Future Generation Computer Systems 36 (2014) 221–236

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Multi-objective energy-efficient workflow scheduling using list-based heuristics

Juan J. Durillo *, Vlad Nae, Radu Prodan

Institute of Computer Science, University of Innsbruck, Austria

HIGHLIGHTS

- Mono-objective greedy schedulers perform poorly in terms of energy efficiency.
- MOHEFT outperforms HEFT and greenHEFT in both makespan and energy consumption.
- MOHEFT increases energy efficiency with small hits on makespan.
- Energy consumption reduced by up to 34.5% with only 2% makespan overhead.

ARTICLE INFO

Article history:
Received 2 November 2012
Received in revised form
16 May 2013
Accepted 17 July 2013
Available online 6 August 2013

Keywords:
Energy-efficient scheduling
Workflow scheduling
Multi-objective optimisation

ABSTRACT

Workflow applications are a popular paradigm used by scientists for modelling applications to be run on heterogeneous high-performance parallel and distributed computing systems. Today, the increase in the number and heterogeneity of multi-core parallel systems facilitates the access to high-performance computing to almost every scientist, yet entailing additional challenges to be addressed. One of the critical problems today is the power required for operating these systems for both environmental and financial reasons. To decrease the energy consumption in heterogeneous systems, different methods such as energy-efficient scheduling are receiving increasing attention. Current schedulers are, however, based on simplistic energy models not matching the reality, use techniques like DVFS not available on all types of systems, or do not approach the problem as a multi-objective optimisation considering both performance and energy as simultaneous objectives. In this paper, we present a new Pareto-based multi-objective workflow scheduling algorithm as an extension to an existing state-of-the-art heuristic capable of computing a set of tradeoff optimal solutions in terms of makespan and energy efficiency. Our approach is based on empirical models which capture the real behaviour of energy consumption in heterogeneous parallel systems. We compare our new approach with a classical mono-objective scheduling heuristic and state-of-the-art multi-objective optimisation algorithm and demonstrate that it computes better or similar results in different scenarios. We analyse the different tradeoff solutions computed by our algorithm under different experimental configurations and we observe that in some cases it finds solutions which reduce the energy consumption by up to 34.5% with a slight increase of 2% in the makespan.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Precedence-constrained parallel applications, also known as *workflows*, are a popular paradigm used by scientists for modelling large applications. Most of these applications have to deliver results as fast as possible and thus require parallel or distributed computation for decreasing their execution time. A major challenge in this case is to achieve the "best" *schedule* of the workflow tasks onto the available resources that minimises its execution time—a well-known NP-complete problem. Nowadays, high-performance parallel computing is available to almost any scientist or researcher. Along with their many advantages, these facilities introduce new challenges such as the considerable amount of energy required in today's data centres. Besides the green implications of energy savings, Hamilton [1] reported that the financial expenditure for the energy consumption of Amazon's data centres in a period of fifteen years accounts for 19% of its total budget and that 23% of the same budget represents the cost of maintaining the cooling infrastructure. Under these circumstances, energy efficiency is becoming an important objective for computing infrastructure providers due to its environmental and financial implications.

* Corresponding author. Tel.: +43 512 507 53284.
E-mail addresses: juan@dps.uibk.ac.at (J.J. Durillo), vlad@dps.uibk.ac.at (V. Nae), radu@dps.uibk.ac.at (R. Prodan).

0167-739X/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.07.005

Reducing the energy consumption while maintaining the quality of the service in terms of performance has therefore become a major research challenge. The existing works mainly address this issue through the use of Dynamic Voltage and Frequency Scaling (DVFS), which enables online adjustment to voltage and frequency in CMOS circuits [2–4]. By applying this technique to different CPUs in a heterogeneous parallel system, it is possible to have resources working at their highest speed, and implicitly at peak energy consumption, while executing time-critical tasks. Conversely, they will be tuned to work at a lower speed with lower energy consumption while executing non-critical tasks. In such systems, the user is confronted with a set of resources working at different speeds and with different characteristic energy consumptions.

In these circumstances, workflow scheduling must be formulated as a *multi-objective optimisation problem (MOP)* aiming at optimising two possibly conflicting criteria: *makespan* and *energy* consumption for executing the workflow. The main characteristic of MOPs is that no single solution exists that is optimal with respect to all objectives, but a set of tradeoff solutions known as *Pareto set* or *Pareto front*, depending on if we refer to the domain or co-domain of the functions to be optimised. The property of the solutions on the Pareto set/front is that they cannot be simultaneously improved with respect to all objectives. In our particular case, it is likely that no single schedule simultaneously minimises makespan and energy consumption. Instead, there will be a set of solutions with a different tradeoff between makespan and energy consumption.

In this paper, we present a holistic approach for an energy-efficient workflow scheduler for heterogeneous systems, capable of computing a set of tradeoff solutions in a single run. Our method called MOHEFT is an extension of the Heterogeneous Earliest Finish Time (HEFT) algorithm [5], one of the most popular algorithms for workflow scheduling. HEFT is based on the assumption that the execution time of each workflow task is known on each resource, which is not always realistic for highly dynamic distributed systems consisting of a large number of multi-core processors sharing resources (e.g. bus, caches) and affected by the external load caused by other concurrent tasks. To overcome this drawback, MOHEFT relies on empirical models for execution time of workflow tasks and their entailed energy consumption. These models are based on the knowledge extracted from historical executions, reflecting the behaviour of real multi-core CPUs with different levels of energy consumption depending on the number of cores used and their individual level of utilisation, and predicting it for new unseen resources. While we do look into the external load generated by different workflow tasks, we do not consider multi-tenancy in our approach, as scientific applications, including workflows, are typically highly specialised and customised applications owned by individual scientists and tuned for their particular temporary research needs.

The contributions of this paper in the area of energy-efficient scheduling are as follows.

1. *Identification of fine-grained levels of energy consumption in a multi-core CPU*. Thorough extensive experimentation, we measure the energy consumption and performance of different multi-core CPUs with different number of cores used. The experiments show that the performance of individual cores decreases with the concurrent number of cores used, while the energy efficiency increases.
2. *Use of empirical models for energy consumption and performance based on real data*. We build neural network-based empirical models for time and energy consumption of CPU-intensive tasks based on historical executions on heterogeneous sets of machines. Our approach considers external load coming from tasks of the same as the one being modelled; however, we plan in future work to extend it with training information about the types of tasks defining the external load.
3. *A multi-objective energy-efficient scheduling algorithm*. We present an algorithm capable of computing workflow schedules representing tradeoffs between energy consumption and makespan. We prove thorough an extensive experimentation that our algorithm computes solutions with the same or shorter makespan than HEFT and with lower energy consumption than HEFT and greenHEFT, an ad-hoc greedy algorithm for energy optimisation. MOHEFT also outperforms the most popular multi-objective optimisation algorithm, NSGA-II [6], computing solutions with shorter makespan and lower energy consumption.
4. *Analysis of the impact of different workflow characteristics and different resources on the tradeoff solutions*. We carry out a thorough evaluation of our proposal under different circumstances: number of tasks, homogeneous resources in terms of speed and energy consumption, or heterogeneous resources. Our aim is to analyse the performance of MOHEFT versus other algorithms and the impact of different scenarios on the computed tradeoff solutions.

The rest of this paper is structured as follows. The next section formally describes our problem and introduces some background on multi-objective optimisation. Related works on energy modelling, multi-objective workflow scheduling and energy-efficient scheduling are analysed in Section 4. After that, we describe the modelling approach followed in this paper. Section 6 introduces MOHEFT. The evaluation of our algorithm and the analysis of the obtained results is included in Section 7. Finally, we present the main conclusions and future work in Section 8.

2. Formalism

Our problem consists in scheduling a workflow application's tasks on a set of available heterogeneous resources in such a way that the makespan and the energy consumption of its execution are minimised. We introduce in the remainder of this section a simple but realistic formalism that defines the workflow, resource environment, and the metrics targeted.

2.1. Workflow application

We model a *workflow application* as a directed acyclic graph (DAG), $W = (A, D)$ consisting of n tasks or activities: $A = \bigcup_{i=1}^{n} \{A_i\}$, interconnected through control flow and data flow dependences, D, defined as

$$D = \left\{ \left(A_i, A_j, Data_{ij}\right) \mid \left(A_i, A_j\right) \in A \times A \right\},$$

where $Data_{ij}$ represents the size of the data needed to be transferred from activity A_i to activity A_j. In the remainder of this paper, we use the terms activity and task interchangeably. We use $pred\,(A_i) = \{A_k | \forall\,(A_k, A_i, Data_{ki}) \in D\}$ to denote the set of *predecessors* of activity A_i (i.e. A_i's parents to be completed before starting it). Every activity $A_i \in A$ is characterised by its length (or workload) measured for example in the total number of instructions, which affects its execution time and the energy consumption.

2.2. Resource environment

We assume a hardware platform consisting of m heterogeneous resources $R = \cup_{j=1}^{m} R_j$. Each resource $R_j \in R$ is described by a set of nine different characteristics which influence the number of machine instructions per second it is able to process, and the energy it consumes during this. A brief description of these nine characteristics is summarised in Table 1. We use $sched(A_i)$ to denote the resource on which activity A_i is scheduled to be executed.

Table 1
Resource characteristics.

Characteristic	Description
Technology (nm)	Dimension of the CPU lithography (e.g. 32 nm)
Architecture (bits)	CPU architecture (e.g. 32 or 64 bits)
Min frequency (GHz)	Minimum processor frequency
Frequency (GHz)	Nominal processor frequency
Cache size (KB)	Level of cache memory size
Cache sharing	Number of cores sharing the last level cache
Cores	Number of cores on the CPU
Threads	Number of concurrent hardware threads[a]
TDP	CPU thermal design point[b]

[a] Twice the number of physical cores for Hyper-Threading.
[b] Theoretical maximum heat dissipation requirement for not exceeding the maximum junction temperature; also a rough indicator of the power consumption class of the CPU.

2.3. Makespan

For computing the workflow makespan, we first define the *completion time* $T_c^{(A_i)}$ of an activity A_i on resource $R_j = sched(A_i)$ as the maximum completion time of its predecessors, including their data transfers to R_j, plus its own total execution time $t_{(A_i,R_j)}$:

$$T_c^{(A_i)} = \begin{cases} t_{(A_i,R_j)}, & pred(A_i) = \emptyset; \\ \max\limits_{A_p \in pred(A_i)} \left\{ T_c^{(A_p)} + \dfrac{Data_{pi}}{b_{pj}} \right\} & \\ \quad + t_{(A_i,R_j)}, & pred(A_i) \neq \emptyset, \end{cases} \tag{1}$$

where $t_{(A_i,R_j)}$ is the *computation time* of activity A_i on R_j, $Data_{pi}$ is the number of bytes transferred between A_p and A_i, and b_{pj} is the bandwidth of the communication link between $sched(A_p)$ and R_j. If the activities A_p and A_i are scheduled on the same machine, i.e. $sched(A_p) = sched(A_i) = R_j$, we assume that no data transfer is necessary (i.e. $b_{pj} = \infty$). We do not assume in our model any specific type of network connection between resources which may reside in the same local area network or can be distributed in the Internet.

Finally, we compute the workflow makespan as the maximum completion time of all its n activities:

$$T_W = \max_{i \in [1,n]} \left\{ T_c^{(A_i, sched(A_i))} \right\}. \tag{2}$$

2.4. Energy consumption

We represent the total energy E_W consumed by a workflow execution as the sum of the energy E_D consumed for all data transfers and the energy E_C consumed for executing all its computational activities:

$$E_W = E_D + E_C. \tag{3}$$

Given a workflow schedule, we define E_D as follows:

$$E_D = \sum_{\substack{(A_i,A_j,Data_{ij}) \in D \wedge \\ sched(A_i) \neq sched(A_j)}} \mathcal{E}_{ij} \cdot Data_{ij}, \tag{4}$$

where $Data_{ij}$ is the number of bytes transferred between A_i and A_j, and $\mathcal{E}_{ij}$ is a characteristic value representing the energy expended for transferring one byte of data between $sched(A_i)$ and $sched(A_j)$, which neglects the energy consumed by the external networking equipment.

Similarly, we define the energy E_C consumed by the computational activities of a workflow as the sum of the energy $E_{R_j}^{(A_i)}$ consumed by resource $R_j = sched(A_i)$ for executing activity A_i:

$$E_C = \sum_{i=1}^{n} E_{R_j}^{(A_i)}, \tag{5}$$

where n is the total number of workflow activities.

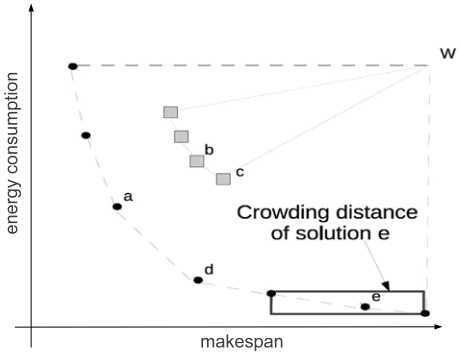

Fig. 1. Comparison of multi-objective tradeoff solutions.

We further define the energy $E_{R_j}^{(A_i)}$ consumed by resource $R_j = sched(A_i)$ for executing each activity A_i as follows:

$$E_{R_j}^{(A_i)} = P_{R_j}^{(s)} \cdot t_{(A_i,R_j)} + \int_{T_s^{(A_i)}}^{T_c^{(A_i)}} P_{R_j}^{(d)}(t) \cdot \mathrm{d}t, \tag{6}$$

where $P_{R_j}^{(s)}$ refers to the static power consumption of the resource R_j in idle state, $P_{R_j}^{(d)}(t)$ represents the additional dynamic power consumption of the same resource stemming from the computations scheduled at time instance t, $t_{(A_i,R_j)}$ is the computation time of A_i on R_j, $T_c^{(A_i)}$ its completion time, and $T_s^{(A_i)} = T_c^{(A_i)} - t_{(A_i,R_j)}$ its start time. While this equation describes in general the energy consumed by an activity on a dedicated resource, we will model in Section 5.1 its dynamic fraction to consider the impact of external load on multi-core processors (see Eq. (7)).

2.5. Problem definition

Given a workflow $W = (A, D)$, our goal is to approximate the *Pareto front* as a set of K workflow schedules $\bigcup_{k=1}^{K} wf\_sched_k(W)$, where every *workflow schedule* $wf\_sched_k(W) = \left\{ \bigcup_{\forall A_i \in A} sched_k(A_i) \right\}$, $\forall k \in [1, K]$ is *Pareto-optimal* with respect to makespan and energy consumption. We give in the next section the background information on multi-objective optimisation theory required for computing and evaluating the Pareto optimality of workflow schedules.

3. Multi-objective optimisation

We introduce in this section a few concepts from the *multi-objective optimisation* theory for a better understanding of this work. We assume without loss of generality that minimisation is the goal for all objectives.

A multi-objective optimisation problem can be formally defined as finding all vectors $\vec{x} = [x_1, x_2, \ldots, x_n]$ which minimise the vector function $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_o(\vec{x})]$. For our problem, $n = |A|$ represents the cardinality of the task set A, and the i-th component of a solution $\vec{x}$ the resource where task A_i is scheduled: $sched(A_i) = x_i, x_i \in R$. For our bi-objective optimisation case, we have $o = 2$, where $f_1(\vec{x})$ is the makespan and $f_2(\vec{x})$ the energy.

As it is not always possible to find a solution that minimises both makespan and energy consumption simultaneously, we introduce the concept of *dominance*. A solution x_1 dominates a solution x_2 if the makespan and energy consumption of x_1 are smaller than those of x_2. Conversely, two solutions are said to be non-dominated whenever none of them dominates the other (i.e. one is better in makespan and the other in energy consumption). In Fig. 1, the

solution labelled a dominates the one labelled b because it has better makespan and energy consumption. Similarly, a dominates c. Meanwhile, a and d are non-dominated because a is better in makespan, but d is better in energy consumption. The set of optimal non-dominated solutions is called Pareto front (the trend line containing the a, d, and e solutions) and represents a set of tradeoff solutions among the different objectives. Every solution in this set represents a different mapping of the workflow tasks with different makespan and energy consumption.

A Pareto front can be seen as a tool for decision support and preference discovery. Its shape can provide insight to researches or scientists, allowing them in many cases to explore the possible space of non-dominated solutions with certain properties, and possibly revealing regions of particular interest which cannot be seen until the Pareto front is computed and visualised. This way, the users do not have to set their preferences before finding a solution; instead the preferences are discovered afterwards. Nevertheless, not all Pareto fronts are valid. A good Pareto front provides accuracy (solutions close to optimal) and diversity (uniformly cover all possible ranges of optimal solutions). A way of measuring the quality of a set of tradeoff solutions X is the *hypervolume* $HV(X)$ [7], representing the area enclosed between the points in X and a reference point W (see Fig. 1), usually selected as the maximum objective value (e.g. highest makespan and energy). The better and the more diverse the points in X are, the higher the $HV(X)$. In Fig. 1, the set with the solid round points is better than the set with the squared solutions because the area enclosed within the dashed lines is larger than the one represented by the dotted lines.

4. Related work

We review the related work in three fields touched by our approach: energy modelling, multi-objective workflow scheduling, and energy-efficient scheduling.

4.1. Energy modelling

The problem formulation presented in Section 2.5 relies on two models: one for the number of instructions a resource is able to process every second and another one for the energy consumed by a resource at an instant t. Here we review the existing models and analyse their applicability to our problem.

Many existing approaches [5,8–12] employ a simplified model for determining the time required by a resource for completing an activity. This model assigns a fixed speed to every resource quantified as the number of machine instructions executed per second. The time required for computing an activity on a resource is then approximated as the length of the task divided by the speed of that resource. Although this model is valid for single core machines, it does not properly describe current distributed systems composed of heterogeneous multi-core machines allowing the execution of independent tasks in different cores. Due to various reasons such as cache sharing between cores and data bus contention, such scenarios always introduce a time overhead for concurrently executing these multiple independent tasks.

Regarding the existing energy consumption models [13–18], they all consider only two levels of energy consumption in a machine corresponding to its idle and full-load states. These models, however, do not properly reflect the current energy-aware multi-core architectures. These systems are characterised by a multitude of distinct power consumption levels depending on the number of cores and their level of utilisation. It is also worth noting that, as a consequence of having shared subsystems between cores, these levels often do not follow a linear utilisation—energy consumption model (i.e. the additional energy incurred by using more cores diminishes with the total number of cores used).

4.2. Multi-objective workflow scheduling

Most of the existing works in multi-objective workflow scheduling combine the different optimisation criteria in a single optimisation goal. Usually, this combination consists of a weighted aggregation of these criteria, where the weights are aimed at expressing user preferences over the objectives. The main difference among existing works relies on the way these preferences are expressed. For example, in [8] reliability (in terms of resource failures) and makespan are combined using a weight vector provided by the user. The same objectives are optimised in [19,11]. In the former, both objectives are given the same importance in the formulation (so the same preference for the goals is assumed), while in the latter the preferences are introduced by means of constraints over the different objectives.

The idea of imposing constraints over the different objectives is also exploited by other works like [20] for optimising makespan and economic cost in utility Grids. Constraints impose a desired quality of service that every objective must satisfy. Once these constraints have been set, the idea is to optimise one preferred objective within the established constraint. Afterwards, several modifications are applied with the aim of improving the solution with respect to a second preferred objective, as long as the constrains are not violated for any of them. This last step is repeated for all objectives. A general disadvantage of preference-based approaches is that the computed solution depends on the combination of the multiple objectives, which is done a priori and without any information about the problem being solved. Thus, if the aggregation function does not capture the user preferences in an accurate way, the computed solution may not be satisfactory for the solved problem. In these approaches, each distinct workflow requires specific and independent constraints and thus, any change to the workflow (or task) will require these constraints to be recomputed. Moreover, the use of non-achievable constraints is prone to failures. Concretely, a user with no knowledge of the workflow's and resources' characteristics might constrain the execution to 100 W h, while the most energy-efficient execution would require 150 W h. Finally, these approaches provide only one solution at a time instead of a complete Pareto front. Our approach computes the whole Pareto set of solutions for each workflow, opening opportunities for a wide range of analyses and allowing the user to accurately select the solution with the preferred makespan-to-energy consumption tradeoff.

Only few approaches compute the whole set of tradeoff solutions among the different objectives. Some of these works address the problem by applying evolutionary algorithms (e.g. genetics, firefly). For example, [12] optimises the makespan and cost of executing tasks in a cloud system, while [13,21] optimise for makespan and energy consumption. While evolutionary methods are capable of computing high quality solutions, they require significant computation time. One possible approach to overcome these difficulties consists in initialising these algorithms with previously computed good solutions, as presented in [12]. A second alternative to compute the whole set of tradeoff solutions is to apply list-based heuristics. The application of these strategies for multi-objective workflow scheduling is investigated in [22,23] for optimising makespan and cost of executing workflows with no regard to energy consumption.

4.3. Energy-efficient scheduling

There exists extensive literature in energy-efficient scheduling, most of it applying the DVFS technique. One line of research consists in the application of genetic algorithms [13–15]. While applying different variants of genetic algorithms, these algorithms have in common the use of individuals representing the mapping of

Table 2
Resource configuration for modelling time and energy consumption of the povray task (all are 64 bit architectures; subsystems such as memory and peripherals are omitted).

CPU						CPU count	PSU count[a]	$P^{(s)}_{R_j}$ (W)	No. of machines
Model	Tech. (nm)	Freq. (GHz)	Cache (KB)	Cores	TDP (W)				
AMD Opteron 8356	65	2.3	2 048	4	95	8	4	494	2
AMD Opteron 6168	45	1.9	12 288	12	115	2	1	176	2
Intel Xeon X5650	32	2.66	12 288	6 (HT[b])	95	2	2	200	2
Intel Xeon E7-4870	32	2.4	30 720	10 (HT)	130	4	2	474	1
AMD Opteron 880	90	2.4	2 048	2	95	4	2	424	1
AMD Opteron 885	90	2.6	2 048	2	95	8	4	836	1

[a] Power supply units, abbreviated here "PSU".
[b] Intel's proprietary Hyper-Threading technology, abbreviated here "HT".

tasks onto machines and the voltage at which each machine should operate. Alongside genetic algorithms, greedy heuristics have also been of interest [16,17]. A common disadvantage of all these works is that the machines' voltage is not changed during the workflow execution, but maintained for the whole execution once it has been set to a specific level. From our point of view, there is room for researching in this area by proposing techniques capable of dynamically changing the voltage during the execution of applications. A different line of research [18] is to analytically compute the minimum speed at which every core of a system should operate in order to meet an imposed time deadline. Then, a mapping of the tasks onto resources is computed accordingly. Authors showed that the energy consumption of the whole system can be reduced by having processors working at the minimum required voltage.

Among the techniques which do no use DVFS, a common strategy is to power-down or turn off to sleep mode resources that are not used [24], removing therefore the static energy consumed by them in idle mode. The use of list-based heuristics which do not tune the voltage of the CPU is analysed in [25] for computing energy-efficient schedules. A game theoretic approach for computing a Pareto-optimal solution is investigated in [26]. The drawback of the last two approaches is that only one schedule is computed at a time, thus requiring several runs for generating different tradeoff solutions.

To the best of our knowledge, our work is the first pure multi-objective approach capable of computing a set of tradeoff solutions for optimising makespan and energy consumption of scientific workflows. Our method also incorporates power-down techniques by accounting only for the energy consumed by the resources doing real computation. Although we do not use DVFS for reducing the energy consumed by a CPU, it can be combined in a hybrid approach.

5. Energy and execution time modelling

5.1. Applicability of existing models

In this section we show that the existing models reviewed in Section 4.1 are not applicable in the context of modern multi-core architectures. For this purpose, we conduct an extensive experiment aiming to prove that: (1) the performance of individual cores is impacted by the workload of the other cores; (2) the energy consumption of a multi-core CPU may vary among a multitude of different levels at any instant t; and (3) the energy consumption overhead for powering on a core decreases with the number of already powered on cores.

We consider in a real workflow task called povray belonging to the Persistence Of Vision Raytracer (POV-Ray), a real workflow application which is presented in detail in Section 7. This task renders a set of frames (i.e. images) from a three-dimensional scene descriptor file and it is the most time-consuming part of the workflow. Although parallel implementations of this task are publicly available, we consider in our experiments the sequential implementation provided at http://www.povray.org/ since our goal is to characterise the performance and energy consumption of individual CPU cores.

Table 2 describes the resources used for running these experiments. The external load consists in the simultaneous execution of several other povray instances, which we varied from one to the maximum number of available cores on each architecture. We executed the povray task using different number of frames, ranging from 10 to 1000, and averaged the time and energy consumption required by a single frame for each different level of external load. We measured the time and energy metrics using an instrumentation tool written in Python and C which retrieves online measurements from multiple LAN-enabled Voltech PM1000+[1] power measurement devices connected to the machines.

Fig. 2 shows the time required by two of the considered resources when rendering a single frame under increasing external load conditions. We observe on both machines a significant performance overhead introduced by the external load. Concretely, the time needed for computing one frame considerably increases when more cores are concurrently utilised. For the Intel architecture, we compute a slowdown of 34.2% in the case of running simultaneously 40 instances of the povray activity (i.e., 40 cores being used simultaneously) compared to no external load (i.e., only one core doing computation). An additional slowdown of 38.1% from 40 to 80 threads of external load is due to the additional overhead of the Hyper-Threading technology. For the AMD architecture, we compute a slowdown of 36.9% between no and full external load conditions (i.e. executing one and 32 instances of the povray activity, respectively).

Fig. 3 presents the variation in energy consumption of one CPU core with increasing external load as in the previous case. The decrease in energy consumption per core with increasing external load is due to the fact that the cores share a set of common subsystems (e.g. cache memory, memory bus, memory controller) that consume relatively the same amount of energy when utilised by any number of cores, as they quickly reach saturation state. For clarification, we present a concise analytical formulation of this statement. We measured the dynamic fraction of the energy consumption introduced in Eq. (6):

$$E_{dyn} = \int_{T_s^{(A_i)}}^{T_c^{(A_i)}} P^{(d)}_{R_j}(t) \cdot \mathrm{d}t = c \cdot E_{core} + E_{shared}, \tag{7}$$

where c represents the number of active cores, E_{core} represents the energy consumption of the active core running A_i, and E_{shared} is the energy consumption of all shared subsystems in their activated states. For computing the energy expenditure per core, we normalised this metric by the number of active cores (also by considering Hyper-Threading units):

$$E^{(unitary)}_{dyn} = \frac{E_{dyn}}{c} = E_{core} + \frac{E_{shared}}{c}. \tag{8}$$

[1] http://www.voltech.com/products/poweranalyzers/PM1000.aspx.

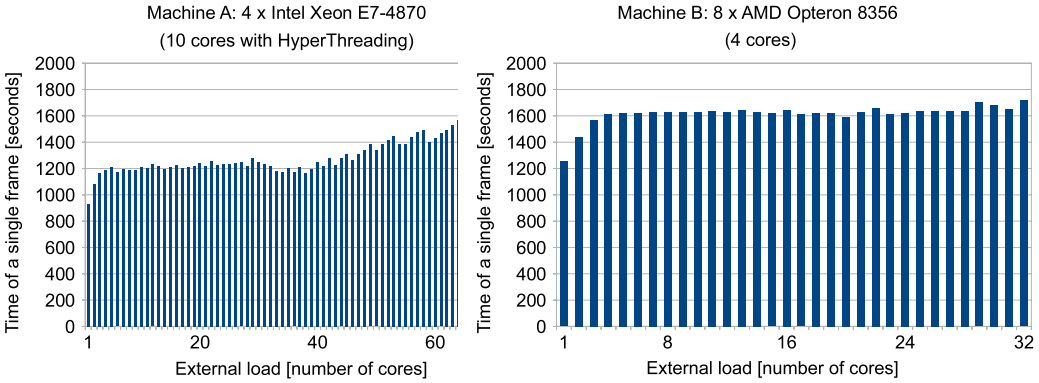

Fig. 2. Time for rendering one frame on a single core of two multi-core machines (AMD and Intel with Hyper-Threading) with increasing external load until reaching machine saturation.

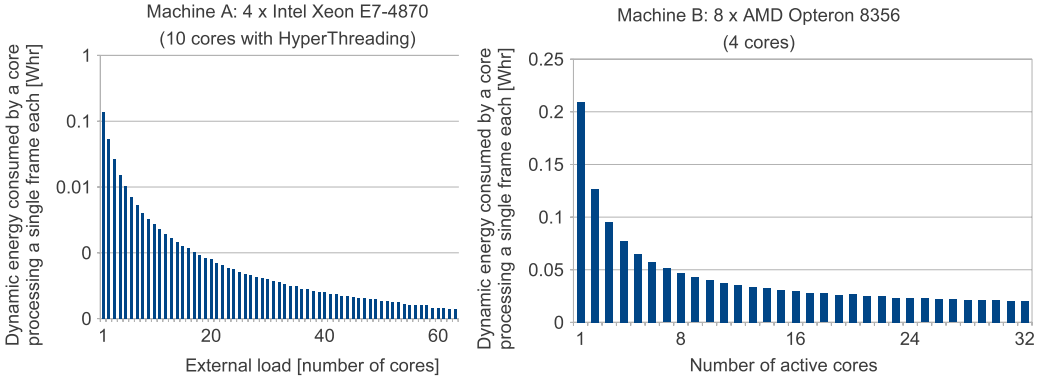

Fig. 3. Energy consumption of a single core of two multi-core machines (AMD and Intel with Hyper-Threading) with increasing external load until it reaches machine saturation.

It is obvious from this formulation that the per-core energy consumption is a function generically expressed as

$$E_{dyn}^{(unitary)} = f\left(\frac{1}{c}\right) + \mathcal{C}, \tag{9}$$

where $\mathcal{C}$ represents a constant. This accurately describes the variation in the energy consumption seen in Fig. 3 and confirms our assumption that multi-core systems exhibit more than two power consumption levels. In light of these findings, we affirm that the existing models for performance and energy consumption of multi-core machines must be adjusted, or new ones need to be researched.

5.2. Neural network-based predictor

Scientific workflows can be composed of a wide range of activities stressing different subsystems of the resources running them: some CPU-centric (e.g. performing mostly mathematical computations), others input/output-centric (e.g. mostly reading and writing data to a disc or to the network). Depending on the different operations they perform, and implicitly the subsystems they stress, activities determine different levels of power consumption of a resource. In between the previously identified CPU-intensive and input/output-intensive activities there exists additionally a wide spectrum of different types of activities, stressing the resources' subsystems in different proportions, having consequently different energy consumptions and running times. Under these circumstances, it is reasonable to affirm that the resource performance and the energy consumption models should not only be based on the underlying hardware architecture, but also on the activity type itself.

Currently there are two approaches to designing such models: theoretical and empirical models. The main drawback of theoretical models is the complexity to derive them. To the best of our knowledge, no valid theoretical model for either energy prediction or execution time has been proposed so far. Empirical models are based on modelling the knowledge extracted after measuring the energy consumption and execution time of different tasks on different resources. The main drawback of this latter approach is the difficulty of completely covering the infinite configuration space defined by the activity types and resource types (i.e. it is impossible to instrument the executions of all activity types on all resource types). One alternative for overcoming this drawback is the use of predictors for estimating the execution time and energy consumption of workflow activities on unknown (i.e. not previously evaluated) resources.

In this paper we investigate the use of neural network predictors [27] for workflow activity execution time and energy consumption estimations. Our approach is currently limited to CPU-intensive tasks (`povray`-like) and external load coming from tasks of the same `povray` type. In the future we will extend our model by training it with information about the type of activities defining the external load. To each activity type we associate two neural networks: one for estimating its execution time on any (activity configuration – input size – resource) combination, the other for providing the corresponding energy consumption estimations. We consider this approach to be a valid alternative in the context of scientific workflows, where the type and number of activities is limited and a priori known. We designed a predictor based on a Multi-Layer Perceptron (MLP) with one hidden layer, which requires as input the different characteristics of the machine (as described in Table 1) the activity will be executed on, along with the

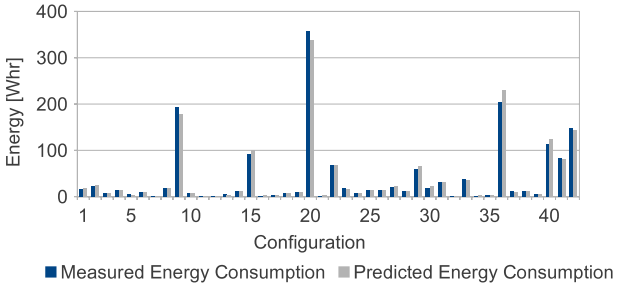

Fig. 4. Neural network prediction versus real energy consumption in various configurations.

input size (e.g. the number of frames to render in the case of the `povray` task analysed in Section 5.1) and number of cores already in use on the target machine. The output is either the estimated execution time $t_{(A_i,R_j)}$ of its associated task A_i on the given resource R_j in the case of the execution time predictor, or the energy expended by the resource to execute the activity $E_{R_j}^{(A_i)}$ in the case of the energy predictor.

To validate our approach we use data collected from executions of the real `povray` workflow task (analysed in Section 5.1) on our diverse set of resources (shown in Table 2) for neural network training and cross-validation. We divide the available data into two subsets, the *training set* comprising 85% of the data, utilised in the training process, and the *testing set* (the remaining 15%) for validation. The output of the network when presenting entries in this last set is compared with the real measurements. The accuracy of our models is then determined by the deviation of the differences between these two values.

We conducted 20 experiments consisting of network training–testing cycles, using in each instance a different selection of the training and testing sets, and we measured an average prediction accuracy of 97% and a minimum of 95%. The accuracy of our method can be visually assessed in Fig. 4. In particular, this figure presents a sample of a validation trace comprising real measured energy consumption values and the corresponding estimation values by the trained neural network predictor. The values represent the energy expended for rendering 1000 frames for different configurations consisting of various machines with different external loads (in no particular order).

6. MOHEFT: multi-objective heterogeneous earliest finish time

In this section we describe our proposed multi-objective scheduling algorithm for computing a set of tradeoff solutions (instead of a single one) as an extension to the HEFT list scheduling algorithm. For a better understanding, we start by describing the mono-objective version of the algorithm and extend it afterwards for dealing with multiple objectives. Finally, we present also green-HEFT, a heft-like heuristic for minimising the energy consumption.

6.1. HEFT: heterogeneous earliest finish time algorithm

HEFT [5] is a popular list-based heuristic scheduling algorithm for optimising the makespan [5] in workflow applications, described in pseudocode in Algorithm 1. The method consists of two phases: ranking and mapping. In the ranking phase (line 2), we compute the order in which the activities are being mapped using the B-rank metric representing the distance of the activity to the end of the workflow. The idea of this ranking is to execute first the tasks with most number of successors. Further details about how to sort the tasks can be found in [5]. Once the execution order

Algorithm 1 HEFT algorithm.

Require: $W = (A, D)$, $A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $R = \bigcup_{i=1}^{m} R_i$ ▷ Set of resources
Ensure: $sched(W) = \bigcup_{i=1}^{n} sched\,(A_i)$ ▷ Workflow schedule
1: **function** HEFT(W, R)
2: $B \leftarrow$ B-RANK(A) ▷ Order the tasks according to B-rank
3: $sched(W) \leftarrow \emptyset$ ▷ Initialise workflow schedule with empty set
4: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the n ranked tasks in S
5: $T_c^{(\min)} \leftarrow \infty$
6: **for** $j \leftarrow 1, m$ **do** ▷ Iterate over all m resources
7: $T_c^{(S_i)} \leftarrow \max_{A_p \in pred(S_i)} \left\{ T_c^{(A_p)} + \frac{Data_{pi}}{b_{pj}} \right\} + t_{(B_i,R_j)}$ ▷ Compute completion time of S_i
8: **if** $T_c^{(B_i)} < T_c^{(\min)}$ **then** ▷ Save the minimum completion time
9: $T_c^{(\min)} \leftarrow T_c^{(B_i)}$
10: $R_{\min} \leftarrow R_j$
11: **end if**
12: **end for**
13: $sched(W) \leftarrow sched_W \cup (B_i, R_{\min})$ ▷ Schedule the task on $R_{\min}$
14: **end for**
15: **return** $sched(W)$
16: **end function**

is determined, the second phase assigns every task to the resource where it completes earliest (lines 4–14) in the order computed in the first phase. For every task (line 4), its completion time on every resource (line 6) is computed (line 7), and is finally mapped onto the resource where it finishes earliest (line 13). After all tasks have been mapped, the workflow schedule is returned (line 15).

6.2. MOHEFT: multi-objective heterogeneous earliest finish time algorithm

As described before, HEFT builds a solution by iteratively mapping tasks onto resources. That mapping is aimed at minimising the completion time of every task; therefore it chooses in every iteration only the resource which minimises this goal. When multiple objectives are considered for computing a set of tradeoff solutions, we must allow the creation of several solutions at the same time instead of approximating a single one. We achieve this by scheduling each task to all resources that provide a tradeoff between the considered objectives.

The MOHEFT algorithm extends HEFT with these ideas, as depicted in pseudocode in Algorithm 2. The only additional input parameter of MOHEFT is the size of the set of tradeoff solutions K. Similar to HEFT, our method ranks first the tasks using the B-rank (line 2). Then, instead of creating an empty solution as HEFT does, it creates a set S of K empty solutions (lines 3–5). Afterwards, the mapping phase begins (lines 6–15) in which MOHEFT iterates first over the list of ranked tasks (line 6). The idea is to extend every solution in S by mapping the next task onto all possible resources creating m new solutions stored in a temporal set S' which is initially empty (line 7). For creating these new solutions, we iterate over the set of resources (line 8) and the set S (line 9), and add the new extended intermediate schedules to the new set S' (line 10). This strategy results in an exhaustive search if we do not include any restrictions. We therefore only save the best K tradeoff solutions from the temporary set S' to the set S (lines 13–14). We consider that a solution belongs to the best tradeoff if it is not dominated by any other solution and if it contributes to the diversity of the set. For this last criterion, we employ the *crowding distance* defined in [6] and graphically depicted in Fig. 1, which gives a measure of the area surrounding a solution where no other tradeoff solution is placed. Our criterion is to prefer solutions with a higher crowding distance, since the set will represent a wider area of different tradeoff solutions. After assigning all the tasks (line 16), the algorithm returns the set of K best tradeoff solutions.

Algorithm 2 MOHEFT algorithm.

Require: $W = (A, D), A = \bigcup_{i=1}^{n} A_i$ ▷ Workflow application
Require: $R = \bigcup_{i=1}^{m} R_i$ ▷ Set of resources
Require: K ▷ Number of tradeoff solutions
Ensure: $S = \bigcup_{i=1}^{K} sched(W)$ ▷ Set of K tradeoff workflow schedules
1: **function** MOHEFT(W, R, K)
2: $B \leftarrow$ B-RANK(A) ▷ Order the tasks according to B-rank
3: **for** $k \leftarrow 1, K$ **do** ▷ Create K empty workflow schedules
4: $S_k \leftarrow \emptyset$
5: **end for**
6: **for** $i \leftarrow 1, n$ **do** ▷ Iterate over the n ranked tasks
7: $S' \leftarrow \emptyset$
8: **for** $j \leftarrow 1, m$ **do** ▷ Iterate over all m resources
9: **for** $k \leftarrow 1, K$ **do** ▷ Iterate over all K tradeoff schedules
10: $S' \leftarrow S' \cup \{S_k \cup (B_i, R_j)\}$ ▷ Add new mapping to all intermediate schedules
11: **end for**
12: **end for**
13: $S' \leftarrow$ SORTCROWDDIST(S', K) ▷ Sort according to crowding distance
14: $S \leftarrow$ FIRST(S', K) ▷ Choose K schedules with highest crowding distance
15: **end for**
16: **return** S
17: **end function**

6.3. GreenHEFT: a list-based heuristic for minimising energy consumption

For the sake of comparisons, we also designed a mono-objective heuristic for optimising energy consumption only, called *greenHEFT*. The new heuristic consists as HEFT of a ranking and a mapping phase. The former phase is as in HEFT based on the B-rank metric. The differences with HEFT arise in the second phase. While HEFT assigns tasks to the resources where they finish earliest, greenHEFT maps the tasks onto resources where their execution entails the lowest energy consumption.

6.4. MOHEFT complexity

Given a set of n activities and m resources, the computational complexity of HEFT (and greenHEFT) is $O(n \cdot m)$. MOHEFT only introduces two main differences with respect to HEFT: the creation of several solutions in each iteration of the algorithm, and the possibility of considering resources providing a tradeoff solution. These two modifications only require an additional loop in MOHEFT (see Algorithm 2, lines 9–11). Considering that the set of tradeoff solutions is K, the extra loop in MOHEFT performs only K iterations, rendering a complexity of $O(n \cdot m \cdot K)$. Usually, the number of tradeoff solutions is a constant much lower than n and m. For example, a workflow can be composed of thousands of tasks and the set of tradeoff solutions can be accurately represented with tens of solutions. Thus, the complexity can be approximated as almost $O(n \cdot m)$, as in HEFT (or greenHEFT).

7. Evaluation

We run experiments for evaluating the MOHEFT solutions, aiming at

- demonstrating that they are at least as good as those delivered by the mono-objective HEFT for makespan and greenHEFT for energy;
- demonstrating that MOHEFT computes higher quality tradeoff solutions than classical multi-objective optimisation algorithms;
- analysing the results of MOHEFT for
  - workflows with different shapes and number of activities;
  - reduced number of resources;
  - different types of resources in terms of clock frequency and static energy consumption;
  - workflows composed of different activity types.

7.1. Experimental setup

In this section we describe the experimental setup used in our experiments in terms of workflow applications and resource infrastructure.

7.1.1. Workflows

We conduct our evaluation using a real scientific workflow called the Persistence Of Vision Raytracer (POV-Ray) [28]. This workflow application is available as a free tool for creating three-dimensional graphics, and it is known to be a time and resource consuming process used not only by hobbyists and artists, but also in biochemistry research, medicine, architecture and mathematical visualisation. The POV-Ray workflow is composed of three different activities: `povray` which renders a set of frames (i.e. images) from a three-dimensional scene descriptor file, `png2yuv` which merges the resulting files into a raw `YUV` video, and `ffmpeg` which encodes the raw video in the `MPEG` video format. Additionally, for a more complete evaluation of the proposed algorithm, we also generate synthetic workflows with two variable characteristics: workflow shape and activity characteristics.

Workflow shape. We consider four types of synthetic workflows with different shapes and number of activities using the workflow generator described in [29]. The length of each activity is given by the number of frames generated by sampling a random uniform distribution. The different generated types are

- *Type*-1, containing a high number of independent activities;
- *Type*-2, containing a high number of independent activities, each having one successor and one predecessor;
- *Type*-3, containing a low number of independent activities such that at most two activities can be run in parallel;
- *Type*-4, alternating workflow regions with a high number of independent activities with regions with a low number of independent activities.

Activity characteristics. We further consider a wide range of synthetic activities differentiated by the ratio between their CPU time (exclusively used) and their I/O time (i.e. reading/writing from/to disc, or network). For example, a ratio denoted as 80%:20% indicates that the activity spends 80% of its execution by exclusively using the CPU and the remainder of 20% performing I/O operations. We generated these activities based on instrumentation data collected from the real `povray` workflow activity, used as a template for activities with a (100%:0%) CPU:I/O ratio, along with time and energy measurements we performed for network transfers and disc operations. The POV-Ray workflow can be accurately characterised through these two parameters as having a *Type*-1 shape and a (100%:0%)-type dominant activity. Thus, we present for simplification the POV-Ray execution results along with the synthetic workflows of its corresponding type.

7.1.2. Resources

We developed a synthetic resource generator which takes as input 50 distinct CPU specifications (as provided by their manufacturers) and generates complete resource setups consisting of multiple clusters comprising multiple nodes, each node being a multi-core machine. Each synthetic machine is defined by the metrics presented in Table 1, along with other characteristics such as the power consumption in idle state $P_{R_j}^{(s)}$ and the amount of installed memory. The resource generator can be configured to generate values for any of the resulting machines' characteristics according to a given distribution. For example, in the case of a targeted distribution frequency with 1.8 GHz minimum, 2.0 GHz median, and 2.6 GHz maximum, it will generate a set of resources whose CPU frequencies vary between 1.8 and 2.6 GHz, with a

Table 3
The most relevant characteristics of the generated resource setup; the statistical profile of each metric is presented as a *(minimum; median; maximum)* triplet.

Setup	Machines	Cores	Freq. (GHz)	Cache (MB)	RAM (GB)	$P_{R_j}^{(s)}$ (W)
RS1	100	(1; 2; 8)	(1.6; 3.1; 4.0)	(3; 12; 30)	(4; 24; 256)	(36; 426; 872)
RS2.LF	100	(6; 6; 16)	(1.6; 1.7; 1.7)	(6; 18; 18)	(8; 40; 128)	(31; 88; 365)
RS2.MF	100	(2; 8; 12)	(2.3; 2.4; 2.7)	(4; 15; 24)	(8; 40; 128)	(46; 458; 801)
RS2.HF	100	(2; 8; 8)	(3.0; 3.1; 3.5)	(3; 20; 20)	(8; 40; 128)	(35; 476; 812)
RS3.LP	100	(4; 6; 16)	(1.6; 3.3; 4.0)	(8; 16; 18)	(4; 40; 128)	(23; 38; 66)
RS3.MP	100	(4; 6; 16)	(1.6; 3.0; 4.0)	(8; 16; 18)	(4; 56; 128)	(103; 117; 146)
RS3.HP	100	(4; 6; 16)	(1.6; 3.0; 4.0)	(8; 16; 18)	(4; 48; 128)	(403; 417; 446)

Table 4
The experimental space.

Section	Resources		Workflows		Focus of the investigation
	Setup	Size[a]	Characteristic[b]	Size	
7.2	RS1	$>10 \cdot n$	(100%:0%)	20–200	MOHEFT vs. HEFT/greenHEFT
7.3	RS1	$>10 \cdot n$	(100%:0%)	20–200	MOHEFT vs. NSGA-II
7.4	RS1	$>10 \cdot n$	Uniform	20–200	Workflow size impact
7.5	RS1	10%..75%	(100%:0%)	200	Resources number impact
7.6	RS2.*	$>10 \cdot n$	(100%:0%)	200	Resources frequency impact
7.7	RS3.*	$>10 \cdot n$	(100%:0%)	200	Static power impact
7.8	RS1	$>10 \cdot n$	(100..10%:0..90%)	200	Activity type impact

[a] n denotes the total number of activities; $x\%$ represents a setup with only $x\%$ of the resources utilised in Pareto-optimal solutions.
[b] $(x\% : y\%)$ denotes activities with a ratio of $x\%$ computation to $y\%$ I/O operations.

higher agglomeration around the 2.0 GHz value. We generated seven resource setups as outlined in Table 3: one RS1 with the machine characteristics as uniformly distributed as possible but prioritising a uniform distribution of the CPU frequency, three with emphasis on resources with low (RS2.LF), medium (RS2.MF) and high (RS2.HF) CPU frequencies, and three with emphasis on resources with low (RS3.LP), medium (RS3.MP), and high (RS3.HP) idle power consumption $P_{R_j}^{(s)}$. We computed the execution time and energy consumption of each workflow activity on each resource using the empirical neural network models developed in Section 5.2.

Table 4 summarises the coverage of our evaluation space consisting of various resource setups and workflow types. We define the resources through a tuple comprising one of the resource setups presented in Table 3 and the size representing the number of cores utilised. We define a workflow setup through the activity characteristics and the workflow size (i.e. number of activities). In each experiment we employ all four workflow shapes defined in Section 7.1.1.

7.2. Experiment-1: MOHEFT versus HEFT and greenHEFT

In the first experiment, we compare the results computed by MOHEFT with the mono-objective versions for optimising makespan (HEFT) and energy (greenHEFT). We consider workflows having between 20 and 200 activities and different heterogeneous distributed systems randomly selected from RS1, having between 10 and 100 resources. As MOHEFT, HEFT and greenHEFT are deterministic algorithms, we only run once each algorithm per configuration and summarise the comparative results in terms of the makespan and the energy consumption of the computed schedules in Tables 5 and 6. Each table cell shows the normalised improvement of the schedule with lowest makespan/energy consumption computed by MOHEFT over the schedule computed by HEFT/greenHEFT. The symbol—means that both algorithms compute the same results.

The results in Table 5 show that MOHEFT is at least as good as HEFT in a large number of cases, and often better. For example, MOHEFT reported a makespan 1.1% shorter than the one computed by HEFT for *Type*-1 workflows with 200 activities and 100 resources. The explanation is that HEFT is a greedy heuristic and, hence, it selects in every iteration the resource minimising the workflow makespan up to this task. Thus, HEFT takes the best local decision, but does not consider its impact on the activities yet to be scheduled. In contrast, MOHEFT builds several solutions in parallel achieving a better exploration of the search space, leading us to compute better solutions in some cases. As expected, MOHEFT computes solutions with better makespan than greenHEFT in all the evaluated cases. We observe the biggest differences for *Type*-1 workflows with the highest number of activities (0.939% shorter). For the other three workflow types, the improvements are not influenced by the workflow size.

We observe in Table 6 that MOHEFT finds workflow schedules with lower energy consumption than greenHEFT. Again, the explanation has to do with a better exploration of the search space. As expected, MOHEFT also outperforms HEFT for this objective function too. The best improvements of MOHEFT over HEFT and greenHEFT can be observed again for *Type*-1 workflows, emphasising the potential of our technique for scheduling many independent activities. As with the makespan, the improvements of MOHEFT over the other techniques for *Type*-2 and *Type*-3 workflows are independent of the number of activities and resources. For *Type*-4 workflows, however, MOHEFT achieves increasingly better results for a higher number of activities due to a larger number of parallel activities in the workflow (similar to *Type*-1).

We conclude that greedy mono-objective algorithms perform poorly in terms of energy efficiency in distributed systems with multi-core processor architectures. Overall MOHEFT always provides the best solutions for both makespan and energy objectives, matching the individual solutions provided by HEFT and greenHEFT, and in many cases even improving them.

7.3. Experiment-2: MOHEFT versus NSGA-II

In this experiment we analyse the quality of the set of tradeoff solutions computed by MOHEFT. We compare our approach with NSGA-II [6], the most popular evolutionary multi-objective optimisation algorithm, and with NSGA-II*, an improved version of it

Table 5
Makespan comparison between MOHEFT, HEFT and greenHEFT.

		Number of Resources				
		10	25	50	75	100
Workflow shape *Type*-1						
Activities	2	– / 0.711	– / 0.649	– / 0.505	– / 0.659	– / 0.659
	50	– / 0.873	– / 0.831	– / 0.691	– / 0.837	– / 0.838
	100	0.001 / 0.932	– / 0.905	– / 0.805	– / 0.908	– / 0.909
	150	– / 0.949	– / 0.926	– / 0.847	– / 0.932	– / 0.932
	200	– / 0.954	0.030 / 0.930	– / 0.862	0.012 / 0.939	0.011 / 0.939
Workflow shape *Type*-2						
Activities	20	– / 0.010	– / 0.291	– / 0.440	– / 0.343	– / 0.343
	50	0.005 / 0.009	– / 0.290	– / 0.440	– / 0.343	– / 0.343
	100	0.001 / 0.009	– / 0.289	– / 0.440	– / 0.343	– / 0.343
	150	– / 0.008	– / 0.289	– / 0.439	– / 0.342	– / 0.342
	200	0.001 / 0.008	– / 0.289	– / 0.439	– / 0.342	– / 0.342
Workflow shape *Type*-3						
Activities	20	– / 0.017	– / 0.298	– / 0.444	– / 0.346	– / 0.346
	50	– / 0.017	– / 0.298	– / 0.444	– / 0.346	– / 0.346
	100	– / 0.017	– / 0.298	– / 0.444	– / 0.346	– / 0.346
	150	– / 0.017	– / 0.298	– / 0.444	– / 0.346	– / 0.346
	200	– / 0.017	– / 0.298	– / 0.444	– / 0.346	– / 0.346
Workflow shape *Type*-4						
Activities	20	0.002 / 0.234	0.002 / 0.305	– / 0.438	0.004 / 0.347	0.004 / 0.347
	50	– / 0.169	– / 0.297	– / 0.440	– / 0.343	– / 0.343
	100	0.001 / 0.257	– / 0.323	– / 0.439	– / 0.355	– / 0.355
	150	– / 0.285	– / 0.312	– / 0.437	– / 0.350	– / 0.350
	200	– / 0.249	– / 0.315	– / 0.438	– / 0.349	– / 0.349

Table 6
Energy consumption comparison between MOHEFT, HEFT and greenHEFT.

		Number of Resources				
		10	25	50	75	100
Workflow shape *Type*-1						
Activities	20	0.488 / 0.961	0.787 / 0.326	0.933 / –	0.926 / 0.306	0.949 / 0.306
	50	0.425 / 0.606	0.727 / 0.404	0.915 / –	0.896 / 0.398	0.931 / 0.398
	100	0.421 / 0.621	0.696 / 0.319	0.927 / –	0.887 / 0.416	0.901 / 0.416
	150	0.415 / 0.635	0.703 / 0.312	0.927 / –	0.882 / 0.426	0.886 / 0.426
	200	0.401 / 0.644	0.693 / 0.295	0.933 / –	0.878 / 0.368	0.859 / 0.369
Workflow shape *Type*-2						
Activities	20	0.353 / –	0.656 / –	0.845 / –	0.625 / –	0.625 / –
	50	0.353 / –	0.657 / –	0.845 / –	0.626 / –	0.626 / –
	100	0.354 / –	0.657 / –	0.846 / –	0.626 / –	0.626 / –
	150	0.354 / –	0.657 / –	0.846 / –	0.626 / –	0.626 / –
	200	0.354 / –	0.657 / –	0.846 / –	0.626 / –	0.626 / –
Workflow shape *Type*-3						
Activities	20	0.348 / –	0.653 / –	0.844 / –	0.624 / –	0.624 / –
	50	0.348 / –	0.653 / –	0.844 / –	0.624 / –	0.624 / –
	100	0.348 / –	0.653 / –	0.844 / –	0.624 / –	0.624 / –
	150	0.348 / –	0.653 / –	0.844 / –	0.624 / –	0.624 / –
	200	0.348 / –	0.653 / –	0.844 / –	0.624 / –	0.624 / –
Workflow shape *Type*-4						
Activities	20	0.230 / 0.094	0.651 / 0.006	0.846 / –	0.678 / –	0.678 / –
	50	0.270 / 0.058	0.653 / –	0.845 / –	0.625 / –	0.625 / –
	100	0.249 / 0.134	0.646 / 0.017	0.846 / –	0.903 / –	0.903 / –
	150	0.276 / 0.196	0.649 / 0.009	0.846 / –	0.912 / –	0.912 / –
	200	0.273 / 0.150	0.646 / 0.007	0.846 / –	0.901 / –	0.901 / –

that initialises the population with already known good solutions in terms of makespan and energy consumption computed using HEFT and greenHEFT [12]. We used the jMetal framework [30] for implementing both algorithms. The solution representation and genetic operators (i.e. crossover and mutation) are based on the work described in [12]. We run both algorithms until a maximum number of 100,000 schedules are evaluated for each workflow. We consider the classical NSGA-II configuration, consisting of a population size of 100, crossover probability of 0.9, and mutation probability as the inverse of the number of tasks. As NSGA-II and NSGA-II* are stochastic algorithms, we run them for five times and use the best solution for comparison with MOHEFT. Our comparison relies on the use of the HV quality indicator described in Section 3 that assigns to a set of tradeoff solutions a numeric value in the range [0..1) reflecting its quality. The higher this value, the higher the quality of the set of tradeoff solutions. A value of 0 means that all the tradeoff solutions in that set are dominated by the solutions computed by the other evaluated algorithms.

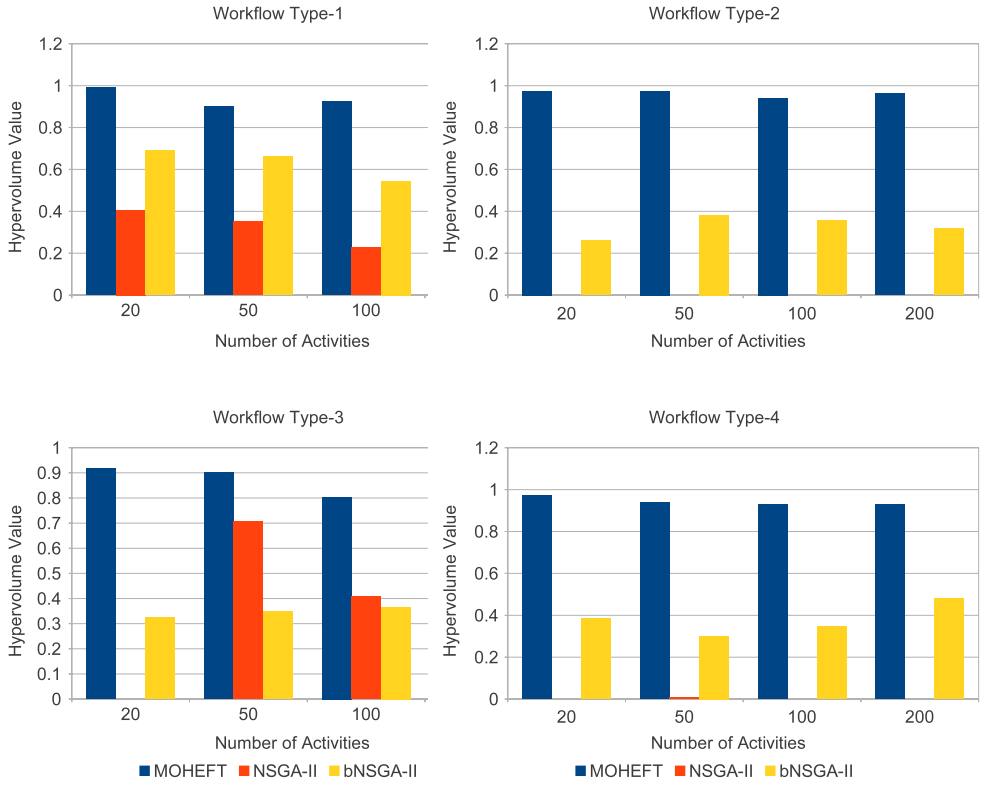

Fig. 5. Hypervolume of the Pareto fronts computed by MOHEFT, NSGA-II and NSGA-II* for workflows of four types with different number of activities.

We consider for this comparison workflows of each type with different number of tasks (from 20 to 200 tasks) and the same resource setup employed in Experiment-1. The results summarised in Fig. 5 show that MOHEFT computes tradeoff solutions of higher quality than NSGA-II and NSGA-II* (labelled as bNSGA-II in the figures) in all cases. The HV of the NSGA-II solutions is often 0, meaning that for that configuration MOHEFT and NSGA-II* computed schedules with shorter makespan and lower energy consumption. This is the case for all workflows of *Type*-2 and *Type*-4, where many dependences among activities exist. The performance advantage of MOHEFT over the other algorithms increases with the number of activities which negatively affects the performance of the evolutionary algorithms.

On the other hand, NSGA-II* outperformed NSGA-II for all workflows of *Type*-1, *Type*-2, and *Type*-4; however, for workflows of *Type*-3 with a low number of independent activities NSGA-II obtained better results. This means that initialising the population with the solutions computed by HEFT and greenHEFT is not only useless for this workflow type, but it also misleads NSGA-II*.

7.4. Experiment-3: impact of the number of activities and workflow shape

We now analyse the impact of the number of activities and workflow shape on the scheduling results. We consider for this experiment workflows composed of 20–200 activities with the 100%:0% characteristic. We employ the entire RS1 resource setup; thus the number of heterogeneous resources is much larger than the number of tasks (i.e. the number of available cores is more than 10 times larger than the number of activities to be scheduled, as shown in Table 3).

The results in Fig. 6 show that the shape of the workflow influences the computed set of tradeoff solutions. For example, for workflows of *Type*-1 in a highly heterogeneous resource setup, the computed Pareto fronts have convex shapes. Such Pareto fronts contain schedules which significantly improve the energy consumption with only a minimal hit in makespan. For example, for a workflow with 200 activities it is possible to reduce the energy consumption by up to 34.5% with a resulting makespan only 2% longer than the shortest solution. For *Type*-2 and *Type*-3 workflows, the resulting set of tradeoff solutions has a linear shape, meaning that diminishing the energy consumption will result in a proportional increase in the makespan. For *Type*-2 workflows, the slope of the computed Pareto front is approximately −0.5 and is almost independent of the number of workflow activities. Specifically, this means that for every Whr saved, the workflow makespan increases by two seconds. In the case of *Type*-3 workflows, the slope is around −0.6. To have an idea of the tradeoff between makespan and energy consumption, in workflows of *Type*-2 the energy consumption can be reduced by up to 17.4% for the sake of a 2% longer makespan; in *Type*-3 workflows, the similar makespan overhead brings about 20% less energy consumption. Finally, the shapes of the fronts computed for workflows of *Type*-4 are neither convex nor linear, but rather jagged with constant fronts and sudden high jumps (not evident in Fig. 6 due to the large scale of the horizontal axis). This Pareto front pattern denotes that for this workflow type energy savings cannot be achieved with a gradual increase in makespan (i.e. fine-grained tuning is not possible, but visual exploration of the tradeoff solutions is required).

7.5. Experiment-4: impact of the number of resources

In this set of experiments we analyse the influence of the number of resources on the schedule. For each workflow, we only consider the resources on which at least one activity has been scheduled by any of the tradeoff solutions computed in the previous section in the RS1 resource setup. Then, we reduce this pool to 75%, 50%, 25%, and 10% of its original size by randomly removing resources. We maintain the number of activities constant at 200. For some workflows, we omit the results for the 50%, 25% or 10% cases because of identical solutions.

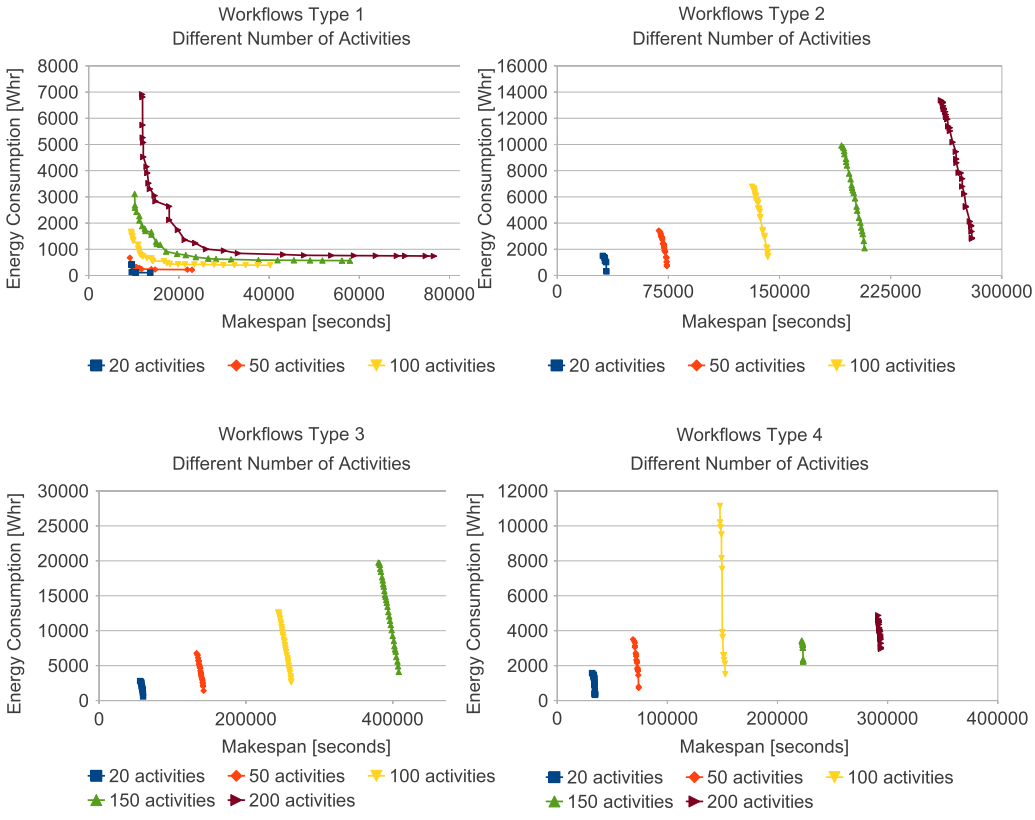

Fig. 6. Pareto tradeoff solutions computed by MOHEFT for workflows of four types and different number of activities.

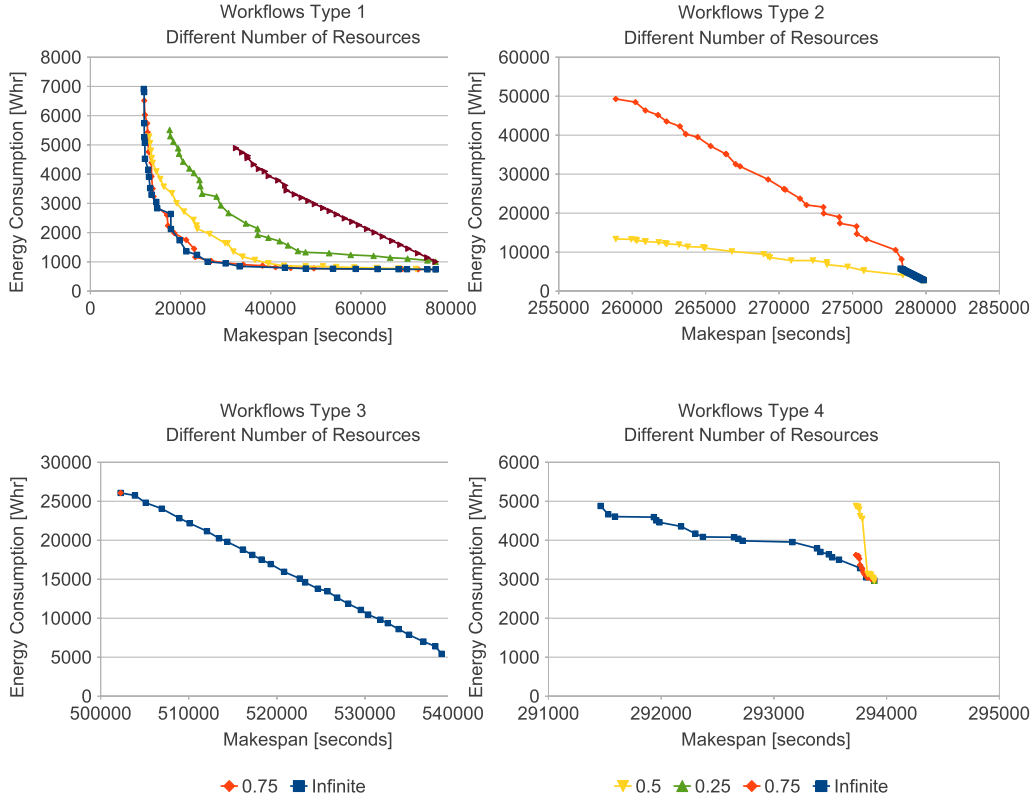

Fig. 7. Pareto tradeoff solutions computed by MOHEFT for different number of resources.

The results depicted in Fig. 7 show that reducing the number of resources also modifies the schedules computed by MOHEFT. For *Type*-1 workflows, the Pareto front is gradually transformed from the initial convex shape towards a linear shape. In this case, the opportunities for energy reduction reduce together with the number of available resources. For example, with 75% of resources, the

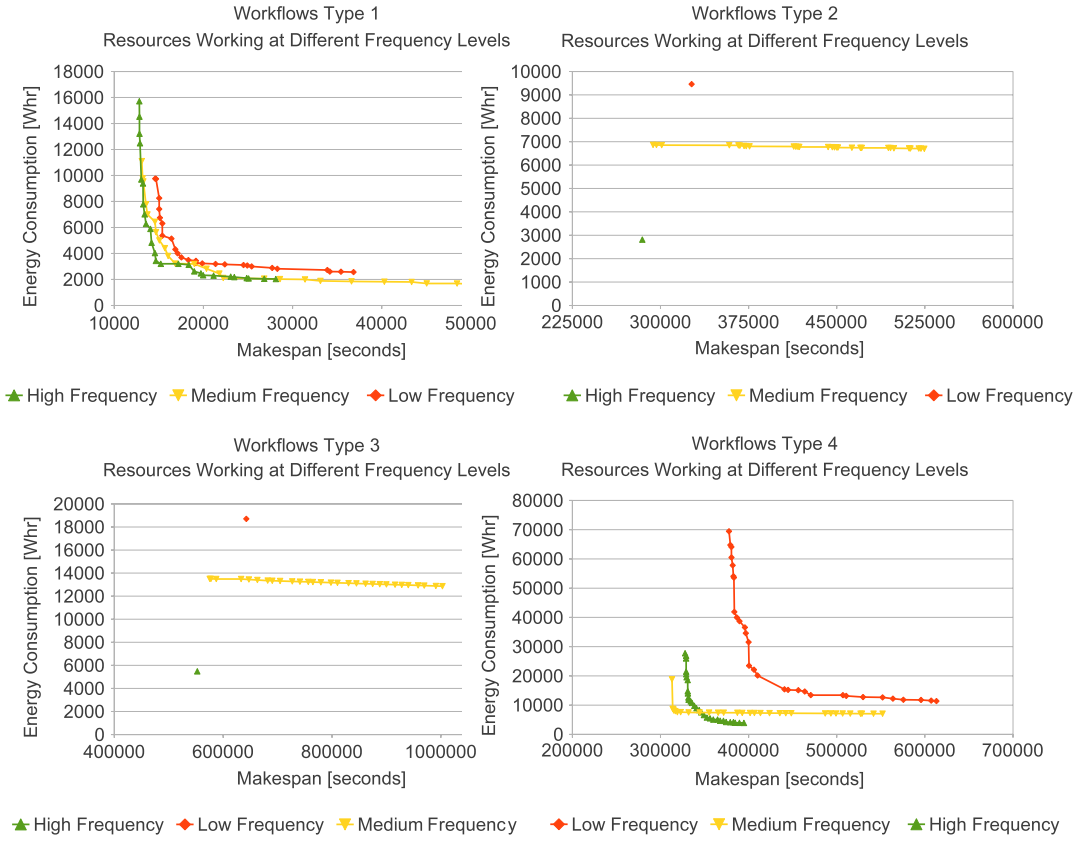

Fig. 8. Pareto tradeoff solutions computed by MOHEFT for different *low*, *medium* and *high* resource clock frequencies.

energy can be reduced by up to 26% incurring in only 10% of makespan overhead; when the number of resources reduces to only 10% of the original size, the percentage of energy savings is only up to 7% for the same percentage of makespan increase. For *Type*-2 workflows, the shape of the Pareto front is linear and does not change, but the absolute value of its slope increases. This scenario indicates that the reduction in the number of machines is accompanied by an increase in energy consumption. At the same time, the steeper slopes indicate better opportunities for energy reduction with small changes in makespan. The same is valid for *Type*-3 workflows, but in this case the reduction in the number of resources results in a diminution of the Pareto front to only one solution. As examples of the enormous energy savings that can be obtained in the last two scenarios we highlight here some results. In the case of *Type*-2 workflows, this percentage reaches 94.29% with 75% of the available resources with only an 8% longer makespan, and 51.21% with only 50% of these resources with an almost negligible impact on makespan (less than 1%). These figures are similar for *Type*-3 workflows. Finally, *Type*-4 workflows show similar behaviour to *Type*-2 ones. The main finding of this experiment is that reducing the number of resources leads to schedules with higher energy consumption, but to more possibilities for optimising for energy consumption while maintaining an acceptably short makespan at the same time.

7.6. Experiment-5: impact of the resource clock frequency

The clock frequency influences the energy consumed by a CPU and the number of processed instructions per second. In this section, we analyse the impact of resource clock frequency on the results computed by MOHEFT. We employ the three resource setups characterised by different CPU frequency levels: *low* (`RS2.LF`), *medium* (`RS2.MF`), and *high* (`RS2.HF`). We set the number of workflow activities to 200 and employ activities with the (100%:0%) characteristic.

Fig. 8 shows the Pareto fronts computed for each workflow type in every resource setup. In the case of workflows of *Type*-1, the shape of the Pareto front is independent of the resource clock frequency. By increasing the clock frequency, we notice a clear improvement in both makespan and energy consumption in all cases. For this type of workflow, energy savings increase with the frequency. For example, reducing the makespan by up to 5% results in a reduction of up to 35% of energy consumption in a setup of resources operating at *low* frequency, and up to 60% in a setup of resources operating a *high* frequency. Interestingly, for workflows of *Type*-2 and *Type*-3 and homogeneous resource clock frequencies, MOHEFT was not able to find any tradeoff between makespan and energy consumption (see 8). This stems from the fact that MOHEFT has very few alternatives in terms of resources and thus, little makespan or energy consumption improvement potential. In the case of homogeneous resources operating at *low* frequency, MOHEFT was able to find a single solution with high makespan and high energy consumption. Conversely, it found only one solution with low makespan and low energy consumption for homogeneous resources operating at *high* frequency. This indicates that the energy consumption is strongly influenced by the workflow execution time due to the static power consumption ($P_{R_j}^{(s)}$). Surprisingly, the makespan of the Pareto fronts for complex *Type*-4 workflows is not proportional to the average frequency of the available resources (see Fig. 8). In other words, a higher frequency does not always guarantee shorter makespan, as shown by some solutions computed for the resources operating at medium and high frequencies. The results of this experiment indicate that resources operating at high clock frequencies provide in general better solutions in terms of makespan and energy consumption due to the high impact of the static energy.

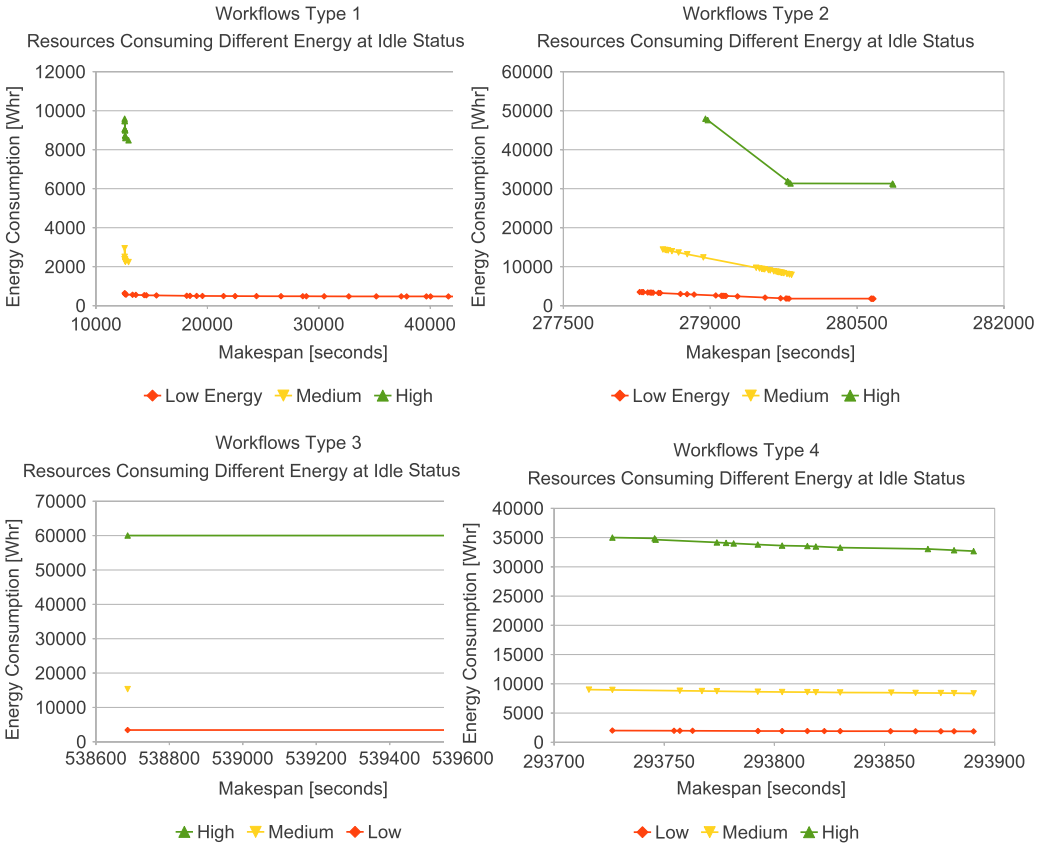

Fig. 9. Pareto tradeoff solutions computed by MOHEFT for resource with different static energy consumption at idle state.

7.7. Experiment-6: impact of static energy consumption at idle state

Having observed that static energy has a strong influence on energy consumption, we analyse in this section the impact of resources with different levels of static energy on the tradeoff solutions. We consider the three sets of resources characterised by different levels of power consumption in idle state ($P_{R_j}^{(s)}$):

- *low* (`RS3.LP`) consuming approximately 40 W in idle state for CPU and other subsystems (e.g. power supply, memory, peripherals), and corresponding to Intel Xeon E3-based machines[2];
- *medium* (`RS3.MP`) consuming around 120 W and corresponding to a wide range of desktop-type machines based on Intel[3] or AMD[4] CPUs;
- *high* (`RS3.HP`) consuming around 420 W which are similar to multi-processor server architectures with multiple redundant power supplies (e.g. four-CPU AMD Opteron 8356 described in Table 2).

We consider in this experiment the same workflows as in Experiment-4.

The Pareto fronts computed by MOHEFT in Fig. 9 show that the savings in energy are small or expensive in comparison with makespan reduction. This pattern is common to all workflow types except *Type*-2 workflows for machines with high and medium static energy consumptions. In *Type*-1 workflows, reducing energy consumption is expensive in terms of makespan increases. For example, reducing the energy in a 25% would require a makespan almost 340% longer than the shorter computed makespan. For *Type*-2 workflows and setups with machines with high and medium energy consumption, up to 40% of energy can be saved with small hits on makespan (less than a 2% overhead). In the rest of the cases, the Pareto fronts are linear with a slope close to zero. In these cases, there is almost no tradeoff between makespan and energy consumption.

Our conclusion is that homogeneous resources in terms of static energy barely bring potential for energy optimisation, except for the case of *Type*-2 workflows. In the rest of the cases, the schedule with the shortest makespan consumes as much energy as the one with the lowest energy consumption.

7.8. Experiment-7: impact of CPU and I/O utilisation

In the final experiment, we aim at analysing the impact of the activity type on the MOHEFT results by considering activities which alternate periods of computation with periods waiting for I/O operations. We consider activities using the CPU for 25%, 50%, 75%, and 100% of their execution and waiting for I/O operations the rest of the time. We use the same set of resources as in Experiments 1–3, based on the `RS1` setup with more than ten times as many available cores as the number of activities.

The results in Fig. 10 show that the higher the CPU utilisation gets, the shorter the makespan and the lower the energy consumption are. The explanation lies in the same amount of instructions computed by each task, which is obviously faster and more energy efficient for a higher CPU utilisation. We also observe that the CPU and I/O utilisations change the shape of the Pareto fronts for all workflows except *Type*-1. This is mostly visible when activities use the CPU only for a small fraction of time. The shape of the fronts for the other workflow types consists of two segments, both representing a linear dependence between makespan and energy consumption. The first segment allows for high improvements in

2 http://ark.intel.com/products/65735/.
3 http://ark.intel.com/products/series/37168.
4 http://www.amd.com/uk/products/desktop/processors/amdfx/Pages/amdfx-model-number-comparison.aspx.

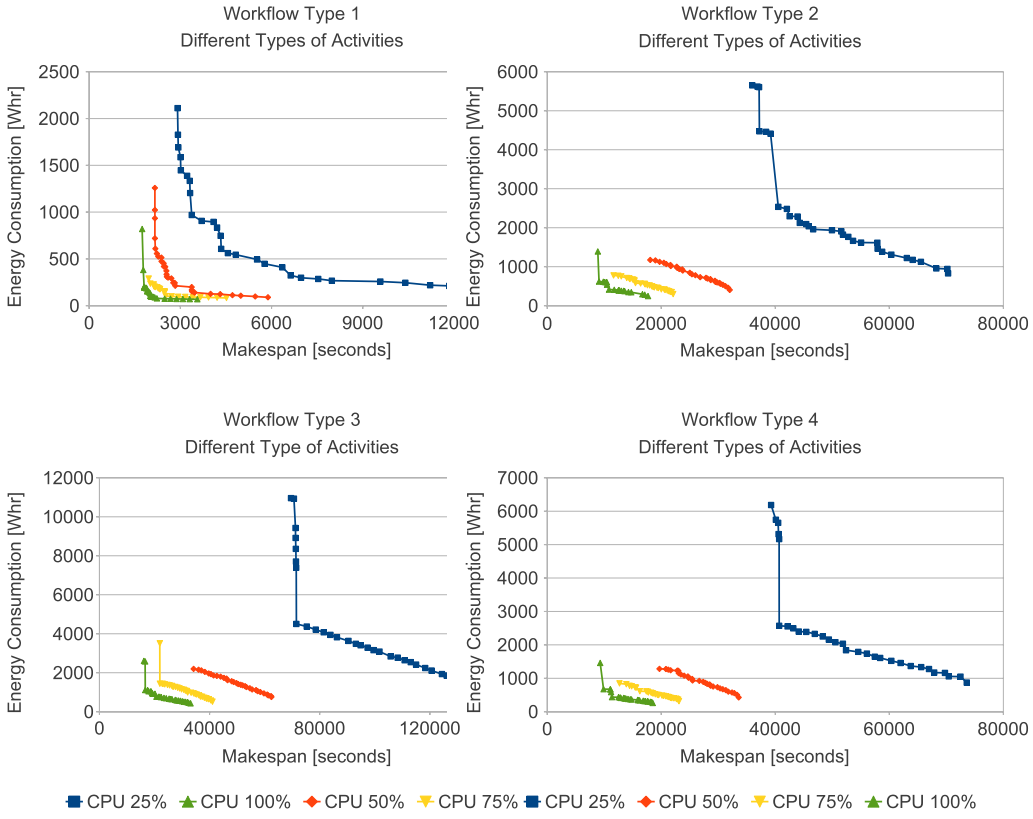

Fig. 10. Pareto tradeoff solutions computed by MOHEFT for workflow activities with different CPU and I/O utilisations.

Table 7
Summary of results.

Exp.	*Type*-1	*Type*-2	*Type*-3	*Type*-4	Conclusion
Exp-3	Convex	Linear	Linear	Not classified	Best energy-makespan tradeoffs can be obtained for workflows with high parallelism and large number of tasks
Exp-4	Convex to linear	Linear	Linear	Not classified	Reducing the number of resources increases makespan and energy consumption, but tradeoffs are still possible
Exp-5	Convex	Horizontal	Horizontal	Not classified	Faster resources are preferred for both makespan and energy consumption
Exp-6	Convex/horizontal	Horizontal	Horizontal	Horizontal	Static power consumption strongly influences the energy efficiency and presents higher energy-makespan tradeoff potential than dynamic power consumption
Exp-7	Convex	Linear	Linear	Linear	I/O-intensive workflows exhibit high makespans and energy consumptions, but present a wider range of energy-makespan tradeoffs possibilities

energy consumption with a small makespan overhead, while the second segment involves a higher makespan overhead for every Whr saved.

7.9. Summary

Table 7 summarises the main results of Experiments 3–7 targeting the evaluation of MOHEFT in various scenarios for different workflow types, activity characteristics and underlying resources. For every experiment, we present our main finding and classify the type of Pareto front computed for every type of workflow. We identified in our experiments three classes of Pareto fronts: convex, sloped linear, and horizontal. Convex Pareto fronts, frequently obtained for *Type*-1 workflows, comprise highly energy-efficient solutions with small makespan overheads. Sloped linear Pareto fronts, most frequently encountered for *Type*-2 and *Type*-3 workflows, indicate a linear dependence between makespan and energy consumption allowing a gradual tuning. Horizontal Pareto fronts represent a particular case of the sloped linear fronts with the slope very close to zero and denoting situations in which only minor improvements in energy efficiency can be obtained with a huge negative impact on makespan. We denote the peculiar Pareto fronts which do not belong to any of these generic classes as "not classified". These special Pareto fronts require individual visualisation in order to select the desired tradeoff solution between makespan and energy consumption, as we performed in our evaluation.

8. Conclusions and future work

We tackled in this paper the problem of energy-efficient workflow scheduling in heterogeneous parallel systems. We proposed a novel multi-objective list-based workflow scheduling heuristic called MOHEFT, capable of computing tradeoff solutions between makespan and energy consumption. This algorithm relies on empirical models for predicting the energy consumption and execution time of workflow activities, designed based on historical data collected from real workflow task executions. We compared MOHEFT with HEFT, the state-of-the-art scheduling algorithm for optimising workflow makespan, and greenHEFT, a customised version of HEFT for optimising energy. We demonstrated through extensive experimentation that our proposed heuristic is at least as good as the other two algorithms in all evaluated scenarios, and better in many cases. We also compared MOHEFT with NSGA-II, a state-of-the-art evolutionary algorithm used in the multi-objective

optimisation community. The experiments showed that MOHEFT computed Pareto sets of higher quality than NSGA-II. Finally, we analysed the MOHEFT's response to changes in workflow characteristics, activities and underlying resources. For each studied scenario, we evaluated the viability of energy optimisations, as well as the corresponding tradeoffs in makespan. In future work we plan to extend our algorithm to consider multi-tenant commercial cloud systems, including other optimisation objectives such as economic cost. We will also study the impact of activity types on the external load and the quality of our empirical prediction method, including other types of IO-intensive applications.

References

[1] James Hamilton, Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for Internet-scale services.

[2] Eric Humenay, David Tarjan, Kevin Skadron, Impact of process variations on multicore performance symmetry, in: Proceedings of the Conference on Design, Automation and Test in Europe, DATE'07, EDA Consortium, San Jose, CA, USA, 2007, pp. 1653–1658.

[3] T. Burd, T. Pering, A. Stratakos, R. Brodersen, A dynamic voltage scaled microprocessor system, in: Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International, 2000, pp. 294–295, 466.

[4] A.P. Chandrakasan, S. Sheng, R.W. Brodersen, Low-power CMOS digital design, IEEE Journal of Solid-State Circuits 27 (4) (1992) 473–484.

[5] H. Topcuoglu, S. Hariri, Min-You Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, T. Meyarivan, A fast elitist multi-objective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2000) 182–197.

[7] J. Knowles, L. Thiele, E. Zitzler, A tutorial on the performance assessment of stochastic multiobjective optimizers, Technical Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 2006.

[8] Saurabh Kumar Garg, Rajkumar Buyya, H.J. Siegel, Scheduling parallel applications on utility grids: time and cost trade-off management, in: Proceedings of the Thirty-Second Australasian Conference on Computer Science—Volume 91, ACSC'09, Australian Computer Society, Inc, Darlinghurst, Australia, Australia, 2009, pp. 151–160.

[9] E. Ilavarsan, P. Thambidurai, Low complexity performance effective task scheduling algorithm for heterogeneous computing environments, Journal of Computer Science 3 (2) (2007) 94–103.

[10] Xiaofeng Wang, Rajkumar Buyya, Jinshu Su, Reliability-oriented genetic algorithm for workflow applications using max–min strategy, in: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 108–115.

[11] I. Assayad, A. Girault, H. Kalla, A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints, in: International Conference on Dependable Systems and Networks, DSN'04, IEEE, Firenze, Italy, 2003.

[12] Jia Yu, Michael Kirley, Rajkumar Buyya, Multi-objective planning for workflow execution on grids, in: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, GRID'07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 10–17.

[13] M. Mezmaz, N. Melab, Y. Kessaci, Y.C. Lee, E.-G. Talbi, A.Y. Zomaya, D. Tuyttens, A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, Journal of Parallel and Distributed Computing (71) (2011) 1497–1508.

[14] J. Kolodziej, S.U. Khan, F. Xhafa, Genetic algorithms for energy-aware scheduling in computational grids, in: 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2011.

[15] J. Kolodziej, S.U. Khan, L. Wang, A. Zomaya, Energy efficient genetic-based schedulers in computational grids, Concurrency and Computation: Practice and Experience (2012) 1497–1508.

[16] C.O. Diaz, M. Guzek, J.E. Pecero, G. Danoy, P. Bouvry, S.U. Khan, Energy-aware fast scheduling heuristics in heterogeneous computing systems, in: 2011 International Conference on High Performance Computing and Simulation, HPCS, July 2011, pp. 478–484.

[17] P. Lindberg, J. Leingang, D. Lysaker, S.U. Khan, J. Li, Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems, The Journal of Supercomputing (2010) 478–484.

[18] N. Min-Allah, H. Hussain, S.U. Khan, A.Y. Zomaya, Power efficient rate monotici scheduling for multi-core systems, Journal of Parallel and Distributed Computing (2012) 48–57.

[19] Mourad Hakem, Franck Butelle, Reliability and scheduling on systems subject to failures, in: Proceedings of the 2007 International Conference on Parallel Processing, ICPP'07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 38–46.

[20] Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, Marios D. Dikaiakos, Scheduling workflows with budget constraints, in: S. Gorlatch, M. Danelutto (Eds.), Integrated Research in Grid Computing, in: CoreGrid series, Springer-Verlag, 2007.

[21] María Arsuaga-Ríos, Miguel A. Vega-Rodríguez, Multi-objective firefly algorithm for energy optimization in grid environments, in: Marco Dorigo, Mauro Birattari, Christian Blum, Anders Lyhne Christensen, Andries P. Engelbrecht, Roderich Groß, Thomas Stützle (Eds.), Swarm Intelligence, in: Lecture Notes in Computer Science, vol. 7461, Springer, Berlin, Heidelberg, 2012, pp. 350–351.

[22] Louis-Claude Canon, Emmanuel, Mo-greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines, in: International Heterogeneity in Computing, 2011.

[23] J.J. Durillo, H.M. Fard, R. Prodan, MOHEFT: a multi-objective lilst-based method for workflow scheduling, in: 4th IEEE International Conference on Cloud Computing Technology and Science, December 2012.

[24] Samir Khuller, Jian Li, Barna Saha, Energy efficient scheduling via partial shutdown, in: Proceedings of the Twenty-First Annual ACM–SIAM Symposium on Discrete Algorithms, SODA'10, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2010, pp. 1360–1372.

[25] Cesar O. Diaz, Mateusz Guzek, Johnatan E. Pecero, Pascal Bouvry, Samee U. Khan, Scalable and energy-efficient scheduling techniques for large-scale systems, in: Proceedings of the 2011 IEEE 11th International Conference on Computer and Information Technology, CIT'11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 641–647.

[26] S.U. Khan, I. Ahmad, A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids, IEEE Transactions on Parallel and Distributed Systems 20 (3) (2009) 346–360.

[27] Christopher M. Bishop, Neural Networks for Pattern Recognition, first ed., Oxford University Press, USA, 1996.

[28] http://www.povray.org/.

[29] J. Yu, R. Buyya, K. Ramamohanarao, Workflow scheduling algorithms for grid computing, in: F. Xhafa, A. Abraham (Eds.), Metaheuristics for Scheduling in Distributed Computing Environments, Springer, Berlin, 2008, pp. 109–153.

[30] Juan J. Durillo, Antonio J. Nebro, jMetal: a java framework for multi-objective optimization, Advances in Engineering Software 42 (2011) 760–771.

Juan J. Durillo received the M.S. and Ph.D. degrees in computer science from the University of Málaga, Spain, in 2006 and 2011, respectively. Currently, he is working as post-doctoral researcher in the Department of Computer Science, University of Innsbruck, Austria. Dr. Durillo has authored more than ten science papers in international journals and has contributed to more than 20 papers in international conferences or book chapters. His research interests include multi-objective optimisation, parallel and distributed computing, and search-based software engineering.

Vlad Nae is currently a Post-doctoral Fellow at the Institute of Computer Science, University of Innsbruck. He received his Diploma Engineer degree in 2006 from the Politehnica University of Bucharest, Romania. In December 2011 he received a Ph.D. in Computer Science from the University of Innsbruck. His research interests are energy efficiency and resource management in distributed systems, particularly in the area of highly interactive applications such as massively multiplayer online games. He is the author of 19 publications including four journal articles and three book chapters.

Radu Prodan received the Ph.D. degree from the Vienna University of Technology, Vienna, Austria, in 2004. He is currently associate professor at the Institute of Computer Science, University of Innsbruck, Austria. His research in the area of parallel and distributed systems comprises programming methods, compiler technology, performance analysis and scheduling. He participated in several national and European projects. He is currently coordinating the Austrian projects and was workpackage Leader in the IST-034601 (edutain@grid) and 26185 (SHIWA) projects. He is the author of more than 70 journal and conference publications and one book. He was the recipient of an IEEE Best Paper Award. He is a member of the ACM.