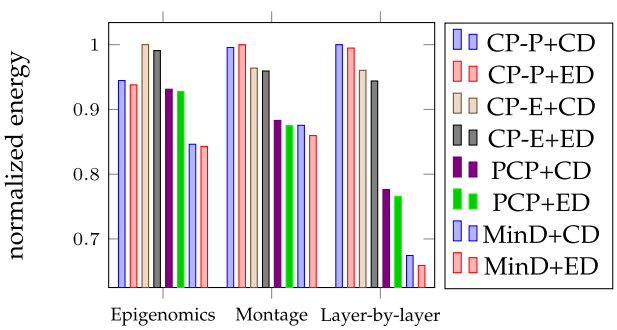

Fig. 14. Energy consumption for different DAG types.

Acknowledgments

This work has been supported by the European Union in the framework of European Social Fund through the project:

Supporting Educational Initiatives of the Warsaw University of Technology in Teaching and Skill Improvement Training in the Area of Teleinformatics. Moreover, the authors would like to acknowledge the funding from National Research Fund (FNR), Luxembourg in the framework of ECO-CLOUD (C12/IS/3977641), Green@Cloud (INTER/CNRS/11/03), and IShOP (POL-LUX/13/IS/6466384) projects. The presented experiments were carried out using the HPC facility of the University of Luxembourg [44].

REFERENCES

[1] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1400–1414, Aug. 2012.

[2] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Comput. Syst.*, vol. 29, no. 1, pp. 158–169, 2013.

[3] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Y. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," in *Adv. Comput.*, vol. 82, pp. 47–111, 2011.

[4] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 3, pp. 299–316, Jun. 2000.

[5] T. D. Braun, H. J. Siegel, N. Beck, L. L. Blni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

[6] DatacenterDynamics. (Jan. 2014). DCD industry census 2013: Data center power [Online]. Available: http://www.datacenterdynamics.com/focus/archive/2014/01/dcd-industry-census-2013-data-center-power

[7] A. Dogan and F. Ozguner, "Trading off execution time for reliability in scheduling precedence-constrained tasks in heterogeneous computing," in *Proc. 15th Int. Parallel Distrib. Process. Symp.*, Apr. 2001, p. 8.

[8] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, and T. Fahringer, "A multi-objective approach for workflow scheduling in heterogeneous environments," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2012, pp. 300–309.

[9] C. Fiandrino, D. Kliazovich, P. Bouvry, and A. Y. Zomaya, "Performance and energy efficiency metrics for communication systems of cloud computing data centers," *IEEE Trans. Cloud Comput.*, 2015. (To be published).

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," in , *Proc. Ann. Discr. Math.*, 1979, pp. 287–326.

[11] M. Guzek, P. Bouvry, and E.-G. Talbi, "A survey of evolutionary computation for resource management of processing in cloud computing [review article]," *IEEE Comput. Intell. Mag.*, vol. 10, no. 2, pp. 53–67, May 2015.

[12] M. Guzek, A. Gniewek, P. Bouvry, J. Musial, and J. Blazewicz, "Cloud brokering: Current practices and upcoming challenges," *IEEE Cloud Comput.*, vol. 2, no. 2, pp. 40–47, Mar. 2015.

[13] M. Guzek, D. Kliazovich, and P. Bouvry, "HEROS: Energy-efficient load balancing for heterogeneous data centers," in *Proc. 8th IEEE Int. Conf. Cloud Comput.*, Jun. 2015, pp. 1–8.

[14] M. Guzek, J.E. Pecero, B. Dorronsoro, and P. Bouvry, "Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems," *Appl. Soft Comput.*, vol. 24, pp. 432–446, 2014.

[15] M. Guzek, J. E. Pecero, B. Dorronsoro, P. Bouvry, and S. U. Khan, "A cellular genetic algorithm for scheduling applications and energy-aware communication optimization," in *Proc. Int. Conf. High Perform. Comput. Simul.*, Jun. 2010, pp. 241–248.

[16] M. Guzek, S. Varrette, V. Plugaru, J. E. Pecero, and P. Bouvry, "A holistic model of the performance and the energy efficiency of hypervisors in a high-performance computing environment," *Concurrency Comput.: Practice Experience*, vol. 26, no. 15, pp. 2569–2590, 2014.

[17] L. Hongyou, W. Jiangyong, P. Jian, W. Junfeng, and L. Tang, "Energy-aware scheduling scheme using workload-aware consolidation technique in cloud data centres," *Commun., China*, vol. 10, no. 12, pp. 114–124, Dec. 2013.

[18] IHS. (Feb. 2014). Cloud—related spending by businesses to triple from 2011 to 2017 [Online]. Available: http://press.ihs.com/press-release/design-supply-chain/cloud-related-sp ending-businesses-triple-2011-2017

[19] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

[20] J. Kang and S. Ranka, "Assignment algorithm for energy minimization on parallel machines," in *Proc. Int. Conf. Parallel Process. Workshops*, Sep. 2009, pp. 484–491.

[21] D. Kliazovich, S. T. Arzo, F. Granelli, P. Bouvry, and S. U. Khan, "e-STAB: Energy-efficient scheduling for cloud computing applications with traffic load balancing," in *Proc. IEEE Int. Conf. Green Comput. Commun.*, Aug. 2013, pp. 7–13.

[22] D. Kliazovich, P. Bouvry, and S. U. Khan, "GreenCloud: A packet-level simulator of energy-aware cloud computing data centers," *J. Supercomput.*, vol. 62, no. 3, pp. 1263–1283, 2012.

[23] D. Kliazovich, P. Bouvry, and S.U. Khan, "DENS: Data center energy-efficient network-aware scheduling," *Cluster Comput.*, vol. 16, no. 1, pp. 65–75, 2013.

[24] D. Kliazovich, J. E. Pecero, A. Tchernykh, P. Bouvry, S. U. Khan, and A. Y. Zomaya, "CA-DAG: Modeling communication-aware applications for scheduling in cloud computing," *J. Grid Comput.*, pp. 1–17, 2015.

[25] W. Lin, C. Liang, J. Z. Wang, and R. Buyya, "Bandwidth-aware divisible task scheduling for cloud computing," *Softw.: Practice Experience*, vol. 44, no. 2, pp. 163–174, 2014.

[26] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J. -M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 33:1–33:36, 2014.

[27] S. McCanne and S. Floyd. (1989). The network simulator—ns2 [Online]. Available: http://www.isi.edu/nsnam/ns/

[28] J. Mei and K. Li, "Energy-aware scheduling algorithm with duplication on heterogeneous computing systems," in *Proc. ACM/IEEE 13th Int. Conf. Grid Comput.*, Sep. 2012, pp. 122–129.

[29] J. Mei, K. Li, and K. Li, "Energy-aware task scheduling in heterogeneous computing environments," *Cluster Comput.*, vol. 17, no. 2, pp. 537–550, 2014.

[30] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuyttens, "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems," *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1497–1508, 2011.

[31] B. Mochocki, X. S. Hu, and G. Quan, "Transition-overhead-aware voltage scheduling for fixed-priority real-time systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 2, Apr. 2007, Art. no. 11.

[32] University of Luxembourg. (2010). GreenCloud simulator [Online]. Available: http://greencloud.gforge.uni.lu

[33] A. -C. Orgerie, M. D. de Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 47:1–47:31, 2014.

[34] J. Pouwelse, K. Langendoen, and H. Sips, "Energy priority scheduling for variable voltage processors," *Proc. Int. Symp. Low Power Electron. Des.*, 2001, pp. 28–33.

[35] Cisco. *Cisco data center infrastructure 2.5 design guide*. Indianapolis, IN, USA: Cisco press. Mar. 2010.

[36] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models," in *Proc. Conf. Power Aware Comput. Syst.*, 2008, pp. 3–7.

[37] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, "Scheduling workflows with budget constraints," in *Proc. CoreGRID Integration Workshop Integrated Res. GRID Comput.*, 2007, pp. 189–202.

[38] M. Sharifi, S. Shahrivari, and H. Salimi, "PASTA: A power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources," in *Computing*, vol. 95, no. 1, pp. 67–88, 2013.

[39] O. Sinnen, *Task Scheduling for Parallel Systems (Wiley Series on Parallel and Distributed Computing)*. Hoboken, NJ, USA: Wiley, 2007.

[40] O. Sinnen and L. A. Sousa, "Communication contention in task scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 6, pp. 503–515, Jun. 2005.

[41] F. Tao, Y. Feng, L. Zhang, and T. W. Liao, "CLPS-GA: A case library and pareto solution-based hybrid genetic algorithm for energy-aware cloud service scheduling," *Appl. Soft Comput.*, vol. 19, pp. 264–279, 2014.
[42] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *J. Scheduling*, vol. 5, no. 5, pp. 379–394, 2002.
[43] H. Topcuoglu, S. Hariri, and M. -Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[44] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos, "Management of an academic HPC cluster: The UL experience," in *Proc. Int. Conf. High Perform. Comput. Simul.*, Jul. 2014, pp. 959–967.
[45] K. Ye, Z. Wu, C. Wang, B. B. Zhou, W. Si, X. Jiang, and A. Y. Zomaya, "Profiling-based workload consolidation and migration in virtualized data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 878–890, Mar. 2015.
[46] J. Yu, and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in *Proc. Workshop Workflows Support Large-Scale Sci.*, Jun. 2006, pp. 1–10.

Mateusz Żotkiewicz received the BE degree from Coventry University, the MSc degree from Warsaw University of Technology, and the PhD degree in informatics from Telecom SudParis and in telecommunications from Warsaw University of Technology. He was a visiting researcher at Universitat Politecnica de Catalunya in 2013 and at University of Luxembourg in 2015. He is an assistant professor at Warsaw University of Technology (WUT). His research interests concentrate on network planning and optimization. He is an author of more than 50 research papers published in international telecom and OR journals and conference proceedings.

Mateusz Guzek received the PhD degree in holistic, energy-efficient optimization of cloud computing infrastructures from the University of Luxembourg in 2014. He is a research associate at the Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg. His research interests include cloud computing, resource allocation, including scheduling, load balancing and cloud brokering, dynamic multi-agent systems organizations, and a wide range of optimization techniques, from heuristics and exact methods to evolutionary computation. He is member of the IEEE.

Dzmitry Kliazovich received an award-winning PhD degree in information and telecommunication technologies from the University of Trento, Italy. He is a research fellow at the Faculty of Science, Technology, and Communication of the University of Luxembourg. He received a large number of scientific awards, mainly from the IEEE Communications Society. He chaired a number of highly ranked international conferences and symposia. He is the author of more than 100 research papers. He is an associate editor of the *IEEE Communications Surveys and Tutorials* and of the *IEEE Transactions of Cloud Computing Journals*. He is a vice chair of the *IEEE ComSoc Technical Committee on Communications Systems Integration and Modeling*. He is a coordinator and principal investigator of the Energy-Efficient Cloud Computing and Communications initiative funded by the National Research Fund of Luxembourg. His main research interest include energy efficient communications, cloud computing, and next-generation networking. He is senior member of the IEEE.

Pascal Bouvry received the PhD degree in computer science from the University of Grenoble (INPG), France. He is a professor in the computer science and communication research unit of the Faculty of Science, Technology and Communication, the University of Luxembourg and a faculty member at the Luxembourg Interdisciplinary Center of Security, Reliability, and Trust. His research interest include cloud & parallel computing, optimization, security and reliability. He is in the *IEEE Cloud Computing* and *Elsevier Swam and Evolutionary Computation* editorial boards. He is also acting communication vice-chair of the *IEEE STC on Sustainable Computing* and co-founder of the *IEEE TC on Cybernetics for Cyber-Physical Systems*. A full biography is available on page http://pascal.bouvry.org. Contact him at pascal.bouvry@uni.lu. He is member of the IEEE.