Multi-objective Energy-Efficient Workflow Scheduling using List-based Heuristics

Juan J. Durillo, Vlad Nae, Radu Prodan

University of Innsbruck
6020-Innsbruck, Austria
juan, vlad, radu @dps.uibk.ac.at

DOI: https://doi.org/10.1016/j.future.2013.07.005

Multi-objective Energy-Efficient Workflow Scheduling using List-based Heuristics

Juan J. Durillo, Vlad Nae, Radu Prodan

University of Innsbruck
6020-Innsbruck, Austria
juan, vlad, radu @dps.uibk.ac.at

Abstract

Workflow applications are a popular paradigm used by scientists for modeling applications to be run on distributed computing systems for obtaining high-performance. Nowadays, the increase in the number and type of different distributed systems facilitated the access to high-performance computing to almost any 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. In order to decrease the energy consumption in distributed systems, different methods have been proposed. Among them, energy-efficient scheduling is receiving increasing attention today. Current schedulers are, however, either based on simplistic energy consumption models which do not match the reality, or use techniques like DVFS available on all the types of distributed systems. In this paper, we present a multi-objective workflow scheduler able to compute a 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 real distributed systems. We compare our algorithm two classical mono-objective scheduling algorithms and show that our approach computes better or similar results in different scenarios. Furthermore, we analyse the different tradeoff solutions computed by our algorithm under different experimental configurations and we observe that in some cases MOHEFT finds solutions which reduce the energy consumption by up to 34.5% by just increasing the makespan by 2% versus the optimal solution.

1. Introduction

Precedence-constrained parallel applications, also known as *workflows*, are one of the most popular paradigms used by scientists for modelling large applications. Most of these applications have to deliver results under certain hard time constraints requiring executions in distributed systems. The challenge stemming from these requirements is achieving an efficient task-to-resource machine

mapping, also called *schedule*, in order to minimize the execution time of the workflow – an NP-complete problem.

Nowadays, high performance parallel computing is available to almost any scientist or researcher. Besides expensive supercomputers, we have experienced in the last decade a proliferation of other distributed systems such as commodity clusters and grids. Furthermore, the new cloud computing distributed systems paradigm that arose in the recent years introduces a new operational model where resources are managed by specialized data centres and rented only on demand and for the period of time they need to be used. High performance computing can be viewed under this paradigm as a service which hides most of the complexity of operating a large number of distributed resources.

Along with its many advantages, cloud computing introduces several new challenges. For example, a critical problem in distributed systems is the considerable amount of energy that data centres require. Besides the green implications of energy-savings, Hamilton [12] reported that the financial expenditure for the energy consumption of Amazon.com data centres in a period of fifteen years accounts 19% of its total budget, and that 23% of the same budget represents the cost of maintaining the cooling infrastructure (which depends on the energy consumption too). Under these circumstances, energy efficiency is presently becoming an important objective for computing infrastructure providers due to its environmental implications and high financial impact.

Reducing the energy required by a distributed system can be accomplished through two non-orthogonal approaches: hardware and software. The hardware approach implies the design of more energy-efficient components, manly processors which represent the main energy consumers in computing systems. Advances in hardware energy efficiency are, however, difficult to achieve and are characterised by a long delay (sometimes even years) between the initial proposal and the final commercial product due to the long, multi-phased manufacturing process (design, prototyping, testing, mass production). Thus, relying on hardware advantages may not be a satisfactory solution for short term energy savings.

Concerning the software approach, existing techniques try to reduce the energy consumption (when possible) through the use of Dynamic Voltage and Frequency Scaling (DVFS), which enables on-line adjustments to voltage and frequency in CMOS circuits [13, 4, 6]. By applying this technique to different CPUs in a distributed 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.

Extrapolating this idea to heterogeneous distributed systems, the energy efficiency of workflow executions might be increased by scheduling critical tasks on fast but probably energy-expensive resources, and non-critical tasks on slower less energy consuming ones. In such systems, the user is confronted with a set of resources working at different speeds and with different characteristic energy consumptions.

Scheduling workflows in these circumstances can be formulated as a multi-

objective optimization problem (MOP) which aims at optimising two possibly conflicting criteria: *makespan* and entailed *energy* consumption of 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 if we refer to the domain or co-domain of the functions to be optimized. The property of the solutions on the Pareto set/front is that they cannot be simultaneously improved with respect to all objectives. Concretely in our case, there is likely that no single workflow schedule that simultaneously minimises its makespan and energy consumption exists, but rather a set of Pareto-optimal tradeoff solutions.

Although several energy-efficient approaches to workflow scheduling have been proposed so far, they present several disadvantages which hinder or even prevent their wider adoption in distributed systems:

1. DVFS technology is limited to adjusting the frequency and voltage of the entire CPU[1]. This limits the applicability of DVFS for scheduling at the single resource level since it is not possible to execute tasks requiring different DVFS levels on the same multi-core machine.
2. DVFS is most of the times not available to the guest operating systems in virtualised environments and is managed only by the hypervisors. This directly affects scheduling on cloud resources, where virtual machines of different users may run on different cores of the same physical CPU where exposing DVFS functionality is infeasible.
3. Most of the available energy consumption models are theoretical and have not been thoroughly validated with realistic experiments.
4. Most of the existing models simplify the machine energy consumption to two distinct levels corresponding to its idle and fully-loaded state. In reality, the energy consumption greatly varies and in continuous fashion with the level of CPU utilisation which, in turn, is dependent on the executed task's characteristics. This behaviour is even more evident in multi-core CPUs with individual cores having different levels of utilisation.
5. Existing multi-objective approaches requires the user indicate a-priori preferences in order to reduce the search space. One technique is to combines the multiple objectives in a single objective, for example by assigning different a-priori weights to the different objectives and optimizing the resulting aggregation function. The disadvantage is that the computed solution depends on the combination of the multiple objectives, which is made a priori and without any information about the problem being solved and may not capture the user preferences in an accurate way. Another approach is optimise a single preferred objective and keep the others within user-defined constraints [21]. The main weakness is that it requires a predefined order in which to optimise the objectives, hence including

[1]Intel scheduled the release of their newest Haswell architecture in 2013 which promises "more fine grained control" of the CPU sub-components for energy efficiency, but no concrete details have been released so far.

some sort of preference information. Finally, reasonable a-priori values for the constraints are often unknown until the first schedule is computed.

In this paper, we present a holistic approach towards the design of an energy-efficient workflow scheduler able to compute a set of trade-off solutions. Our scheduler, called MOHEFT, is an extension of the popular Heterogeneous Earliest Finish Time (HEFT) algorithm. MOHEFT relies on empirical models for the energy consumption and execution time of workflow tasks. These models are based on the knowledge extracted from historical real task executions, and reflect 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. The complete list of contributions of this paper in the area of energy-efficient scheduling are:

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 an empirical model based on historical execution time and energy consumption measurements of real workflow tasks on a heterogeneous set of machines.
3. *A multi-objective energy-efficient scheduling algorithm.* We present an algorithm capable of approximating different Paretio-optimal workflow scheduling solutions as a trade-off between energy consumption and makespan. We validate the algorithm through comparisons to with the popular HEFT algorithm for minimizing makespan and greenHEFT, a extension of the former for optimising energy comsumption.
4. *Analysis of the impact of different workflow characteristics and different resources on the trade-off solutions.* We cover a multi-dimensional experimental space, analysing the impact of multiple activity, workflow and resource properties on the trade-off solutions of our algorithm.

The rest of this paper is structured as follows. Next formally describes our problem and introduces some background on multi-objective optimization. Section 3 reviews existing models for tasks energy consumption and execution time, and describes the modelling approach followed in this paper. Section 4 presents MOHEFT. The evaluation of our algorithm and the analysis of the obtained results is included in Section 5. The next section is aimed at reviwing existing works on multi-objective workflow and energy-efficient scheduling. Finally, we presents the main conclusions and future work in Section 7.

2. Formalism

Our problem consists of scheduling the workflow tasks on the available resources in such a way that the makespan and the energy consumption of its

Table 1: Resource characteristics.

Characteristic	*Description*
Technology [nm]	Dimension of the CPU lithography (e.g. 32nm)
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[i]
TDP	CPU Thermal Design Point[ii]

[i] Twice the number of phisical cores for hyper-threading. [ii] Theoretical maximum heat dissipation requirement for not exceeding the maximum junction temperature; also a rough indicator of the power consumption class of the CPU

execution are minimized. We introduce in the remainder of this section a simple but realistic formalism that defines the workflow, resource environment, and the metrics targetted.

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 dependencies, D, defined as:

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

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 $pred\,(A_i) = \{A_k |\, (A_k, A_i, Data_{ki}) \in D\}$ to denote the set of *predecessors* of activity A_i (i.e. activities to be completed before starting A_i). Every activity $A_i \in A$ is characterised by its length (or workload) measured for example in total number of instructions. The execution time and the energy consumption entailed by the activity execution depends on its length and the resource on which it executes.

2.2. Resource Environment

We assume that our hardware platform consists of a set 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.

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 $t_{(A_i,R_j)}$:

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

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 one TCP stream between $sched(A_p)$ and R_j ($b_{pj} = \infty$, if $sched(A_p) = R_j$).

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 between 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.

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:

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

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.

2.5. Problem Definition

Given a workflow $W = (A, D)$, our goal is to approximate the *Pareto-optimal workflow schedules* $sched(W) = \bigcup_{i=1}^{n} sched\,(A_i)$ with respect to makespan and energy consumption, where n is the total number of activities.

2.6. Multi-Objective Optimization

In this section, we introduce a few concepts from the *multi-objective optimization* theory for a better understanding of this work. We assume without loss of generality that minimisation is the goal for all the objectives, as any maximisation problem can be defined in terms of a minimization too.

A general, multi-objective optimisation problem can be formally defined as finding all the vectors $\vec{x} = [x_1, x_2, \ldots, x_n]$ which minimise the vector function $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_t(\vec{x})]^T$. For our particular problem, $n = |A|$ represents the cardinality of the task set A and the i-th component of a solution $\vec{x}$ represents 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 $t = 2$, where $f_1(\vec{x})$ represents the makespan and $f_2(\vec{x})$ the energy consumption.

Multi-objective optimization introduces the concept of *dominance*. A solution $\vec{x_1}$ dominates a solution $\vec{x_2}$, if both the makespan and the energy consumption entailed by the schedule $\vec{x_1}$ are smaller than those of $\vec{x_2}$. 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 consumes less energy). In Figure 1 for example, the solution labelled a dominates the one labelled b because it has better makespan and consumes less energy. Similarly, a dominates c too. Meanwhile, a and d are non-dominated because a is better in makespan, but d consumes less energy. A set of non-dominated solutions is called *Pareto set* (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 schedule of the workflow with different makespans and energy consumption.

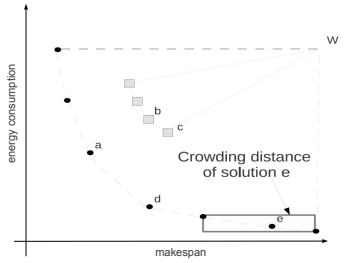

Figure 1: Comparison of multi-objective tradeoff solutions.

A Pareto front can be seen as a tool for decision support and preference discovery. Its shape can provide insight to researches or scientists (from now

decision makers), 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 known. In this way, the users does not have to set their preferences before finding a solution, instead the preferences are discovered afterwards.

3. Energy and Execution Time Modelling

3.1. Background

The problem formulation presented in the previous section relies on two models: one for the number of instructions a resource is able to process every second, and a second one for the energy consumed by a resource at an instant t. This section provides a concise background overview of existing models and analyses their applicability to our problem.

Many existing approaches [22, 10, 14, 23, 2, 25] employ a simplified model for determining the time required by a resource for completing an activity. This model consists in assigning 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 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 which allow individual cores execute different tasks (sometimes even belonging to different users). Due to various reasons such as cache sharing between cores and data bus contention, such scenarios always introduce a time overhead for concurrently executing multiple independent tasks on a multi-core, multi-processor machine.

Regarding existing energy consumption models [19, 16, 17, 8, 18, 20], 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, which exhibit a multitude of distinct power consumption levels depending on the number of cores and their level of utilisation. The problem of these simplified models is further exacerbated by the recent advances in energy-aware hardware features of the new CPU architectures. For example, some CPUs automatically switch between different power modes according to the current load. It is also worth noting that, as 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).

3.2. Applicability of Existing Models

In order to validate our claim that the existing models reviewed in Section 3.1 are not applicable in the context of modern multi-core, multi-processor architectures, we conduct an extensive number of experiments through which we characterise the execution performance and energy consumption of workflow

Table 2: Resource configuration for modelling time and energy consumption of the `povray` workflow task (all machines are 64 bit architectures and the and the minimum operational frequency is omitted).

CPU Model	*Tech. [nm]*	*Freq. [GHz]*	*Cache size [KB]*	*Cache sharing*	*Cores*	*Threads*	*TDP [W]*	*CPU no.*	*Instances*
AMD Opteron 8356	65	2.3	2048	1	4	4	95	8	2
AMD Opteron 6168	45	1.9	12288	6	12	12	115	2	2
Intel Xeon X5650	32	2.66	12288	6	6	12	95	2	2
Intel Xeon E7-4870	32	2.4	30720	1	10	20	130	4	1
AMD Opteron 880	90	2.4	2048	1	2	2	95	4	1
AMD Opteron 885	90	2.6	2048	1	2	2	95	8	1

tasks on such architectures. Specifically, through these experiments, we 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 inherent to powering on a core decreases with the number of cores already powered on.

We measure the execution time and the associated energy consumption of a workflow activity called `povray`, belonging to a the Persistence Of Vision Raytracer (POV-Ray), a real workflow thoroughly presented in Section 5. We focus our analysis on this activity which renders a set of frames (i.e. images) from a three-dimensional scene descriptor file as it is highly parallelisable and the most time-consuming part of the workflow. We have measured the execution time and energy consumption of the `povray` activity using different workload sizes on a diverse pool of resources, as described in Table 2. The maximum activity concurrency level of the resource pool is 264, stemming from the cumulated number of hardware threads available in our resource pool. We achieve the time and energy instrumentation and measurement through a tool we developed in Python and C which retrieves online measurements from multiple LAN-enabled Voltech PM1000+[2] power measurement devices connected to the machines.

Figure 2 shows the time required by two multi-core machines for rendering a 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 other cores are concurrently utilised. For the Intel architecture, we compute a slowdown of 34.2% in case of 39 threads of external load compared to no external load. An additional slowdown of 38.1% from 39 to 79 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. zero and 31 external load threads).

Figure 3 presents the variation in energy consumption of one CPU core with increasing external load (number of running threads) on the remaining CPU cores. The decrease in energy consumption per core with increasing external

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

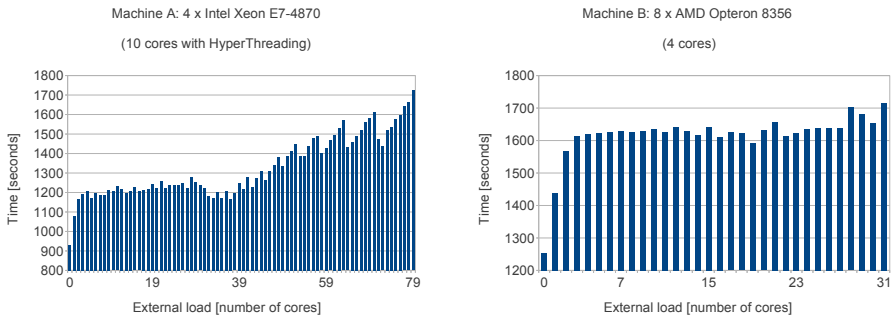

Figure 2: Time for rendering one frame on a single core of two multi-core machines (AMD and Intel with Hyper-Threading) with gradually increasing external load until reaching machine saturation.

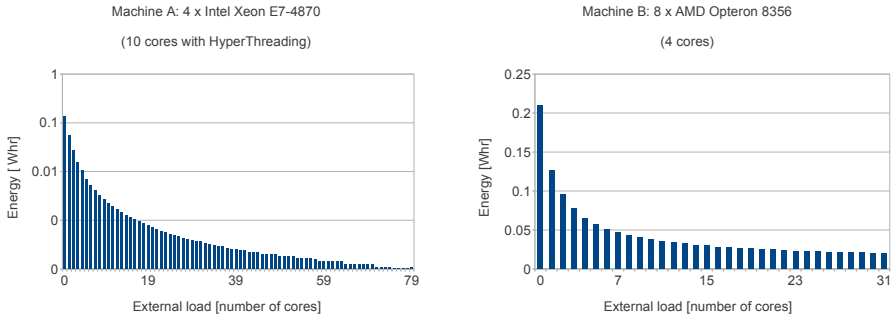

Figure 3: Energy consumption of a single core of two multi-core machines (AMD and Intel with Hyper-Threading) with increasing external load until it reaches machine saturation.

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 Equation 6:

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

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):

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

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

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

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

3.3. Proposed Energy Consumption and Execution Time Models

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 have 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 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 [3] for workflow activity execution time and energy consumption estimations. 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.

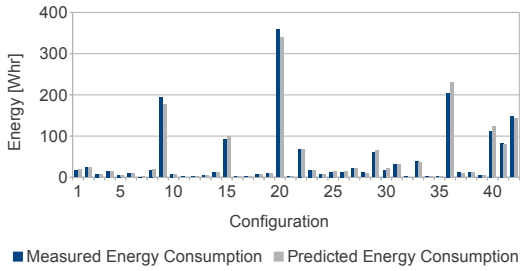

Figure 4: Neural Network predictions vs real energy consumption data on several examples.

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 input size (e.g. the number of frames to render in the case of the `povray` task analysed in Section 3.2) and the external load on the resource. 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. We train the predictors with historical data from executions with configurations uniformly sub-sampling the activity types – resource types evaluation space.

To validate this proposed approach we use the data collected from executions of the real `povray` workflow task (analysed in Section 3.2) on our diverse set of resources (shown in Table 2) for neural network training and, eventually, for cross-validation. Concretely, 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 which is realised by presenting the never encountered before input to the neural network and measuring the deviation of the predicted value (energy or execution time) from the real, measured value.

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%. Figure 4 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 $1,000$ frames for different configurations consisting of various machines with different external loads (in no particular order).

4. MOHEFT: Multi-Objective Heterogenous Earliest Finish Time Algorithm

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

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: $sched(W) \leftarrow \emptyset$ ▷ Initialize 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: $T_c^{(S_i)} \leftarrow \max\limits_{A_p \in pred(S_i)} \left\{ T_c^{(A_p)} + \frac{Data_{pi}}{b_{pj}} \right\} + t_{(B_i,R_j)}$ ▷ 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**

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 multiples objectives. Finally, we present also greenHEFT, heft-like heuristic for minimising the energy consumption.

4.1. HEFT: Heterogeneous Earliest Finish Time Algorithm

The Heterogeneous Earliest Finish Time Algorithm (HEFT) [22] is a popular list-based heuristic scheduling algorithm for optimizing the makespan [22] 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 [22]. Once the execution order 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).

4.2. MOHEFT: Multi-Objective Heterogenous 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

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**

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 saves the best K tradeoffs 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 [7] and graphically depicted in Figure 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.

4.3. greenHEFT: A List-based Heuristic for Minimizing Energy Consumption

For the sake of comparisons, we also designed a mono-objective heuristic for optimising energy consumption only. called *greenHEFT*. The new heuristic works in the same way as the original HEFT with the difference in the second

phase every task in the ranking is assigned to the processor where it consumes less energy.

5. Evaluation

We carried out extensive experiments to evaluate the solution delivered by MOHEFT for makespan and energy-efficient workflow scheduling, aiming at:

- demonstrating that they at least as good as those obtained using the mono-objective HEFT for optimising makespan and greenHEFT for optimising energy;
- analysing the results of MOHEFT for:
  - scheduling workflows with different shapes and number of activities;
  - reduced number of resources;
  - different types of resources (i.e. different clock frequency and different levels of static energy consumption);
  - workflows composed of different activity types.

5.1. Experimental Setup

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

5.1.1. Workflows

We conduct our evaluation using a real scientific workflow called the Persistence Of Vision Raytracer (POV-Ray) workflow [1], which is a free tool for creating three-dimensional graphics, 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 employ four types of synthetic workflows with different shapes and lengths using the workflow generator described in [24]:

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

Table 3: The two-dimensional evaluation space, where the focus parameter of each evaluation is highlighted.

Section	***Resource setup***		***Workflow type***		
	Size[i]	*Type*[ii]	*Shape*	*Size*	*Characteristic*[iii]
5.2	∞	uniform	all	20-200	(100%:0%)
5.3	∞	uniform	all	**20-200**	**uniform**
5.4	**10%-75%**	uniform	all	200	(100%:0%)
5.5	∞	**frequency**	all	200	(100%:0%)
5.6	∞	**static power**	all	200	(100%:0%)
5.7	∞	uniform	all	200	**(100..10%:0..90%)**

[i] ∞ denotes more than ten times more cores than activities; x% represents a setup with only x% of the resources utilised in Pareto-optimal solutions [ii] Focus parameter varied in three classes: *high*, *medium* and *low*, or *"uniform"* for uniformly distributed parameters [iii] (x%:y%) denotes activities with a ratio of x% computation to y% I/O operations

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 executions results along with the synthetic workflows of its corresponding type.

5.1.2. Resources

We developed a synthetic resource generator which takes as input CPU descriptions 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, 2.0G Hz median, and 2.6 GHz maximum, it will generate a set of resources whose CPU frequencies vary between 1.8 GHz and 2.6 GHz, with a higher agglomeration around the 2.0GHz value. The degree of affinity towards the median value is controlled through a fourth scale parameter. We generated in our evaluation seven types of resource setups: one with all the values of the machine characteristics uniformly distributed, three with emphasis on machines with high, medium and low CPU frequencies, and three with emphasis on machines with high, medium, and low idle power consumption $P_{R_j}^{(s)}$.

Table 3 summarises the coverage of our two-dimensional evaluation space, consisting of various resource setups and workflow types. We define a resource

setup through two parameters: the number of machines and the type of setup described above. We define a workflow setup through the two characteristics defined in Section 5.1.1: workflow shape, activity characteristics and number of activities.

5.2. Experiment-1: MOHEFT versus HEFT and greenHEFT

In the first experiment, we compare the results computed by the MOHEFT algorithm with the mono-objective versions for optimising makespan (HEFT) and energy consumption (greenHEFT). We consider workflows having between 20 and 200 activities and different heterogeneous systems with 10 – 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 4 and 5. 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 the results obtained by both algorithms have been the same.

Compared to HEFT, the results in Table 4 show that MOHEFT is at least as good in a large number of cases, and often even better. For example, MOHEFT reported a makespan 1.1% faster 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, selects in every iteration the resource that minimises 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 and leading to better solutions in several cases. As expected, MOHEFT computed 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 independent activities, which also increases with the number of activities. For the other three workflow types, the improvements are not influenced by the number of workflow activities.

With respect to energy consumption, we observe in Table 5 that MOHEFT also found workflow schedules with lower energy consumption than greenHEFT, again due to better search space exploration. As expected, MOHEFT 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. Similarly to the makespan objective, 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 mono-objective greedy mono-objective algorithms perform poorly in terms of energy-efficiency in distributed system with multi-core processor architectures. Overall MOHEFT always provides the best solutions for both

Table 4: 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 5: 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 / –

makespan and energy objectives, matching the individual solutions provided by HEFT and greenHEFT, and in many cases even improving on them.

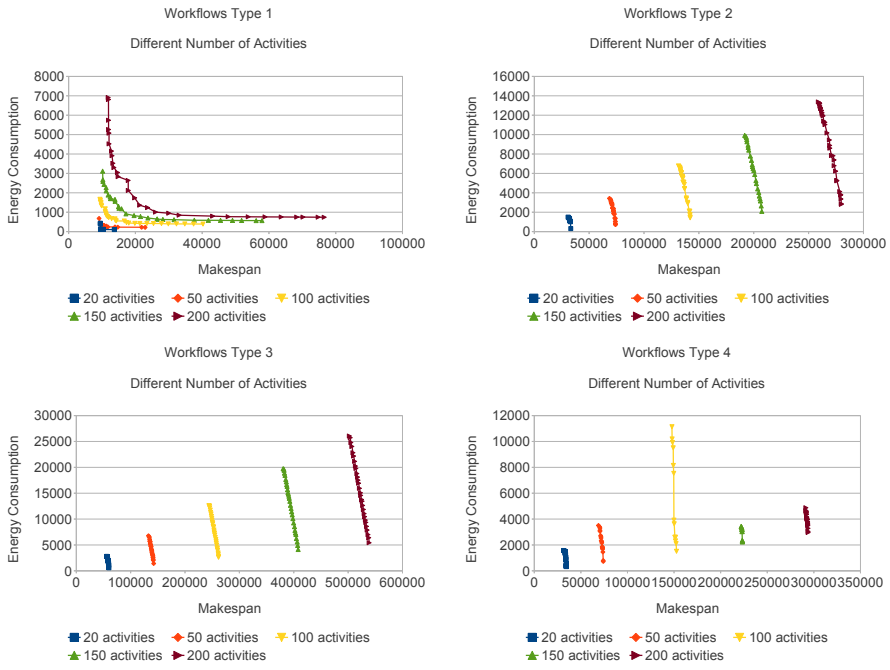

Figure 5: Pareto tradeoff solutions computed by MOHEFT for workflows of four types and different number of activities.

5.3. Experiment-2: 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. For all experiments in this section, we considered that the workflows are composed of $20 - 200$ activities with the $100\% : 0\%$ characteristic. We further considered that the number of heterogeneous resources is much larger than the number of tasks.

The results in Figure 5 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 Pareto set of tradeoff solutions has a convex shape, which means that it will be possible to find schedules which significantly improve the energy consumption with only a minimally 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 increase in makespan of only 2% versus the best 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 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. Concretely, this signifies that for every Whr saved, the workflow makespan will double. In the case of *Type-3* workflows, the slope is around -0.6. Finally, the shape of the fronts

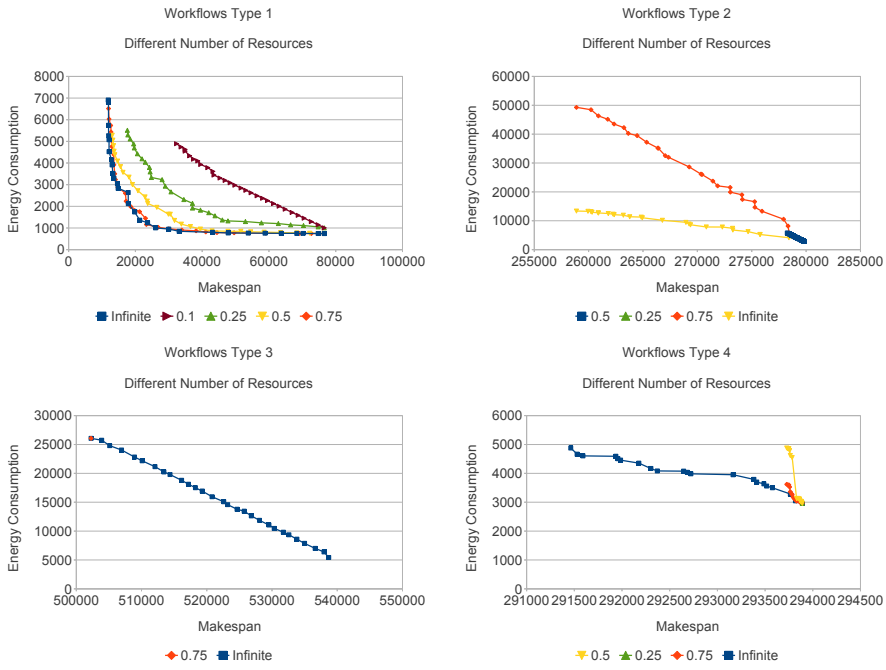

Figure 6: Pareto tradeoff solutions computed by MOHEFT for different number of resources.

computed for workflows of *Type-4* are neither convex nor linear, but rather jagged with constant fronts and sudden high jumps (not evident in Figure 5 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).

5.4. Experiment-3: Impact of the Number of Resources

This set of experiments is intended to analyse the influence of the number of resources on the MOHEFT results. For each analysed workflow, we considered the resources on which at least one activity has been scheduled by any of the tradeoff solutions computed in the previous section. Then, we reduced this pool to 75%, 50%, 25%, and 10% of its original size by randomly removing resources. For these experiments, we consider that the number of activities is 200 in all the cases and the number of resources is much larger than the number of tasks. For some workflows we have omitted the results with 50%, 25% or 10% of the resources because the obtained solutions had exactly the same values. The results summarised in Figure 6 show that reducing the number of resources also modifies the schedules computed by MOHEFT. In the case of the *Type-1* workflow, a reduction in the number of resources gradually transforms the Pareto front from the initial convex shape towards a linear shape. For *Type-2* workflows the shape of the Pareto front is linear and does not change, but

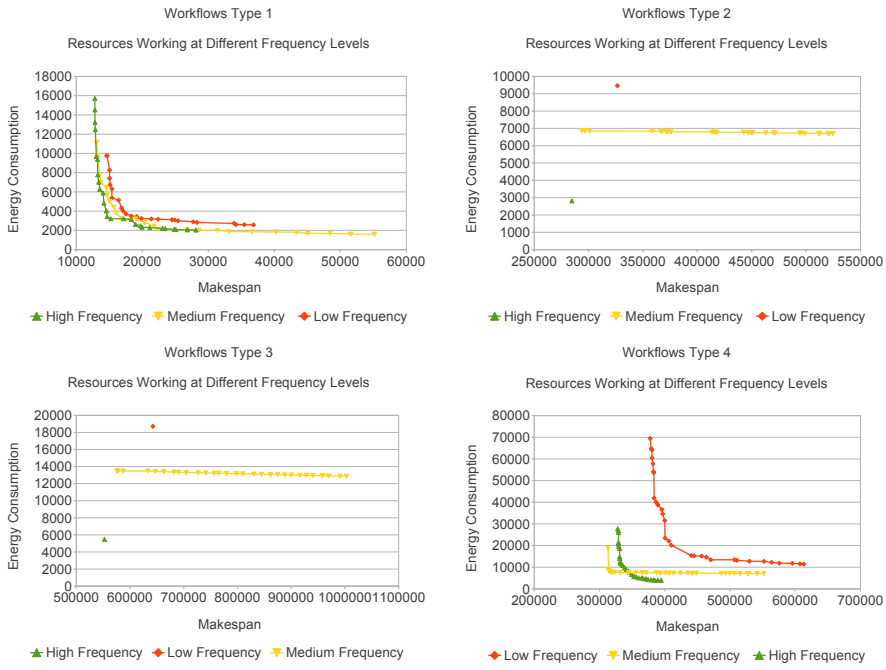

Figure 7: Pareto tradeoff solutions computed by MOHEFT for different *low*, *medium* and *high* resource clock frequencies.

its slope increases, indicating an increasing difficulty in optimizing the energy consumption. In the case of *Type-3* workflows, reducing the resource setup size results in the reduction of the Pareto front to just one solution. Finally, in *Type-4* workflows, a similar effect as with *Type-2* workflows can be observed, the Pareto front changing towards a higher difficulty for computing tradeoff solutions as the the size of the resource setup decreases.

The main finding of this experiment is that the reduction in the number of resources leads to schedules with a higher energy consumption and an increasing difficulty in optimising for energy consumption. This is a consequence of the smaller probability of finding energy-efficient resources in a limited resource pool.

5.5. Experiment-4: Impact of the Resource Clock Frequency

As presented in the introduction, the clock frequency influences the energy consumed by a CPU and the number of instructions per second processed. In this section, we analyse the impact of resource clock frequency on the MOHEFT Pareto solutions. We considered resources working at three different frequency levels: *low*, *medium*, and *high*. In the *low* level, resources are homogeneous and work at 1.6 GHz, in the *medium* level they are heterogeneous and work in the range 2 – 2.5 GHz, and in the *high* level they are again homogeneous and work

at 3.5 GHz. We set the number of workflow activities to 200 with the (100%:0%) characteristic.

We observe in Figure 7 that the clock frequency influences the shape of the Pareto fronts for some workflows. In the case of workflows of *Type-1*, the shape of the Pareto front is maintained independently of the resource clock frequency. With the increase in clock frequency, we notice a clear improvement in both makespan and energy consumption for in all studied cases. Of particular interest in these experiments are the solutions computed for the workflows *Type-2* and *Type-3*. For resources with homogeneous clock frequencies, MOHEFT was not able to find any tradeoff between makespan and energy consumption (see 7 for these workflow types). This stems from the fact that MOHEFT has very few distinct alternatives in terms of resources, thus, little makespan improvement potential. In case the resources are homogeneous and operate a *low* frequency, a single solution with high makespan and high energy consumption has been found. Conversely, with homogeneous resources operating at a *high* frequency, only one solution with low makespan and energy consumption has been computed. The energy consumption is strongly influenced by the workflow execution time due to the static power consumption ($P_{R_j}^{(s)}$). In complex *Type-4* workflows, an interesting observation can be made (based on Figure 7): the makespan of the computed Pareto fronts is not proportional to the average frequency of the available resources. More concretely, a higher frequency does not always guarantee better makespan solutions, as shown by the better makespan obtained in some solutions for the medium frequency than for the high one.

5.6. Experiment-5: Impact of Static Energy Consumption at Idle State

As defined in Section 2, the energy consumption is composed of two terms: static and dynamic. To analyse the impact of the static energy, we consider resources with three different levels of energy consumption at idle status: *low*, *medium*, and *high*. Resources operating at *low* level consume approximately 20 Watt per hour at idle state, at *medium* level around 120 Watts per hour, and at high level around 400 Watts per hour. We consider in this new experiment the same workflows as in the Experiment-4.

We observe in Figure 8 that the savings in total energy are really small compared to makespan if the static energy consumed in idle state is homogeneous. This pattern is common to almost all the workflows, except the *Type-1* workflows for machines with *high* and *medium* static energy consumptions where the computed fronts are again convex, like in Experiment-1. Thus, as happened in that case, high improvements in energy consumption can be achieved by small increases in the workflow makespan. In the rest of cases, the computed Pareto fronts are linear with a slope close to zero. Obviously, optimising for energy does not bring any benefit in these cases.

5.7. Experiment-6: 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

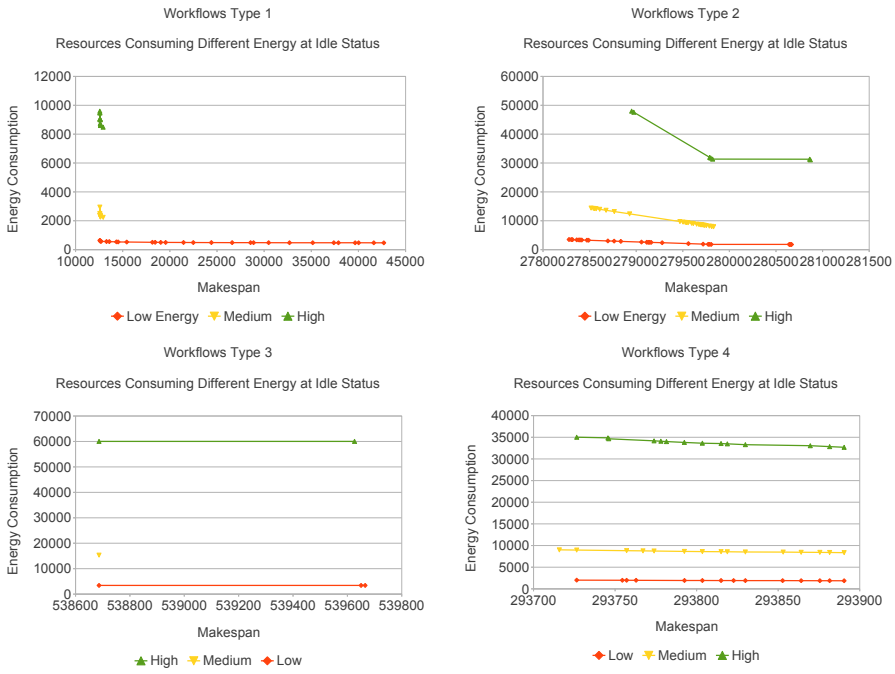

Figure 8: Pareto tradeoff solutions computed by MOHEFT for resource with different static energy consumption at idle state.

computation with periods waiting for I/O operations. Specifically, we consider activities using the CPU 25%, 50%, 75%, and 100% of their execution and waiting for I/O operations the rest of the time. We used the same set of resources as in Experiment-2.

The results in Figure 9 show that, the higher the CPU utilisation, the shorter the makespan and the lower the energy consumption are. The explanation for this behaviour lays in the same amount of instructions that has to be 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 changes the shape of the Pareto fronts for all the workflows except *Type-1*. These changes are mostly visible when the activities use the CPU only for a small fraction of time. The shape of the computed fronts for the other workflow types consists on two parts, as displayed in Figure 8. Both parts represent a linear dependence between makespan and energy consumption. The first part allows for high improvements on energy consumption with only small makespan overhead; meanwhile, the second part involves higher makespan overheads for every Whr saved.

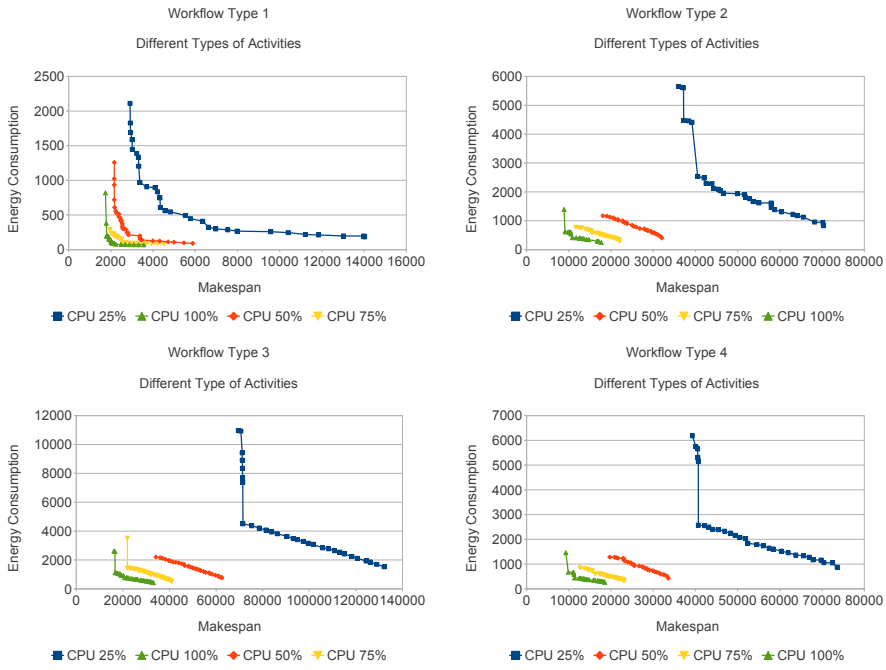

Figure 9: Pareto tradeoff solutions computed by MOHEFT for workflow activities with different CPU and I/O utilisations.

5.8. Summary

Table 6 summarises the main results of our experiments which targeted the evaluation of the MOHEFT behaviour in different scenarios with the goal to evaluate its response to changes in workflow type, 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. In our experiments we have identified 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, hence constant, proportional tradeoffs between energy efficiency and makespan. Horizontal Pareto fronts represent a particular case of the sloped linear fronts with the slope very close to zero and denote situations in which only minute improvements on energy efficiency can be obtained with a huge negative impact on makespan. The particular Pareto fronts which do not belong to any of these generic classes are denoted "not classified" in the summary table (Table 6), and were individually analysed when encountered in our evaluation.

Table 6: Summary of results.

Exp.	*Type-1*	*Type-2*	*Type-3*	*Type-4*	*Conclusion*
Exp.-2	convex	linear	linear	not classified	Best energy-makespan tradeoffs obtained for workflows with high parallelism and large number of tasks
Exp.-3	convex to linear	linear	linear	not classified	Reducing the number of available resources increases makespan and energy consumption, but tradeoffs are still possible
Exp.-4	convex	horizontal	horizontal	not classified	Higher speed resources are preferred for both makespan and energy consumption optimisation
Exp.-5	convex / horizontal	horizontal	horizontal	horizontal	Static power consumption of resources strongly influences the energy efficiency of the schedules and presents higher energy-makespan tradeoff potential than dynamic power consumption
Exp.-6	convex	linear	linear	linear	I/O-intensive workflows are expensive in terms of both energy and makespan, but also present a wider range of possible energy-makespan tradeoffs

6. Related Work

In this work, we empirically modelled the energy consumed by a distributed system based on realistic data. Then, we presented a bi-objective workflow scheduling algorithm for makespan and energy optimisation. As a consequence, we made contributions three different fields: energy modelling, and multi-objective workflow-scheduling, and energy-efficient scheduling. We analysed the energy modelling related work in Section 3. We review in this section the existing works in the areas of multi-objective and energy-efficient scheduling.

6.1. Multi-Objective Workflow Scheduling

Most of the existing works in multi-objective workflow scheduling combined the different criteria in a single optimisation goal. Usually, this combination consists of a weighted aggregation of the different optimization criteria, where the weights are aimed at expressing user preferences over the objectives. The main difference among existing works is in the way in which the preferences are expressed. For example, in [10] reliability (in terms of resource failures) and makespan are combined using a weight vector which is provided by the user. The same objectives are optimised in [11] and [2]. 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 has been exploited in other works, for example in [21] 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 constrain. 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. This fact implies that, 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. Additionally, the objective functions require to be normalized to the same interval in to properly capture these preferences, which requires the optimal solutions in terms of each objective to be known. Concerning to constraint-based approaches, the main drawback is that reasonable a-priori values for the constraints are often unknown until the first schedule is computed.

Only few approaches compute the whole set of tradeoff solution among the different objectives. Among these approaches, we can distinguish two types of techniques: genetic algorithms and list heuristic-based techniques. Example of applications of genetic algorithm for multi-objective workflow scheduling are [25] and [19]. The former optimises the makespan and cost of executing tasks in a cloud system, while the latter optimises for makespan and energy consumption using DVFS techniques, as will be commented later in this section. Genetic algorithms are able of computing high quality solutions, but require significantly higher computation time. One possible approach for overcoming these difficulties to initialized the algorithms with known good solutions, as presented in [25]. List-based heuristics have been applied in [5] and [9] for optimising makespan and cost of workflows.

6.2. Energy-Efficient Scheduling

There exists extensive literature in energy-efficient scheduling, most of them applying DVFS for tuning the energy consumed. Among the different approaches, we identified different lines of research.

One line consists in the application of genetic algorithms [19, 16, 17]. While applying different variants of genetic algorithms, these algorithms have in common the use of individuals representing the mapping of tasks onto machines, and the voltage at which each machine should operate. Alongside genetic algorithms, greedy heuristics have also been of interest [8, 18]. A disadvantage of genetic and greedy heuristics is that the machines' voltage is not changed during the computations, but maintained for the whole execution once it has been set to a specific level. From our point of view, there is room of research in this area by proposing techniques able of dynamically changing the voltage during the execution of applications.

A different line of research [20] 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 found 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 resources to sleep mode resources which are not used [15], which reduces the static energy consumed in idle mode.

To the best of our knowledge, our work is the first truly multi-objective approach able to compute a set of tradeoff solutions for optimizing makespan and energy consumption in distributed systems. Our method also incorporates power-down techniques by accounting only for the energy consumed by the machines doing real computation. Although we do not use DVFS for reducing

the energy consumed by a CPU, these techniques can be combined in a hybrid approach.

7. Conclusions and Future Work

This paper tackles the problem of energy-efficient workflow scheduling in distributed heterogeneous systems. Our approach consists in a multi-objective list-based heuristic, called MOHEFT, able to compute tradeoff solutions between makespan and energy consumption. This algorithm relies on empirical models for predicting the energy consumption and execution time of workflow activities. These empirical models have been designed based on historical data collected from real workflow task executions.

We have compared MOHEFT with HEFT, the state-of-the-art, scheduling algorithm for optimizing makespan, and greenHEFT, a customized version of HEFT for optimizing energy that we introduce in this paper. Trough extensive experiments, we have shown that our proposed algorithm is at least as good as the other two algorithms in all evaluated scenarios, and performing better in many cases. Furthermore, we have analysed how MOHEFT behaves in different conditions, namely its response to changes in different characteristics of the workflows, activities and underlying resources. For each of the studied scenarios we evaluate the viability of energy efficiency optimisations as well as the corresponding tradeoffs in makespan.

In future work we plan to further include other objectives in our multi-objective scheduling algorithm, such as economical cost of workflow executions, which is of particular interest in the context of the emerging commercial clouds.

References

[1] `http://www.povray.org/`.

[2] I. Assayad, A. Girault, and 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*, Firenze, Italy, June 2003. IEEE.

[3] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1 edition, January 1996.

[4] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, pages 294 –295, 466, 2000.

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

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

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[8] C.O. Diaz, M. Guzek, J.E. Pecero, G. Danoy, P. Bouvry, and S.U. Khan. Energy-aware fast scheduling heuristics in heterogeneous computing systems. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 478 –484, july 2011.

[9] J.J. Durillo, H.M. Fard, and 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.

[10] Saurabh Kumar Garg, Rajkumar Buyya, and 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, pages 151–160, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc.

[11] Mourad Hakem and Franck Butelle. Reliability and scheduling on systems subject to failures. In *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP '07, pages 38–, Washington, DC, USA, 2007. IEEE Computer Society.

[12] James Hamilton. Cooperative expendable micro-slice servers (cems): Low cost, low power servers for internet-scale services.

[13] Eric Humenay, David Tarjan, and Kevin Skadron. Impact of process variations on multicore performance symmetry. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pages 1653–1658, San Jose, CA, USA, 2007. EDA Consortium.

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

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

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

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

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

[19] 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. *Journal of Parallel and Distributed Computing*, (71):1497–1508, 2011.

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

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

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

[23] Xiaofeng Wang, Rajkumar Buyya, and 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, pages 108–115, Washington, DC, USA, 2009. IEEE Computer Society.

[24] J. Yu, R. Buyya, and K. Ramamohanarao. Workflow scheduling algorithms for grid computing. In F. Xhafa and A. Abraham, editors, *Metaheuristics for Scheduling in Distributed Computing Environments*, pages 109–153. Springer Berlin, 2008.

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