PhD-FSTC-2014-33
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 02/09/2014 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Cesar DIAZ
Born on 21 April 1975 in Villavicencio, (Colombia)

ENERGY-EFFICIENT SCHEDULING IN GRID COMPUTING AND RESOURCE ALLOCATION IN OPPORTUNISTIC CLOUD COMPUTING: MODELS AND ALGORITHMS

Dissertation defense committee

Dr Pascal Bouvry, dissertation supervisor
Professor, Université du Luxembourg

Dr Johnatan E. Pecero
Research Assistant, Université du Luxembourg

Dr Frederic Guinand, Chairman
Professor, Université du Havre, Le Havre, France

Dr Prof. Imed Kacem
Professor, Université Paul Verlaine, Metz, France.

Abstract

Resource allocation among Heterogenous Computing Systems (HCS) components, such as cluster, grid, or cloud computing can be considered as a service. These systems manage millions of computational resources to solve several difficult computational problems. Resource allocation and scheduling among these systems are still a hot topic for research purposes. A goal of this research is to find an efficient use of these resources proposing a resource allocation and efficient scheduling techniques. Firstly, the relevance of energy consumption in processing elements as well as techniques and policies to support it are presented. It emphasizes in resource allocation algorithms in opportunistic environments and low complexity scheduling heuristics in grid computing environment. In particular, a series of low complexity, scalable, and energy-efficient algorithms for scheduling in grid computing and a resource allocation technique for opportunistic environment are presented. The latest aforementioned technique was evaluated in an opportunistic cloud environment. Three fast and energy-efficient batch mode scheduling novel heuristics were designed, developed, and evaluated to produce fast tasks mapping in HCS. To fully understand their capabilities and limitations, these aforementioned heuristics were studied and compared with a variety of system parameters for their performance and scalability.

To Mery and Miguel

Declaration of Authorship

I, Cesar Diaz, declare that this thesis titled, *"Energy-efficient scheduling in grid computing and resource allocation in opportunistic cloud computing: models and algorithms"* and the work presented in it are my own.

I confirm that:

- This work was done wholly while in candidature for a research degree at the University of Luxembourg.
- No part of this thesis has previously been submitted for a degree or any other qualification at the University of Luxembourg, or any other institution.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Contents

Chapter 1

Introduction

Contents

Heterogenous Computing Systems (HCS) such as cluster, grid or cloud, can employ thousands or even millions of computational resources with different performance capabilities and cost-effective to solve several difficult problems in multiple application domains in research and industries. HCSs are widely used as powerful parallel and distributed platforms for executing applications requiring a large amount of computing power and/or data. Therefore, an efficient use of the resources is a critical issue for performance guarantees.

HCS with a large number of computing resources incurs in huge amount of energy consumption and this trend will continue without properly energy consumption optimization. The optimization on the energy waste is a major concern of interest, especially on cloud data centers. Since an efficient reduction in the consumption of energy brings benefits not only at the business and system's performance levels, but also at the social level by minimizing the impact on the environment for example in the air quality, CO2 emission and climate change. This issue forces HCS managers to adopt innovative ways to improve energy efficiency.

There is significant research effort addressing reduction of energy consumption of today's HCS systems. Engineers and researchers make advances in different aspects of the domain to ensure increasing energy optimization. Practical approaches in many HCS facilities, at present, adopt energy management measures at hardware levels. At this level HCS managers look for energy saving solutions to installing low-power cooling devices and heat sinks, low-power battery computers, using low-power processors, choosing products or devices whose performance match their energy requirements, enhancing resource and infrastructure maintenance, better components of the system even with green and clean energy suppliers, deployment of specific technologies such as UPS, chillers, economizers.

Nowadays, compared with low-energy hardware saving approaches, low-energy software based approaches are practical and commonly used. Focusing on solutions to the hardware level as a first step in the optimization process of energy consumption is natural and central,

however, it is well known that even with optimized hardware, poor software can lead to important energy waste. HCS managers are looking for significant improvements in energy efficiency, at present, on software based solutions. One of the main concern is to provide high performance with low energy consumption, that is to provide energy efficiency. Researchers are working on rethinking algorithms and applications, improving software design, developing and implementing energy-aware resource allocation and scheduling policies. It is in this context that this research work has been investigated.

This thesis concern the design, analysis and development of efficient heuristics dealing with the resource management, specifically with the scheduling problem in HCS considering performance and energy issues. This dissertation addresses the following main hypothesis:

- If I take advantage of the heterogeneity resources of a HCS by designing efficient scheduling strategies, it will reduce energy consumption without sacrificing the performance of the system.
- If an application is designed with low-complexity, then energy savings will be obtained.
- If a resource allocation strategy considering resource consolidation is designed taking advantage of busy or idle desktop resources, then the reduction of energy will be obtained.

The first two hypothesis concern HCS based on Grid computing and the last hypothesis mainly concern an opportunistic cloud-based infrastructure composed of private desktop computing resources.

1.1 Motivations

Over the last years, HCS have become an increasingly important part of most business applications. These systems have been continuously upgrading to high performance computing systems meeting the increasing demands of computational power by many scientific and industrial applications. These systems are also designed to satisfy variable pick loads and most of HCS are overprovisioned. Unfortunately, this traditional practice in HCS of overprovisioning the physical infrastructure has a very negative impact overall HCS efficiency and therefore causes resource underutilization. Hundred or thousands of resources (mainly servers) are idling most of the time (plugged in 24 hours a day but that are rarely utilized) wasting considerable amount of energy, which in turn increases HCS operational costs and electricity bills, raises social and environmental concerns, and decreases performance of the system.

It is worth to notice the negative impact of energy consumption inefficiency. In terms of monetary costs, the energy consumption of computing equipment (*"67 billion kilowatt-hours from the grid in 2010, according to Census Bureau figures reviewed by the Electric Power Research Institute for The Times"*[1]) is costly and runs to several thousands dollars per year. The report to congress on server and data center energy efficiency by the U.S. environmental protection agency ENERGY STAR program [137] estimated that the energy

[1]Taken from *NewYork Times* "Power, Pollution, and The Internet" by James Glanz. Published September 22, 2012

used by the servers and data centers consumed about 61 billion kilowatt-hours in 2006 for a total electricity cost of about $4.5 billion. System's performance of HCS facilities is also concerned with energy consumption. Heat dissipation due to the huge number of transistors integrated into current processors architectures, which reaches to nearly one billion, has a severe impact on the overall performance worsening system reliability resulting not only in system performance degradation, but also in high energy bill for air-conditioning system and heat removal devices. For example, the same report mention that a fully populated rack of blade servers requires up to 20-25 kW of power to operate, and it is expected to require an additional 20-25 kW of power for the cooling and power conversion equipment to supports it. Finally, no less important and associated to the previous issues is the impact that CO2 emissions due to energy consumption and heat dissipation may have on the society and environment. Large-scale HCS systems are becoming big polluters. The CO2 emissions attributable to HCS and large server farms are increasing, and it is comparable to the levels produced by the global airline industry. The Uptime Institute [83] estimates that carbon dioxide emissions will quadruple between 2010 and 2020. Air quality and climate change are concerned by CO2 emissions having a big impact in life quality of society.

In this thesis, I focus on developing batch mode scheduling heuristics for HCSs to reduce the energy consumption of the system without compromise the performance. Moreover, I also focus on how some strategies in resource allocation for opportunistic environments may reduce energy consumption.

1.2 Methodology

To know the real energy consumption of one physical machine, a *wattmeter* was connected to one of the computers lab used by this research work. Figure 1.1 shows the behavior of the CPU when is running a *Java* program which ensure 100% use of the cores during five minutes. The processor has in total eight cores, four physical cores and four virtual cores. The average energy value was measured during these time for each processor state.

As previously mentioned, the optimization of energy consumption can be handled across different levels of the solution stack in hardware, software and system design [41, 128]. In this research work, the main focus is on HCS management software by resource allocation and scheduling policies. HCS management software tools can help to optimize energy consumption as they facilitate the implementation of different policies throughout the HCS and provide features like provisioning, monitoring, deployment, increase resource utilization, workload management, resources heterogeneity exploitation and configuration management that support system efficiency [128].

Scheduling algorithms and resource allocation strategies can be designed by using sophisticated frameworks [117]. The most common are approximation algorithms frameworks, meta-heuristics frameworks, parallel frameworks. In modern HCS there is a variability in the load increase because workloads are often very unstable. There is a necessary trade-off between reacting as fast as possible to minimize the duration when the application underperforms because of insufficient resource capacities, and a slower approach to avoid situations where the load has already decreased when computing resources become available. HCS managers prefer simple mechanism than sophisticated implementations [63]. The main reasons for such preference can be that the gains of using sophisticated approaches are low to be worth

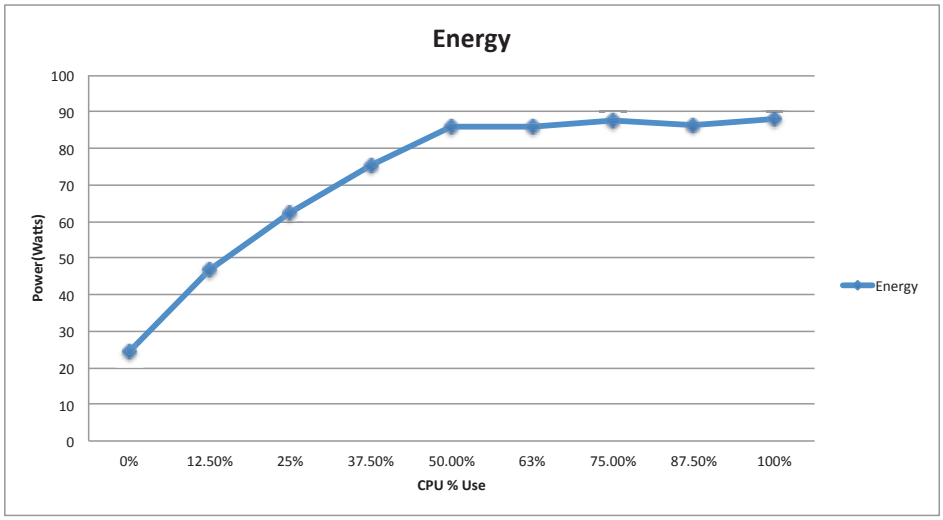


Figure 1.1: Energy measured by a physical device.

the effort. Other possible reason is that implementing and evaluating these techniques is a difficult exercise, this is the reason that real HCS rely on simpler techniques such that it can be easy their adaptation and implementation into real HCS. In this dissertation, it is adopted an approach based on designing very simple mechanisms. Regarding energy efficiency, sophisticated scheduling techniques need more computational time and effort to compute and execute the planned schedule and it is translated to increase the energy consumption, this thesis opted for low-complexity policies are designed in this research work.

In large HCS systems, different components of the computing facilities contribute to the energy consumption. The main components, beside the building facilities, are CPU, memory units, communication paths or networks, storage, and others. The optimization of energy consumption should be done in all of these elements in a holistic approach. In Figure 1.2 shows the energy consumption of the processor, following the same measuring setup of Figure 1.1, it suffices to prove that the processor have the major part of the energy consumed by the CPU, up to 50% of the total consumption. For this reason, in this thesis, the focus is on processing elements, which is still one of the main resources that consumes a major part of the total energy used by a computer system.

One of the methodologies used to validate the performance of the scheduling heuristics proposed in this work is the approximation factor. It is also used to qualify the efficiency of the scheduling algorithms. The approximation factor of a strategy is defined as the ratio of the reference objective to the optimal one. As in most of real hard problems, it is difficult or even almost impossible to determine the optimal solution, a lower bound of the target objective is considered. The approximation factor allows to show that the computed solution obtained by the evaluated heuristics does not exceed the optimal value for any problem instances by more than a certain factor. The analysis is conducted as follows. First, the performance ratio (i.e., relative error) of the approximation factor of each strategy under each metric is evaluated. This is done relative to the best performing strategy for each metric. Thereafter, the performance ratios to evaluate performance of the strategies is averaged, and show if some strategies tend to dominate results in all of the test cases, with the expectation that it

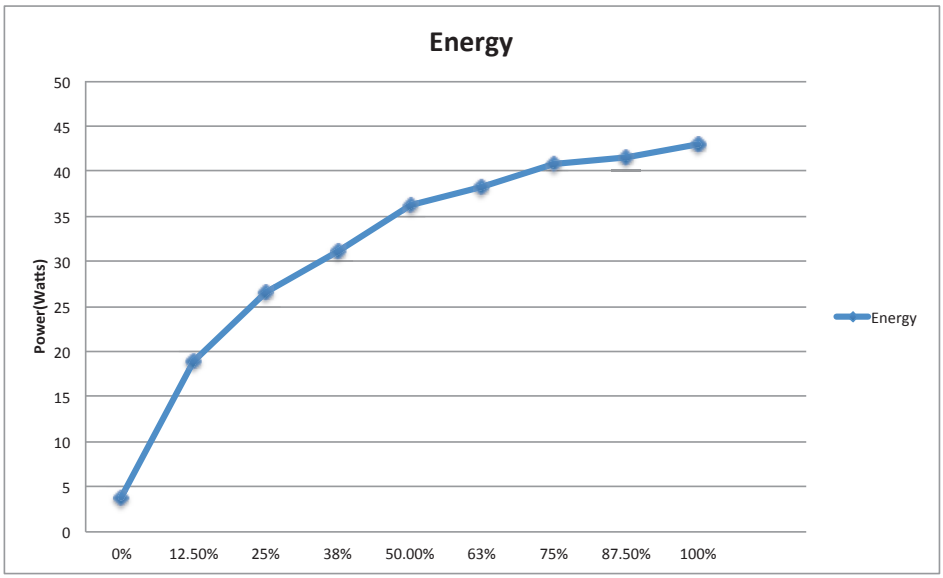


Figure 1.2: Energy measured by a program called *intel®power gadget 3.0*. The program provides the power consumption of the processor with a 20 seconds sample rate.

will also perform well under other conditions. For example, with different HCS configurations and workloads. However, they do not show the negative effects of allowing a small portion of the problem solutions to dominate the conclusions. To analyze the possible negative effects of allowing a small portion of the problem instances with large deviation to dominate the conclusions, and to help with the interpretation of the data generated by the benchmarking process, it is also used performance profiles of the strategies. The performance profile is a non-decreasing, piecewise constant function, which presents the probability that a performance ratio is within a factor of the best ratio [53, 54]. The given function is the cumulative distribution function. Strategies with large probability for smaller ratio are to be preferred.

The workload considered to evaluate our proposed scheduling heuristic is based on *Expected Time to Compute* model (ETC) presented by Ali et al. [8]. Each task is considered as an indivisible unit of workload and must be processed completely in a single machine without interruptions. The tasks waiting for its execution will be scheduled in a batch mode. Batch mode scheduling strategies are used when jobs have to be scheduled after predefined time intervals [26]. They are simple, powerful, and efficient in contrast to more sophisticated schedulers that need longer execution time to compute the mapping. Moreover, these strategies take advantage of using job and resource characteristics in deciding which job to assign to which machine. All of the machines are idle and available at time zero, which is possible by considering an advance reservation. To present the heterogeneity of tasks and machines, a variety of ETC matrices are generated. The model is implemented by the *coefficient-of-variation* (COV) based method [8] considering different machine and task heterogeneities. In this model is consider the relation of the tasks and machines as well, by introduced consistency scenarios.

Alternatively, to evaluate the resource allocation algorithms over opportunistic cloud environments, real workloads were considered.

1.3 List of contributions

The major contributions contained in this PhD thesis include:

1. Three energy-aware low computational complexity scheduling algorithms are proposed, designed, and evaluated to exploit the heterogeneity of the resources and applications emphasize in performance, complexity, energy consumption, and scalability in HCS. The algorithms are evaluated and compared with the best reported in the literature. The set of experimental results shows that low computational complexity heuristics perform as efficiently as known ones considering the makespan criterion, and outperform them in terms of flowtime, runtime execution, memory used, and number of best solution found criteria. Detail analysis show that low computational complexity heuristics are the best performing heuristics in the original-consistency cases showing similar behavior in the partial-consistency scenarios for makespan.

2. A policy for resource allocation modeling used by *UnaGrid* is proposed and evaluated in order to reduce the energy consumption of the opportunistic grid environment. *UnaGrid* is an opportunistic infrastructure developed and implemented at University of Los Andes, whose main purpose is to provide a virtualization mechanism that guarantees the properties aforementioned. Opportunistic grids are a good alternative to develop high performance computing. Virtualization is a good approach in order to provide a complete isolation between both environments computing and end-user ones.

3. An energy model based on the experimentation, first in opportunistic grid and cloud environments is presented and validated.

4. Energy savings in resource allocation on an opportunistic infrastructure, specifically for *UnaCloud* are presented. A new algorithm is proposed that is a variant of First-Fit Decreasing, it places the VMs first on PMs already in use by a physical user. The proposed algorithm have been validated against three well known state-of-the-art heuristics. Different scenarios using real workload traces have been simulated.

1.4 Dissertation outline

The remainder of this document is organized as follows: This first part of the dissertation starts with an introduction to services in Chapter 1. A brief exposition of scheduling and resource allocation algorithms in opportunistic environment is presenting in Chapter 2. Chapter 3 provides a detailed exposition of our proposed low complexity scheduling heuristics. A detailed description of Opportunistic cloud computing, ranging from Desktop Computing to Volunteer Computing is presented in Chapter 4. Chapter 5 describes the particular Infrastructure where the experiments were developed, furthermore, shows all the process and results of the infrastructure benchmarking. Chapter 6 report some energy savings on a particular opportunistic cloud infrastructure focus on a consolidation algorithm with special policies. This manuscript concludes the work presented during this dissertation in Chapter 7, presenting also some future work.

Chapter 2

Scheduling Heuristics

Contents

2.1 Introduction

Resource allocation is the process to allocate resources in the appropriate way according to an objective function. Scheduling can be understood in general as allocating resources over time. The scheduler is a key component for performance guarantees. Its aim is to assign tasks to computing resources considering system- and user-centric objectives and satisfying given efficiency criteria, usually related to the total execution time, resource utilization, economic cost, energy constraints, etc. The scheduling problem is NP-hard [82], even in the case of homogeneous machine systems and applications without precedence constraints. Moreover, reducing energy consumption for heterogenous computing systems can bring benefits such as reducing operating costs, increasing system reliability, and environmental aspect. It is crucial to search low computational complexity algorithms to schedule tasks, and/or energy-aware resource allocation algorithms, to exploit the heterogeneity of the resources and applications. Low overhead is one of the key issues for large-scale problems. There are some situations in which only low complexity scheduling heuristic can be used, where the scheduling process is performed during the execution of the allocated tasks.

Consider that m *machines* M_j $(j = 1, ..., m)$ have to process n *tasks* T_i $(i = 1, ..., n)$. Problems of allocating resources over time, or allocation of resources to tasks over given time periods can be understood as scheduling problems. From now on we make the assumption that each machine can process at most one task at a time and that each task can be processed

on at most one machine at a time [73]. Scheduling problems can be classified, based on classification scheme introduced by Graham et al. [73], in three fields, **machine environment** denoted by α, **task characteristics** denoted by β and **optimality criteria** denoted by γ. Before define each field, the task data is specify. That is, *i.)* each T_i consists of a number of operations $O_{1i}, ..., O_{mi}$, associated with each operation O_{ij} there is a *processing time* p_{ij} to solve this operation and a set of machines $\mu_{ij} \subseteq \{M_1, ..., M_m\}$ where O_{ij} may be processed, usually, all μ_{ij} are one element sets (**dedicated machines**) or all μ_{ij} are equal to the set of all machines (**parallel machines**) [29]; *ii.)* a *release date* r_i, on which the first operation of T_i becomes available for processing; *iii.)* a *due date* d_i by which T_i should ideally be completed; *iv.)* a w_i, indicating importance or weight and *v.)* a *cost function* $f_i(\tau)$ which measures the cost of completing T_i at time τ.

Machine environment: is characterized by a string $\alpha = \alpha_1\alpha_2$ of two parameters, where α_1 is characterized by four values: $\alpha_1 \in \{\circ, P, Q, R\}$, $\circ$ denote the empty symbol, if task T_i consists of only one operation that can be processed on any M_j machine, the processing time will be p_i. If $\alpha_1 = \circ$, it is a dedicated machine, $p_{i1} = p_i$. If $\alpha_1 = P$, this is *identical parallel machines*, where the processing times are $p_{ij} = p_j\,(i = 1, ..., m)$; and if $\alpha_1 = Q$, it is *uniform parallel machines*, where processing times differ with a given *speed factor*= q_i of machine $M_j, (j = 1, ..., m)$, $p_{ij} = q_i p_j$; s_j the speed of the machine M_j so under conditions stated above, the speed factor is defined as $q_i = 1/s_j$, and the processing time $p_{ij} = \frac{p_j}{s_j}$.

Task characteristics: are specified by $\beta \subset \{\beta_1, \cdots, \beta_8\}$ which are defined as follows: $\beta_1 \in \{pmtn, \circ\}$. If $\beta_1 = pmtn$ indicates that **preemption** (task splitting) is allowed. This means that processing may be interrupted and resumed at a later time. If $\beta_1 = \circ$ No preemption is allowed. $\beta_2 \in \{res, res1, \circ\}$. If $\beta_2 = res1$: the presence of only a *single resource* is assumed. If $\beta_2 = res$: the presence of s limited *resources* $R_h(h = 1, \cdots, s)$ is assumed, each T_j requires the use of r_{hj} units of R_h at all times during its execution. $\beta_3 \in \{prec, tree, \circ\}$. If $\beta_3 = prec$: describes **precedence relations** between the tasks. It is derived from a directed acyclic graph $G = \{V, A\}$ in arbitrary form, where $V = \{1, \cdots, n\}$ vertex corresponds with the tasks, and $(j, k) \in A$, we write $T_j < T_k$ require that T_j must be completed before T_k starts. If $\beta_3 = tree$, then G is a *rooted tree* with an outdegree (all arcs are directed towards a root) at most one for each vertex or indegree (all arcs are away from a root) at most one for each vertex. If $\beta_3 = \circ$: no precedence relation is specified. $\beta_4 \in \{r_j, \circ\}$. If $\beta_4 = r_j$: then release dates may be specified for each task. If $r_j = 0$ then $\beta_4 = \circ$. $\beta_5 \in \{m_j \leq \overline{m}, \circ\}$. If $\beta_5 = m_j \leq \overline{m}$: with $\alpha_1 = J$, a *constant upper bound* on m_j is specified. If $\beta_5 = \circ$: no such bound is specified. $\beta_6 \in \{p_{ij} = 1, \underline{p} \leq p_{ij} \leq \overline{p}, \circ\}$. If $\beta_6 = p_{ij} = 1$: each operation has unit processing time. If $\beta_6 = \underline{p} \leq p_{ij} \leq \overline{p}$: constant lower and upper bounds on p_{ij} are specified. If $\beta_6 = \circ$: aforementioned bounds are not specified. $\beta_7 \in \{d_j, \circ\}$. If $beta_7 = d_j$, then a deadline d_j is specified for each task T_j, i.e. task T_i must finish no later than time d_i. Finally, in some scheduling problems, sets of tasks must be grouped into batches. A batch is a set of tasks which must be processed jointly on a machine. The completed time of all tasks in a batch is defined as equal to the completed time of the batch. We can assume that this set-up time is the same for all batches and sequence independent. Group the tasks into batches and schedule these batches are known as a batching problem. There are two types, denoted by *p-batching problems* (where the length of a batch is equal to the maximum of processing times of all tasks in the batch), and *s-batching problems* (where the length of a batch is equal to the sum of processing times

of all tasks in the batch). $\beta_8 \in \{p-batch, s-batch, \circ\}$ corresponding the first two of each aforementioned cases. If $\beta_8 = \circ$: no batch problem is specified.

Optimality criteria: The third field γ refers to the optimality criterion chosen. First, let us denote by C_j the **completion time** of task T_j (time that the task is completed), its associated cost by $f_j(C_j)$, and **flowtime** (time since the client submits a request until is completed) by $F_j = C_j - r_j$. There are two types of total cost functions $\gamma \in \{f_{max}, \sum f_j\}$.

$$f_{max} = max\{f_j(C_j) \mid j = 1, \cdots m\} \tag{2.1}$$

$$\sum f_j \in \left\{\sum C_j, \sum T_j, \sum U_j, \sum w_j C_j, \sum w_j T'_j, \sum w_j U_j\right\} \tag{2.2}$$

The scheduling problem is minimize the total cost function showed in Eqs. 2.1 and 2.2. The most common objective functions are:

- **makespan:** known as well as *schedule length* $C_{max} = \max\{C_j \mid j = 1, \cdots, n\}$ defined as the maximum completion time of the tasks.
- **total flowtime:** $F = \sum_{j=1}^{n} F_j$.
- **mean flowtime:** $\bar{F} = \frac{1}{n}\sum_{j=1}^{n} F_j$.
- **weighted (total) flowtime:** $F_w = \sum_{j=1}^{n} w_j F_j$.
- **mean weighted flowtime:** $\bar{F}_w = \frac{\sum_{j=1}^{n} w_j F_j}{\sum_{j=1}^{n} w_j}$

Other objective functions depend on due dates d_j which are associated with tasks T_j. We can also compute for each task:

- the *lateness* $L_j = C_j - d_j$;
- the *tardiness* $T'_j = \max\{0, C_j - d_j\}$;
- the *earliness* $E_j = \max\{0, d_j - C_j\}$;
- the *absolute deviation* $D_j = \mid C_j - d_j \mid$;
- the *squared deviation* $(C_j - d_j)^2$;
- the *unit penalty* $U_j = \begin{cases} 0 & if\ C_j \leq d_j \\ 1 & otherwise \end{cases}$

Figure 2.1 shows a possible schedule for n=3 tasks and m=1 machine. A scheduling algorithm is procedure to construct a schedule from an input instance.

Other interpretation about scheduling problems could be classify in deterministic scheduling problems, which no variable with a non-deterministic description appears. Alternatively, stochastic scheduling models, which any variable with a probabilistic description appears. Each, aforementioned, classification could be *static* or *dynamic*. Static, known as well as *offline*, the entire input instance is known in advance and the goal usually is to give an efficient algorithm that produces the optimum or near to optimum schedule. Dynamic, known as well as *online*, in which some parameters are unknown in advance, just it is revealed to the algorithm only when the task arrives, thus *online algorithm* at any time has to make decisions based on the partial knowledge of the tasks that have arrived by that time; this approach is even more realistic in many practical situations.

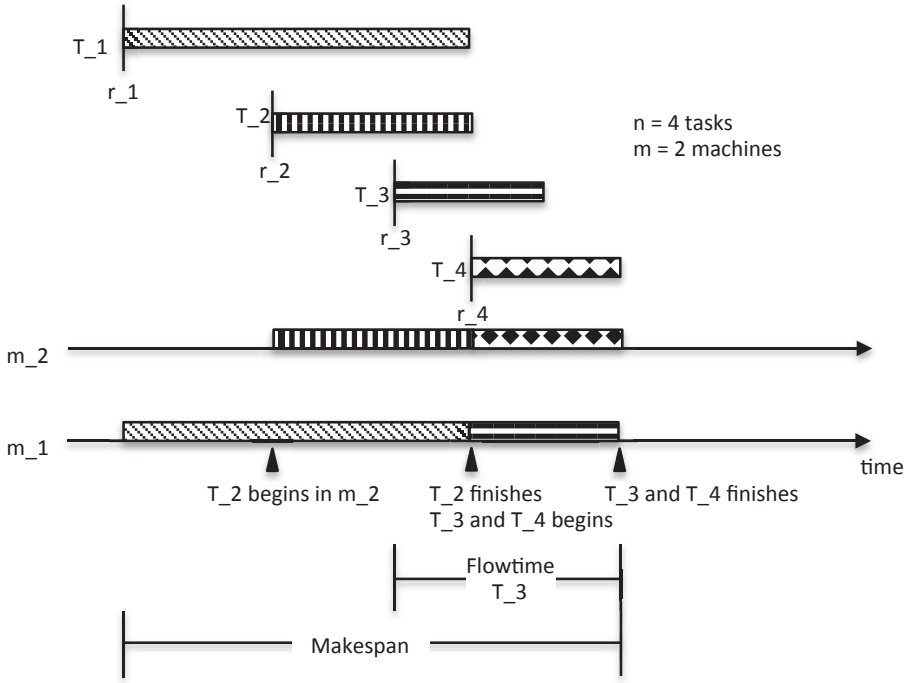


Figure 2.1: An example of a schedule

2.2 Heterogeneous Computing Scheduling Heuristics

For a complete state-of-the-art please refer to *Technical report* of Dong and Akl [56], they presented not only a complete review of Grid scheduling algorithms but also they presented the taxonomy of these as well as a good overview of the scheduling problem.

2.2.1 Related Work: low complexity scheduling heuristics

The scheduling heuristics related to HCS have drawn a great deal of attention. We present the related work to the proposed research.

The Opportunistic Load Balancing heuristic assigns each task in an arbitrary order to the next available machine regardless of the execution time for the task on that machine [66, 14]. This greedy heuristic tries to balance the load on the machines. However, it has poor solutions, because it does not consider execution times. Minimum Execution Time assigns each task to the machine with the lowest execution time of that task, without considering machine's availability [66, 14]. The Minimum Completion Time heuristic assigns each task to the machine with the minimum expected completion time of the task. The heuristic tries to improve the performance of the previously described heuristics.

Work Queue is a straightforward and adaptive scheduling algorithm for scheduling independent tasks. The heuristic selects a task randomly and assigns it to the machine as soon as it becomes available, that is a machine with minimum workload [76].

The well-known Min-min (Max-min, Min-max) algorithm works in two phases, starting with a set of all unmapped tasks, the algorithm establishes the minimum (maximum) comple-

tion time for the task, in the first phase. The task with the minimum (maximum) completion time is selected and assigned to the corresponding machine [26]. The task is then removed from the set and the process finishes when all of the tasks are mapped.

The Sufferage heuristic [112] computes for each task the difference between the second earliest completion time (on a machine m_j) and the earliest completion time (on a machine m_i) in the first step. This difference is called the sufferage value. In the second step, the heuristic selects the task with the maximum sufferage value. Then, the task is assigned to the corresponding machine with minimum completion time. The heuristic gives precedence to those tasks with high sufferage value. Sufferage II and Sufferage X extend original Sufferage heuristic [35].

Segmented Min-Min [167] sorts the tasks according to the ETCs. The tasks can be sorted into ordered list by the average ETC, the minimum ETC, or the maximum ETC. Then, the task list is partitioned into segments with the equal size. The segment of larger tasks is scheduled first and the segmented of smaller tasks last. For each segment, Min-min is applied locally to assign tasks to machines. Low complexity heuristics [OPTIM2011] differ from Segmented Min-Min in that the heuristics do not partition the list. Moreover, scheduling decisions in low complexity heuristics is guided by a Score Function. There is a kind of similar work presented by Tabak et al. [33] improving Min-Min, Max-Min and Sufferage, nevertheless, their application do not use Score Function. Ezzatti et al. [60] present an efficient implementation of the Min-Min, their algorithm works in two phases: in the first phase use radix exchange sort and in the second phase the normal min-min, their algorithm present the same complexity of our heuristics. High Standard Deviation First [120] considers the standard deviation of the expected execution time of a task as a selection criterion. The task with the highest standard deviation must be assigned first for scheduling. Thereafter, the second part of the Sufferage heuristic is applied.

The QoS guided Min-Min extends the Min-min algorithm. The algorithm takes into account different levels of quality of service required by the tasks and provided by Grid resources such as memory, cpu speed and bandwidth. The idea is to execute tasks requiring high levels of QoS only on resources with high QoS, whereas a task requiring low levels of QoS can be executed on resources with low or high QoS [77]. The tasks with high QoS levels are mapped first, then the tasks with low QoS are considered. The priorities of tasks with the same QoS level are set in the same way of the Min-min heuristic.

The Directed Acyclic Graph (DAG) mapping problem in HCS can be based on mapping multiple groups of independent tasks [98]. The mapping process is performed during the execution of the early mapped tasks. Some solutions use them to improve various algorithms. The most important issue of these kind of solutions is taken into consideration the communication costs [15, 75]. Oliver et al [151] proposed two algorithms under a task graph or DAG but involvement communication not only in network but also among processors, their results shows the important overhead missing in works that have zero costs in communication, furthermore, scheduling the edges on the processors has an impact on the operating techniques of scheduling heuristics. The heuristics proposed in [111] take advantage of task heterogeneity. These heuristics are based on the idea of defining an order of task execution. The authors proposed the Task Priority Diagram (TPD) technique. TPD is a precedence task graph, based on a Hasse diagram that defines a partial order between tasks based on their ETC values. TPD contains the information about the mapping sequence of tasks. The best

heuristics proposed in [111] are based on TPD. The runtime of all of the above mentioned algorithms is at least $O(t^2m)$.

Energy-aware scheduling heuristics in large scale computing has been studied from many years ago, techniques as Dynamic Voltage Frequency Scaling (DVFS)[163, 165] (equivalently, dynamic speed scaling, dynamic frequency scaling, dynamic voltage scaling) and Dynamic Power Management (DPM) [22]. Power down mechanism focus on identifying the optimal threshold times to transition to low-power modes during idle periods. Dynamic Frequency Scaling (DFS) [106] already incorporated into many existing technology and commercial processors as for example SpeedStep by Intel or LongHaul by VIA Technologies, enables processors to dynamically changing its working voltage and speed without stopping or pausing the execution of any instruction. DVFS reduces energy consumption of processors based on the fact that power consumption in CMOS circuits has direct relation with frequency and the square of the supplied voltage [96, 95]. DPM on the other hand, consolidates applications on a minimum set of computing resources to maximize the number of resources that can be powered down while maximizing utilization of the used ones. Machines are powered down (i.e. inactive) when not used, and job placement decisions attempt to power a node back on only when is absolutely necessary. These techniques have been used at the level of resource manager including scheduling algorithms [92, 103, 130, 158]. Bilal et al [25] present a complete survey focus on Green Data Center Networks, they based on a taxonomy presented in this work concentrate on different network architectures (e.g. electrical, optical, and hybrid), traffic (i.e. characterization and management), performance monitoring (i.e. for energy efficiency), virtual machine placement, and experimentation techniques (e.g. simulation and emulation).

Several work have been produced in the cloud scheduling field, Mezmaz et al. [118], for instance, presented a new hybrid metaheuristic scheduling for cloud computing based on parallel bi-objective and energy consumption. Alternatively, Marc Eduard [68] focus on *high availability* and proposed some scheduling algorithms searching for an optimal allocation of components on nodes in order to ensure a homogenous spread of component types on every node. Some other research focus on cloud brokering using scheduling model, Simarro et al. [110] proposes a modular broker architecture for optimal service deployment pricing, their results show that users are benefited with 4% - 6% of their budget their service performance. Multitask workflows on cloud platforms can be tackle using using Multi-Objective Scheduling (MOS), Zhang et al. [170] proposes a new MOS scheme based on Vectorized Ordinal Optimization (VOO) achieved problem scalability on any virtualized cloud platform.

There are some parallelization of scheduling heuristics and efficient use of graphics processing units. In [65], the authors present a parallel Min-min algorithm for the GPU and the CPU architectures. The algorithm parallelizes the search process used to find the best pair task/machine that minimizes the completion time. Ref. [122] develops a parallel implementation on GPU for Min-min and Sufferage. The parallel implementation on GPU performs the evaluation of the criteria proposed by each heuristic. That is, for each unassigned task, the evaluation of the criteria for all machines is made in parallel on the GPU architecture, building a vector that stores the identifier of the task, the best the value obtained for the criteria, and the machine to get that value.

2.3 Resource Allocation

Hussain et al. [81] show a complete review on resource allocation in high performance distributed computing systems, their work focus on cluster computing, grid computing, and cloud computing systems (see [81] for the complete bibliography).

2.3.1 Related work: Resource Allocation in Opportunistic Cloud Environments

The problem of VM allocation can be divided in two: the first part is admission of new requests for VM provisioning and placing the VMs on hosts, whereas the second part is optimization of current allocation of VMs. In this work we focus on the first part of the VM allocation problem. This problem can be modeled as a bin-packing problem with variable bin sizes [62].

The problem in its single-objective variant is a NP-hard problem, and thus is expensive to compute with increasing numbers of PMs and VMs. Different heuristics and linear programming based solutions are used for getting a near optimal solution. Several proposed power-aware packing algorithms use a variant of First Fit Decreasing (FFD) heuristic to reduce resource fragmentation. Static consolidation considers that resource utilization does not change during execution and the number of reconfigurations depends solely of creation and deletion of VMs. Dynamic VM management assumes that resource needs change over time, and thus VMs can be moved during their execution in order to improve the optimality placement, one framework that considers dynamic management of VMs is Snooze [62].

Verma et al. [161] modeled the VM workload placement as an instance of the one dimensional bin-packing problem and extended FFD to perform the placement. Li et al. [104] proposed the EnaCloud framework and a modified version of the Best-Fit algorithm is implemented.

Hermenier et al. [78] proposed Entropy. Entropy uses constraint programming to determine a global optimal solution, if one exists, based on a depth first search. The algorithm is enhanced with FFD to lower the number of nodes used in a virtualized cluster.

Buyya et al. [32] presented simulation-driven results for a workload consolidation algorithm based on a modified version of the Best Fit Decreasing algorithm. The algorithm sorts all the VMs in decreasing order of current utilization and allocate each VM to a host that provides the least increase of power consumption due to this allocation. This allows leveraging heterogeneity of the nodes by choosing the most power-efficient ones. Beloglazov and Buyya [20] proposed a resource management policy for virtualized cloud data centers. The objective is to consolidate VMs leveraging live migration and switch off idle nodes to minimize power consumption, while providing required Quality of Service.

Murtazaev and Oh [121] proposed a greedy algorithm called Sercon. It extends First-Fit and Best-Fit heuristics by considering the minimization of number of VM migrations. The algorithm first sorts the PMs according to the load by VMs in decreasing order. Then, it takes the least loaded PM and sorts the current VMs on it in decreasing order of weights. The algorithm tries to allocate them one-by-one on the first most loaded PMs, if not all VMs can be placed on a different PM the operation is cancelled, such that only migrations which lead to free PMs will be performed.

Feller et al. [61] dealt with the workload consolidation problem and model it as an instance of the multi-dimensional bin-packing problem. The authors proposed a nature-inspired workload consolidation algorithm based on the Ant Colony Optimization framework. The proposed algorithm outperforms FFD, however, at greater complexity.

Marshall et al. [113] propose a cloud infrastructure that combines on-demand allocation of resources with opportunistic provisioning of cycles from idle cloud nodes to other processes such as HTC (i.e. Condor) by deploying backfill VMs. They found that backfill VMs contribute to an increase of the utilization of the IaaS from 37.5% to 100%.

Most of the related consolidation algorithms consider a pool of dedicated computing resources. However, In our work [CCSA2013] differs from related state of the art since we deal with an opportunistic cloud-based infrastructure. Bin packing strategies were been evaluated in our work [CCSA2013]. Alternatively, strategies as turn on or turn off the computer labs are evaluated by Ponciano and Brasileiro [134].

Failures, volatility (i.e., intermittent presence), and lack of trust are common characterizes which define the platform where this opportunistic cloud infrastructure is developed, because it is based on desktop and volunteer computing. Choi and Buyya [42] regard such unstable situations and propose a new Group-based Adaptive Result Certification Mechanism providing also dynamic scheduling algorithms according to its properties such as volunteering service time, availability, and credibility.

Chapter 3

Low Complexity Heuristics

Contents

3. LOW COMPLEXITY HEURISTICS

Heterogeneous Computing Systems (HCS) are widely used as powerful parallel and distributed platforms for executing applications requiring a large amount of computing power or data. In such systems, the heterogeneity is characterized by different hardware architectures, operating systems, memory capacity, different application requirements and constraints. One of the big challenges is to fully exploit heterogeneity, maximize the cost-effectiveness and performance of the system.

Many HCS middleware components (e.g., naming, authentication, authorization, accounting, communication and scheduling) are affected by scaling. Scalability is one of the most desired design goals for developers of HCS [155]. Three components are important: size, geographical position, and administrative scalability [155, 125]. To take advantage of capabilities of HCS, a scheduler allocates tasks to the available resources to optimize performance objectives. Generally speaking, the scalability of a scheduler indicates its ability to make fast mapping decisions with growing amount of work, and its capability to manage the increasing number of resources in an efficient way.

The scheduler is a key component for performance guarantees. Its aim is to assign tasks to computing resources considering system- and user-centric objectives and satisfying given efficiency criteria, usually related to the total execution time, resource utilization, economic cost, energy constraints, etc. The scheduling problem in HCS is NP-hard [82], even in homogeneous version without precedence constraints. Therefore, significant efforts for solving the scheduling problem have been extensively developed and numerous methods have been reported in the literature [82, 26, 111, 112, 120, 166, 109, 168, 169, 147, 79]. However, most of them have a high computational complexity, which becomes a drawback when algorithms are used in real systems. Efforts to reduce the computational complexity have been made to cope with the aforementioned issues: parallelization of scheduling heuristics [133, 123, 124] and efficient use of graphics processing units (GPUs) [65].

In the context of exascale computing paradigm, the scheduler deals with thousands of resources, users, and tasks. Traditional techniques to support large HCS do not scale to the largest system. These problems arise due to limitations of local resource manager scalability and granularity [139]. The design of scheduling techniques that use resources efficiently is becoming increasingly important. Moreover, there exist scenarios in which only fast scheduling heuristics with low computational complexity can be used. Especially, when the scheduling process is online [111]. For example, when the HCS system is used as an application service provider to respond to online computational requests. The tasks from the list or queue can be scheduled in a batch mode to increase the system utilization ratio. Although time consuming heuristics can achieve a shorter schedule length, the scheduling process would delay the actual task completion. This delay becomes crucial if tasks have real-time constraints [111]. Therefore, low computational complexity and scalable scheduling heuristics are the preferred choice in this scenario.

In our scheduling problem, tasks have no deadline and precedence constraints. A schedule is evaluated by the schedule length or makespan. We address offline scheduling under consideration of zero release times for the independent tasks. While real HCS systems exhibit online behavior, it is known that tasks typically remain in batches or queues for a significant time. Therefore, an offline scheduling strategy is beneficial for the efficient scheduling of current batchs [140]. Moreover, many offline scheduling algorithms exhibit a good performance also in the online case. Based on theoretical results, it is known that the performance bounds of

offline scheduling strategies can be approximated for the online cases [149, 138, 79].

In this chapter, we will present the *"low complexity heuristics"* proposed by us in [OPTIM2011] and we are specially interested in investigating the most important batch mode scheduling heuristics that address the problem of scheduling independent tasks, producing fast and efficient tasks mapping to HCS resources against our heuristics. Batch mode scheduling strategies are simple, powerful, and efficient in contrast to more sophisticated schedulers that need longer execution time to compute the mapping. We present a comprehensive performance evaluation study using simulation of the following heuristics: the well-known *Min-min* [82, 26], the best two *Task Priority Diagram* (TPD) based heuristics [111] founded in the literature, and our three *low computational complexity* heuristics proposed in [OPTIM2011]. They use tasks and resource characteristics for resource allocation. The heuristics are based on the list scheduling approach. Tasks are sorted in a priority list, then the task with the highest priority (i.e., at the top of the list) is scheduled first until the list is empty. The scheduling decisions are based on a *score function*(see Section 3.2). The score function exploits resource capabilities and task requirements by scheduling tasks to a computing resource that not only minimizes the completion time, but also executes individual tasks faster by trying to balance the load of the machines.

The main purpose of this chapter is to study and compare the performance and scalability of the above mentioned heuristics with a variety of system parameters to fully understand their capabilities and limitations. Our interest is to know whether proposed heuristics outperform traditional and related approaches. In terms of performance, we consider the optimization of schedule length or makespan, which is a system-centric measure representing the utilization of a platform and the main flowtime, which is a (QoS) metric that the allows to guarantee good response times.

To evaluate the performance of the studied algorithms, we analyze the results of numerous simulations with variation of heterogeneity of resources and applications. We evaluate the scalability by increasing the system and problem sizes, and consider task consistency. We show that our *low computational complexity* strategies outperform known approaches in most studied scenarios.

3.1 Problem Definition

We address the following scheduling problem: n computationally tasks $T = \{t_1, ..., t_n\}$ must be scheduled on a HCS composed of $M = \{m_1, ..., m_m\}$ machines, as they were defined in Chapter 2. They are interconnected with high-speed links so network times are neglected.

Tasks are assumed to be independent. The independent tasks model typically arises in HCS systems as grid or volunteer-based computing infrastructures, where independent applications are submitted for execution [156]. It also arises in the case of Bag-of-Tasks applications. They are used in a variety of scenarios, including Single-Program Multiple Data applications used for multimedia processing, simulations fractal calculations, computational biology, parameter sweeps, and parallel domain decomposition of numerical models for physical phenomena.

Each task is considered as an indivisible unit of workload and must be processed completely in a single machine without interruptions. From now on we make the assumption that an estimate of the computational workload is known. The computational model that we

consider is the *Expected Time to Compute* model (ETC). The ETC matrix of size $n \times m$ is well defined. Each element, $ETC[t_i][m_j]$ indicates the expected time to compute task t_i on machine m_j. The ETC matrix for a given HCS system can be obtained from user supplied information, task profiling, experimental data, and analytical benchmarking [7, 70, 91, 93, 132]. Moreover, some existing batch schedulers handle such an information in a conservative way: at a given time, all decisions are taken assuming that the available information is exact. When an external event occurs, the scheduling process is restarted with the new state of the system, without rescheduling of the already running jobs [59].

The scheduling problem can now be formulated as follows. Given a HCS system composed of a set of m machines and a set of n tasks, any task is scheduled without preemption on machine m_j from time $\sigma(t_i)$, with an execution time given by $ETC[t_i][m_j]$. The task t_i completes at time $C_i = \sigma(t_i) + ETC[t_i][m_j]$. The objective is to minimize the maximum completion time $C_{max} = max(C_i)$, known as makespan. Table 3.1 describes the variables used in this section.

Name	*Description*
n	Number of tasks
m	Number of machines
$\sigma(t_i)$	The start time of task i
C_i	Completion time of task i
C_{max}	Maximum completion time or makespan

Table 3.1: Variables

We focus on the analysis of scheduling systems, where all of the tasks are given and processed in the same batch. A set of available and ready tasks will be executed up to the completion of the last one. All tasks that arrive during this time interval will be processed in the next batch. There are several general HCS or grid production systems in use whose primary focus is task or job batching and resource scheduling, some examples are OAR [5], TORQUE [6], Portable Batch System (PBS) [3], Load Sharing Facility (LSF) [4], Condor [2]. They are used as a job scheduler mechanism by several meta schedulers.

3.2 Low Computational Complexity Algorithms

This section describes our proposed *low computational complexity* heuristics *Min-Max-Min*, *Max-Max-Min*, and *Mean-Max-Min* [OPTIM2011]. These three heuristics are based on the *Min-min* algorithm. The main idea is to avoid the loop of pairs of machine-task in the *Min-min* algorithm corresponding to the first phase [OPTIM2011, 71]. One alternative is to consider one task at a time to determine the task that should be scheduled next. The three heuristics perform tasks sorting in a list by a predefined priority before scheduling as Segmented Min-Min does. Thereafter, the tasks are selected from the list in constant time without reordering, after each allocation. Once the order of the tasks is determined, the task with the highest priority is selected and assigned to the machine that minimizes a predefined *score function*. The *score function* considers not only the expected completion time, but also

the execution time of tasks. The aim is to minimize the makespan and balance the load of the system.

Framework 1 depicts the general structure of the heuristics. The heuristics start computing the *Priority* of each of the task according to some objective (line 1). Thereafter, the heuristics sort a list of tasks (line 2). The order of the tasks is not modified during the execution of the heuristics. Next, the heuristics allocate the tasks to the machines and determine their starting date (main loop line 3). One task is scheduled at a time. The heuristics always consider the task t_i with the highest priority at the top of the list (line 4). Given a task, the *score function* $SF(t_i, m_j)$ Eq. (3.1) is evaluated for all the machines (lines 5 and 6). The machine for which the value of the *score function* is optimized for task t_i is selected for the task allocation (line 8). The task is removed from L (line 9), then the list is updated (line 10) and the main loop is restarted.

The score of each mapping event is calculated as in Eq. (3.1). For each machine m_j,

$$SF(t_i) = \lambda \cdot \frac{C_i}{\sum_{k=1}^{m} C_{ik}} + (1 - \lambda) \cdot \frac{ETC[t_i][m_j]}{\sum_{k=1}^{m} ETC[t_i][m_k]}, \tag{3.1}$$

where C_{ik} is the completion time of the task t_i on machine k and $ETC[t_i][m_k]$ is the expected time to compute a task t_i on machine k. The first term of Eq. (3.1) aims to minimize the completion time of the tasks t_i, while the second term aims to assign the task to the fastest machine or the machine on which the task takes the minimum expected time to complete. λ (lambda) is the weight parameter that tries to balance both aforementioned terms.

Algorithm 1: Framework 1 Low computational complexity heuristics

```
1 Compute Priority of each task t_i ∈ T according to some predefined objective;
2 Build the list L of the tasks sorted in a given order of Priority;
3 while L ≠ ∅ do
4     Select the task t_i in the top of L;
5     for each machine m_j do
6         Evaluate Score Function SF(t_i);
7     Assign t_i to the machine m_j that minimize SF(t_i);
8     Remove task t_i from L;
9     Update the list L;
```

The heuristics differ in the way used to compute the priorities. *Minimum*, *maximum*, and *average* expected time to complete the task are used. We presented three heuristics:

3.2.1 Heuristic Min-Max-Min:

is represented in Algorithm 2. The algorithm uses the minimum execution time of tasks to determine the priority. Thereafter, the tasks are sorted in decreasing order of the execution time and scheduled based on the minimum completion time.

Algorithm 2: Min-Max-Min

```
1 forall the task t_i do
2     for each machine m_j do
3         Select the minimum execution time for each task t_i;
4 Sort the tasks in decreasing order of minimum execution time;
```

3.2.2 Heuristic Max-Max-Min:

described in Algorithm 3, uses the maximum execution time of tasks as a priority. The tasks are sorted in decreasing order of their maximum execution time, and scheduled based on the minimum completion time.

Algorithm 3: Max-Max-Min

```
1 forall the task t_i do
2     for each machine m_j do
3         Select the maximum execution time for each task t_i;
4 Sort the tasks in decreasing order of maximum execution time;
```

3.2.3 Heuristic Mean-Max-Min

represented by Algorithm 4, considers the average execution time of tasks as a priority. The tasks are sorted in decreasing order and scheduled based on the minimum average completion time.

Algorithm 4: Mean-Max-Min

```
1 forall the task t_i do
2     for each machine m_j do
3         Compute the mean execution time for each task t_i;
4     Sort the tasks in decreasing order of average execution time;
```

3.2.4 Computational Complexity

The computational complexity determines how efficiently the problem can be solved. To compute the computational complexity of the *Min-Max-Min*, *Max-Max-Min*, and *Mean-Max-Min* heuristics start computing the value of the priorities for the tasks, and the construction of the sorted list, this have an overall cost of $O(mn\ log\ n)$. The execution of the main loop (lines 3 to 9) in Framework 1 has an overall cost of $O(mn)$. Therefore, the asymptotic overall cost of the heuristics is $O(max(mn\ log\ n, mn))$, that is, it require $O(mn\ log\ n)$ running time, which is less than one order of magnitude to the related *Min-min* approaches.

3.3 Evaluated Heuristics

3.3.1 Task Priority Algorithms

Luo et al. [111] proposed a set of heuristics that exploit the task consistency to perform better scheduling and mapping decisions. Most of them are based on the tasks execution order. The rational inference is that a task requiring more computational effort is more critical to be scheduled than a task with shorter computational requirements. The order defines the priority between tasks. For that purpose, the authors proposed the Task Priority Diagram (TPD) technique. TPD is a precedence task graph, based on a Hasse diagram that defines a partial order of tasks using the ETC matrix [111]. TPD contains the information about the mapping sequence of tasks. Lou et al. concluded that the heuristics TPD with support of minCT-minCT (minimum completion time in the first phase, and minimum completion time in the second phase) and TPD with support of minCT-minSD (minimum completion time for every unmapped task in the first phase, and minimum standard deviation in the second phase) perform as efficient as Min-min.

We follow the generic framework of the scheduling algorithm with TPD in the decreasing order of task processing times as originally proposed in [111]. In this framework, the first cycle is repeated until all of the tasks have been assigned. In each iteration, T is the set of the unmapped tasks with maximal elements in the current TPD, represented by G. The framework use the First-Mapping Step function which develop the respectives two algorithms able to select a task $t_{i'}$ from T and map the task on machine $m_{j'}$ with a minimum completion time or minimum standard deviation. After $t_{i'}$ is mapped, node $t_{i'}$ and its edges are removed from G, and the set T, containing the elements of the new G' is also updated.

Algorithm 5: Framework 2. TPD heuristic: Mapping Algorithm with TPD [111]

1 G = Hasse-Diagram-Generator($E_{n\times m}$) {E is the input ETC matrix};
2 $T' = \{t'|t'$ is a maximal element of $G\}$;
3 **while** $T' \neq \emptyset$ **do**
4 $\quad F_{i'j'}$ =First-Mapping-Step(T');
5 $\quad$ map $t_{i'}$ to $m_{j'}$ and update the load on $m_{j'}$;
6 $\quad$ delete $t_{i'}$ and the edges of $t_{i'}$ from G;
7 $\quad G = G'$;
8 $\quad T' = \{t|t$ is a maximal element of the new $G\}$;

Following Luo et al. [111], Algorithm 6 and Algorithm 7 use the minimum completion time to select the machine onto which the unscheduled tasks in T' can be mapped (line 1 in both algorithms). Based on this mapping, the second metric is used to select the best task with its corresponding destination machine (line 2 in both algorithms). In Algorithm 6, the task with the overall minimum completion time is selected and mapped to the corresponding machine, while Algorithm 7 prefers the mapping step, that minimizes the standard deviation of the system loads after update.

Luo et al [111] defined the current machine free-time vector as $F = (f_1, ..., f_m)$, which

means that machine j is free after f_j time-units for $1 \leq j \leq m$ [111]. The machine free-time vector after update is $F_{i,j} = (f_1, ..., f_j + e_{ij}, ..., f_m)$, represents a mapping step where task i with expected time to compute e_i is mapped to machine j.

Algorithm 6: First-Mapping-Step ($\{F_{ij} | t_i \in T, 1 \leq j \leq m\}$, minCT-minCT) [111]

1 $\overrightarrow{F} = \{F_{ij'} | F_{ij'} = \text{minCT}(\{F_{ij} | 1 \leq j \leq m\}), t_i \in T\}$;
2 $F_{ij} = \text{minCT}(\overrightarrow{F})$;
3 **return** F_{ij};

Algorithm 7: First-Mapping-Step ($\{F_{ij} | t_i \in T, 1 \leq j \leq m\}$, minCT-minSD) [111]

1 $\overrightarrow{F} = \{F_{ij'} | F_{ij'} = \text{minCT}(\{F_{ij} | 1 \leq j \leq m\}), t_i \in T\}$;
2 $F_{ij} = \text{minSD}(\overrightarrow{F})$;
3 **return** F_{ij};

3.3.2 Min-min Algorithm

One of the most widely used batch scheduling heuristic in HCS is the *min-min* list scheduling method [82]. A pseudocode of the Min-min heuristic is presented in Algorithm 8. The min-min heuristic greedily picks the task that can be executed the soonest, considering the current machine load. The heuristic works in two phases. In the first phase (lines 1 to 4), it calculates the minimum completion time for every unscheduled task (the first min). In the second phase, the task with the overall minimum is selected (line 6), and mapped in a machine (line 7). The task is then removed (line 9) from the set of unscheduled tasks, and the process is repeated until all tasks are mapped.

Algorithm 8: Min-min heuristic [82]

Data: The ETC matrix.
1 **while** $T \neq \emptyset$ **do**
2 **for** *each unmapped task* $t_i \in T$ **do**
3 Select the machine that gives the task its minimum completion time;
4 Build the list of tasks L adding this minimum founded;
5 Find the task $t_k \in L$ with the minimum earliest completion time;
6 Assign task t_k to the machine that gives the earliest completion time;
7 Update the ready time of each machine;
8 Delete the task t_i from T;

3.3.3 Computational Complexity

In the following, for simplicity, we refer to TPD-minCT-minCT [111] algorithm as, Luo_1, TPD-minCT-minSD [111] as Luo_2, Min-min [82] as m_m, and Min-Max-Min, Max-Max-Min, Mean-Max-Min [SUPE2013] as Low_1, Low_2, and Low_3, respectively (Table 3.2). This involves no loss of generality.

Table 3.2 summarizes the heuristics evaluated in this paper. First column shows the name used in this paper (in the experimental section). Second column presents the name of the heuristics in the literature. Third column remarks the computational complexity of each algorithm. Last column provides the reference where heuristics were introduced. It is important to highlight that the main difference of *low-complexity heuristics* among all evaluated heuristic is their simplicity to resolve big scheduling problems.

No.	*Name*	*Complexity*	*Reference*
Luo_1	TPD-minCT-minCT	$O\left(n^2m\right)$	[111]
Luo_2	TPD-minCT-minSD	$O\left(n^2m\right)$	[111]
m_m	Min-min	$O\left(n^2m\right)$	[82]
Low_1	*Min-Max-Min*	$O\left(nm\log n\right)$	[SUPE2013]
Low_2	*Max-Max-Min*	$O\left(nm\log n\right)$	[SUPE2013]
Low_3	*Mean-Max-Min*	$O\left(nm\log n\right)$	[SUPE2013]

Table 3.2: Complexity of heuristics

3.4 Numerical Example

We illustrate the scheduling of the low computational complexity heuristics on a numerical example, and compare them with the TPD-based heuristics and Min-min. Let us consider the example reported in [111] with the ETC input matrix shown in Table 3.3. The instance is composed of eight tasks and four machines.

Figure 3.1 shows the TPD graph generated by Luo_1 and Luo_2, (See [111] for more information about the TPD graph). Table 3.4 shows the different task sequences computed for each heuristic to map the tasks. The sequences generated by Luo_1 and Luo_2 are based on the TPD as shown in Figure 3.1 from top to the bottom (t_7). m_m generates the sequence using the TPD graph from the bottom to the top (t_0). The task in the lower level of the TPD graph has a higher priority for mapping. Low_1, Low_2, and Low_3 generate the sequences according to ETC values used to define tasks' priorities.

We observe the performance behavior of Low_1, Low_2, and Low_3 for different lambda values in the interval [0, 1] (Figure 3.2). From the plot, we observe that the heuristics have the best performance for lambda values in the interval [0.6, 0.9], the same values for the three heuristics. We report the best makespan for lambda equal to 0.8.

Figure 3.3 depicts the optimal schedule computed using an exhaustive search algorithm.

The resulting schedules computed by the evaluated heuristics are depicted in Figures 3.4-3.9. The makespan computed by Luo_1 is equal to 2350, Luo_2 obtains a makespan equal to

Task No.	m_0	m_1	m_2	m_3
t_0	2000.0	2200.0	2400.0	2600.0
t_1	900.0	1100.0	1300.0	1500.0
t_2	950.0	1050.0	1350.0	1550.0
t_3	980.0	1080.0	1330.0	1480.0
t_4	600.0	700.0	800.0	1490.0
t_5	550.0	1070.0	750.0	1000.0
t_6	920.0	600.0	600.0	1000.0
t_7	100.0	200.0	300.0	400.0

Table 3.3: ETC for running example

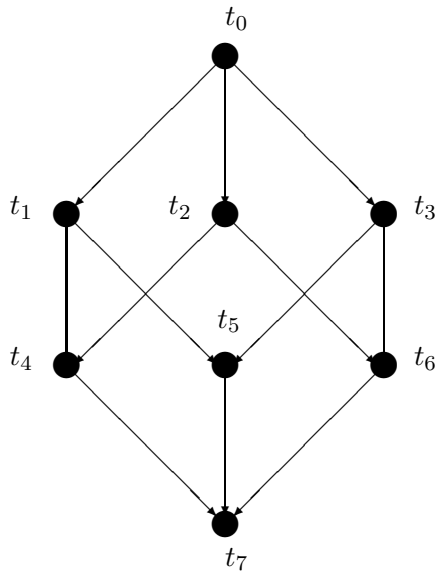


Figure 3.1: Task Priority Diagram TPD, which contains the information about the mapping sequence of tasks and generated by Luo_1 and Luo_2.

2490, m_m computes a makespan of 3200 units of time, the makespan of Low_1 is equal to 2550, Low_2 and Low_3 compute a makespan equal to 2350 units of time.

We can notice that different sequences lead to different performance for the heuristics. In the given example, due to the tasks' priorities combined with the score function *low computational complexity* heuristics perform as efficient as the related heuristics with the TPD graph and outperform Min-min. From the numerical example, we can observe that the low complexity heuristics do not need the TPD graph to compute a similar schedule than the TPD-based heuristics, neither the first phase iteration than Min-min.

The error deviation from the optimal schedule is 33% for m_m, 16% for Low_1, 14% for Luo_2, and 9% for Luo_1, Low_2 and Low_3. The performance behavior of the investigated heuristics is validated in the experimental section.

Luo_1	Luo_2	m_m	Low_1	Low_2	Low_3
t_0	t_0	t_7	t_0	t_0	t_0
t_2	t_3	t_6	t_2	t_3	t_2
t_1	t_1	t_5	t_1	t_2	t_3
t_3	t_5	t_4	t_4	t_1	t_1
t_6	t_2	t_3	t_3	t_4	t_4
t_5	t_6	t_1	t_5	t_6	t_5
t_4	t_4	t_2	t_6	t_5	t_6
t_7	t_7	t_0	t_7	t_7	t_7

Table 3.4: Tasks' sequence computed by each heuristic.

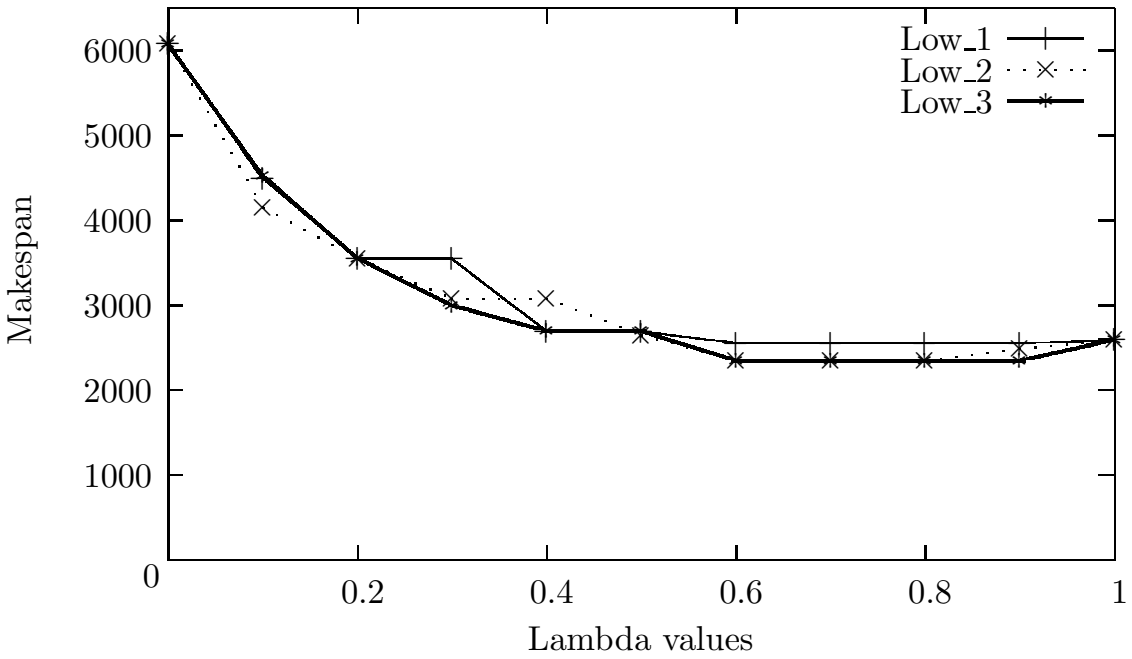


Figure 3.2: Makespan for different lambda values computed by the low complexity heuristics.

3.5 Experimental Validation

In this section, we discuss how we conduct the qualitative analysis of proposed heuristics. To provide a fair comparison, the assumptions and system parameters are kept the same in all of the approaches.

We show that quiet simple schedulers Low_1, Low_2, and Low_3 with low complexity can provide good performance for different scheduling scenarios and instances. We characterize the best scenarios to which the low complexity heuristics outperform related approaches, especially those scenarios where heterogeneity in machines and tasks is important. Note, that

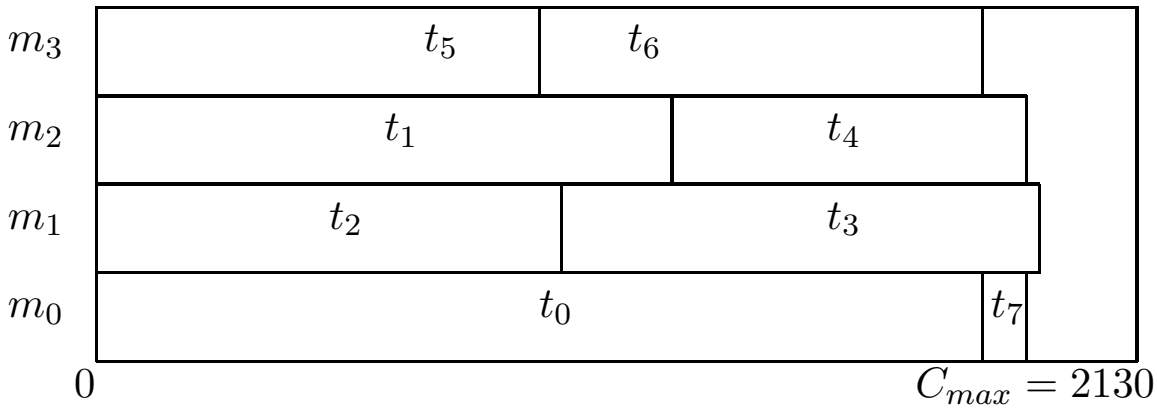


Figure 3.3: Optimal Schedule.

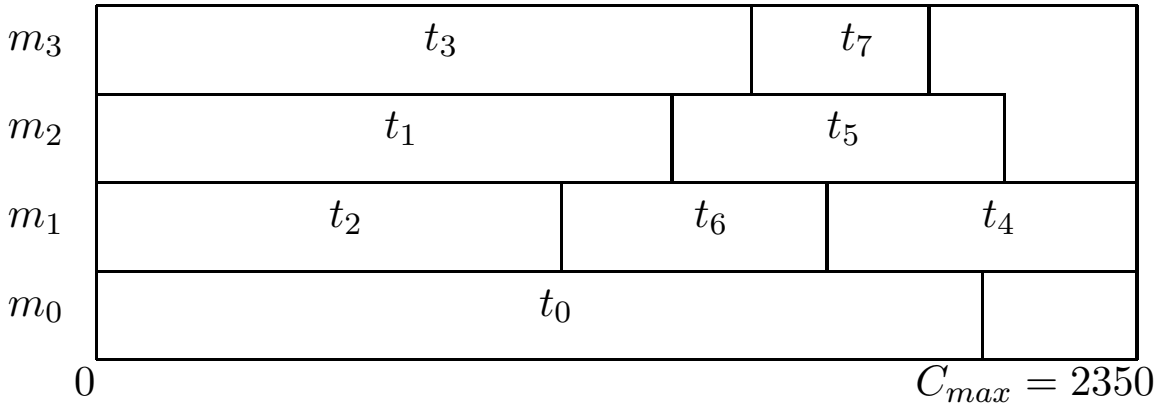


Figure 3.4: Schedule computed by Luo_1.

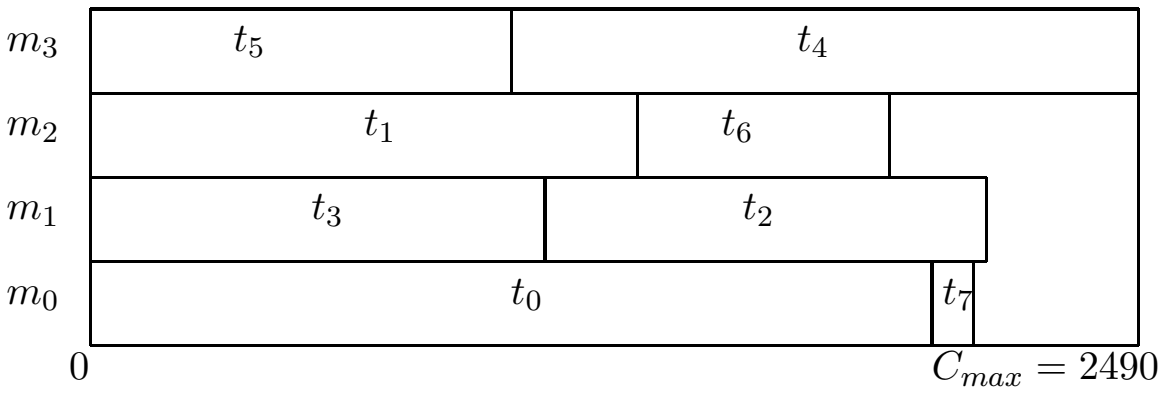


Figure 3.5: Schedule computed by Luo_2.

the low complexity heuristics do not use the TPD technique to define the tasks sequences. Besides the performance behavior it does not require additional management overhead such as evaluating tasks priorities at each iteration of the heuristics.

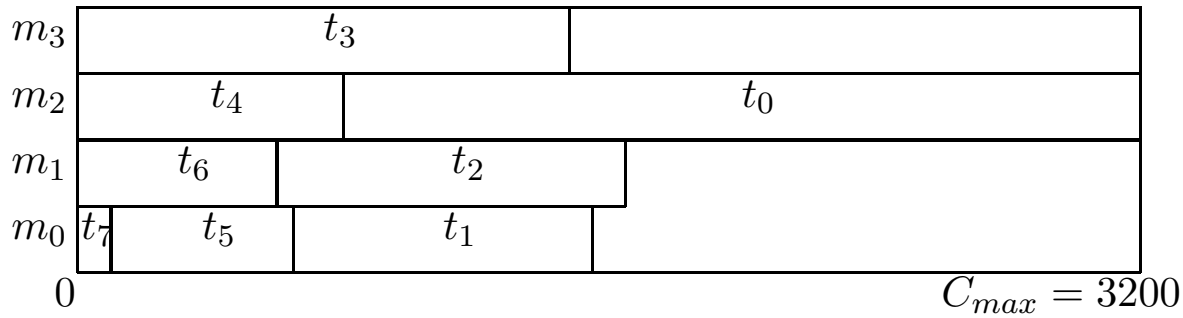


Figure 3.6: Schedule computed by m_m.

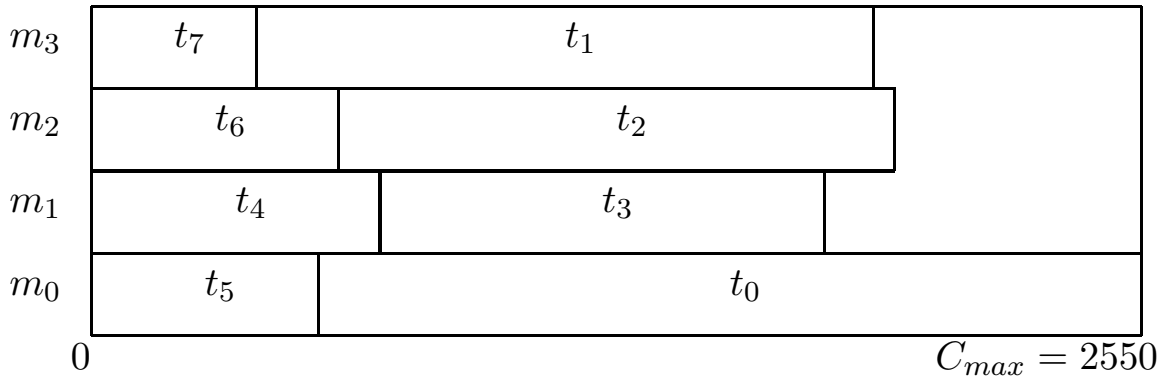


Figure 3.7: Schedule computed by Low_1. The tasks are sorted by shortest execution time first on each machine.

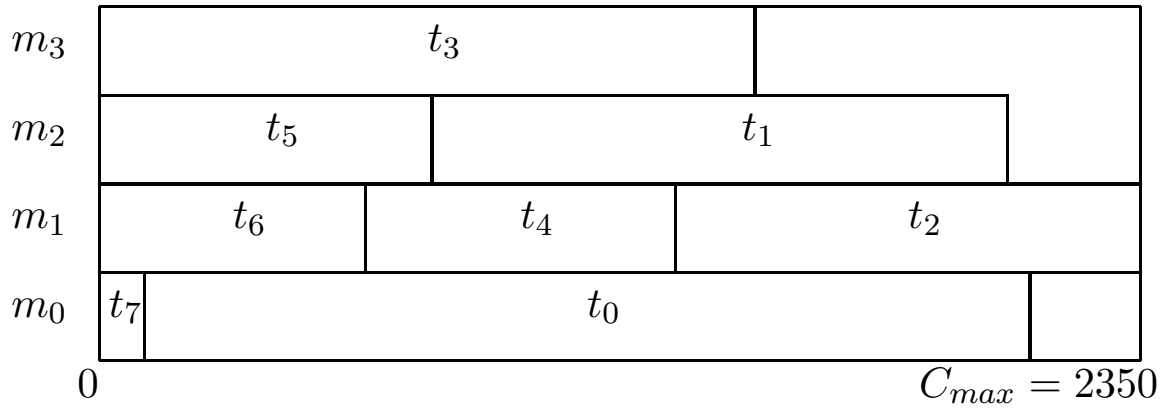


Figure 3.8: Schedule computed by Low_2. The tasks are sorted by largest execution time first on each machine.

3.5.1 Evaluation Method

To provide an effective guidance in choosing the best strategy, we perform an analysis of makespan metric as in [111]. Makespan optimization criterion considers the largest completion time of any job in the system. It is applicable to compare different algorithms. However, it does not depict the information about the efficiency of the scheduling algorithms and the

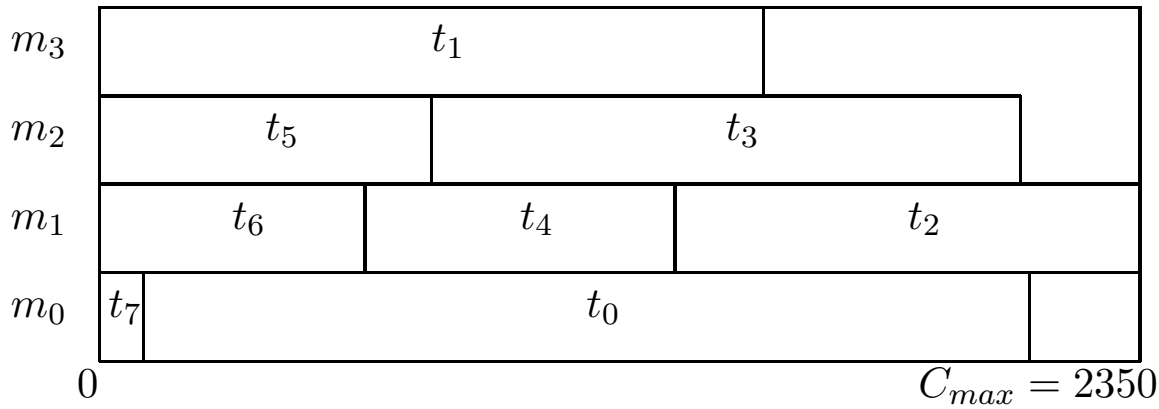


Figure 3.9: Schedule computed by Low_3. The tasks are sorted by the average execution time of tasks.

schedule quality. We also use the approximation factor to qualify the efficiency of the scheduling algorithms. The approximation factor of the strategy is defined as $\rho = C_{max}/C^*_{max}$, where C^*_{max} is the optimal makespan. As we are, in general, not able to determine the optimal makespan, in our experimental evaluation, we use the lower bound of the optimal makespan C^*_{max} instead of the optimal makespan with:

$$C^*_{max} \geq \check{C}^*_{max} = max\left\{p_{max}, \frac{\sum_{i=1}^{n} min_{j=1\cdots m}(ETC[t_i][m_j])}{m}\right\}, \tag{3.2}$$

where

$$p_{max} = max\left\{min_{j=1\cdots m}\left(ETC[t_i][m_j]\right)\right\}. \tag{3.3}$$

We show that the makespan obtained by algorithms does not exceed the optimal makespan for any problem instances by more than a certain factor. This factor is called an approximation factor in the off-line case and used in the following analysis.

The analysis is conducted as follows. First, we evaluate the performance ratio γ (relative error) of the approximation factor of each strategy under each metric. This is done relative to the best performing strategy for the metric, as follows:

$$\gamma = \frac{strategy\ metric\ value}{best\ found\ metric\ value}. \tag{3.4}$$

Thereafter, we average the performance ratios to evaluate performance of the strategies, and show if some strategies tend to dominate results in all of the test cases, in average. Our goal is to find a robust and well performing strategy under all of the test cases, with the expectation that it will also perform well under other conditions. For example, with different HCS configurations and workloads. This approach provides the mean relative errors. The average values give us valuable conclusions. However, they do not show the negative effects of allowing a small portion of the problem solutions to dominate the conclusions.

To analyze the possible negative effects of allowing a small portion of the problem instances with large deviation to dominate the conclusions, and to help with the interpretation of the data generated by the benchmarking process, we presented performance profiles of our

strategies.

The performance profile $\rho(\tau)$ is a non-decreasing, piecewise constant function, which presents the probability that a performance ratio γ is within a factor τ of the best ratio [53, 54]. The function $\rho(\tau)$ is the cumulative distribution function. Strategies with large probability $\rho(\tau)$ for smaller τ are to be preferred. Moreover, three extra metrics are used to compare heuristics in more details.

- The *number of the best solutions* found (B) It is the number of times a particular algorithm obtains the shortest makespan.
- The *number of best solutions found together with another heuristics* (EB) It counts the cases, where one heuristic found the shortest makespan together with at least one other heuristic.
- The *Percentage gain of the mean flowtime* (%Gain) it shows how much the results of one our *low complexity heuristic* is better or worst than the results obtained by Luo_1.

3.5.2 Experimental Setup

In this section, we present a common HCS trace-based simulation setup with an emphasis on the representative workload and testing environment to obtain reproducible and comparable results. However, while machines in real HCS (like computational grid) often exhibit different forms of heterogeneity, such as different hardware, operating system, dynamic behavior, software, many different types of tasks and other restrictions, the model of any study on HCS scheduling must be an abstraction of reality to perform performance evaluation in a repeatable and controllable manner [79]. On the other hand, key properties of HCS should be observed to provide benefits for real deployments. Hence, we assume that the tasks arrive at the system before the scheduling event and the scheduler knows the parameters of all tasks. We consider that all of the machines are idle and available at time zero, which is possible by considering an advance reservation. We assume that the resources are stable and dedicated. This simplified model matches some real HCS deployments, and we consider that the assumptions are reasonable. The evaluated algorithms may serve as a starting point for algorithms that can be implemented in real HCS systems.

To provide a fair comparison, we use an experimental framework proposed in [111]. ETCs (see Section 3.1) are used for performance assessment. To present the heterogeneity of tasks and machines, a variety of ETC matrices are generated. Machine heterogeneity is represented by a parameter called $V_{machines}$. It leads to the variation of execution times of a given task across the computing resources. Task heterogeneity is represented by a parameter called V_{tasks}. It represents the degree of variation among the execution time of tasks for a given machine. The model is implemented by the *coefficient-of-variation* (COV) based method [8] considering different machine and task heterogeneities.

The combinations of $V_{machines}$ and V_{tasks} is used to characterize different HCS environments and computational tasks. For instance, *low* values of machine heterogeneity represent computing systems composed of similar (almost homogeneous) computer resources. On the contrary, computing systems integrated by resources of different types and capacities are represented by *high* values of $V_{machines}$. *Low* values of V_{tasks} parameter represents cases when

tasks are almost homogeneous (i.e., when the computational requirements of tasks are quite similar), with nearly the same execution times for a given machine. *High* values of V_{tasks} parameter describes scenarios, in which applications are required to exhibit short and large execution time. We also consider *full-consistent*, *partial-consistent*, and *original-consistent* scenarios [111], which correspond to consistent, semi-consistent, and inconsistent scenarios introduced in [8].

The full-consistent scenario implies that if a given machine m_j executes task t_i faster than machine m_k, then machine m_j executes all tasks faster than machine m_k. The original-consistent scenario represents a case where machine m_j may be faster than machine m_k for some tasks and slower for others. Finally, the partial-consistent scenario consists of both of the aforementioned scenarios, that is, it contains a consistent sub-matrix within inconsistent sub-matrix. Moreover, the ETC matrix considers an overhead implied by moving the executable code and data associated with each task.

To simulate different heterogeneous scenarios, we vary μ_{task}, V_{task}, and $V_{machine}$ parameters. V_{task}, is within the range of [0.1, 1.1], $V_{machine}$ is within the range of [0.1, 0.6], both with the increment 0.1, μ_{task} is equal to 1000. Table 3.5 shows 16 combinations of tasks, and machines that we use for evaluation. We have generated 144,000 instances (1000 for each parameter combination, in total 48 combinations with three different tasks-machine sizes). We have simulated heterogeneous computing systems composed of 16, 32, and 64 machines and 512, 1024, and 2048 tasks.

Scenarios					
V_{task}	$V_{machine}$	V_{task}	$V_{machine}$	V_{task}	$V_{machine}$
0.1	0.1	0.6	0.6	-	-
0.2	0.1	0.7	0.6	0.6	0.2
0.3	0.1	0.8	0.6	0.6	0.3
0.4	0.1	0.9	0.6	0.6	0.4
0.5	0.1	1	0.6	0.6	0.5
0.6	0.1	1.1	0.6	-	-

Table 3.5: 16 Heterogeneity scenarios.

We have executed the low complexity heuristics for λ-values in the interval [0, 1] with the increment 0.1. We report the best value for each instance. The complexity of Low_1, Low_2, and Low_3 is slightly increased by a constant, which is still lower than the complexity of related approaches.

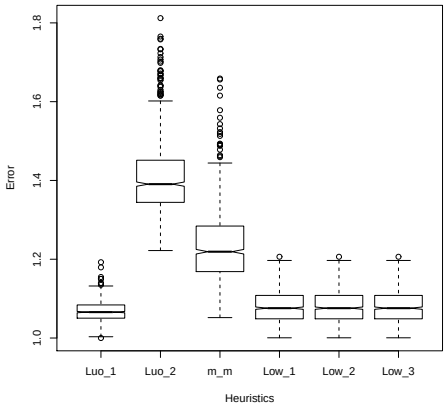
3.6 Experimental Results

The primary goal of the experimental evaluation is to compare six heuristics: Luo_1, Luo_2, m_m, and the low computational complexity heuristics Low_1, Low_2, and Low_3 (Table 3.2). We also present the results regarding the approximation factor, and number of best solutions found objectives. Following the general direction depicted in Luo et al. [111], we average the simulation results using 1000 different ETCs of the same type.

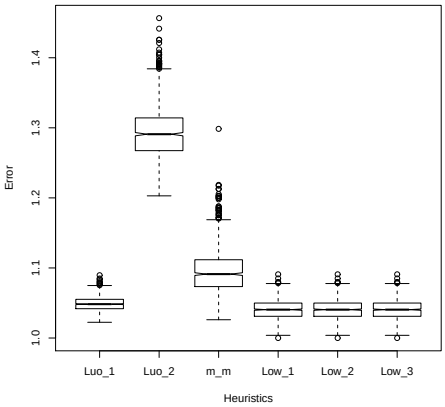
3.6.1 Performance Ratio of the Approximation Factor

Figure 3.10 shows the performance ratio (relative error) of the approximation factor. Figure 3.10(a) and Figure 3.10(b) show two representatives figures of the results for 512 tasks scheduled on 16 machines. Figure 3.10(c) and Figure 3.10(d) show two representatives figures of the results for 1024 tasks scheduled on 32 machines. Figure 3.10(e) and Figure 3.10(f) show two representatives figures of the results for 2048 task scheduled on 64 machines.

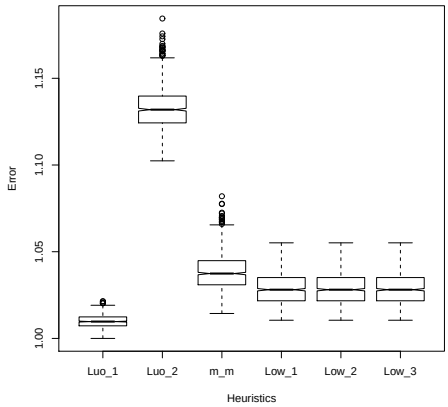
We present the results of several scenarios: partial-consistent scenario with $V_{task} = 1.1$ and $V_{machine} = 0.6$ (1.1×0.6) (Figure 3.10a), with (0.4×0.1) (Figure 3.10c); original-consistent scenario with (0.6×0.5) (Figure 3.10b), with (1.1×0.6) (Figure 3.10d), with (0.2×0.1) (Figure 3.10e), and with (1.1×0.6) (Figure 3.10f).



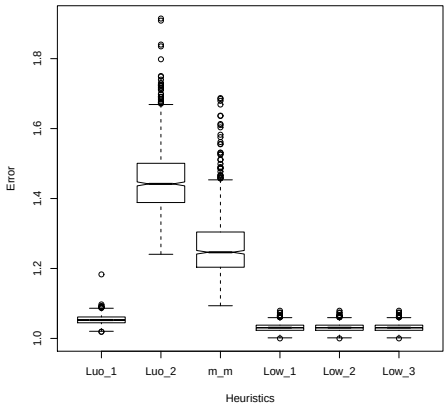
(a) partial-consistency with $V_{task} = 1.1$ and $V_{machine} = 0.6$ for 512×16



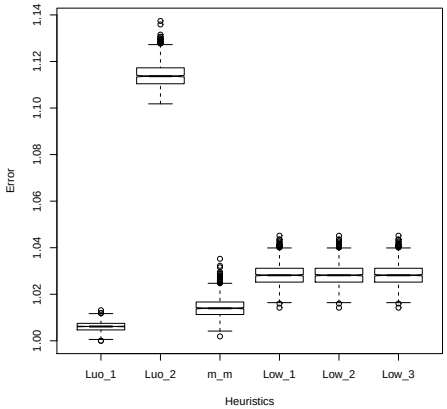
(b) original-consistency with $V_{task} = 0.6$ and $V_{machine} = 0.5$ for 512×16



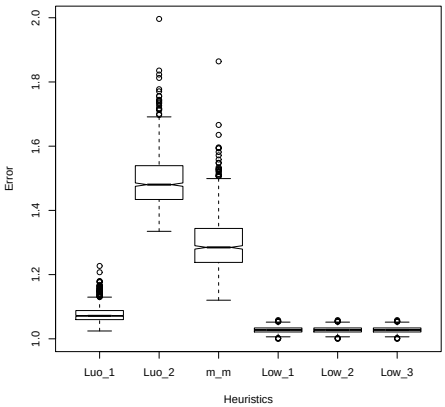
(c) partial-consistency with $V_{task} = 0.4$ and $V_{machine} = 0.1$ for 1024×32



(d) original-consistency with $V_{task} = 1.1$ and $V_{machine} = 0.6$ for 1024×32.

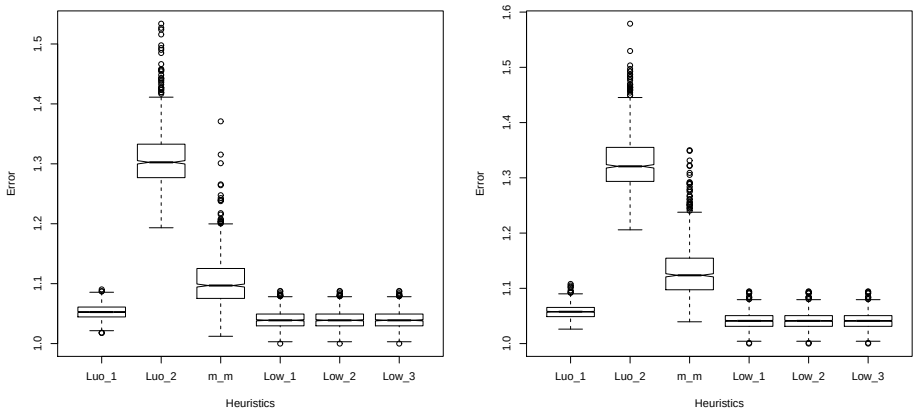


(e) original-consistency with $V_{task} = 0.2$ and $V_{machine} = 0.1$ for 2048×64.



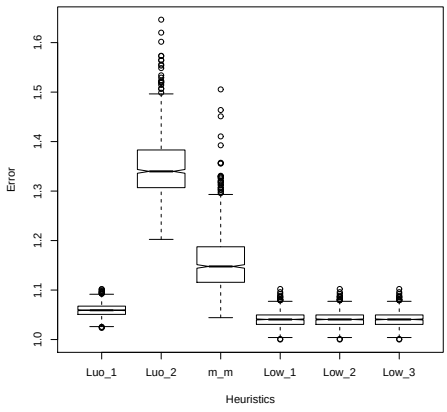
(f) original-consistency with $V_{task} = 1.1$ and $V_{machine} = 0.6$ for 2048×64.

Figure 3.10: Approximation factor error for makespan

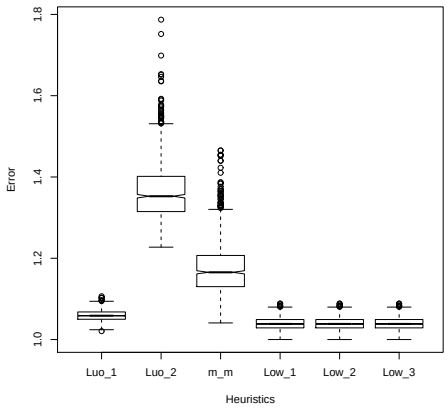


(a)

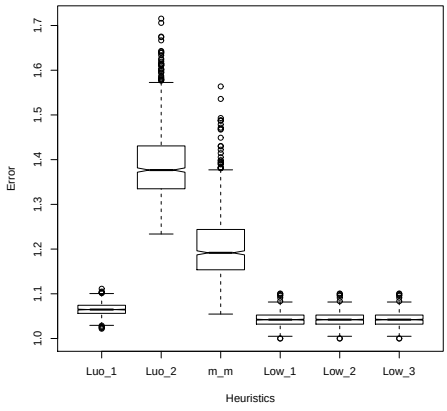
(b)



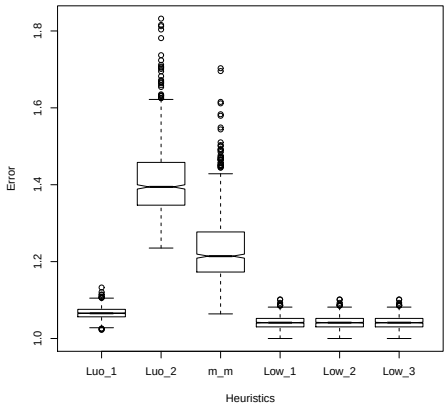
(c)



(d)



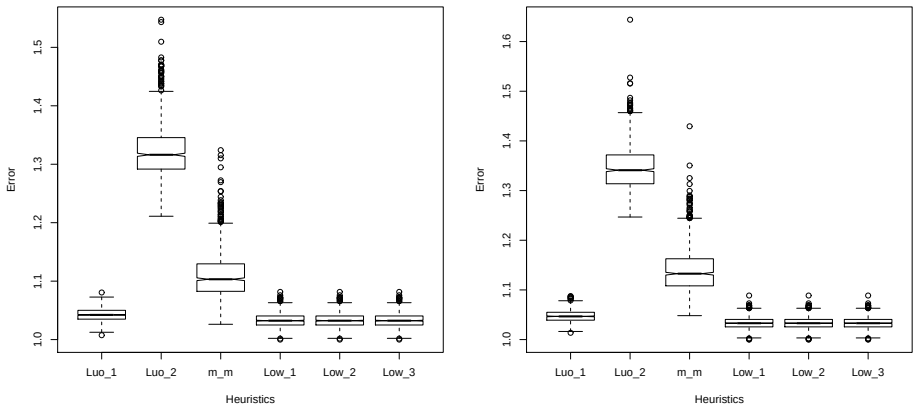
(e)



(f)

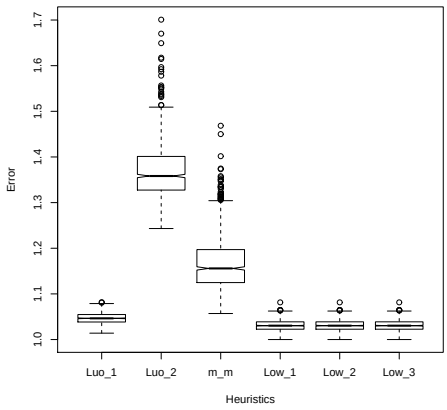
Figure 3.11: Approximation factor error of original-consistency scenario considering high machine heterogeneity (0.6) and high task heterogeneity. The task heterogeneity varies from 0.6 to 1.1 and the ETC matrix size is 512x16

The Luo_2 heuristic has worst results in all of the scenarios and experiments. The results are from 15% to 50% worst than the lower bound of the optimal solutions. The Luo_1, Low_1, Low_2, and Low_3 heuristics produce high quality results that are from 1% to 30% from the lower bound of the optimal solutions. Figure 3.10 (a) shows similar results for Luo_1, Low_1, Low_2, and Low_3 heuristics that are better than m_m. Figure 3.10 (e) shows slightly better results of Luo_1, compared m_m, Low_1, Low_2, and Low_3 heuristics. In all other scenarios Low_1, Low_2, and Low_3 heuristics produce better results. The best behavior of Low_1, Low_2, and Low_3 *low computational complexity* heuristics is in original-consistency cases, which refers to the most generic and real scenarios [26]. It can be validated in Figure 3.11. The figure depicts the results for the original-inconsistent instances This is mainly due to the high heterogeneity of these instances. The benefit of exploiting the heterogeneity of the applications and resources to maximize the performance of the system is more apparent. The score function exploits resource capabilities and task requirements by scheduling tasks to a machine that not only minimizes the completion time, but also executes individual tasks faster. On the other hand, results of Low_1, Low_2, and Low_3 are improved with increasing the size of the computing system from 16 to 64 machines (Figure 3.12 and Figure 3.13). The low complexity heuristics show their better scalability compared with related heuristics because of the low complexity feature and the score function.

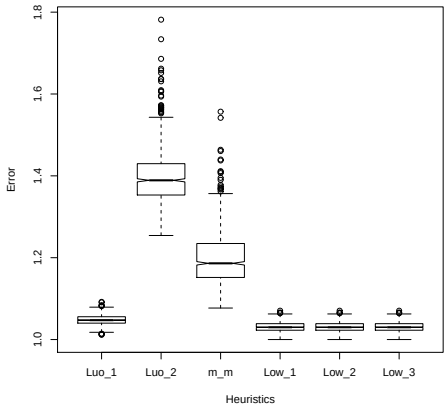


(a)

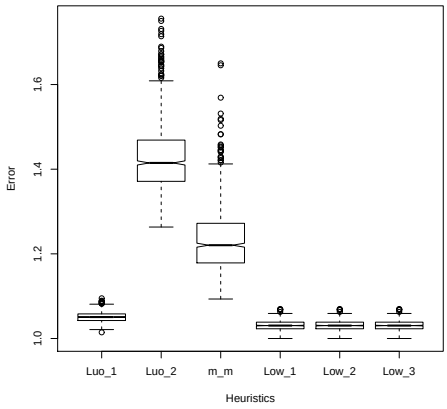
(b)



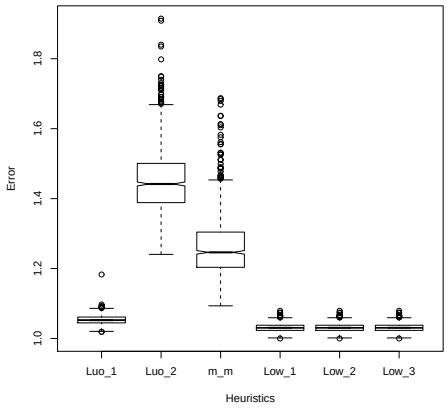
(c)



(d)

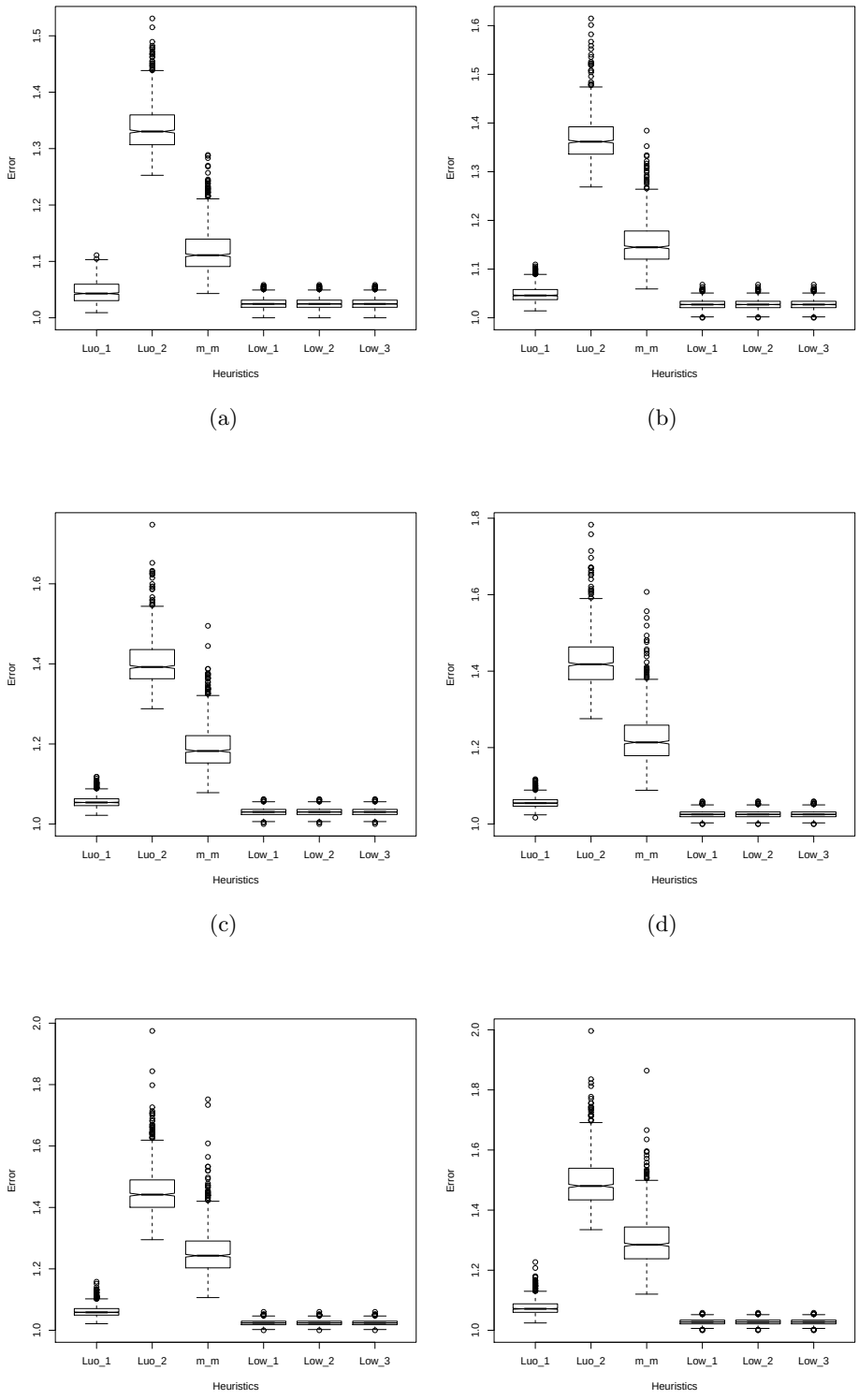


(e)



(f)

Figure 3.12: Approximation factor error of original-consistency scenario considering high machine heterogeneity (0.6) and high task heterogeneity. The task heterogeneity varies from 0.6 to 1.1. The ETC matrix size is 1024x32

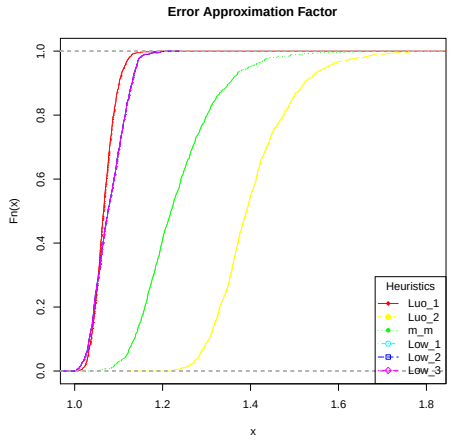


(a) (b)

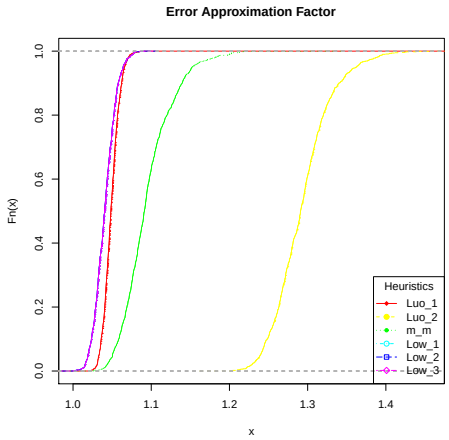
(c) (d)

(e) (f)

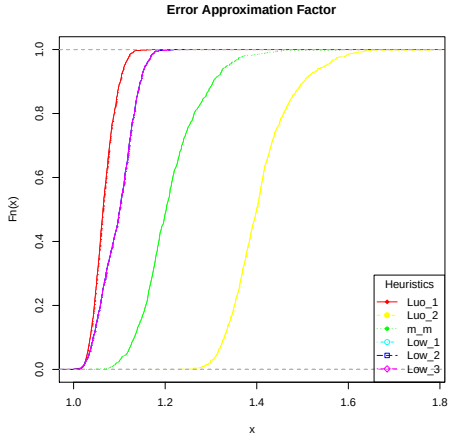
Figure 3.13: Approximation factor error of original-consistency scenario considering high machine heterogeneity (0.6) and high task heterogeneity. The task heterogeneity varies from 0.6 to 1.1. The size of the ETC matrix is 2048 tasks and 64 machines



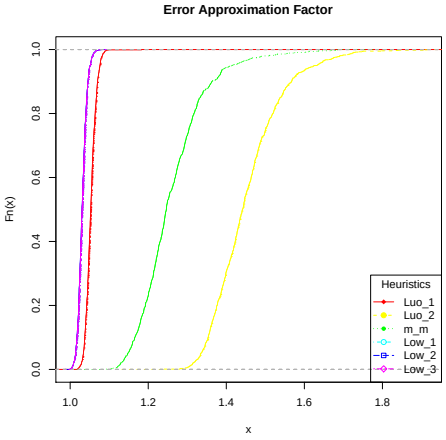
(a) partial-consistency with $V_{task} = 1.1$ and $V_{machine} = 0.6$ for 512×16



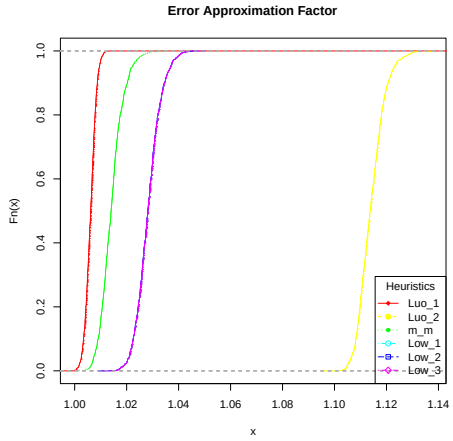
(b) original-consistency with $V_{task} = 0.6$ and $V_{machine} = 0.5$ for 512×16



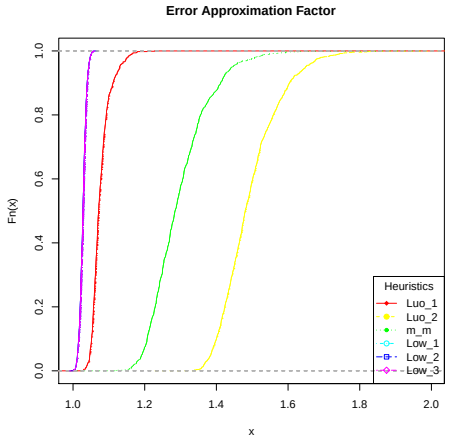
(c) partial-consistency with $V_{task} = 0.4$ and $V_{machine} = 0.1$ for 1024×32



(d) original-consistency with $V_{task} = 1.1$ and $V_{machine} = 0.6$ for 1024×32



(e) original-consistency with $V_{task} = 0.2$ and $V_{machine} = 0.1$ for 2048×64



(f) original-consistency with $V_{task} = 1.1$ and $V_{machine} = 0.6$ for 2048×64

Figure 3.14: Performance profile of the approximation factor error

3.6.2 Performance Profile

In this section, we present the performance profiles of the strategies to complement the study of averages. Figure 3.14 shows the performance profiles of ρ, performance ratios γ (see Section 3.5.1), in the interval $\tau = [1, ..., 2]$ to provide objective information for analysis of a test set. Strategies with large probability $\rho(\tau)$ for smaller τ are to be preferred. For instance, in Figure 3.14(b), for heuristic Luo_1 $\rho(1.05) = 0.8$ and for heuristics Low_1, Low_2, and Low_3 $\rho(1.05) = 0.95$ means that heuristic Luo_1 performed at most 5% worse than the best strategy on 80% of instances considered and heuristics Low_1, Low_2, and Low_3 upon 95%. We show the results of the same scenarios as presented in Figure 3.10. We found small discrepancies in the performance ratios on a substantial percentage of the problems. The Luo_1, Low_1, Low_2, and Low_3 heuristics have the highest probability of being the better strategies. The probability that they are the winners on a given problem within factors of 1.1 of the best solution is about 1. If we choose being within a factor of 1.01 as the scope of our interest, then either Luo_1 or Low_3, would suffice with a probability 0.95. Luo_1 dominates other heuristics in partial-consistent scenarios with (0.4×0.1) (Figure 3.14c), and the original-consistent scenarios with (0.2×0.1) (Figure 3.14e). The Low_1, Low_2, and Low_3 heuristics dominate other heuristics in the original-consistent scenario with (1.1×0.6) (Figure 3.14f). In other scenarios, the Luo_1, Low_1, Low_2, and Low_3 heuristics show the similar results.

The higher values of V_{tasks} in the original-consistent scenarios, the better behavior of the *low computational complexity* heuristics. They show better results compared to Luo_1 in almost all original-consistent scenarios with the higher values of V_{tasks} and $V_{machine}$. Figure 3.17 validates the results.

3.6.3 Number of Best Solutions Found

Tables 3.6 and 3.7 present the evaluation results of the heuristics considering the number of best solutions found (B), and the number of best solutions found together with another heuristics (EB). They show the better behavior of low complexity heuristics in (512×16) case study. For instance, heuristics Low_2 and Low_3 have the best solutions in 895 instances of the $V_{tasks} = 1.1$ and $V_{machines} = 0.6$ original-consistent scenario. Note that EB is the complement to B results.

3.6.4 Flowtime Comparison

The most natural measure of the quality of service received by a client is the flowtime, which is defined as the time since the client submits a request until it is completed. The flowtime is closely related to the user experience, as it measures the amount of time an user has to wait to get his jobs serviced [16] .

In this section, we compare one of our proposed *low complexity heuristics* with the best heuristic reported in the literature, Luo_1, for the same case study (i.e., makespan) considering the flowtime criterion. We only show the comparison results for Low_3 against Luo_1 because results for Low_1 and Low_2 are comparable to Low_3.

We calculate the percentage of the gain ($\%Gain$) of the mean flowtime (Eq. 3.5) obtained

V_{tasks}		Heuristics					
		1	2	3	4	5	6
0.1	B	358	1	431	0	0	0
	EB	203	0	203	0	7	7
0.6	B	226	0	28	0	0	0
	EB	0	0	0	0	746	746
0.7	B	172	0	4	0	0	0
	EB	0	0	0	0	824	824
0.8	B	164	0	0	0	0	0
	EB	0	0	0	0	836	836
0.9	B	135	0	0	0	0	0
	EB	0	0	0	0	865	865
1	B	112	0	0	0	0	0
	EB	0	0	0	0	888	880
1.1	B	105	0	0	0	0	0
	EB	0	0	0	0	895	895

Table 3.6: B and EB table, original-consistent ETCs with $V_{machine} = 0.6$ for 512×16

$V_{machines}$		Heuristics					
		1	2	3	4	5	6
0.1	B	511	0	0	0	0	0
	EB	0	0	0	0	489	489
0.2	B	381	0	2	0	0	0
	EB	0	0	0	0	617	617
0.3	B	349	0	5	0	0	0
	EB	0	0	0	0	646	646
0.4	B	360	0	14	0	0	0
	EB	0	0	0	0	626	626
0.5	B	295	0	15	0	0	0
	EB	0	0	0	0	690	690
0.6	B	226	0	28	0	0	0
	EB	0	0	0	0	746	746

Table 3.7: B and EB table, original-consistent ETCs with $V_{task} = 0.6$ for 512×16

by Low_3 over Luo_1.

$$\%Gain = \left(1 - \frac{flowtime_{Low\_3}}{flowtime_{Luo\_1}}\right) \times 100. \tag{3.5}$$

The positive number shows advantage of our Low_3 compared with Luo_1. Table 3.8 shows the results for the percentage gain of the mean flowtime averaged over the 144 heterogeneous scenarios and different consistencies (FC, PC, and OC refers to full, partial, and original consistency respectively) for the three instances size: (a) 512×16, (b) 1024×32, and (c) 2048×64. Luo_1 is better if V_{mach} is 0.1, in full-consistency scenario for 512×16, and with increasing V_{tasks} increases from 0.1 to 0.6. In all other scenarios, our *low computational complexity* strategies outperform Luo_1 from 0.89% to 12.33%. In summary, 87% of the results Low_3 (same to Low_1 and Low_2) is better than the results obtained by Luo_1 for 512×16. For 1024×32, approximately 90% of the results and for 2048×64 approximately 92% are better than the results obtained for heuristic Luo_1. The results show the best scalability behavior of the *low complexity heuristics* against Luo_1.

		512×16			1024×32			2048×64		
		FC	PC	OC	FC	PC	OC	FC	PC	OC
V_m for $V_t = 0.6$	0.1	−0.15	**2.82**	**3.01**	**0.01**	**2.12**	**2.85**	**0.11**	**1.83**	**2.57**
	0.2	**0.11**	**4.3**	**4.41**	**0.28**	**3.25**	**3.97**	**0.4**	**2.7**	**3.49**
	0.3	**0.53**	**5.33**	**5.38**	**0.62**	**3.89**	**4.65**	**0.73**	**3.11**	**4.37**
	0.4	**0.97**	**6.26**	**6.58**	**1**	**4.1**	**5.67**	**1.10**	**3.39**	**4.84**
	0.5	**1.4**	**7.3**	**7.66**	**1.36**	**6.15**	**6.48**	**1.39**	**4.83**	**5.39**
	0.6	**1.7**	**8.53**	**9.09**	**1.46**	**6.87**	**7.47**	**1.33**	**5.02**	**6.54**
V_t for $V_m = 0.1$	0.1	−0.89	**0.79**	**1.03**	−0.56	**0.5**	**0.66**	-0.28	-0.48	**0.33**
	0.2	−0.66	**1.58**	**1.81**	−0.37	**1.35**	**1.52**	-0.15	**0.41**	**1.23**
	0.3	−0.49	**2.05**	**2.27**	−0.23	**1.84**	**2**	-0.05	**0.93**	**1.73**
	0.4	−0.36	**2.35**	**2.57**	−0.13	**1.60**	**2.35**	**0.01**	**1.29**	**2.07**
	0.5	−0.24	**2.61**	**2.84**	−0.05	**1.89**	**2.62**	**0.07**	**1.59**	**2.34**
	0.6	−0.15	**2.82**	**3.01**	**0.01**	**2.12**	**2.86**	**0.11**	**1.83**	**2.6**
V_t for $V_m = 0.6$	0.6	**1.7**	**8.53**	**9.09**	**1.46**	**6.87**	**7.47**	**1.33**	**5.02**	**6.54**
	0.7	**2.04**	**9.46**	**9.88**	**1.82**	**7.98**	**8.44**	**1.58**	**6.27**	**7.63**
	0.8	**2.31**	**10.27**	**10.57**	**2.06**	**8.94**	**9.28**	**1.84**	**7.43**	**8.54**
	0.9	**2.59**	**10.98**	**11.22**	**2.31**	**9.88**	**10.11**	**2.09**	**8.53**	**9.39**
	1	**2.75**	**11.6**	**11.78**	**2.58**	**10.7**	**10.76**	**2.26**	**9.46**	**10.13**
	1.1	**3.01**	**12.12**	**12.33**	**2.82**	**11.30**	**11.54**	**2.45**	**10.31**	**10.91**

Table 3.8: Percentage gain (%Gain) of the mean flow time for Low_3 over Luo_1.

3.6.5 Time and Memory

Due to the low complexity of *low-complexity heuristics*, the results of the scheduling time calculation is efficient as well as the memory that each algorithm uses.

Time and memory used for all heuristics were measured for each size instance. Each heuristic has been executed separately and the experiments are independent. Figure 3.15 shows the time results in milliseconds of these executions. The logarithmic scale was used to

emphasize the results. As can be observed, *low-complexity heuristics* have better behavior. The algorithms use less time to compute a schedule over all the set of instances. Figure 3.16 shows the memory used in MB to calculate each heuristic as well. We can observe in this figure, that *low-complexity heuristics* outperform the related algorithms for all the sizes of the instances and the gain in memory space is more important when the size of the instances scales. The evaluated heuristics use much more memory when the instances scale because the algorithms evaluate the completion time for the remaining tasks to be scheduled at each step of the loop, furthermore, two of these evaluated heuristics, use TPD graph. On the contrary, *low-complexity heuristics* have a good scalability and low overhead, memory use is reduced, because each step of the algorithm only considers one task to be scheduled, the task with the highest priority.

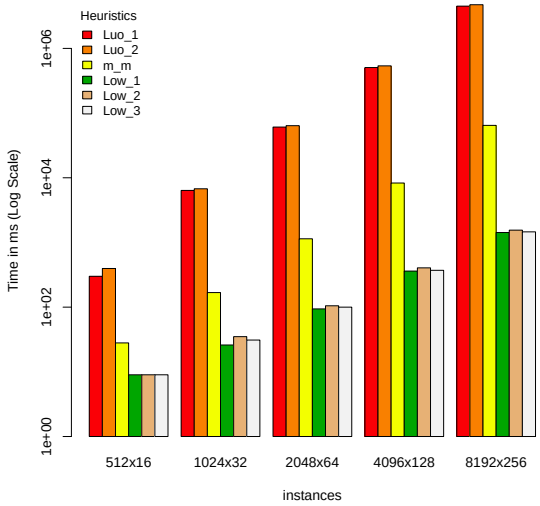


Figure 3.15: Time consumed for each heuristic.

3.6.6 Summary

We summarize results as follow. For full-consistency the instances present less heterogeneity; the TPD based heuristics are able to construct a *full* TPD graph allowing to generate a good sequences of task execution leading to good schedules. For partial and original consistency the instances present more heterogeneity, then the low complexity heuristics are able to exploit the heterogeneity to optimize the schedule, on the opposite the TPD based heuristics construct a TPD graph composed of more independent tasks, hence the execution order of tasks is harder to evaluate. Moreover, low complexity heuristics run faster and show good scalability than related heuristics allowing them to be preferred in large-scale computing systems.

Based on the analysis of the experiments results, we can conclude the following:

- Heuristics with TPD profit of the consistency for the first sorting arranged, that is for full-consistency scenarios, but for original and partial scenarios the performance

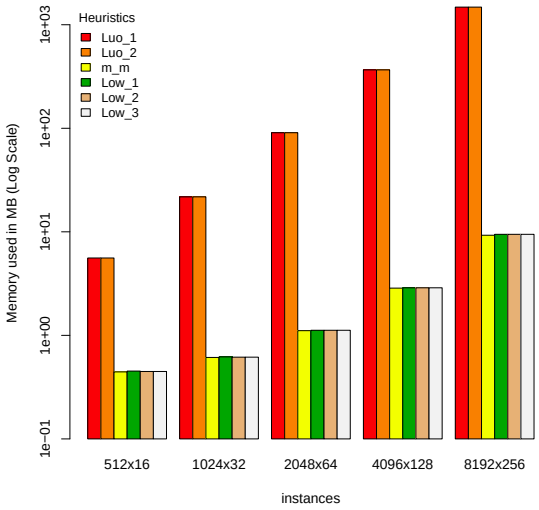


Figure 3.16: Memory used for each heuristic to calculate scheduling

decrease.

- *low computational complexity* heuristics are the best performing heuristics among the heuristics in *original-consistency* cases showing similar behavior in *partial-consistency* scenarios.

Therefore, for a practical mapping problem, Algorithm 9 generates the efficient results considering makespan criterion.

Algorithm 9: Best performance heuristic

```
if ETC input is full-consistency case then
    return Heuristic 1;
else
    return low computational complexity heuristics;
```

3.7 Extra Experimentation Evaluating Energy Efficient

For extra experimentation, we compare our proposed algorithms by analyzing the results of numerous simulations featuring high heterogeneity of resources, and/or high heterogeneity of applications. Simulations studies are performed to compare these heuristics with the well-known *min-min* [94, 82, 26]. The main objective of this extra experimental section is to contribute to the optimization of energy consumption. The energy consumption of an idle resource at any given time is set using a minimum voltage based on the processor's architecture. we assume that energy is the amount of power used over a specific time interval [107]. Fur-

thermore, we consider that resources in the target system are incorporated with an effective energy-saving mechanism for idle time slots [115, 101].

3.7.1 Energy Model

The energy model used in this work is derived from the power consumption model in digital complementary metal-oxide semiconductor (CMOS) logic circuitry. The power consumption of a CMOS-based microprocessor is defined to be the summation of capacitive power, which is dissipated whenever active computations are carried out, short-circuit and leakage power (static power dissipation). The capacitive power (P_c) (dynamic power dissipation) is the most significant factor of the power consumption. It is directly related to frequency and supply voltage, and it is defined as [31, 102]:

$$P_c = AC_{eff}V^2 f, \tag{3.6}$$

where A is the number of switches per clock cycle, C_{eff} denotes the effective charged capacitance, V is the supply voltage, and f denotes the operational frequency. The relationship between circuit delay (T_d) and the supply voltage is approximated by (Eq. 3.7):

$$T_d \propto \frac{C_{ef}V}{(V - V_{th})^{\alpha}}, \tag{3.7}$$

where C_L is the load capacitance, V_{th} is the threshold voltage, and α is the velocity saturation index which varies between one and two (α=2). Because the clock frequency is proportional to the inverse of the circuit delay, the reduction of the supply voltage results in reduction of the clock frequency. It would be not beneficial to reduce the CPU frequency without also reducing the supply voltage, because in this case the energy per operation would be constant.

The energy consumption of any machines in this paper is defined as:

$$E_c = \sum_{i=1}^{n} AC_{ef}V_i^2 fETC[i][M[i]], \tag{3.8}$$

where $M[i]$ represents a vector containing the machine m_j where task t_i is allocated, V_i is the supply voltage of the machine m_j. On the other hand, the energy consumption during idle time is defined as:

$$E_i = \sum_{j=1}^{m} \sum_{idle_{jk} \in IDLES_j} AC_{ef}Vmin_j^2 I_{jk}, \tag{3.9}$$

where $IDLES_j$ is the set of idling slots on machine m_j, V_{minj} is the lowest supply voltage on m_j, and I_{jk} is the amount of idling time for $idle_{jk}$. Then the total energy consumption is defined as:

$$E_t = E_c + E_i \tag{3.10}$$

3.7.2 Experimental Evaluation

We compare the proposed heuristics and the min-min heuristic by simulation using randomly built ETCs. As we already mentioned, the ETC model used can be characterized by three parameters [26, 8]: the first parameter is machine heterogeneity, on which, this time, we can distinguish among *low* and *high* machine heterogeneities. The second parameter is task heterogeneity, we can also distinguish among *low* and *high* task heterogeneities. And the third parameter is the consistency. Table 3.9, shows the twelve combinations of heterogeneity types (task and machines) and consistency classifications in the ETC model. The consistency categories are named for the correspondent initial letter (*f* stands for full-consistent, *p* for partial-consistent, *o* for original-consistent, *lo* stands for low heterogeneity and *hi* for high heterogeneity). Hence, a matrix named *f_lolo* corresponds to a full-consistent scenario with low task heterogeneity and low machine heterogeneity.

Consistency		
full-consistent	*partial-consistent*	*original-consistent*
f-lolo	p-lolo	o-lolo
f-lohi	p-lohi	o-lohi
f-hilo	p-hilo	o-hilo
f-hihi	p-hihi	o-hihi

Table 3.9: Consistency and heterogeneity combinations in the ETC model

3.7.2.1 Experiments

For the generation of these ETC matrices we have used the coefficient of variation based method (COV) introduced in [8]. To simulate different heterogenous computing environments we have changed the parameters μ_{task}, V_{task} and $V_{machine}$, which represent the mean task execution time, the task heterogeneity, and the machine heterogeneity, respectively as we define previously. The value of V_{task} is larger for a higher task heterogeneity. The value of $V_{machine}$ is larger for a higher machine heterogeneity. We have used the following parameters: V_{task} and $V_{machine}$ equal to 0.1 for low case respectively and 0.6 for high case, and $\mu_{task} = 100$. The heterogeneous ranges were chosen to reflect the fact that in real situations there is more variability across the execution time for different tasks on a given machine than that across the execution time for a single task on different machines [111].

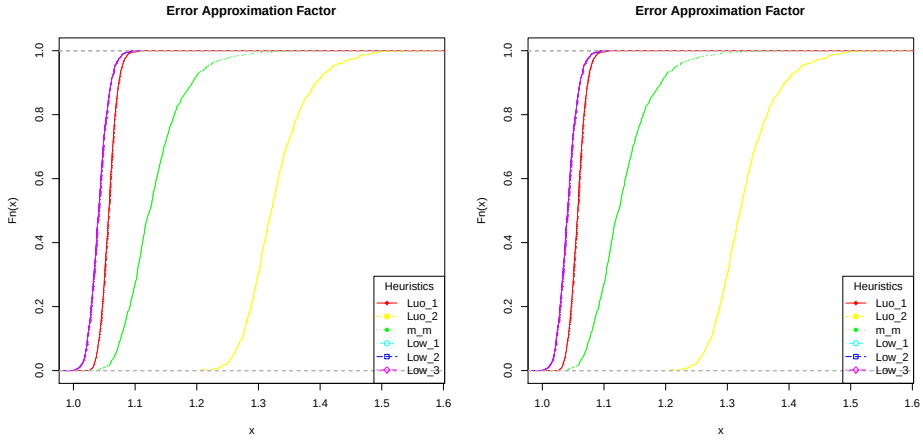
As we are considering batch mode algorithms, we assume in both cases that all tasks have arrived to the system before the scheduling event. Furthermore, we consider that all the machines are idle or available at time zero, this can be possible by considering advance reservation. We have generated 1200 instances, 100 for each twelve cases to evaluate the performance of the heuristics. We have generated instances with 512 tasks in size to be scheduled on 16 machines. Additionally, we have considered different voltages for the machines. We randomly assigned these voltages to machines by choosing among three different set. The first set considers 1.95 and 0.8 Volts for maximum or active state and minimum or idle state, respectively. The second set is 1.75 Volts at maximum state and 0.9 Volts at idle

state. Finally, the last set considers 1.6 Volts at maximum level and 0.7 Volts at idle level.

3.7.2.2 Results

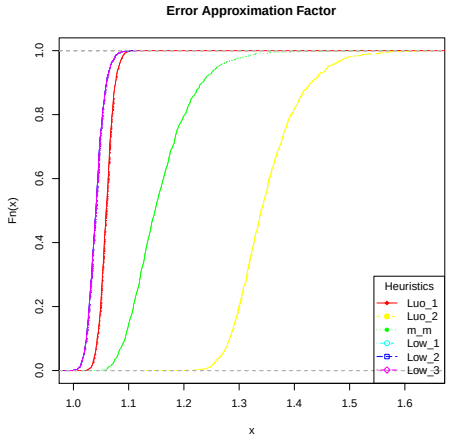
The results for the algorithms are depicted from Figure 3.18 to 3.20. We show normalized values of makespan, flowtime and energy for each heuristic against min-min for λ-Score-Function-values in the interval [0, 1]. The normalized data were generated by dividing the results for each heuristic by the maximum result computed by these heuristics. We only show the curves for the high task and high machine heterogeneity for the three different scenarios which are the most significant results. The legends *m-m n_mksp*, *m-m n_flow* and *m-m n_energy* in the figures stand for makespan, flowtime and energy of min-min.

We can observe from these figures that the proposed heuristics follow the same performance behavior according to the scenarios. Relative values range are biggest for the full-consistent instances than partial-consistent and original-consistent. The results clearly demonstrate that energy efficiency is the best for the full-consistent instances. It may be related to the fact, that the makespan has worse results. However, for value of $\lambda = 0.8$ the proposed heuristics can perform as well as min-min for all the three considered metrics. We can also observe that the proposed algorithms can improve makespan and flowtime results for lambda for partial-consistent and original-consistent instances. Interestingly, if the instance is more original-consistent, our algorithms performs better. The benefit of exploiting the heterogeneity of the applications and resources to maximize the performance of the system and energy is more apparent. This is mainly because these instances are the ones presenting the highest original-consistency and heterogeneity. In terms of flowtime, all the heuristics are as efficient as min-min, however, the proposed heuristics have lower complexity.

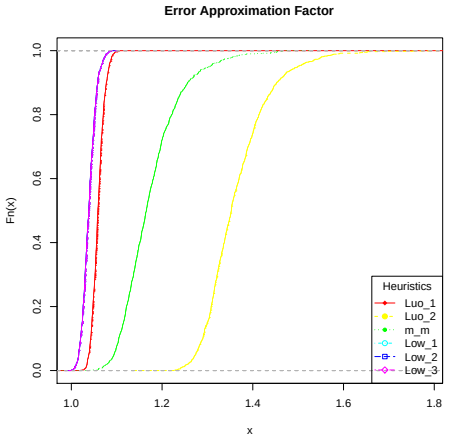


(a)

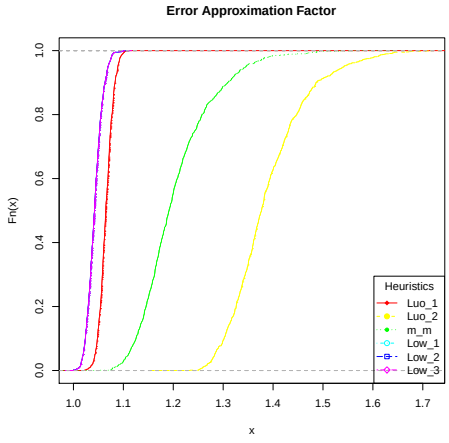
(b)



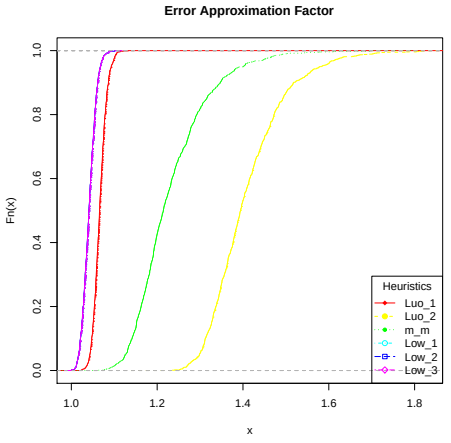
(c)



(d)



(e)



(f)

Figure 3.17: Performance profile of the approximation factor error. The scenario is original-consistency considering high machine heterogeneity (0.6) and high task heterogeneity. The task heterogeneity varies from 0.6 to 1.1 and the ETC matrix size is 512x16

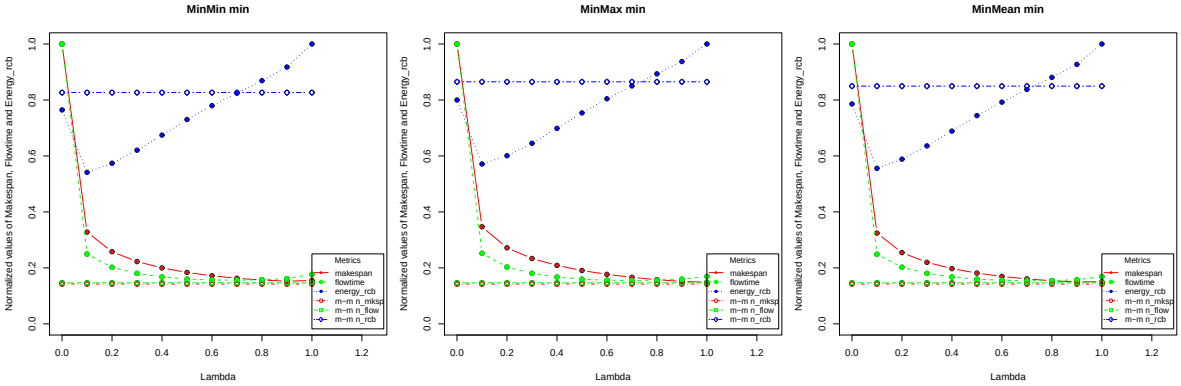


Figure 3.18: Relative performances of the schedules produced by different heuristics in the f-hihi instances.

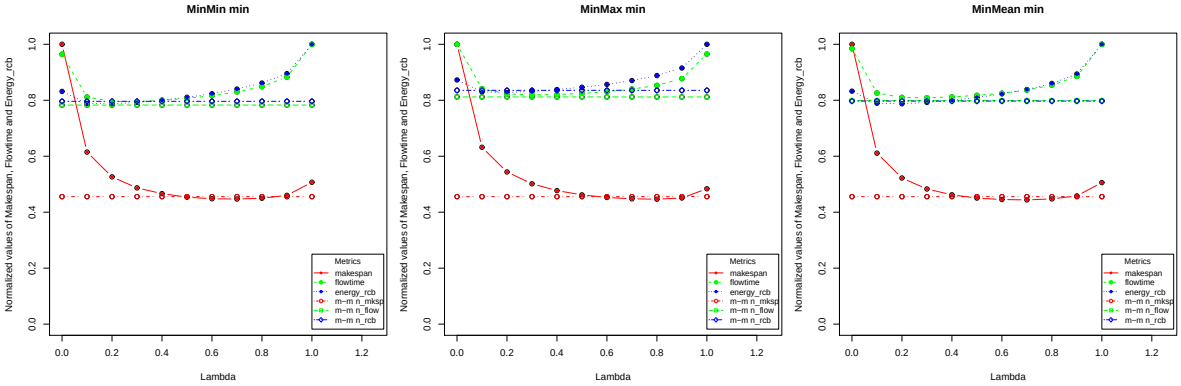


Figure 3.19: Relative performances of the schedules produced by different heuristics in the p-hihi instances.

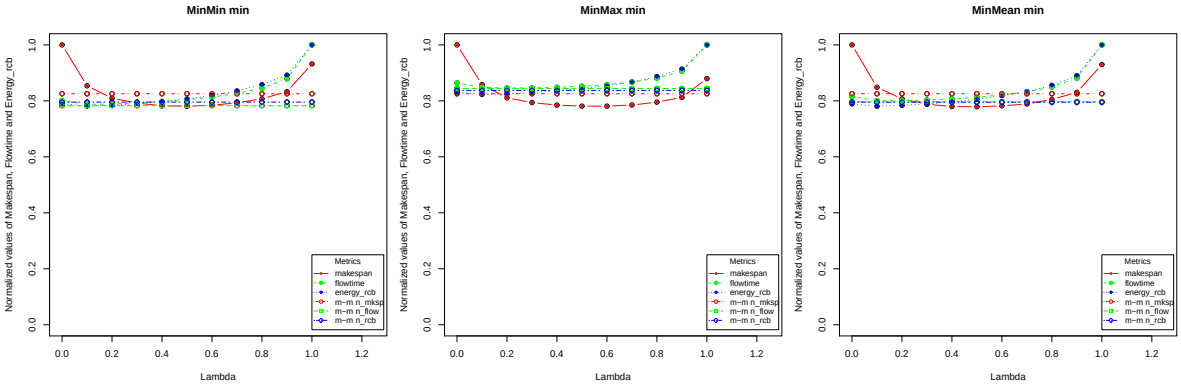


Figure 3.20: Relative performances of the schedules produced by different heuristics in the o-hihi.

Chapter 4

Opportunistic Cloud Computing

Contents

4.1 Desktop, Volunteer and Opportunistic Computing

Large-scale computing platforms and current networking technology enable sharing, selection, and aggregation of highly heterogeneous resources for solving complex real problems. **Opportunistic computing** are distributed platforms built out of the available resources of an existing hardware platform that harvest the computing power of non-dedicated resources when they are idle. They are mainly built using two approaches: (1) a computing node will run a program that will provide access to a set of resources of the host in a limited manner, this is what is done in the BOINC [10] project for example and (2) running on each host two environments side-by-side, one for the user of the machine and one for the computing tasks. For this approach isolation of both environments is critical, the use of virtualization is hence a promising approach.

According to HPC top500 list [157], Latin-American countries do not provide enough dedicated clustering infrastructure compared to the United States or Europe. Dedicated clusters are expensive, so to provide HPC computing facilities, these countries could use the current resources that universities or companies provide. In the event of universities, computer rooms are a good approach to take advantage of. In the case of Latin-America, it's easy to find computer rooms adding more than 5,000 cores, but very hard to find dedicated clusters for research adding more than 200 cores; so an opportunistic computing infrastructure could be a good approach. To implement this approach some issues need to be investigated beforehand, like the impact on energy consumption, isolation of the end-user (e.g. the human user currently using the hardware) and computing (i.e. the cluster task) environments and the division of resources between these two environments keeping in mind that end-users

have priority over any computing environment. To exploit the existence of opportunistic computing, it is important to provide quality of service mechanism for both categories of users, considering transparency for end users, by defining patterns for sharing resources. Moreover, energy is an important and expensive resource that allows the proper functioning of HPC. So, it is essential to provide a mechanism that is also energy efficient.

Virtual machines are a natural solution to provide a complete isolation between both environments computing and end-user ones. The main purpose is to provide a virtualization mechanism that guarantees the properties mentioned above. It must be taken into account that hardware resources must be shared between these two environments in such a way that both kinds of users have the amount of resources needed to perform their tasks. However, the energy efficiency of using virtualization on opportunistic infrastructures to provide a high performance computing environments is still in doubt.

4.1.1 Literature Review

Opportunistic grids is one of the particular implementation of the opportunistic computing. They are used to optimize idle time of desktop machines to perform high performance, high throughput computing [34, 64, 44]. Projects such as Worm [150], Condor [108], GIMPSs [142], SETI@home [11], Distributed.net [114], BOINC [10] are some samples that exploit capabilities of unused computing resources around the world for executing single-purpose application. Worm and Condor exploit idle resources available in an organization. GIMPSs and SETI@home use volunteer computing resources through Internet. Distribiuted.net and BOINC support the execution of multiple-purpose applications over an Internet infrastructure whose resources are managed by a central organization, this last is also the middleware used by SETI@home project [11]. Condor-G [67], InteGrade [72] and Bayaniham.NET [145] allow the aggregation of idle or dedicated resources available in different administrative domains through grid middleware. A common characteristic of all these projects is that, the execution of any grid application is on the physical resources. XtremWeb [69], OurGrid [12], SharedGrid [13], BOINC-CernVM [30], LCH@home [40] execute grid applications in a virtualized environment, Cloud@Home [51] reuse "domestic" computing resources to build voluntary contributors Clouds, *"anyone can experience the power of Cloud Computing, both actively providing his/her own resources and services, and passively submitting his/her applications"* [51]. A virtual machine is executed on each desktop computer. Bio-UnaGrid [162] facilitates the automatic execution of intensive-computing workflows that require the use of existing application suites and distributed computing infrastructures. The infrastructure is a dedicated cluster and a computer lab.

Several approaches have been proposed to reduce power consumption in computer systems by improving the environment where they are deployed, as well as a better design at the hardware and software level. Concerning the deployment environment, the entire infrastructure is considered, mainly including the cooling system of computing resources. Regarding the hardware, researchers are developing more economic computers in terms of energy efficiency. Recently, new hardware designs, for example, next generation memory solutions and solid state drive, are emerging to improve both the computing speed and the energy consumption in computer resources. On the infrastructure software side, developers are designing more efficient algorithms avoiding excessive processing [18, 49].

Other common techniques used to save energy in large scale computing systems are based

on power down mechanisms such as Dynamic Voltage Frequency Scaling (DVFS) [165] (equivalently, dynamic speed scaling, dynamic frequency scaling, dynamic voltage scaling) and Dynamic Power Management (DPM) [22]. Power down mechanism focus on identifying the optimal threshold times to transition to low-power modes during idle periods. Dynamic Frequency Scaling (DFS) already incorporated into many existing technology and commercial processors as for example SpeedStep by Intel or LongHaul by VIA Technologies, enables processors to dynamically changing its working voltage and speed without stopping or pausing the execution of any instruction. DVFS reduces energy consumption of processors based on the fact that power consumption in CMOS circuits has direct relation with frequency and the square of the supplied voltage. DPM on the other hand, consolidates applications on a minimum set of computing resources to maximize the number of resources that can be powered down while maximizing utilization of the used ones. Machines are powered down (i.e. inactive) when not used, and job placement decisions attempt to power a node back on only when is absolutely necessary. These techniques have been used at the level of resource manager including scheduling algorithms [92, 95, 96, 103, 118, 130] (see [158] and the references given there for more details).

Different works deal with energy efficiency issues in clusters and Grids. Orgirie et *al.* [127] propose a set of green policies to reduce the global energy consumption of a large scale experimental grid based on a model where users need to reserve resources and the resources are dedicated to the user during the reservation period. Lammie et *al.* [100] explore energy and performance trade-offs in the scheduling of grid workloads on large clusters. The authors analyze the effect of automated node scaling, CPU frequency scaling, and job assignment. Da Costa et *al.* [43] present a framework based on three components: an ON/OFF model based on an energy-aware resource infrastructure, a resource management system adapted for energy efficiency and a trust delegation component to assume network presence of sleeping nodes. Ponciano and Brasileiro [135] investigate energy-aware scheduling, sleeping and wake-up strategies in opportunistic grids. Sleeping strategies are employed to reduce the energy consumption of the grid during idleness periods; wake-up strategies are employed to choose a set of resources to fulfill a workload demand; and scheduling strategies are employed to decide which tasks to schedule to the available machines.

The energy consumption of under-utilized resources accounts for an important amount of the actual energy use in large scale distributed systems. In this context, resource consolidation through virtualization is an effective way to increase resource utilization and in turn reduces power consumption [101, 21, 97, 148]. The virtualization technology allows to create several VMs on a physical resource, and, therefore, reduces the amount of hardware in use and improve the utilization of the resources as below explained .

4.1.2 Desktop, Volunteer and Opportunistic Computing in the Cloud

Opportunistic computing and the well-known Desktop Grids and Volunteer Computing Systems (DGVCSs) are approaches of distributed systems aimed at the provision of large scale computing infrastructures, by taking advantage of non dedicated resources, most of them desktop computers presenting low levels of use. Those computers are available through Internet or Intranet environments, They have partial availability, they are highly heterogeneous, and they are part of independent administrative domains. They also have allowed the sharing of distributed computing resources, even though those resources are being used for end

users who should not experiment a significant reduction in the quality of service perceived. Furthermore, they offer a high return on investment for applications from a wide range of scientific domains (including computational biology, climate prediction, and high-energy physics). This strategy made the aggregation of millions of volunteer distributed computing resources possible, representing an effective solution to support e-science projects.

4.1.2.1 Opportunistic, Desktop and Volunteer Computer Taxonomy

After a careful analysis of some Opportunistic and DGVCSs projects, we have identified several characteristics which allow us to better study and classify them (a completed work is presented in [36]). These characteristics make it possible to understand the differences not only at a functional level but also on the kind of additional services they may provide. This section presents a summary of Opportunistic, Desktop and Volunteer Computer taxonomy organized around the main characteristics, which differentiate the presented projects: level of scalability, architecture, type of resource provider, scope of the supported applications, supported application models, platforms, portability, granularity for adding resources, license type, ability to specify the desired resources, ease of use and resource usage model.

- *Scalability.* It can be classified by Local Area Networks (LAN) or Internet. For LANs are looking to lever the computational resources in an organization or institution; in this case, the computing resources regularly belong to a single administrative domain, enabling to have a more stable and reliable connectivity, reduces risks associated to security issues and offers a high degree of control over the computing resources that are part of the system. Alternatively, for Internet looks for anonymous geographically distributed computational resources and deals with low-bandwidth communications issues (firewall, NAT, dynamic addressing, etc.), malicious resources and intrusion-related problems, which imply high security risks, unreliable communications and reduced resource availability. Although DGVCSs for LANs gain some benefits as a greater control over the shared resources as well as an availability and security improvement, they are limited to only use the available resources within an organization or institution. This is why the DGVCSs for the Internet are an option to group computational capabilities to the level of thousands or even millions of computers connected over the Internet; the price to pay: security, reliability and a reduced availability.

- *Architecture.* The different components are regularly organized using a centralized or distributed approach. A centralized organization uses the well-known client/server model, where there are users, resources providers and servers. Distributed organizations may be classified into two sub categories, those using a peer-to-peer scheme and those using a hierarchical approach. In a peer-to-peer organization, clients and resource providers do exist, but there is not a centralized server. In hierarchical approach, resource providers are organized in such a way that one system may send a work request to other system with available resources.

- *Type of resource provider.* There are two types of resource providers: voluntary and institutional. Systems with voluntary providers get their computing capabilities from computing whose owners/end users decide voluntarily to donate their underutilized

computing resources. These systems take advantage of underutilized computing resources while company staff performs daily activities. If the provider is institutional, it have greater control, allowing relaxed security policies.

- *Purpose.* Single-purpose, that consist in share their computing capabilities to solve a single specific problem. They are regularly administered by a single organization and are looking to lever the most available resources in environments such as the Internet. General purpose, that support applications for the resolution of different types of problems; they are regularly administered by several organizations, called virtual organizations, looking to solve different problems through the deployment of multiple applications. General purpose DGVCSs can be found in both corporate environments and the Internet. In the design of general purpose DGVCSs should be account adequate levels of portability to guarantee the system, can deploy various types of applications, as well as implementing facilities and tools to allow different virtual organizations to manage their applications. These systems must also implement mechanisms to ensure appropriate use of the computing resources, avoiding a single organization to seize all the computing resources that can be levered by the system.

- *Application model.* According to the applications to be executed, it can be grouped into two main categories: the master/slave model consisting of independent tasks and the parallel programming model which requires communication between processes within a task. In the master/slave model, a master process (server) sends a set of independent tasks to a set of slave processes. The master waits for each slave to execute the job and send back its result. The master receives the results and should have more tasks it assigns a new task to the slave. Tasks running on every slave are totally independent (no communication needed) and can be executed in parallel on different slaves. Execution of a workflow in which each element or workflow task can be run independently also belongs to this category. In this case, itÕs required the use of a tool that facilitates the synchronization between the different independent tasks of the workflow. In the parallel programming model, multiple processes cooperate to execute a common task. Such cooperation is achieved by means of communication using different parallel programming paradigms such as MPI (Message Passing Interface) [152], PVM (Parallel Virtual Machine) [154] or BSP (Bulk Synchronous Parallel) [159]. In these schemes, a task is carried out through the execution of several processes running on different computers. In such applications, priority should be given to different issues such as synchronization between processes, message passing, remote access to memory, delays in communications, among others. These systems are much more complex than those based on a client/server model because they have to deal with the aforementioned issues on platforms not intended to do so (shared resources, Internet scale, unpredictable availability, etc.)

- *Platform.* Depending on the platform used by resource providers to take advantage of idle computing resources, it can be classified into middleware based, Web-based, and virtualization-based. Middleware based is characterized by the need of installing a specific middleware on the resource providersÕ operating system. This middleware allows DGVCSs applications to be executed onto the resource provider system. Additionally, it provides management, security, configuration and accounting mechanisms, as well as

tools to control the level of intrusion to end users. In Web-based, applications must be developed in Java and made available as part of a Web page. Resource providers access that Web page through a regular browser and execute the code as an applet. Once executed, task results are typically returned to a server or to a centralized storage system for analysis. Finally, virtualization-based consists to use virtual machines to facilitate and expedite the installation, configuration and deployment of the applications required to take advantage of idle processing resources as well as to increase their portability. In these DGVCSs, resource-providers regularly have installed a virtualization tool such as VMware [164], Virtual Box (Sun Microsystems, Inc.), KVM (Kernel-based Virtual Machine) [19], Xen [17] or Hyper-v (Microsoft). Resources are configured, so they implement specific sharing policies and an image of a virtual machine containing all the software (O.S., libraries, middleware and applications) required to execute in the DGVCS context is locally stored. Once configured, the virtual machine runs based on configured parameters and it begins to form part of the system to execute the different tasks.

- *Portability.* It is related to the deployment capacity on the different operating system, that resource providers may have.The amount of available resources is limited only to those resources that use the operating system on which the software works. On the contrary, There are independent from the operating system, they are able to group together more computing resources using one of these strategies: 1) to use a language that is independent from the operating system, such as Java; 2) to create and compile the source code for each of the operating systems on which it is expected to work; and 3) to use virtualization tools to allow to run on a particular virtualization software, rather than onto a specific operating system.

- *Granularity of resource aggregation.* It could be individual or cluster aggregation. On individual aggregation, each computing machine is considered an independent resource provider, and the tasks are sending to each one of those resource providers. On the other hand, on cluster aggregation, the basic element is a pool of resources. Each pool regularly has a central element that receives the tasks and identifies the computers that are available to run them. Once a task is finished, its results are returned to the pool.

- *License.* Taking into account the type of licensing, it can be classified as proprietary or Open Source.

- *Specification of resources.* It opens two categories, those that allow users the specification of the required resources for the execution of their tasks, and those offering the same type of resources regardless the kind of jobs to be executed. In the first category, the specification is often expressed by a command or a file. When sending a job, the user specifies the required resources and the scheduler uses this information to select the resources to be assigned. This ensures a task run at a resource provider that has the required capabilities.

- *Usability from user perspective.* A DGVCS growth can be associated with the facility provided for the installation of the system agents in resource providers; this facility allows that sometimes millions of users can be part of the system. Some agents may

require experts in information technology (IT) to perform the deployment and configuration of the agent on resources, limiting the scalability of the system in environments such as the Internet. On the other hand, there exist general public oriented DGVCSs, which donÕt need any IT knowledge to configure a resource to be part of them. We identify three categories: when it is necessary IT staff, when it is just requiring some IT knowledge, and for conventional users.

- *Resource usage model*. Taking into account the form as it is planned and achieved the resource clustering, it can be classified into permanent and controlled. In permanent, resources are concurrently shared between tasks and end-user tasks. By the correct assignment of priorities, it takes advantage of exclusively idle or underutilized capabilities. On controlled, resources are used according to a policy defined by resource owners. Several mechanisms are implemented to enforce those policies: to use resources only when the screen saver is activated, to use configuration templates to define conditions to allow resource usage (i.e. based on a processor, RAM or disk activity level), to define time windows (i.e. resource usage is authorized only at nights), to analyze usage patterns, etc.

4.2 Energy Saving Strategies in Opportunistic Computing

In recent years, several approaches have been proposed to analyze and reduce power consumption in computer systems through better design of both the hardware and the software as well as by improving the environment where they are deployed. Virtualization technologies have been used in data centers by different reasons, one of them is energy consumption saving. Different analyses have shown than virtualization technologies allow reducing the energy consumption by more than 30% [136] and depending on the energy optimization techniques, the virtualization technologies, and the application and operating systems executed on virtual machines, this percentage may vary [105]. In conventional computing clusters, the energy consumption is mainly based on the consumption of servers and cooling systems. In desktops grids, the energy consumption is based on the consumptions of the desktops machines used to execute the grid tasks, cooling consumption is not kept into account due to desktops machines are regularly available in large open spaces. Few efforts have been developed to analyze the energy consumption in opportunistic grids. From the energy consumption point of view, most of desktops grids select the resources to execute grid tasks using an algorithm that keep into account only the best physical resource to execute the jobs without having in mind the energy consumption used to execute the tasks. Around the Condor project an effort [108] have been developed to analyze the energy consumption of a Condor cluster, and how to optimize the energy consumption when desktops used to execute the grid tasks are selected. A more detailed and general project, termed DEGISCO, have been proposed in [146]. This DEGISCO proposal wants to look several aspects of energy consumption and computational performance of different desktops grids around the world, however at the moment of this publication; the project is in its initial phase. Ponciano et al. [134] show some strategies in opportunistic grids to save energy, they evaluated three strategies under opportunistic environment, sleeping, wake-up and scheduling strategies, their results surpassing 80% reduction in energy consumption in a scenario when the contention for resources in the grid was low.

The technique of use task consolidation to save energy is not new. Hsu et al. [80] presented one that aims to optimize energy consumption of virtual cluster in cloud data center, their simulation results show that, this technique, can reduce power consumption up to 17% in managing task consolidation for cloud systems.

Chapter 5

UnaCloud Suite

Contents

In this chapter will introduce UnaCloud Suite by UnaCloud IaaS and UnaCloud PaaS from University of los Andes. These IaaS and PaaS were the environments where the resource allocation experiments were developed. This chapter is taken from different of our publications [CCGrid2013], [CCSA2013], and [ICA3PP2013], for a complete description of UnaCloud, please refer to these papers.

5.1 UnaCloud: Opportunistic Cloud Computing Infrastructure as a Service

We consider an opportunistic based cloud infrastructure called UnaCloud [143]. UnaCloud infrastructure is an open source implementation that combines the main features and advantages of cloud computing and those already provided by desktop grid and volunteer computing systems (DGVCSs) [38]. UnaCloud uses a commodity and non-dedicated underlying infrastructure, implementing opportunistic design concepts to provide computational resources such as CPU, RAM and storage, while profiting from the unused capabilities of desktop computer laboratories. The aim of UnaCloud is to support an Infrastructure-as-a-Service (IaaS) cloud computing service model. This effort is aimed at the provision of computing infrastructures for the development of e-Science projects and to support computing related activities.

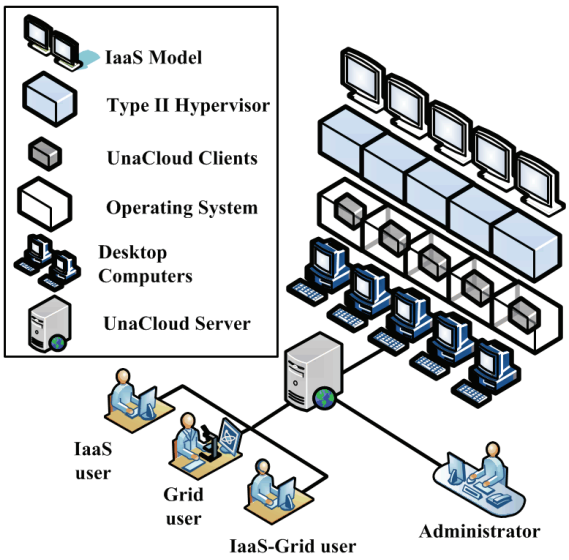


Figure 5.1: UnaCloud deployment architecture.

Although its goal is to use all of the computers on campus, UnaCloud currently has access to three computer labs with 109 desktop computers, whose aggregate capabilities may deliver up to 592 processing cores (70 PMs have four cores each and 39 PMs with eight cores), 572 GB of RAM, 8 TB of storage and 1TB of shared storage in a Network Attached Storage (NAS). Resources are shared by applications from different research groups at the university, through the use and deployment of Customized Virtual Clusters (CVCs). A CVC is a collection of interconnected physical desktops computers, each one executing a single virtual machine (VM) with low priority as a background process. Each VM can take advantage of the unused capabilities while students do their daily activities on the computers. The VM is a template created and personalized by each research group, specifying the operating system, software, applications and middleware they require. Thereafter, UnaCloud is in charge of configuring the complete CVC on a group of specified desktops. Finally, users deploy CVCs on demand through the UnaCloud web portal. Figure 5.1 shows the deployment architecture of UnaCloud, where the UnaCloud Server and the UnaCloud Clients are the main components. The UnaCloud Server is in charge of processing user requests made through the web portal, and it usually communicates with the necessary UnaCloud clients to satisfy the requirements. The UnaCloud client is executed on each desktop and is in charge of initiating, restarting or stopping VMs.

5.1.1 Benchmarking UnaCloud IaaS

This chapter presents the performance evaluation of the opportunistic IaaS UnaCloud using HPL and IOzone. Part of this work have been publishing in [CCGrid2014].

Cloud computing has the possibility to offer on demand, flexible access to appropriate amounts of computation, memory, and storage resources. One of the three service models that composes Cloud Computing is Infrastructure-as-a-Service. The (US) National Institute of Standard Technology (NIST)[1] defines IaaS: *"The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications"* [116], in other words, acquisition and management of physical resources. These resources are provisioned on-demand, that is, when they needed, for the time as they needed, and paying just as they consumed, allowing them expand and contract the available resources. Elasticity, multi-tenancy, reliability, among others are the challenges coming from this service. Nevertheless, the most important characteristic, from user's perspective, of IaaS clouds is good performance, therefore, it is necessary to use one tool that help to get this approach.

Benchmarking is one of the best approaches able to verify that system's performance gets the requirements. Benchmarking computer systems is the process of evaluating their performance and other non-functional characteristics with the purpose of comparing them with other systems or with industry-agreed standards. The nature of a benchmark depends strongly on the intended use of the results.

Benchmarking support process in different situations e.g. use in system design, tuning and operation, use in training, and most common to validate assumptions and models. As Keyvalya Dixit [74, Ch. 9] defines, there are three types of popular benchmarks: kernel, synthetic, and application. *Kernel* is based on the empirical observation of "10% of the code uses 80% of the CPU resources", *Synthetic* is based on instruction mix, and *Application* refers to the user's own application program [74, Ch. 9].

Iosup et al [88] present specific elements to benchmarking the IaaS Cloud. Based on these elements, we associated our work as follows: Our System Under Test *(SUT)* is UnaCloud IaaS [143], the cloud infrastructure built under an opportunistic environment over an university campus. The workload is defined by the characteristic resource to be stressed (e.g. Through CPU-intensive jobs), the job arrival pattern (one of uniform, increasing, and bursty), and the job durations. Our experiments (our benchmarking process) lead to results with high statistical confidence, evolving *complex workloads* that represent multi-tenancy scenarios, on domain-specific scenarios, and on a combination of traditional and cloud-specific metrics.

This information helps to improve the performance of the service in this case of IaaS or tuning their applications to better utilize the performance of existing machines.

In this work, we present a performance analysis of our IaaS opportunistic cloud, to help at users be able to easily compare systems, to us identify best-practices and lower costs, and to show the performance in one specific kind of cloud under opportunistic environment. We analyze in this study the performance and the Input/Output velocity of two widespread

[1] See http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

virtualization frameworks, namely Virtual Box and VMware ESXi, running a single VM instance and compare them over an opportunistic cloud environment.

Following the architecture proposed by [88] we describe the generic architecture for UnaCloud IaaS benchmarking below:

- *Benchmark Description*: We present two types of benchmarking, to analyze the performance and I/O of the system, as it will be explained in Section 5.1.1.2 we cover the following scenarios, 10, 20 and 40 VM size clusters, over two hypervisors in different zone times reflecting the quantity of users of the system.

- *Workload Description*: We used HPL to analyze performance and IOZone to analyze Input/Output.

- *Workload generator and submitter (policy)*: A control system is in charge of made several runs for each scenario. To do so, this control system uses a service to run programs over the overlying infrastructure. The system has the ability to manage custom program execution lifecycle.

- *Allocation (policy)* The allocation process is done using a FCFS over the available resources. The selection process first uses the physical machines that have an end user on it.

- *Provisioning (policy)* The IaaS deploys the requested resources using the allocation process results. This provisioning is done individually by an IaaS agent running over each physical resource.

- *Monitoring & Logging (policy)* All information is recollected on a central database and a shared file system. Over the DB, we store the usage and monitor information of physical resources. Over the shared file system, we store the results of each execution.

- *Results analysis & Modeling* Finally, the results are gathered and processed using tools like Matlab, R and custom java programs.

Alternatively, we will consider the "spin-up" time (to be denoted later) defined as the time taken for an instance to:

- Find and reserve an available server,

- create a new virtual machine instance on it,

- deploy the application code and the data necessary to run the application onto the recently created virtual machine,

- start the application,

5.1.1.1 Benchmarking Literature Review

Brebner et al. [27] show that the use of a benchmark have two main goals, first is to aid in the design of middleware, and second to evaluate the performance of a middleware. In our

work we use for both, evaluate the performance to show to our customers and re-design the middleware if this is necessary.

A benchmark has two major components, as Sawyer proposes in his chapter *"Doing Your Own Benchmark"* [74, Ch. 11], the workload specification and the measurement specification.

Extra metrics about how measure the IaaS Cloud as introducing Iosup et al [88] have been investigated in different works, for instance, variability, that refers to the performance characteristics of a cloud may vary over time as a result, for example, changes that are not discussed with the users [87]. Other feature, that could be measure is the elasticity, the ability to scale resources on demand up and down [28], but, as explained by Islam et al. [90], it is more *"a property"* of a platform using time and cost as its components and elasticity can be measured with respect of its response to different applications with certain QoS.

In grid computing, several works have been presented to evaluate performance, for instance adequate approaches must combine appropriate performance metrics, realistic workloads, and flexible tools for workload generation, submission and analysis as presented in [89]. Two of those previous authors presented a framework for performance evaluation using synthetic workloads for generation and submission, named GrenchMark [85].

High Performance Linpack (HPL) is a software package developed by Dongarra et al. [57] that solves a random dense linear system in double precision arithmetic on distributed-memory computers. HPL is a *kernel benchmark* that analyst performance as it was defined by the Standard Performance Evaluation Corporation (SPEC) in [74, Ch. 9]. Besides, other studies, based on HPL, shown performance fluctuations over cloud-based systems as EC2 and Eucalyptus [58]. Filesystem benchmarks as IOzone ©, generates and measures a variety of file operations than tests Input/Output performance [126]. CCMPerf benchmarking tool is a model based benchmarking that allows developers and end-users of the Corba Component Model (CCM) middleware to evaluate the overhead that CCM implementations impose above and beyond the CORBA implementation, as well as to model interaction scenarios between CCM components [99].

Nowadays, in cloud computing, different performance evaluation have being presented in the literature, Iosup et al. [86] show a very detailed performance analysis about many tasks scientific computing over four very-known cloud service providers. Alternatively, Dong et al. [55] show a performance evaluation of a particular dedicated cloud computing platform in their campus in Southeast University. Our work evaluate the performance of a particular campus cloud platform as well, but it is with *"the service model"*[1] of an opportunistic Infrastructure as a Service.

5.1.1.2 Experiments and Results

In order to make valid overall performance judgments, our evaluations based on quantification are not only focus on providing a comprehensive representation of UnaCloud IaaS, even more, the understanding of the inevitable dependencies between measurements.

We use a small and fixed suite of measurements to make simple and human comprehension feasible comparisons,, we use as metrics: performance (timestamps with system utilization) taken from HPL benchmark and resource consumption data taken from IOzone. We also

[1] See http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf for definition of service models in the cloud.

measure our "spin-up" time in UnaCloud and it is in the rank of 8min. to 12min.

If the synthetic workload captures the right characteristics, the behavior of the synthetic workload and the real application will be similar.

5.1.1.3 Parameter Tuning

For measuring the impact of virtualization over the benchmarks, we use a private and managed infrastructure of 4 physical machines with Core 2 Duo processors and 4GB RAM. Each machine has dual boot to mirror the production infrastructure of UnaCloud. One boot option has Windows 7 as operating system; over this we install VMware Workstation 8 and VBox 4.3. Over each hypervisor we mount an identical (same installation procedure) virtual machine created from scratch. This virtual machine has Debian 7 as SO and all software requirements to run the benchmarks. The second boot option, on physical machines, has an identical configuration of operating system and software as the VMs created on each hypervisor.

Next, the test to measure the impact of PM end users over jobs running over the UnaCloud strategy, we made several tests running benchmarks of different configurations on production environment. For each test we measure the % of the physical machines as the relation of PMs with end user over the cluster size (henceforth, User %). The production environment of UnaCloud is composed by 65 Intel Core i5 and 8GB machines. Each machine boots on Win 7 OS and has Virtual Box 4.3 and VMware Workstation 8 hypervisors. UnaCloud has the capability to deploy cluster on VBox and VMware hypervisors.

The second test uses 1 core and 1 GB per virtual machine. Several tests were made using 10, 20 and 40 VMs per cluster. For each test we store the % of end users over the used PMs. The tests were run over two weeks (all day time) to ensure a good coverage on User %.

5.1.1.4 Experimental methodology

Several tests were made to measure system performance under different scenarios. One set of tests aims to compare the performance and impact of each hypervisor over the Cloud User. A second set of test measures the impact of physical machine users over the virtual machine tasks.

To measure the degradation of performance perceived by UnaCloud users we ran the Iozone and HPL over a test infrastructure composed by 4 physical machines. Over this PMs we install 3 clusters: a cluster over the PMs, another running over VMware Workstation VMs, and finally one running over VirtualBox VMs. Each cluster has Debian 7 as SO and was configured to run the benchmarks (this includes OpenMPI, HPL, IOzone and GotoBlas).

Finally a set of tests were made to determine the impact of real end users over the tasks running on opportunistic VMs. To do this, we set clusters of 10, 20 and 40 VMs that ran HPL and IOzone benchmarks over UnaCloud production environment. Several tests were ran, ensuring that each possible scenario was cover. For these tests we measure the percentage of end users using the PMs while each test is running. To do so, the system reports every minute which PMs has users or not. At the end of the tests, we take the start and end time of each tests and look into the reports, the amount of them with users and those that don't.

The user 's percentage is presented on Eq. 5.1

$$User\% = \frac{totalReportsWithEndUser}{TotalReports} \times 100 \tag{5.1}$$

where *totalReports* represents the amount of reports that were made while the test were running and *totalReportsWithEndUser* represents the amount of reports which doesn't have an end user from *totalReports*.

5.1.1.5 Results

In this section we present the results for each set of tests. The results are presented for processing and storage intensive tasks.

Baseline Performance Comparison:

A test infrastructure was setup to make a baseline comparison of each hypervisor performance. It was made on a test infrastructure, due to the administrative impositions to do it over the UnaCloud production environment. The results are show on Fig. 5.2. The results shows a high impact on perceived performance, showing that for HPL tasks, it is recommendable to use physical machines and not to virtualize.

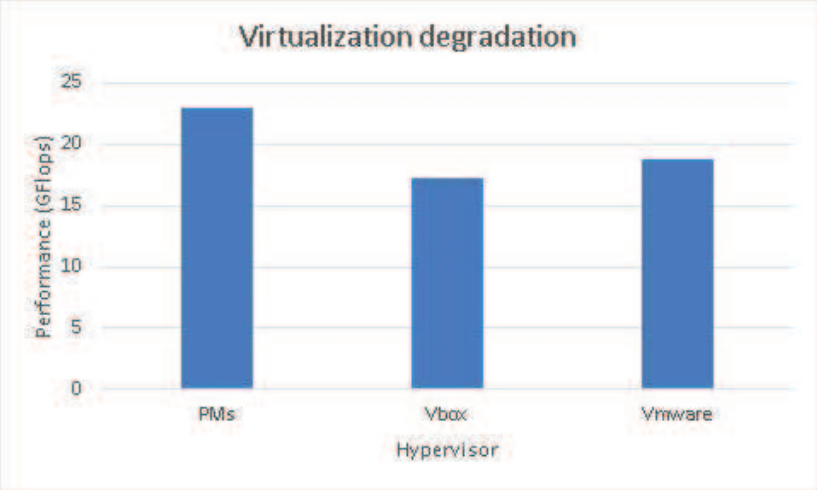


Figure 5.2: Virtualization degradation of HPL benchmark.

Cloud User Impact:

Finally, a set of tests were made to determine the impact of end-user activities on virtual clusters running over UnaCloud. To do so, we ran several tests that measures the platform performance under real scenarios. The executions were made over three weeks, on each execution we measure the amount of end-users using the underlying physical infrastructure.

Results are showing on Fig. 5.3. As can be seen, the impact of Users % (the amount of physical machines with user over the total of virtual machines on the cluster) does not affect the cluster performance. Each point measures the performance for one test, the variability on test with the same User % can be explained taking into account that we do not present averaged values, due to the difficulty on getting representative samples for each user %.

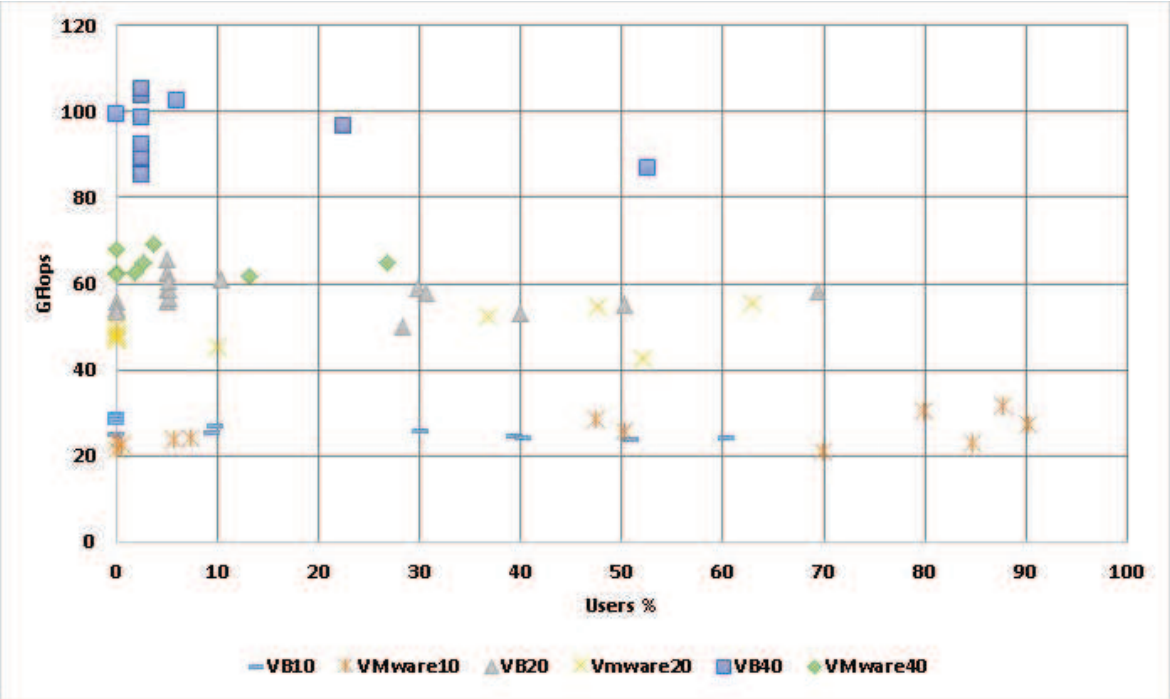
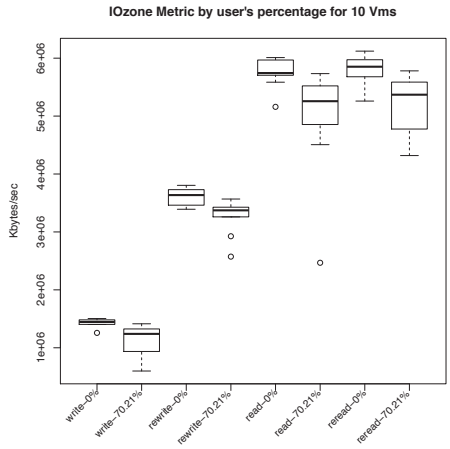


Figure 5.3: End user impact over opportunistic cloud users, HPL Banchmark.

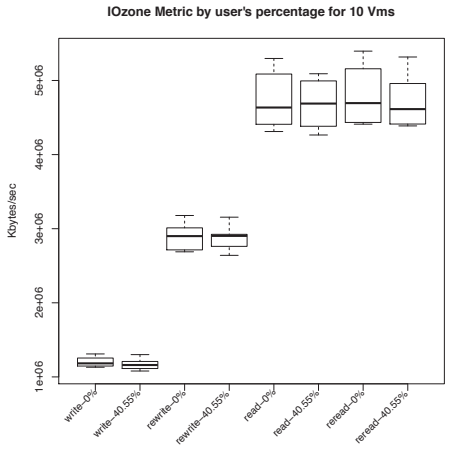
These results also show that, when the cluster size increases, it is more difficult to get a high Users %. This is because the periods of high usage on computer labs are short in comparison with benchmark execution times.

IOzone: One inherent limit to the usage of virtualization in any environment resides in the huge overhead induced on I/O operations. Thus, we present the results of the IOZone benchmark in Figures 5.4 over the opportunistic cloud environment. We chose a typical measure of 64MB *file size* and 1MB *record size* over the experimental spectrum of the IOzone. The Boxplots representation were chosen because its good form to show the significance difference of the data due of User% and each value coming from one machine of the system, these values are stochastic moreover when the space of the sample increase (i.e. 10, 20, and 40 VMs). The different User% reflects the variability of the opportunistic environment. We show in each figures the speed, in kB/sec, to write, rewrite, read, and reread 64MB files with 1MB record size in UnaCloud IaaS.

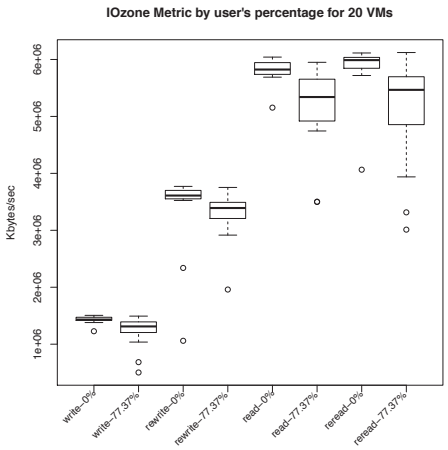
Fig. 5.4(a) shows how is the behavior of the opportunistic environment taken as a basis comparison the User% in 0, we can observe that the difference with User% in 70.21 for VMware is not too significative, it is reflect the good performance of Input/Output measures in an opportunistic environment. Similar behavior show Figs. 5.4(c) and (e) for 20 and 40 VMs respectively and User% in 77.37 and 76.71 respectively. Fig. 5.4(b) shows the same comparison of Fig. 5.4(a) but for other hypervisor, Virtual Box. Taken as a basis comparison the User% in 0 as before, we can observe a similar response with User% in 40.55, this shows a better performance of Virtual Box in opportunistic environment than VMware. Figs. 5.4(d) and (f) show, as a basis comparison, User% in 0 against User% in 55.16, and User% in 2.46 against User% in 51.34 respectively, for 20 and 40 Virtual Box VMs respectively. Those



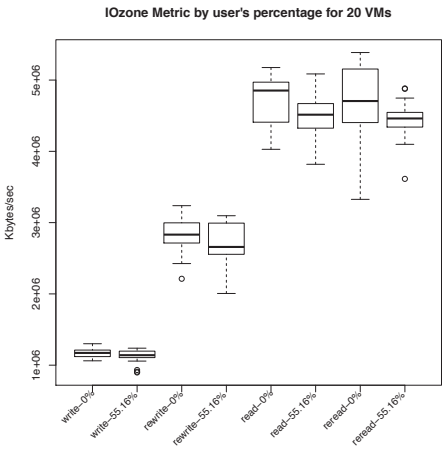
(a) IOzone results for 10 VMs with VMware.



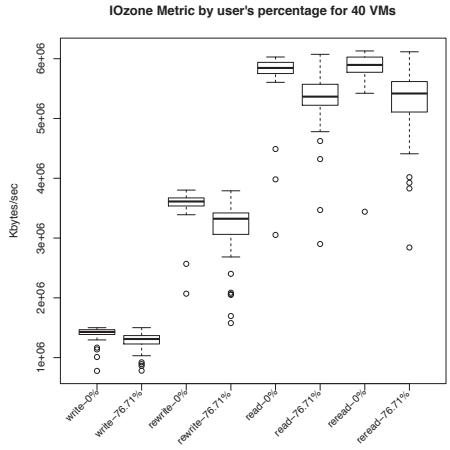
(b) IOzone results for 10 VMs with Virtual Box.



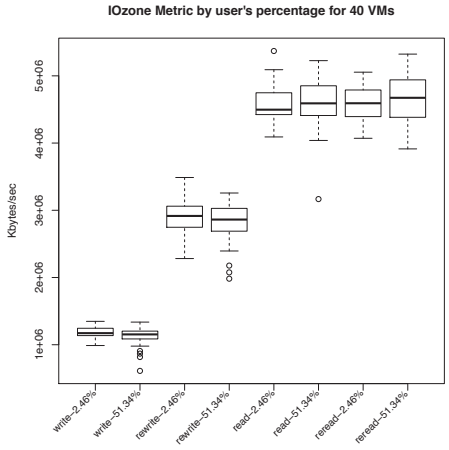
(c) IOzone results for 20 VMs with VMware.



(d) IOzone results for 20 VMs with Virtual Box.



(e) IOzone results for 40 VMs with VMware.



(f) IOzone results for 40 VMs with Virtual Box.

Figure 5.4: IOzone results for each configuration. 10, 20, and 40 VMs, two User%, and two hypervisors. file size=64MB, record size=1MB.

figures show the same aforementioned behavior. Furthermore, an interesting result we can observe from Fig. 5.4(f), the performance on IOzone follows the same behavior when user% is different to 0 against bigger user% value, this means that the I/O velocity have not significant change when is an user or when are 20 users, showing its robustness. Nevertheless, if we are evaluating the overall performance of Input/Output characteristics between hypervisors, VMware has better performance than Virtual Box, its values outperform 10 times approx. (e.g. 6e06 KB/sec for VMware while for Virtual Box is 5e06).

5.1.1.6 Summary

Tests show an important difference between Virtual Box and VMware hypervisors. We summarize the results from two points of view:

- CPU intensive view: Overall performance of HPL benchmark shows an average of 10% of increase when using VBox over VMware clusters. In general, the results are not affected by the amount of end users on the underlying infrastructure for both hypervisors.
- In other hand, IOzone benchmark results show a relevant difference on I/O performance between VBox and VMware. Although the performance impact of users on VBox is lower over VMware results, we can see that VMware performance is better for all tests. So we can conclude that VMware hypervisor has a better I/O performance according to IOzone benchmark.
- Aditionally, we can observe also that VBox is more robust than VMware to manage User% in the system for IOzone benchmark.

According to the results, we recommend using VBox for CPU intensive application. In contrast the best choice to run I/O intensive task are VMware Workstation hypervisor. These results are based on an opportunistic use of the available resources, applying a resource sharing policy between end users and cloud users.

5.2 Building Platform as a Service for High Performance Computing over Opportunistic Cloud Computing

Opportunistic computing is a sustainable alternative to satisfy the growing demand of computing resources. In this section, we introduce a novel cloud platform for developing high performance computing applications over an opportunistic environment. We consider a cloud-based opportunistic infrastructure called UnaCloud [143] over this, the new platform as a service has being developed (UnaCloud PaaS). UnaCloud PaaS is a cloud computing platform oriented to use of opportunistic IaaS to deploy high performance applications. UnaCloud PaaS offers multipurpose platforms for low IT knowledge HPC users, that wants to use an opportunistic infrastructures to deploy and run specific applications. It is created to facilitate the complexity of opportunistic desktop based infrastructures to run applications.

Taking advantage of unused resources opportunistically, we present the main characteristics of UnaCloud PaaS as well as each of its components defined. UnaCloud PaaS can be

deploy two platform types: Condor and MPI. Each platform is specified by a set of roles. A role is a set of homogenous machines regarding software, operating system and configuration.

To show the performance of the platforms, we conduct several experiments, measuring system response time and execution time of running platforms as well as one sample application execution called Gromacs. Furthermore a set of test, using a well know benchmark, were made in one of the platforms. Our experimental settings reflect the need of an opportunistic aware PaaS for the execution of successful platforms over opportunistic infrastructures.

5.2.1 Related Work

On the field of PaaS implementations for HPC we can find many solutions. Manjrasoft©presents Aneka [160], a solution to develop .NET applications using multiple programming models, and run them over hybrid infrastructures. Microsoft Windows Azure [119] offers an entire set of services on a platform for application development and deployment over Microsoft datacenters. It provides APIs, libraries and services for solving specific application problems like storage, cache, application synchronization, scalability and resource acquisition. MagosCloud [45] offers an opportunistic PaaS for web 2.0 services. It is focused on developers and offers a declarative way to express platforms and requirements over a XML schema. Amazon offers special HPC cloud VMs [9] with high performance hardware instances in their data centers. Amazon also [84] offers an Elastic Map Reduce service to execute Map Reduce workflows over a managed and scalable infrastructure. The FEFF Project [141] makes an offer to deploy and manage platforms for spectroscopy analysis and material science software. These clusters are deployed over Amazon EC2.

Sabalcore [144] is a commercial solution for HPC that allows to deploy custom virtual clusters over their specialized infrastructures. It offers solutions stacks and software for engineering modeling, financial simulations, energy computations and more. ViteraaS [52] propose a PaaS model for running HPC programs. ViteraaS is a virtual cluster management tool, which allows users to deploy on-demand virtual clusters of various sizes. The objective of ViteraaS is to simplify the creation/deletion of virtual HPC clusters and job submission.

Unlike the commercial and academic PaaS models implementations, UnaCloud PaaS is specially designed to use opportunistic infrastructures to deploy managed platforms for scientific computations. UnaCloud PaaS makes use of UnaCloud IaaS opportunistic infrastructure to deploy customized virtual clusters (CVC) over it. Once this CVCs are deployed, UnaCloud PaaS configures them to execute and manage user applications.

5.2.2 UnaCloud Platform Architecture for HPC

UnaCloud PaaS is a Platform-as-a-Service implementation that provides managed and scalable platforms to deploy HPC applications over opportunistic infrastructures. It uses UnaCloud IaaS services to deploy and manage virtual clusters over the available infrastructure. Once these clusters are deployed, UnaCloud PaaS configure and installs all software and configuration requirements to build platforms for user program executions. Each execution is accomplished, managed and monitored. Each platform execution runs on a virtualized environment that is completely isolated from other executions.

A platform, in UnaCloud PaaS, is defined as a set of machines that are configured at hardware, operating system, network and software levels, and they are offered as a scalable

and managed service for application executions. UnaCloud PaaS uses the concept of *role*. Each role is defined by its main module (which identifies the platform type), its size and a set of software modules. A software module is a collection of configurations and programs that are applied and installed on a virtual machine to satisfy a requirement. Each module have a set of input parameters, whose values are established by the system and users. For example, currently at UnaCloud PaaS there are two main modules: Condor and OpenMPI. Those modules are used by two platform types that can be deployed: Condor (BoT) and MPI. Also, there is a set of software modules that can be applied to a platform role before its execution. For example an user can choose to add Gromacs [23] to a MPI platform. Across these modules, an user can add software dependencies required by its application or program. An user can also add files to a platform execution. A file can be chosen from the local machine where the user is consuming the PaaS services.

Finally a platform execution has a list of commands that are executed on specific environments. The content and environment of a command is defined by the user. The environment refers to the shared folder where the command is executed and the multiplicity of the command.

5.2.2.1 UnaCloud PaaS Cloud Features

The characteristics of UnaCloud PaaS are summarized below:

- *Fault tolerance:* One of the most important feature needed to successful deploy platforms and applications over opportunistic infrastructures is the fault tolerance. UnaCloud PaaS provides a component that led the failures of the platform called *Failure manager* and it is described later.

- *Efficiency:* Through the opportunistic environment, UnaCloud PaaS provides a framework for developing distributed application taking advantage of the share utilization of the machine by its service oriented architecture.

- *Usability:* UnaCloud PaaS provides Web interfaces, whose operation is almost intuitive for a basic IT knowledge. Additionally it provides an API to access UnaCloud PaaS services from other applications.

- *self-service:* The design of UnaCloud PaaS permits to users consumes unilaterally platform resources by a self-service model.

- *Broad Network Access:* UnaCloud PaaS provides platform executions services that are available over internet and are consumed through standard secure remote access mechanisms like https and ssh.

- *On-demand services customization:* UnaCloud PaaS provides ways to customize execution environments required on demand by end-users using the *API Client* component defined in the following sections. This customization is able to meet large scale computational requirements.

- *Scalability:* UnaCloud PaaS uses an opportunistic commodity horizontal scaling infrastructure service that is based on a private cloud deployment model.

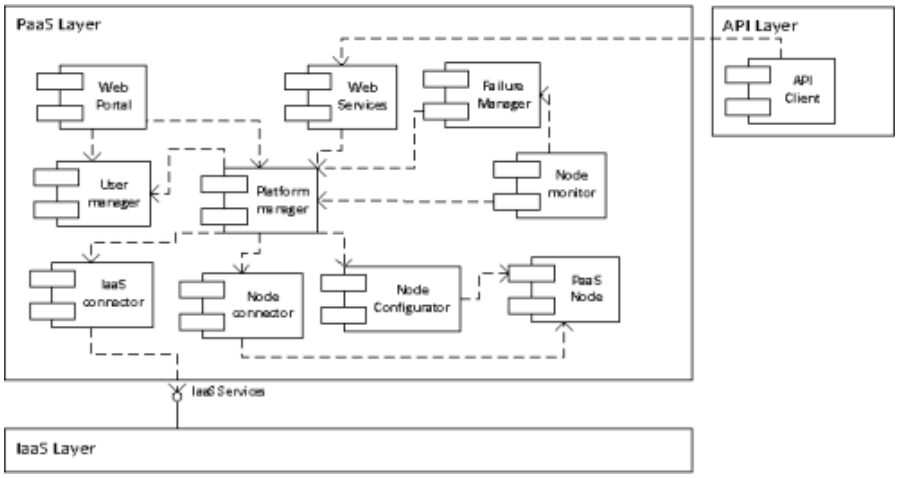


Figure 5.5: UnaCloud PaaS component diagram.

- *Interoperability:* UnaCloud PaaS is based in loose coupling and interoperability services operating over highly heterogeneous, distributed and non-dedicated infrastructures.

- *Extensiblity:* Based on open source tools, UnaCloud PaaS is broadly diffused in order to facilitate its extensibility.

- *Security:* UnaCloud uses authentication, authorization, confidentiality and non-repudiation mechanisms to secure the PaaS model deployment. Also, each platform runs on an isolated virtualized environment.

- *Measured service:* UnaCloud PaaS records and reports, by logs, all events regarding platform executions. It also takes traceability of used resources and the operations over them.

5.2.2.2 UnaCloud PaaS Components

Figure 5.5 shows the component structure of UnaCloud PaaS. It is divided in three major layers. IaaS layer, which provides infrastructure services. Currently the IaaS layer is provided by UnaCloud IaaS. API layer, that provides a specification to implement a web service client and it is connected to one of the two interfaces of PaaS layer. And finally, a PaaS layer, it is the main module of UnaCloud PaaS. It provides services to deploy and manage platforms from two interfaces: a web user interface and web services. The web services are consumed by an API layer as above mentioned. This provides abstract access to UnaCloud PaaS features due of its service oriented architecture.

Each UnaCloud PaaS component is described below.

+ **Web Portal:** is the main way of access to UnaCloud PaaS services. It provides a set of pages and web forms to manage all system entities. It also provides a set of forms to deploy and manage platform executions.

+ **Web Services:** is a component that exposes the platform execution services and operations through Web Services. This component only exposes services to start, manage and stop platform executions. System administration should be done through web portal.

+ **API Client:** it is a component specification that offers an abstraction to web services. It facilitates the complexity of the use of web services, so the user can consume UnaCloud PaaS operations in terms of objects and methods, and not by complex web services.

+ **User manager:** It is in charge of user account management. It includes passwords, user permissions and user traceability. This component is used by other components to check user permissions and limits before any resource or security related action.

+ **Node connector:** It allows the server to connect to the PaaS nodes. It uses standard mechanisms like SSH as tunnel to execute remote commands on each node. The main purpose of this component is to execute remote commands on deployed clusters.

+ **Platform manager:** This is the main component of the system. It is in charge of coordinate and orchestrate all other component to deploy cloud platforms. It has the logic to deploy, manage and control the platform executions. It is also in charge of storing a historic log of all deployed platforms.

+ **Node monitor:** This component is in charge of monitoring all node instances of all running platforms to determine if there is a node with a hardware failure. If so, it reports it to the Failure manager. The monitoring process involves taking running commands SO ids and check for process health. When a running VM cannot be accessed, it is marked as failed and sent to Failure Manager

+ **Failure manager:** It is the component which have the algorithms and business logic to recover a platform execution after a failure on one of its components. It uses the platform manager to orchestrate this process. The recovery process depends on deployed platform. It include checkpointing and platform restart thecniques.

+ **Node Configurator:** This component configures and manage configuration settings for the platform nodes. It implements an interface between the external configuration manager and UnaCloud PaaS. A node configuration is specified by a set of modules and parameters that are used to install and configure software and tools.

+ **IaaS connector:** This component connects to the underlying infrastructure provider to deploy and manage virtual machines. This component get the information of VM deployment retrieved after a cluster start operation on the underlying IaaS system and transform it into a UnaCloud PaaS managed object.

+ **PaaS node:** This last component is mounted on every virtual machine used by UnaCloud PaaS. It contains some basic configurations to be compatible with Node connector and Configurator components. It is composed by an SSH server and a configurator manager client.

5.2.3 Implementation

The implementation takes the design and architectural decisions of UnaCloud to provide the following services to end users:

1. *Platform deployments:* Two platforms are offered to end users: Condor and OpenMPI. These platforms can be consumed by a web interface that allows the customization of each platform. Some software modules are offered to be added to the platforms: Gromacs, Blast, High Performance Linpack and Java.

2. *Platform execution monitoring:* Platform executions are monitored, so it is restored in failure cases. Also, the program execution is checked so, at successful termination the user is notified about the results.

3. *Platform execution management:* The user can stop and pause running executions.

4. *User folder management:* The user can manage its user folder to add, move and delete files that can be used on platform deployments.

5. *PaaS management:* Finally, the implementation offers a way to manage all configurations, entities and services of UnaCloud PaaS. It can add and delete software modules, platforms, files, users and more.

5.2.3.1 Parameter Tunning

To achieve the implementation, it was used UnaCloud IaaS infrastructure. UnaCloud IaaS has been deployed in three computer laboratories at Universidad de los Andes. Each laboratory has 35 computers with Intel Core i5 (3.01GHz) processors, 8GB of RAM and Windows 7 as their main operating system. In addition, UnaCloud Servers (PaaS and IaaS) were deployed on virtual machines running on a server, which is located in the data center (for availability reasons).

A set of tests was made to measure system response time and execution time of running platforms varying the number of virtual machines for each platform software. For each configuration, we ran an executable with an exact duration of 60 seconds.

UnaCloud PaaS was tested using an MPI application with production dataset inputs provided by the Department of Chemical Engineering at University of Los Andes. The application executes a Molecular Dynamics simulation using the GROMACS [23] package of the transmembrane domain from the Outer membrane protein A (OmpA) of *Escherichia coli*. The purpose of the simulation is to calculate the annihilation free energy of the transmembrane domain by coupling the intramolecular interactions of the protein in vacuo. The same simulation is executed varying the number of virtual machines and the number of cores per VM.

5.2.4 Testing and Results

A set of tests was made to evaluate the performance of the proposed PaaS solution according to the objectives of the present work.

Platform	Modules	VMs	VM Start time (s)	Config. time (s)	Run time (s)
MPI	-	1	60	90	83
MPI	-	2	60	100	104
MPI	-	4	61	111	81
MPI	-	8	61	137	99
MPI	-	16	60	208	99
MPI	-	32	60	271	99
MPI	Gromacs	1	55	330	89
MPI	Gromacs	2	60	325	100
MPI	Gromacs	4	60	352	85
MPI	Gromacs	8	80	414	110
MPI	Gromacs	16	70	463	90
MPI	Gromacs	32	61	518	105
Condor	-	1	90	111	136
Condor	-	2	110	137	116
Condor	-	4	120	156	151
Condor	-	8	121	163	151
Condor	-	16	120	207	131
Condor	-	32	121	298	144

Table 5.1: System Response Times

5.2.4.1 System response and run times

As aforementioned, several platform executions were launched varying the number of virtual machines of the main role and the platform software modules.

In Table 5.1 we can see the virtual machines start time, configuration time and run time, in seconds, of different platforms varying its total size. We can see that virtual machines starts in the same time, independently of the size. However, condor platforms take about the double to start its VMs. It is because this platform has two roles (master/slave), in contrast to MPI platforms that have one (exec). On MPI cases it has a mean error of +30 seconds, it was due the fact that each 60 seconds the platforms are inquired to determine if the executables have finished. On condor cases the error is about +60 seconds because there is a time expended on queue management.

In Figure 5.6 we present the configuration time for each platform setup as the number of VMs is increased. A linear regression and the Pearson product-moment correlation coefficient (R2) is shown for each setup. We can conclude that configuration time is linear dependent with the size of the platform.

Cores	VMs	Cores/VM	Gflops	ns/day	T(h)
1	1	1	4.84	0.89	8.06
2	2	1	5.39	1.00	7.30
5	5	1	8.52	1.58	4.59
10	10	1	7.97	1.47	4.88
15	15	1	11.12	2.06	3.61
20	20	1	11.18	2.07	3.65
2	1	2	9.02	1.67	4.31
4	2	2	7.41	1.37	5.27
10	5	2	11.84	2.19	3.34
20	10	2	10.9	2.2	3.59
30	15	2	9.93	1.84	3.93
40	20	2	10.02	1.85	3.99
4	1	4	11.19	2.07	3.48
8	2	4	7.58	1.40	5.26
20	5	4	770	1.43	5.07
40	10	4	5.27	0.98	8.00
60	15	4	5.07	0.94	9.15
80	20	4	5.79	1.07	17.15

Table 5.2: Gromacs Simulation Results

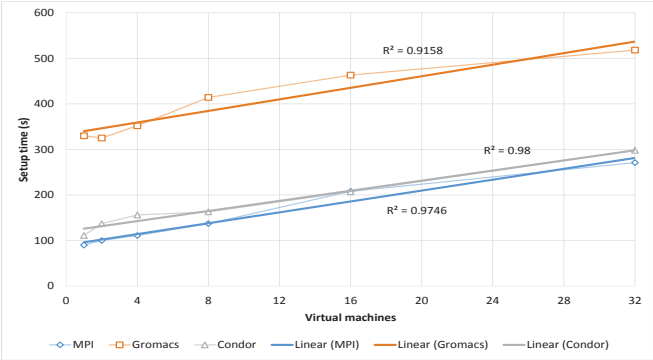


Figure 5.6: Configuration time of different platforms varying the amount of VMs

5.2.4.2 Sample application execution

Several executions were made varying the amount of VMs and cores per VM. It was measured the execution time (T) in hours, the amount of nanoseconds of simulation per day and the Gflops obtained from each test. Every test was executed 3 times, the mean values are presented on Table 5.2. In total, the tests takes 12 days of human execution time and more than 374 days of machine time. Without a system to manage those platform executions it could be impossible to run all these tests. Thank to our failure recovery algorithms and strategies, these test could be executed on a reasonable time.

5.2.4.3 Benchmarking

Finally a set of tests were made to measure the performance of one of the platforms, MPI platform. We use the High-Performance Linpack Benchmark implementation provided by the Innovative Computing Laboratory of the University of Tennessee. We use OpenMPI to run it in parallel. As implementation of the Basic Linear Algebra Subprograms is was used GotoBLAS2 [39] provided by the Texas Advanced Computing Center.

Several tests were made varying the number of VMs and the amount of cores per VM. Figure 5.7 shows the result. As we can see, there is not a despicable potential that can be farmed from UnaCloud opportunistic infrastructure.

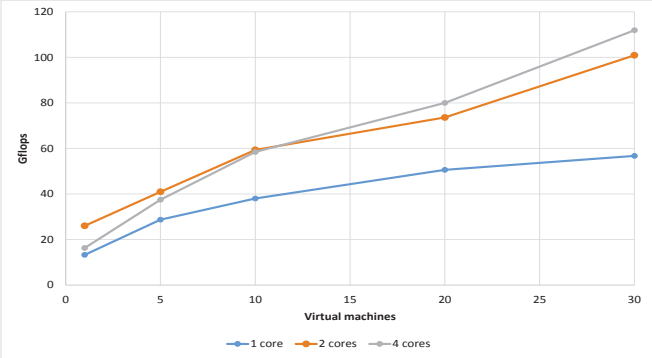


Figure 5.7: Cluster Gflops varying the number of VMs and the cores per VM

Chapter 6

Resource allocation algorithms in Opportunistic Cloud Computing

Contents

6.1 Energy-aware VM Allocation on An Opportunistic Cloud Infrastructure

With the increasing demand of computing resources needed by large-scale applications on science, research and industry, the energy consumption of large-scale distributed systems has increased dramatically. Energy consumption is one of the biggest issues in most of these systems raising monetary, and system performance concerns. Cloud computing, based on virtualization technologies, appears as a new alternative to supply on demand the resources required by the applications. Virtualization is a mechanism to abstract the hardware and system resources of physical servers. Virtualization technology is a solution to provide a complete isolation to consolidate servers onto a larger virtualized system that uses their resources to the fullest, reducing costs, and energy footprint. It is one of the most effective ways to reduce capital expenditures. Considering the salient features of virtualization and regarding that cloud computing mostly rely on this technology, cloud is an inherently energy-efficient computing model [148, 101].

Cloud computing has emerged as one of the most important transformational trends in research and business for enabling on-demand services that include applications, computing, storage, networking from a common pool of different resources. Cloud computing makes these resources easier to use, and typically cloud delivers the services over the internet. It

is composed of virtualized resources that are allocated to represent a single unified image of computing resources. Some benefits of the cloud include economies of scale, cost reductions on technology infrastructure and capital costs, flexibility, elasticity, self-service and metering, management services, increased collaboration services, energy efficiency among others.

There are different kind of cloud deployment models: public, private, community, hybrid. Public clouds refer to a set of services, interfaces and resources owned and operated by a third party for use by other companies. The resources are dynamically provisioned from an off-site third-party provider, and are based on a pay-as-you-go model. A private cloud is a virtualized group of servers operated solely for a single organization. Private clouds can be more cost effective for companies that already own a lot of computing resources. Some benefits of private cloud services are data security, reliability concerns and corporate governance. A community cloud is a model where the cloud infrastructure is shared by several organizations and supports a specific community with joint concerns. The hybrid cloud model refers to the combination of a private cloud with strategic use of public cloud services.

The most common models for delivering cloud services are: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). The IaaS model delivers computing services including hardware, networking, storage, and memory. PaaS is a combination of infrastructure, middleware and development tools that can be used to develop, deploy, manage and run a software solution on a cloud platform. In the SaaS model, users access on-demand applications software and databases.

We consider a private cloud that implements an IaaS model on the computing resources of an university. The IaaS model is oriented to the to the provision of computing resources for the development of scientific projects and to support related activities (i.e., to execute applications such as BLAST, Hmmer or Gromacs). It uses a commodity underlying infrastructure implementing opportunistic design concepts to provide computational resources such as CPU, RAM and Storage, while profiting from the unused capabilities of desktop computer laboratories in a non-intrusive manner offering some of the most important advantages of cloud computing, as for example up-front investment elimination and the appearance of infinite resources available on demand. The private cloud, called UnaCloud [143], is able to deploy, manage and deliver opportunistic IaaS model based on preexisting, non-dedicated and (almost) homogeneous computing resources of computer laboratories. UnaCloud differs from other existing private cloud implementations since it is deployed on the shared computing resources with physical users (e.g., students in the university), while most of the private clouds are deployed on dedicated computing resources. UnaCloud also differs from volunteer, opportunistic grid systems, and community clouds in which private computer owners donate a portion of their idle computing resources to be used by anyone inside a community for supporting a specific project. Due to the large amount of available computing resources on a campus university, the investigated private cloud represents an economically attractive solution for deploying a cloud model avoiding not only the underutilization of non-dedicated computing resources, but also financial investments in hardware and maintenance costs associated.

Although Unacloud may aggregate desktops from independent individuals it is meant to work with machines within a computer laboratory, where each laboratory is managed by a single administrator. It gives UnaCloud several advantages in terms of homogeneity, control and risk of failures. In our tests, a computer laboratory with 35 machines presents a maximum

of two failures a day. Besides, UnaCloud has been used to run Bag of Tasks applications, with many short-lived jobs, making unnecessary to deal with checkpoints and fault tolerance issues [37].

Regardless of the fact that UnaCloud offers the performance capability of deploying Virtual Machines (VMs) in a sustainable way by using idle resources opportunistically, it lacks of energy saving consideration during the placement of the VMs. UnaCloud implements a random placement of VMs to the idle resources of the Physical Machines (PMs) leading to suboptimal resources allocation reducing the gain on energy consumption. Therefore, we aim to investigate energy savings on UnaCloud that can serve as a basis to related opportunistic private cloud models.

Recent trends in the development of new hardware, such as low power energy efficient CPUs, multicore machines, low computer monitors, have mitigated energy consumption to a certain degree on cloud infrastructures. Software based approaches are also a solution to optimize energy consumption. Techniques such as Dynamic Voltage and Frequency Scaling and Dynamic Power Management have been extensively studied and deployed to make the Cloud infrastructure components power efficient [163, 24]. The consolidation of VMs to reduce the number of underutilized or over-loading computing resources, and shutting down the unused resources or to transition parts into a lower power state is another efficient energy saving strategy. This technique is enabled by virtualization technologies that facilitate the running of several VMs on a single resource concurrently. Several approaches to consolidate VMs have been proposed, however most of these algorithms target dedicated resources, however, in our work we consider shared resources with physical users.

We consider, in this chapter, the consolidation of VMs through efficient VM allocation. That is, given a set of VMs and a set of PMs resources the aim is to find the best mapping of VMs to a minimum set of computing resources already in use according to constraints on VMs and resources provided by the PMs to optimize the Energy Consumption Rate (ECR) of the considered private cloud. We investigate four consolidation heuristics. Three of the heuristics rely on classical bin-packing algorithms, and one is an ad hoc (opportunistic) consolidation algorithm proposed in [50], that place VMs first on PMs already in use by a physical user. We empirically evaluate these heuristics on real workload traces. We use cloud workloads based on real production traces collected from the archives of UnaCloud. The traces have been collected during one year. Real UnaCloud scenarios provide a realistic VMs stream for performance evaluation based on simulations of VMs consolidation algorithms. We present results obtained by simulations, while the ultimate goal is to implement the best strategy on UnaCloud. The main reason is that UnaCloud is a production cloud infrastructure already in use making it difficult the implementation for testing the algorithms. However, based on the generated results a work in progress is considering the implementation on UnaCloud.

6.1.1 Energy-efficiency in an Opportunistic Cloud Environment

6.1.1.1 Parameter tunning

UnaCloud aggregates computing resources of desktop-based computer laboratories. Those desktop computers can be modeled as a Set of Physical Machines (SPMs) each one with some hardware specifications including CPU cores, RAM memory, hard disk and networking. Researchers require the execution of a Set of Virtual Machines (SVMs) each one requiring a

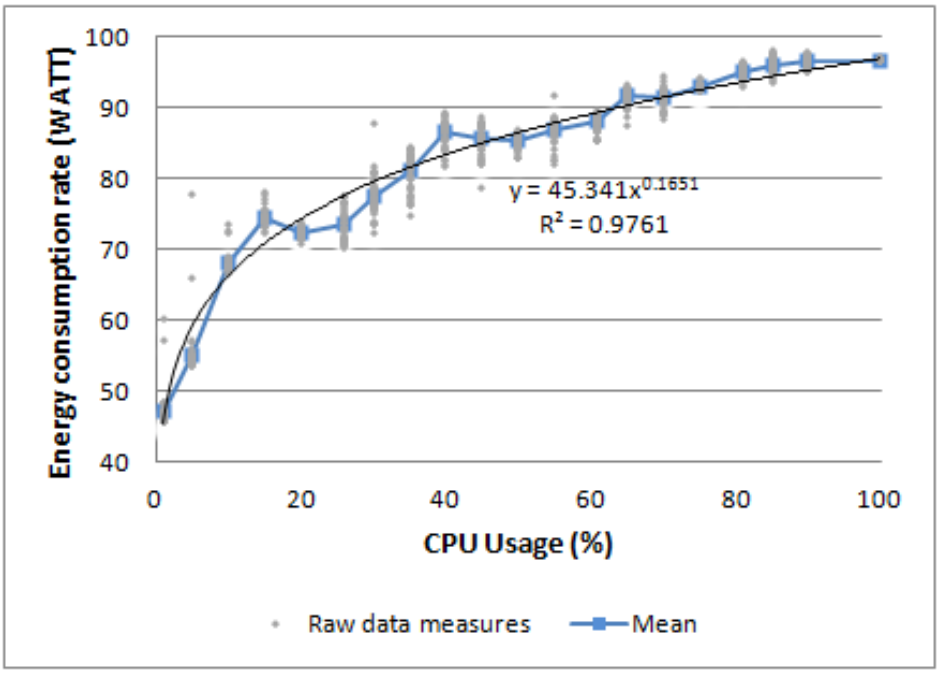


Figure 6.1: Desktop Computer energy consumption. $f(x) = y(x) = 45.341x^{0.1651}$

minimum hardware specification. Due to opportunistic environment, we assume that there are two states (idle and busy) of a PM to be selected to deploy a VM, and according to its state the ECR and the estimated execution time of a VM executing a CPU-intensive task can change.

To analyze the relation between the CPU usage and the ECR consumed by a PM, experimental tests of a previous and recent work [37] show that the CPU usage and ECR are not directly proportional. Figure 6.1 shows the function $f(x)$ (based on a regression calculated on experimental tests) that allows to estimate the ECR of a PM according to its CPU usage. Here we present CPU usage as the percentage provided by the operating system which is in aggregation of the utilization across all cores of the CPU.

Table 6.1 provides the ECRs required to execute a VM according to the state of the PM. We assume that a VM will execute a CPU-intensive task during a τ time. While the VM is in execution the ECR of the PM will increase to $f(x)$, where $f(x)$ is a real function that returns the ECR of a PM given its CPU usage percentage. The idle state represents a turned on PM without any user (a student, administrative, etc.) using it while the VM is in execution. The busy state represents a turned on PM with an user using it and a VM in execution.

When the physical machine is in idle state, the ECR consumed by the VMs is equal to the difference between $f(x)$ and $f(0)$, where $f(0)$ is the ECR consumed by the PM while is in idle state and $f(x)$ is the ECR when a virtual machine is in execution (by number of cores required from VMs determining the CPU usage). In the busy state, the ECR consumed by the VMs is equal to the difference among $f(x) + ECR_{MON}$, and ECR_{user}, where ECR_{MON} is the ECR of the monitor when there is a user using the PM, and ECR_{user} is the mean ECR of a physical machine when there is a user using it and there is not a VM in execution. In busy state the execution time of the VM takes more time because there is an user consuming

Computer state	Execution with VM	ECR for an intensive CPU Task (ECR with VM - without VM)
1 (idle)	τ	$f(x) - f(0)$
2 (busy)	$\frac{100}{L_{free}} \times \tau$	$f(x) + ECR_{MON} - ECR_{user}$

Table 6.1: ECR used by a VM executing a CPU-intensive task.

Computer state	Execution time	Mean ECR (W)
1 (idle)	τ	$f(x) - 47$
2 (busy)	$0.1 \times \tau_{user} + \tau$	$f(x) + 20 - 87$

Table 6.2: Experimental results of energy consumption

computational resources (and competing by the resources required by the VM), therefore the execution time of the VM can be calculated as $100/L_{free} \times \tau$, where L_{free} is the percentage of CPU dedicated to the virtual machine and τ is the estimated running time (provided on demand when a VM is requested) of the VM. We assume that the execution time of an opportunistic CPU intensive task is linearly proportional to the amount of processor used in the PM that is running it. To estimate the percentage of CPU used by users of the PMs, during daylight working hours we executed different tests that show that the CPU utilization does not exceed 10% on average [38]. That is $L_{free} = 90\%$.

To estimate the ECR on different states, the results of Table 6.1 can be completed with the function $f(x)$ depicted in Figure 6.1. Additional parameters such as ECR_{MON} and ECR_{USER} were calculated using specific tests. In tests using a commodity desktop computer, the ECR_{MON} was equal to 20W and the ECR_{USER} was equal to 87W [38].

τ is the sum of τ_{user} and τ_{free} if τ_{user} is less than τ. τ_{user} is given by the time of the PM with an user when it is executing VMs and τ_{free} is the rest of the time to finish the VM, Eq 6.1 shows the relation of the execution time of the VM when there is an user in the PM.

$$\begin{aligned} \tau &= \tfrac{100}{L_{free}}\tau_{user} + \tau_{free} = \tfrac{10}{9}\tau_{user} + \tau_{free} \\ 1.1 \times \tau_{user} + \tau - \tau_{user} &= 0.1 \times \tau_{user} + \tau \end{aligned} \tag{6.1}$$

Table 6.2 shows that from the energy consumption point of view, the best desktop PMs to deploy a VM are those in a busy state.

6.1.1.2 Energy Model

In this section we show a mathematical description to calculate the ECR required for executing a SVM on an opportunistic cloud infrastructure.

Once the consolidation process has finished, the program calculates the power consumption. $P_i^{core}(t)$ is the power of a core i at time t that belongs to a PM. Eq 6.2 shows how is

defined this power.

$$P_i^{core}(t) = s_i(t) \cdot \left((1 - y_i(t))\, P_i^{idleC} + y_i(t) P_i^{workC} \right) \tag{6.2}$$

where P_i^{idleC} and P_i^{workC} are power consumed in idle and work state of the core. $s_i(t)$ denotes if the core is on or not at time t as $s_i(t) = 1$ and $s_i(t) = 0$ respectively. $y_i(t)$ denotes if the core is working or not at time t. When a core is on, without working, consumes P_i^{idleC} but if the core is on and working, it consume P_i^{workC}. The model assumes that power consumption of all system components is essentially constant regardless of the machine activity.

The power consumption of a PM is denoted by $P_j^{mach}(t)$ as Eq. 6.3 shows.

$$P_j^{mach}(t) = z_j(t) \cdot \left((1 - w_j(t))\, P_j^{idleM} + w_j(t) P_j^{workM}(t) \right) \tag{6.3}$$

where P_j^{idleM} and P_j^{workM} are power consumed in idle and busy states of the PM. $z_j(t)$ denotes different states of the PM as Eq. 6.4 shows. The value of $z_j(t) = 1.1$ refers to the extra power consumption because of competing by the resources required as aforementioned. $w_i(t)$ denotes if the core is working or not at time t. Hence, power consumed by machine have direct relation to power consumed by core.

$$z_j(t) = \begin{cases} 1 & if\ mach\ is\ on \\ 1.1 & if\ mach\ is\ on\ and\ user \\ 0 & otherwise \end{cases} \tag{6.4}$$

We assume that P_j^{idleM} is the sum of P_i^{idleC} of all cores belong to machine j and is the same when machine is in idle state. P_j^{workM} is calculated from the equation resulted from Figure 1 as Eq. 6.5 shows.

$$P_j^{workM}(t) = V_{j_{ini}} x_j(t)^{\left(\log_{10} \frac{V_{j_{max}}}{V_{j_{ini}}}\right)/2} \tag{6.5}$$

where $x_j(t)$ is taken from Figure 1 as Eq. 6.6 shows. We have specific homogeneous machines with a behavior as it shows in Figure 1 and related with Eq.6.5, we can deduce $V_{j_{ini}} = 45.341$ and if $V_{j_{max}} = 97$ we have $P_j^{workM}(t) = 45.341 x_j(t)^{0.1651}$.

$$x_j(t) = \frac{T\_u\_c}{T\_c} \times 100 \tag{6.6}$$

Where T_u_c refers to Total used cores and T_c refers to Total cores. As we assumed for P_j^{idleM}, we assume the power when a PM j is at full charge, it is when all the cores are working, therefore we can deduce:

$$\begin{gathered} P_j^{idleM} = \textstyle\sum_i^{T\_c} P_i^{idleC} \\ P_j^{mach}(t_{max}) = P_j^{workM}(t_{max}) = P_i^{workC} \times T\_c \\ P_j^{workM}(t) = P_i^{core} \cdot T\_u\_c \end{gathered} \tag{6.7}$$

Now, from equations 6.5 and 6.7 we can conclude:

$$\begin{aligned} P_j^{idleM} &= V_{j_{ini}} \\ P_{j_{max}}^{mach} &= V_{j_{max}} \end{aligned} \tag{6.8}$$

To calculate the energy consumed by a PM, once the placement process has finished, we sort all the VMs assigned to each machine by execution time (et) in descending order:

$$\tau_1 > \tau_2 \tag{6.9}$$

where τ_1 denotes the biggest execution time of VM in the machine assigned and τ_2 the next execution time in descending order. Using equation 6.10 we can calculate the energy consumed by a PM

$$E_j = \sum_{k=1}^{T\_v-1} P_j^{mach}(\tau_k) \cdot (\tau_k - \tau_{k+1}) + P_j^{mach}(\tau_{T\_v}) \cdot (\tau_{T\_v}) \tag{6.10}$$

where T_v refers to total of VMs assigned to a PM j. The total energy consumed by the system is denoted as E_{total} (see Eq. 6.11)

$$E_{total} = \sum_{j}^{T\_m} E_j \tag{6.11}$$

where T_m refers to the total PMs in the system.

6.1.2 Energy-Aware VM Allocation Strategies for the Opportunistic Cloud

Resource allocation and scheduling techniques can be used in opportunistic cloud solutions to minimize the ECR or maximize the performance offered to researchers. To minimize the ECR, the resource allocation techniques should try to use a minimum Set of Physical Machines (SPMs) in busy state to execute the Set of Virtual Machines (SVMs) required by researchers, however this strategy regularly increases the execution time of the meta-task (the execution of a SVM). To maximize the performance provided to researchers the resource optimization and scheduling techniques should try to assign to each PM (preferably in idle, hibernated or turned off state) only one VM, this strategy decreases the penalty caused by the execution of multiple VMs on the same PM, and increases the ECR of the opportunistic solution due to more PMs are in execution. In this section three different energy-aware resource optimization techniques, for using in opportunistic cloud solutions, are introduced. These techniques take into account requirements of VMs and PMs. For VMs we take into account requirements associated to the number of CPU cores requested and the execution time; and for PMs requirements associated to CPU cores available and the status of the PM. The resource allocation algorithms were defined considering the following assumptions and goals:

- Reduce the ECR required to execute a metatask (SVMs).
- Several VMs can be executed on a PM.
- The assignation of VMs to PMs is made in batch mode.
- There are enough CPU cores to supports the CPU requirements of the SVC.
- It is better to assign a VM to a PM with an user (busy state) or to a PM (with available cores) that is already executing another VM (busy state).

- Physical machines are homogeneous.

6.1.2.1 Custom Round Robin Allocation

The simplest algorithm used to assign the execution of SVMs to SPMs is the Round Robin algorithm customized. Algorithm 10 shows that the SVM is received in an arbitrary order and each VM is assigned to a PM (also using an arbitrary order) if the physical machine has enough available CPU cores to its execution.

Algorithm 10: Custom Round Robin Algorithm

```
1 forall the VMs VM_i do
2     Set the VMs;
3     forall the PMs PM_j do
4         if PM_j.available_cores ≥ VM_i.cores then
5             assign VM_i to PM_j;
```

This algorithm is very easy to implement, however, it does not consider the state of the PM where a VM is executed, selecting randomly PMs in busy, idle, hibernated or turned off state, which may increase the number of PMs required to execute the SVM. The goal of using a minimum number of PMs to execute the SVM is not keep into account. Additionally some VMs with low CPU requirements may be initially assigned to PMs with large CPU capabilities, and then VMs with large CPU requirements may not be executed, and so queued until the execution of the next batch scheduling process.

6.1.2.2 1-D Bin Packing Allocation

UnaCloud usually executes VMs to run CPU-intensive applications. Therefore, we consider that the VMs consolidation problem is constrained by a single resource, in this case all the VMs are CPU bound, then the problem corresponds to the one dimensional bin packing problem with different bin capacity (i.e., different number of cores per PM).

The most popular and used heuristics to deal with the one dimensional bin packing problem are First-Fit, First-Fit Decreasing and Best-Fit algorithms [129]. These heuristics use a greedy weight function applied to the items such that every item is assigned a single value. In case of First-Fit Decreasing and Best-Fit algorithms the items are sorted and then placed sequentially into a decreasing order. The heuristics, specially First-Fit Decreasing, are best known to be very effective both in theory with performance guarantees and in practice. The worst-case behaviors of these algorithms and their average behavior over practical instances have been studied [153]. Results show that the algorithms have performance guarantees $d + \delta$, where d is the number of resource dimensions and δ is some constant ≤ 1. Several systems have implemented variants of the three heuristics. Therefore, we consider them to be evaluated in the context of UnaCloud.

6.1.2.3 Sorting VMs and PMs to Minimize the Use of PMs

To minimize the ECR rate used by the opportunistic cloud, the consolidation strategies should try to place the VMs first on a PM in a busy state (i.e., with a VM already assigned on it or with a physical user) that satisfies the VM requirements rather than placing the VM on a PM in a different state. In [50] we proposed a packing algorithm that prioritizes the deployment of VMs on PMs already in use. The algorithm (Algorithm 11), called Sorting in [50], is a variant of First-Fit Decreasing.

Algorithm 11: Sorting VMs and PMs to minimize the use of PMs

```
1 Sorting VMs and PMs;
2 forall the VMs VM_i do
3     Set the VMs;
4     forall the PMs PM_j do
5         if PM_j.available_cores ≥ VM_i.cores then
6             assign VM_i to PM_j;
7             re-order PMs;
```

The algorithm starts by sorting the set of VMs and PMs. VMs are sorted in decreasing order of required cores and estimated running time. PMs are sorted by three attributes: (1) PMs that already have virtual machines running on them, (2) PMs in busy state (an user is using them), and (3) PMs with more available CPU cores. Then, VMs are placed on the PMs in a First-Fit policy. After a VM is placed on a PM the ordered list of the PMs is sorted again. The attributes used to sort PMs allow that PMs with a VM already assigned or in a busy state have priority over the others, hence reducing the ECR rate as described in Section 6.1.1.

6.1.2.4 Sorting VMs and PMs to Minimize the Use of PMs and Executing VMs with Similar Execution Time on the Same PM

Although the algorithm proposed in the previous section reduce the ECR required to execute the SVM, we identified that if VMs with similar execution times are executed on the same PMs the ECR required to execute the SVM is lower. This is due to every PMs used to executed the SVM will execute tasks at peak capacity during a similar period of time, which according to Figure 6.1, allows to have a minimum number of PMs executing VMs during similar execution times.

As a comparison example, supposing that there are 2 PMs (PM_1 and PM_2) each one with 8 cores available and three VMs (VM_1, VM_2 and VM_3) with different execution times (10 hours, 1 hour and 10.5 hours respectively) each one requiring 4 CPU cores to be executed. The algorithm described in the previous section would select PM_1 to execute VM_1 (during 10 hours) and VM_2 (during 1 hour); and PM_2 to execute VM_3 (during 10.5 hours). This implies that only during 1 hour there will be 1 PM running at 100% of its CPU capacity. With the algorithm proposed in this section, VM_1 and VM_3 would be executed on PM_1, and VM_2 would be executed on PM_2. With this new strategy 1 PM will be running at 100%

of its capacity during 10 hours, which reduces the ECR required to execute the SVM. The algorithm proposed to achieve this assignation is the Algorithm 12.

Algorithm 12: Executing VMs with Similar Execution Time within same PM

```
1  Sorting VMs and PMs;
2  forall the PMs PM_j do
3      Set the PMs;
4      forall the VMs VM_i do
5          Set the VMs for all PMs;
6          if !VM_i.assigned and PM_j.available_cores ≥ VM_i.cores then
7              taskI = VM_i;
8              repeat
9                  assign taskI to PM_j;
10                 if PM_j still have cores then
11                     assigns VMs with similar execution time;
12             until PM_j have available cores and !taskI;
```

In Algorithm 12, PMs are sorted using the same three attributes of Algorithm 11. VMs are sorted by the number of cores required and by the execution time in descending form (line 1). All of the PMs are processed (line 2) and all of the VMs are processed (line 4) for each PM. Once a VM pending of execution that requires a number of cores inferior or equal to the available cores of the current PM is found (line 6). Then, VM is assigned to the current PM (line 9) and then if the current PM still have available CPU cores (line 10), the next unassigned VM with similar execution time is selected and assigned to the current PM (line 11), while the PM has available cores and there are not VMs that can be executed within PM (line 13).

Next section presents the evaluation comparison of First-Fit, First-Fit Decreasing, Best-Fit, and Sorting using UnaCloud scenarios.

6.1.3 Experimental Results

This section presents the empirical evaluation of the investigated consolidation algorithms. The aim is to gain a first insight into the performance of the algorithms on different real scenarios before implementing the best one on UnaCloud.

UnaCloud currently has access to three computer labs with 109 desktop computers, whose aggregated capabilities may deliver up to 592 processing cores (70 PMs have four cores each and 39 PMs with eight cores), 572 GB of RAM, 8 TB of storage and 1TB of shared storage in a Network Attached Storage (NAS).

6.1.3.1 Workload

In order to provide performance comparison, we use workloads based on real VMs production traces. Real UnaCloud scenarios provide a realistic VMs stream for performance evaluation

based on simulations of VMs allocation algorithms. Besides, traces from the archives of UnaCloud are used. The traces have been collected during one year. The total number of VMs in the workload requested during the year is up to 9800 each one requiring either 1, 2, 3, 4, 6, or 8 cores, 1, 2, 3, 4, 6, or 8 GB of RAM, and 20 GB of storage.

In order to estimate the energy consumed by a given placement, we use the information provided in Section 6.1.1. The VMs are also characterized by different time periods from 45 minutes up to 43200 minutes ($\approx$ 720 h), the time requested by users on demand to execute VMs. The ECR rate values represent the power drawn by the PMs at the utilization given by the placement over the execution time. We assume that when a PM does not have a VM assigned to it and an user is not working on it the PM can be turned off. Hence, no energy is consumed by the PM and is not included in the computation of the total ECR rate.

6.1.3.2 Experimental Scenarios

Different scenarios have been generated as follows. We consider that a given percentage of PMs has an user working on it. To simulate the scenarios and generate the instances used by the consolidation algorithms, we vary the percentage from 0% up to 50% with the increment 10%. We randomly assign an user to a PM and we generate 30 instances for each scenario. The number of VMs to be placed on the PMs varies from 40 up to 130 with the increment 10. We generate 30 instances at random for each size from the real traces and we used them as workloads for all the simulations discussed below. The maximum number of cores on each size is up to the total available physical cores in order to support the worst packing scenario, in which all the PMs run at least one VM. We assume that a physical user utilizes a PM during 60 minutes up to 240 minutes. We randomly assign the time of the physical user working on the PM (uniformly in the interval [60, 240]).

6.1.3.3 Algorithms Comparison

We measured the amount of provisioned PMs, and the ECR rate of the placement for every algorithm. We report average results. The objective is to explore the behavior of the investigated heuristics and the gain of the opportunistic environment.

Figures 6.2 shows the number of PMs used by each of the heuristics to place all the VMs and Figures 6.3 presents the ECR rate for the considered scenarios.

As we can see in Figure 6.2 the Sorting heuristic utilizes important lower amount of PMs yielding to superior average PM optimization and significant ECR lower consumption. Considering the extreme scenarios, that is, assuming 0% and 50% of PMs with an user, the heuristic optimizes up to 41% more than First-Fit and First-Fit decreasing, and 36% more than Best-Fit when there are no physical users in the PMs. When 50% of the PMs are in an busy state Sorting requires 30% on average less PMs than the related heuristics. The main reason that Sorting needs more PMs in the second scenario is that it places the VMs to a maximum number of PMs in a busy state.

We can observe that Sorting gains significant energy saving (i.e. low ECR) regarding the related heuristics. The average gain of ECR by Sorting when no users are assigned to PMs is up to 38% regarding First-Fit, up to 34% concerning First-Fit Decreasing, and up to 35% with respect to Best-Fit. For the second scenario whit 50% of busy PMs all the investigated heuristics gain more energy than in the first scenario with all the PMs in an idle

state. These results highlight the benefits and advantages of an opportunistic cloud, in this case UnaCloud, as a sustainable infrastructure.

The average ECR gain of Sorting is more important regarding related heuristics than in the first scenario. Sorting optimizes 48% more energy than First-Fit, it can gain up to 45% regarding First-Fit Decreasing, and up to 46% more than Best-Fit. The results highlight that prioritizing busy PMs to place the VMS is a good saving energy option to consolidate VMs.

It can be observed that First-Fit utilizes more PMs (approx. 4%) in the opportunistic environment to consolidate the VMs than Best-Fit in almost all the scenarios, however the ECR rate is lower than Best-Fit. We consider that it is due to the fact that First-Fit uses more PMs in an idle state than Best-Fit, nevertheless for a shorter time of period than Best-Fit, hence the ECR rate is less than Best-Fit.

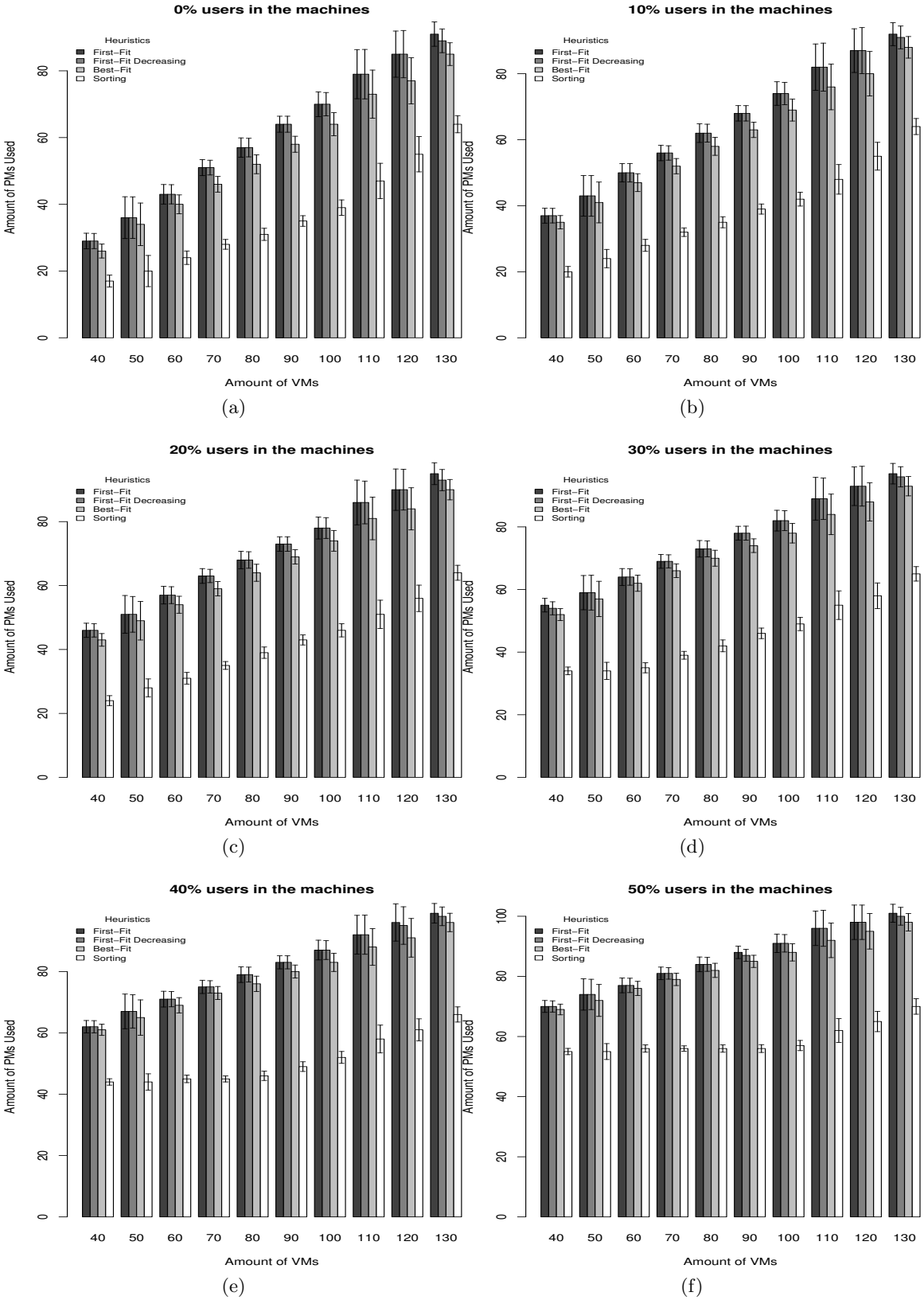


Figure 6.2: Number of PMs to place the VMs

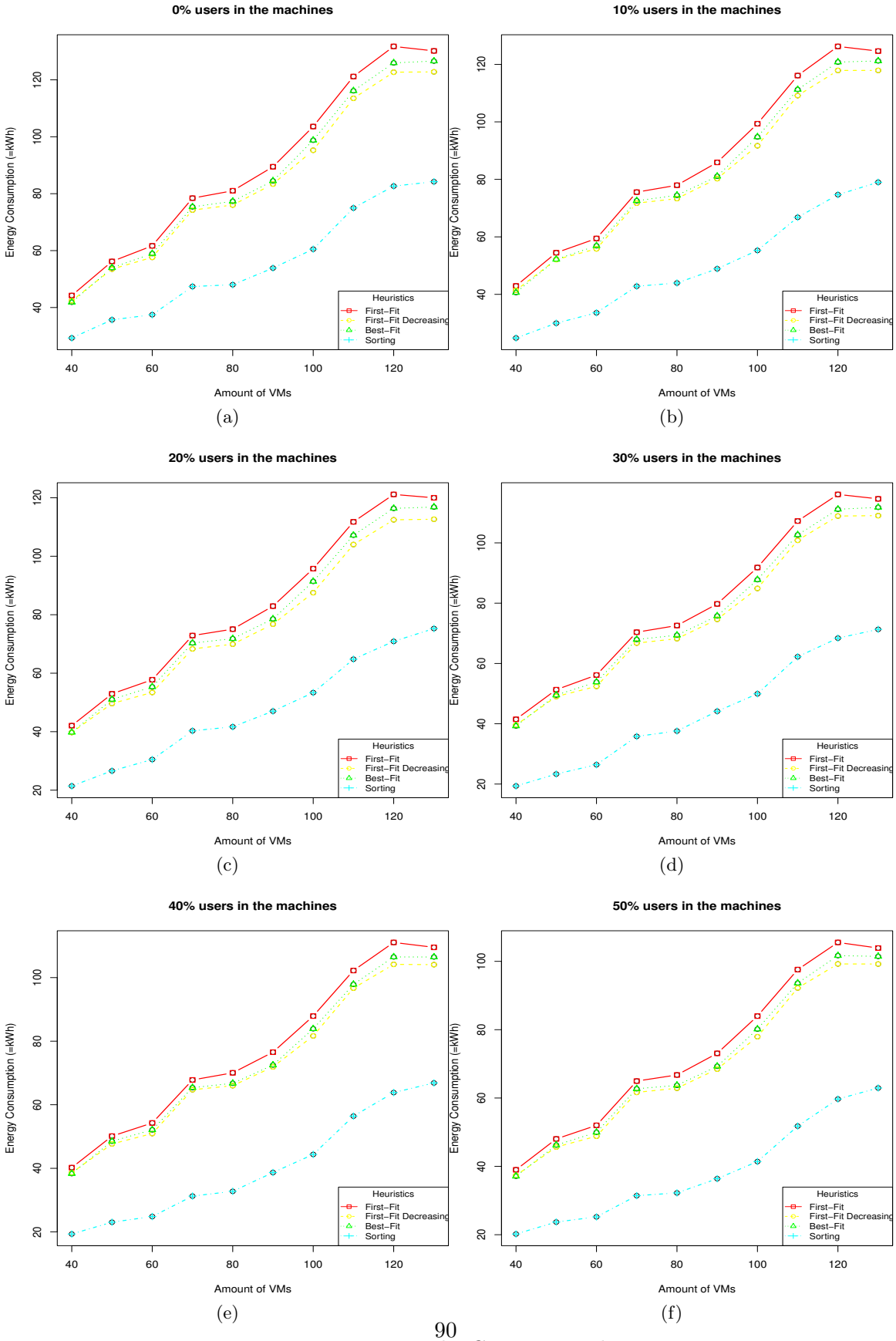


Figure 6.3: Total ECR consumed.

Chapter 7

Conclusions and Perspectives

Contents

7.1 Summary

In this thesis, we studied two different problems, the scheduling problem over grid computing and the resource allocation problem over opportunistic cloud computing. For the first problem, we did not only tackle the problem from two different perspectives, performance and energy consumption, but also reduced the complexity of the proposed algorithms. We compared, our proposed algorithms, with the best scheduling algorithms founded in the literature. Concerning resource allocation problem, we tackled the problem from two perspectives, opportunistic environment and energy consumption. Moreover, we used two environments grid and cloud to evaluate our resource allocation algorithms. We also compared, our resource allocation algorithms, with the most common used in the literature. A selection of such solutions is carefully done.

The major contributions contained in this PhD thesis are:

- We presented a state of the art in scheduling heuristics and resource allocation for opportunistic environments (including grid and cloud computing).
- We proposed and evaluated low computational complexity algorithms in the context of their performance and scalability in HCS. The algorithms are evaluated and compared with the best reported in the literature. The set of experimental results shows that low computational complexity heuristics perform as efficiently as known ones considering the makespan criterion, and outperform them in terms of number of best solution found criteria. Detail analysis show that low computational complexity heuristics are the best performing heuristics in the original-consistency cases showing similar behavior in the partial-consistency scenarios. Moreover, these low computational complexity heuristics have significant good scalability and efficiency

- We presented a description of Opportunistic cloud computing, ranging from Desktop Computing to Volunteer Computing.
- We described a particular Opportunistic cloud infrastructure, named *UnaCloud*, where the experiments were developed.
- We assessed *UnaCloud* infrastructure by a complete benchmarking, measuring performance and storage. Also we presented *UnaCloud* as PaaS.
- We presented and validated an energy model based on the experimentation, first in opportunistic grid then in opportunistic cloud environments.
- We addressed energy savings on an opportunistic infrastructure, specifically for UnaCloud. We proposed a new resource allocation algorithm that is a variant of First-Fit Decreasing. The proposed algorithm was validated against three well known state-of-the-art heuristics. We have simulated different scenarios using real workload traces. The results showed that the opportunistic infrastructure seems to be a good option as a sustainable computing on demand infrastructure.

7.2 Future research lines

As a future work, we are planning to enhance our low complexity heuristic to be fault tolerant, furthermore, we would like to test them in a real Grid or Cloud computing environment.

There are still some open issues in opportunistic environments that we would like to address in the future. We are planing to consider communication issues and heterogeneity of resources during the placement of VMs, the problem can be modeled as a multi-dimensional packing problem. Moreover, interoperability is a big issue that we are going to tackle.

We want continue the evolution of our opportunistic infrastructure, not only as PaaS and IaaS, but also as SaaS and why not Business orchestration.

7.2.1 Advances in Cloud Computing

There are several research in all components to make the Cloud accessible, flexible, elastic, pervasive, ubiquitous, interoperable, portable, multi-tenant, etc,.

At this time, the last buzzword is with BigData due to the increase of content in the web as social networks, the technology to digital storage has been improved, the quantity of information is huge, not only for scientific reasons, that it was a gap at the beginning, now all kind of information is in the web.

7.2.1.1 Future of the combination of Internet Of Things with Cloud Computing

Ubiquitous cloud computing refers to use of Internet resources at any place and any time for any objectives. Today, people can access the Internet from anywhere by fixed or wireless networks. New challenges for implement services in the cloud has been impacted with a new research environment that leverages software and services being provided on user demand. The IoT is a natural extension of the Internet. The foundation of the IoT is *radio-frequency identification* (RFID). This enables the discovery of tagged objects and mobile devices by

browsing an IP address or searching for a database entry. In the future all objects on Earth be radio-tagged, ranging from a milk cartoon to a truck container to a jumbo jet.Computers identify and manage all tagged objects in the same way humans do. Now the machines need services (e.g. think and understanding), a complete new kind of applications will be developed for satisfying users' need, with IoT technology arriving to the market, the services offered will multiply. The IoT system will improve our quality of life and make society cleaner and more secure.

7.2.1.2 Future of the SmartGrids with Cloud Computing

A smart grid includes an intelligent monitoring system that keeps track of all electricity flowing in the system as well as it can be distributed to each new micro grid created. Smart meters, consist in a digital upgrade of current utility meters and capacity to get more variables from the power grid, track energy usage in real time so that both the customer and the utility company know how much is being used at any given time or in real time. Energy is paid for using "time of day" pricing, meaning electricity will cost more at peak times of use. In a combination of Cloud Computing with IoT the commodity for manage energy could be the best. For instance, when power is least expensive the user can allow the smart grid to turn on selected home appliances such as washing machines or factory processes that can run at arbitrary hours. At peak times, when the cost of energy is higher, it could turn off selected appliances to reduce demand, all of this in a remote way.

References

[ICA3PP2013] 57

[2] Condor high throughput computing, 2011. Available online at http://www.cs.wisc.edu/condor/. Cited August 2011. 18

[3] Openpbs, 2012. Available online at http://www.mcs.anl.gov/research/projects/openpbs/. Cited November 2012. 18

[4] Lsf - load sharing facility, 2013. Available online at http://www.vub.ac.be/BFUCC/LSF/. Cited January 2013. 18

[5] Oar resource management system for high performance computing, 2013. Available online at http://oar.imag.fr. Cited January 2013. 18

[6] Torque resource manager, 2013. Available online at http://www.adaptivecomputing.com/products/open-source/torque. Cited January 2013. 18

[7] Abdulla Al-Qawasmeh, Anthony A. Maciejewski, Haonan Wang, Jay Smith, Howard Jay Siegel, and Jerry Potter. Statistical measures for quantifying task and machine heterogeneities. *The Journal of Supercomputing*, 57(1):34–50, 2011. 18

[8] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen. Representing task and machine heterogeneities for heterogeneous computing systems. *Journal of Science and Engineering*, 3(3):195–207, 2000. 5, 29, 30, 44

[9] Amazon. High Performance Computing on AWS. http://aws.amazon.com/hpc-applications/, 2013. [Online; accessed July-2013]. 67

[10] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2256-4. doi: 10.1109/GRID.2004.14. URL http://dx.doi.org/10.1109/GRID.2004.14. 49, 50

[11] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11): 56–61, November 2002. ISSN 0001-0782. doi: 10.1145/581571.581573. URL http://doi.acm.org/10.1145/581571.581573. 50

[12] Nazareno Andrade, Walfredo Cirne, Francisco Brasileiro, and Paulo Roisenberg. Ourgrid: An approach to easily assemble grids with equitable resource sharing. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 61–86. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20405-3. doi: 10.1007/10968987_4. URL http://dx.doi.org/10.1007/10968987_4. 50

[13] Cosimo Anglano, Massimo Canonico, and Marco Guazzone. The sharegrid peer-to-peer desktop grid: Infrastructure, applications, and performance evaluation. *J. Grid Comput.*, 8(4):543–570, December 2010. ISSN 1570-7873. doi: 10.1007/s10723-010-9162-z. URL http://dx.doi.org/10.1007/s10723-010-9162-z. 50

[14] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, pages 79 –87, mar 1998. doi: 10.1109/HCW.1998.666547. 10

[15] E. Bampis, F. Guinand, and D. Trystram. Some models for scheduling parallel programs with communication delays. *Discrete Applied Mathematics*, 72(1-2):5–24, 1997. 11

[16] Nikhil Bansal. *Algorithms for Flow Time Scheduling*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2003. 38

[17] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003. ISSN 0163-5980. doi: 10.1145/1165389.945462. URL http://doi.acm.org.proxy.bnl.lu/10.1145/1165389.945462. 54

[18] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, December 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.443. URL http://dx.doi.org/10.1109/MC.2007.443. 50

[19] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association. URL http://dl.acm.org/citation.cfm. 54

[20] Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 577–578, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4039-9. doi: 10.1109/CCGRID.2010.45. URL http://dx.doi.org/10.1109/CCGRID.2010.45. 13

[21] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82:47–111, 2011. URL http://dblp.uni-trier.de/db/journals/ac/ac82.html#BeloglazovBLZ11. 51

[22] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):299–316, 2000. ISSN 1063-8210. doi: 10.1109/92.845896. 12, 51

[23] H.J.C. Berendsen, D. van der Spoel, and R. van Drunen. Gromacs: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91(3):43 – 56, 1995. ISSN 0010-4655. doi: 10.1016/0010-4655(95)00042-E. URL http://www.sciencedirect.com/science/article/pii/001046559500042E. 68, 71

[24] Kashif Bilal, SameeU. Khan, SajjadA. Madani, Khizar Hayat, MajidI. Khan, Nasro Min-Allah, Joanna Kolodziej, Lizhe Wang, Sherali Zeadally, and Dan Chen. A survey on green communications using adaptive link rate. *Cluster Computing*, 16(3):575–589, 2013. ISSN 1386-7857. 79

[25] Kashif Bilal, Saif Ur Rehman Malik, Osman Khalid, Abdul Hameed, Enrique Alvarez, Vidura Wijaysekara, Rizwana Irfan, Sarjan Shrestha, Debjyoti Dwivedy, Mazhar Ali, Usman Shahid Khan, Assad Abbas, Nauman Jalil, and Samee U. Khan. A taxonomy and survey on green data center networks. *Future Generation Computer Systems*, 36(0): 189 – 208, 2014. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2013.07.006. URL http://www.sciencedirect.com/science/article/pii/S0167739X13001519. Special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications. 12

[26] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Lasislau L. Bölöni, Muthucumara Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.*, 61:810–837, 2001. 5, 11, 16, 17, 34, 42, 44

[27] Paul Brebner, Emmanuel Cecchet, Julie Marguerite, Petr Tma, Octavian Ciuhandu, Bruno Dufour, Lieven Eeckhout, Stphane Frenot, Arvind S. Krishna, John Murphy, and Clark Verbrugge. Middleware benchmarking: approaches, results, experiences. *Concurrency and Computation: Practice and Experience*, 17(15):179 – 185, 2005. ISSN 1532-0634. doi: 10.1002/cpe.918. URL http://dx.doi.org/10.1002/cpe.918. 60

[28] Paul C. Brebner. Is your cloud elastic enough?: performance modelling the elasticity of infrastructure as a service (iaas) cloud applications. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 263–266, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1202-8. doi: 10.1145/2188286.2188334. URL http://doi.acm.org.proxy.bnl.lu/10.1145/2188286.2188334. 61

[29] Peter Brucker. *Scheduling Algorithms*. Springer, Berlin, Germany, 5th edition, 2007. ISBN 978-3-540-69515-8. 8

[30] P. Bunci, C. Aguado Sanchez, J. Blomer, L. Franco, A. Harutyunian, P. Mato, and Y Yao. Cernvm a virtual software appliance for lhc applications. *J. of Physics.*, 219

(4):43 – 53, December 2010. ISSN 1742-6588. doi: 10.1088/1742-6596/219/4/042003. 50

[31] T.D. Burd, T.A. Pering, A.J. Stratakos, and R.W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE J. Solid-State Circuits*, 35(11):1571–1580, 2000. 43

[32] Rajkumar Buyya, Anton Beloglazov, and Jemal H. Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. In *Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 6–20, Las Vegas, USA, July 2010. CSREA Press. 13

[33] B. Barla Cambazoglu, E. Kartal Tabak, and Cevdet Aykanat. Improving the performance of independenttask assignment heuristics minmin,maxmin and sufferage. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1244–1256, 2014. ISSN 1045-9219. doi: http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.107. 11

[34] M. C. Cardoso and F. M. Costa. Mpi support on opportunistic grids based on the integrade middleware. *Concurrency and Computation: Practice and Experience*, 22(3):343–357, 2010. ISSN 1532-0634. doi: 10.1002/cpe.1479. URL http://dx.doi.org/10.1002/cpe.1479. 50

[35] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *Proc. of the 9th Heterogeneous Computing Workshop*, pages 349–363, USA, 2000. 11

[36] Harold Castro, Eduardo Rosales, and Mario Villamizar. Desktop grids and volunteer computing systems. In Nikolaos Preve, editor, *Computational and Data Grids: Principles, Applications and Design*, pages 1–30. IGI Global, 2012. ISBN 9781613501146. doi: 10.4018/978-1-61350-113-9. 52

[37] Harold Castro, Mario Villamizar, German Sotelo, CesarO. Diaz, JohnatanE. Pecero, and Pascal Bouvry. Green flexible opportunistic computing with task consolidation and virtualization. *Cluster Computing*, 16(3):545–557, 2013. ISSN 1386-7857. doi: 10.1007/s10586-012-0222-y. URL http://dx.doi.org/10.1007/s10586-012-0222-y. 79, 80

[38] Harold Castro, Mario Villamizar, German Sotelo, Cesar O. Diaz, Johnatan E. Pecero, Pascal Bouvry, and Samee U. Khan. Gfog: Green and flexible opportunistic grids. In Samee U. Khan, Lizhe Wang, and Albert Y. Zomaya, editors, *Scalable Computing and Communications, Theory and Practice*. Wiley&Sons, Forthcomming. 57, 81

[39] Texas Advanced Computer Center. Gotoblas2. http://www.tacc.utexas.edu/tacc-projects/gotoblas2/, 2013. [Online; accessed 28-January-2013]. 74

[40] CERN. Ladron Hadron Collector. http://lhcathome.web.cern.ch/LHCathome/, 2011. [Online; accessed 01-November-2012]. 50

[41] Hui Chen, Bing Luo, and Weisong Shi. Anole: A case for energy-aware mobile application design. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 232–238, Sept 2012. doi: 10.1109/ICPPW.2012.34. 3

[42] SungJin Choi and Rajkumar Buyya. Group-based adaptive result certification mechanism in desktop grids. *Future Generation Computer Systems*, 26(5):776 – 786, 2010. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2009.05.025. URL http://www.sciencedirect.com/science/article/pii/S0167739X09000557. 14

[43] G. Da Costa, J.-P. Gelas, Y. Georgiou, L. Lefevre, A.-C. Orgerie, J. Pierson, O. Richard, and K. Sharma. The green-net framework: Energy efficiency in large scale distributed systems. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, May 2009. doi: 10.1109/IPDPS.2009.5160975. 51

[44] Raphael de Aquino Gomes and Fbio M. Costa. An approach to enhance the efficiency of opportunistic grids. *Concurrency and Computation: Practice and Experience*, 23(17):2092 – 2106, 2011. ISSN 1532-0634. doi: 10.1002/cpe.1707. URL http://dx.doi.org/10.1002/cpe.1707. 50

[45] J.Y. De la Pava Torres and C. Jimenez-Guarin. Magoscloud secure: A secure, highly scalable platform for services in an opportunistic environment. In *High Performance Computing and Simulation (HPCS), 2012 Intl. Conf. on*, pages 53–59, 2012. doi: 10.1109/HPCSim.2012.6266890. 67

[OPTIM2011] 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 *HPCS, 2011 Int. Conference*, pages 478 –484, july 2011. doi: 10.1109/HPCSim.2011.5999863. 11, 17, 18, 119

[CCGrid2014] Cesar O. Diaz, Johnatan E. Pecero, Pascal Bouvry, German Sotelo, Mario Villamizar, and Harold Castro. Performance evaluation of an iaas opportunistic cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 546–547, May 2014. doi: 10.1109/CCGrid.2014.116. 59

[SUPE2013] CesarO. Diaz, JohnatanE. Pecero, and Pascal Bouvry. Scalable, low complexity, and fast greedy scheduling heuristics for highly heterogeneous distributed computing systems. *The Journal of Supercomputing*, pages 1–17, 2013. ISSN 0920-8542. doi: 10.1007/s11227-013-1038-0. URL http://dx.doi.org/10.1007/s11227-013-1038-0. 23

[49] C.O. Diaz, M. Guzek, J.E. Pecero, P. Bouvry, and S.U. Khan. Scalable and energy-efficient scheduling techniques for large-scale systems. In *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, pages 641–647, Aug 2011. doi: 10.1109/CIT.2011.106. 50

[50] C.O. Diaz, H. Castro, M. Villamizar, J.E. Pecero, and P. Bouvry. Energy-aware vm allocation on an opportunistic cloud infrastructure. In *Proceedings of the 2013 13th IEEE/ACM Int. Symposium CCGRID*, pages 663–670. IEEE Computer Society, 2013. doi: 10.1109/CCGrid.2013.96. 79, 85

[51] Salvatore Distefano, Vincenzo D. Cunsolo, Antonio Puliafito, and Marco Scarpa. Cloudathome: A new enhanced computing paradigm. In Furth Borko and Armando Escalante, editors, *Handbook of Cloud Computing*, pages 575–594. Springer US, 2010. ISBN 978-1-4419-6523-3. doi: 10.1007/978-1-4419-6524-0. 50

[52] F. Doelitzscher, M. Held, C. Reich, and A. Sulistio. Viteraas: Virtual cluster as a service. In *IEEE Int. Conf. (CloudCom)*, pages 652–657, 2011. doi: 10.1109/CloudCom.2011.101. 67

[53] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002. 5, 29

[54] Elizabeth D. Dolan, Jorge J. Moré, and Todd S. Munson. Optimality measures for performance profiles. *SIAM J. on Optimization*, 16(3):891–909, March 2006. ISSN 1052-6234. doi: 10.1137/040608015. 5, 29

[55] Fang Dong, Junzhou Luo, Jiahui Jin, Yanhao Wang, and Yanmin Zhu. Performance evaluation and analysis of seu cloud computing platform. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 1455–1460, 2012. doi: 10.1109/ICSMC.2012.6377940. 61

[56] Fangpeng Dong and Selim G. Akl. Technical report no. 2006-504 scheduling algorithms for grid computing: State of the art and open problems, 2006. URL http://research.cs.queensu.ca/home/akl/techreports/GridComputing.pdf. 10

[57] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003. ISSN 1532-0634. doi: 10.1002/cpe.728. URL http://dx.doi.org/10.1002/cpe.728. 61

[58] Y. El-Khamra, Hyunjoo Kim, S. Jha, and M. Parashar. Exploring the performance fluctuations of hpc workloads on clouds. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 383–387, 2010. doi: 10.1109/CloudCom.2010.84. 61

[59] Lionel Eyraud. A pragmatic analysis of scheduling environments on new computing platforms. *Int. J. High Perform. Comput. Appl.*, 20(4):507–516, November 2006. 18

[60] Pablo Ezzatti, Martin Pedemonte, and Alvaro Martin. An efficient implementation of the min-min heuristic. *Computers & Operations Research*, 40(11):2670 – 2676, 2013. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2013.05.014. URL http://www.sciencedirect.com/science/article/pii/S0305054813001433. 11

[61] Eugen Feller, Louis Rilling, and Christine Morin. Energy-aware ant colony based workload placement in clouds. In *Proceedings of the 2011 IEEE/ACM 12th Int. GRID*, pages 26–33, Washington, DC, USA, September 2011. IEEE Computer Society. 14

[62] Eugen Feller, Louis Rilling, and Christine Morin. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In *Proceedings of the 2012*

12th IEEE/ACM Int. Symposium CCGRID, pages 482–489, Washington, DC, USA, May 2012. IEEE Computer Society. ISBN 978-0-7695-4691-9. 13

[63] Hector Fernandez, Corina Stratan, and Guillaume Pierre. Robust performance control for web applications in the cloud. In *4th International Conference on Cloud Computing and Services Science*, Barcelona, Espagne, April 2014. URL http://hal.inria.fr/hal-01006607. Best paper award. 3

[64] Marcelo Finger, Germano C. Bezerra, and Danilo R. Conde. Resource use pattern analysis for predicting resource availability in opportunistic grids. *Concurrency and Computation: Practice and Experience*, 22(3):295–313, 2010. ISSN 1532-0634. doi: 10.1002/cpe.1478. URL http://dx.doi.org/10.1002/cpe.1478. 50

[65] Pinel Frederic, Dorronsoro Bernabe, and Bouvry Pascal. Solving very large instances of the scheduling of independent tasks problem on the gpu. *Journal of Parallel and Distributed Computing*, pages 1–8, 2012. ISSN 0743-7315. doi: 10.1016/j.jpdc.2012.02.018. Available online 9 March 2012. 12, 16

[66] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel. Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In *Proceedings of the Seventh Heterogeneous Computing Workshop*, HCW '98, pages 184–199, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8365-1. 10

[67] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002. ISSN 1386-7857. doi: 10.1023/A:1015617019423. URL http://dx.doi.org/10.1023/A%3A1015617019423. 50

[68] Marc Eduard Frincu. Scheduling highly available applications on cloud environments. *Future Generation Computer Systems*, 32(0):138 – 153, 2014. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2012.05.017. URL http://www.sciencedirect.com/science/article/pii/S0167739X12001136. Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures. 12

[69] Cecile Germain, Vincent Neri, Gilles Fedak, and Franck Cappello. Xtremweb: Building an experimental platform for global computing. In Rajkumar Buyya and Mark Baker, editors, *Grid Computing Ñ GRID 2000*, volume 1971 of *Lecture Notes in Computer Science*, pages 91–101. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-41403-2. doi: 10.1007/3-540-44444-0_9. URL http://dx.doi.org/10.1007/3-540-44444-0_9. 50

[70] Arif Ghafoor and Jaehyung Yang. A distributed heterogeneous supercomputing management system. *IEEE Computer*, 26(6):78–86, June 1993. 18

[71] Arnaud Giersch, Yves Robert, and Frédéric Vivien. Scheduling tasks sharing files on heterogeneous master-slave platforms. *J. Syst. Archit.*, 52:88–104, 2006. 18

[72] Andrei Goldchleger, Fabio Kon, Alfredo Goldman, Marcelo Finger, and Germano Capistrano Bezerra. Integrade object-oriented grid middleware leveraging the idle computing power of desktop machines: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(5):449–459, April 2004. ISSN 1532-0626. doi: 10.1002/cpe.v16:5. URL http://dx.doi.org/10.1002/cpe.v16:5. 50

[73] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5(2):287–326, 1979. doi: 10.1016/S0167-5060(08)70356-X. 8

[74] Jim Gray. *The Benchmark Handbook for Database and Transaction systems*. Morgan Kaufmann, ACM SIGMOD Anthology, 2nd edition, 1993. ISBN 1-55860-292-5. 59, 61

[75] Frederic Guinand, Aziz Moukrim, and Eric Sanlaville. Sensitivity analysis of scheduling uect trees on two processors. *Parallel Computing*, 30(1):103–120, 2004. doi: DOI: 10.1016/S0167-8191(03)00091-7. 11

[76] Torben Hagerup. Allocating independent tasks to parallel processors: An experimental study. *Journal of Parallel and Distributed Computing*, 47(2):185–197, 1997. 10

[77] XiaoShan He, XianHe Sun, and Gregor von Laszewski. Qos guided min-min heuristic for grid task scheduling. *J. Comput. Sci. Technol.*, 18(4):442–451, July 2003. ISSN 1000-9000. 11

[78] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50, New York, NY, USA, March 2009. ACM. ISBN 978-1-60558-375-4. 13

[79] Adan Hirales-Carbajal, Andrei Tchernykh, Ramin Yahyapour, JoseLuis Gonzalez-Garcia, Thomas Roblitz, and JuanManuel Ramirez-Alcaraz. Multiple workflow scheduling strategies with user run time estimates on a grid. *Journal of Grid Computing*, 10(2):325–346, 2012. ISSN 1570-7873. doi: 10.1007/s10723-012-9215-6. URL http://dx.doi.org/10.1007/s10723-012-9215-6. 16, 17, 29

[80] Ching-Hsien Hsu, Shih-Chang Chen, Chih-Chun Lee, Hsi-Ya Chang, Kuan-Chou Lai, Kuan-Ching Li, and Chunming Rong. Energy-aware task consolidation technique for cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 115–121, Nov 2011. doi: 10.1109/CloudCom.2011.25. 56

[81] H. Hussain, S. U. R. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, W. Yongji, N. Ghani, J. Kolodziej, A. Y. Zomaya, C. Xu, Pavan Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, P. Bouvry, H. Li, L. Wang, D. Chen, and A. Rayes. A survey on resource allocation in high performance distributed computing systems. *Parallel Computing*, 39:709–736, 11/2013 2014. URL http://www.sciencedirect.com/science/article/pii/S016781911300121X. 13

[82] Oscar H. Ibarra and Chul E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *J. ACM*, 24:280–289, 1977. 7, 16, 17, 22, 23, 42, 119

[83] Uptime Institute. 2013 data center industrie survey, 2014. URL http://uptimeinstitute.com/images/stories/DataCenterSurvey_assets/uptime-institute-23

[84] A. Iordache, C. Morin, N. Parlavantzas, E. Feller, and P. Riteau. Resilin: Elastic mapreduce over multiple clouds. In *13th IEEE/ACM CCGrid*, pages 261–268, 2013. doi: 10.1109/CCGrid.2013.48. 67

[85] A. Iosup and D. Epema. Grenchmark: A framework for analyzing, testing, and comparing grids. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 313–320, 2006. doi: 10.1109/CCGRID.2006.49. 61

[86] A. Iosup, S. Ostermann, M.N. Yigitbasi, R. Prodan, T. Fahringer, and D. H J Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):931–945, 2011. ISSN 1045-9219. doi: 10.1109/TPDS.2011.66. 61

[87] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 104–113, 2011. doi: 10.1109/CCGrid.2011.22. 61

[88] Alexandru Iosup. Iaas cloud benchmarking: approaches, challenges, and experience. In *Proceedings of the 2013 international workshop on Hot topics in cloud services*, HotTopiCS 13, pages 1–2, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2051-1. doi: 10.1145/2462307.2462309. URL http://doi.acm.org.proxy.bnl.lu/10.1145/2462307.2462309. 59, 60, 61

[89] Alexandru Iosup, DickH.J. Epema, Carsten Franke, Alexander Papaspyrou, Lars Schley, Baiyi Song, and Ramin Yahyapour. On grid performance evaluation using synthetic workloads. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 4376 of *Lecture Notes in Computer Science*, pages 232–255. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-71034-9. 61

[90] Sadeka Islam, Kevin Lee, Alan Fekete, and Anna Liu. How a consumer can measure elasticity for cloud platforms. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 85–96, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1202-8. doi: 10.1145/2188286.2188301. URL http://doi.acm.org.proxy.bnl.lu/10.1145/2188286.2188301. 61

[91] Michael A. Iverson, Füsun Özgüner, and Lee Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. *IEEE Trans. Comput.*, 48(12):1374–1379, December 1999. ISSN 0018-9340. 18

[92] S.U. Khan and I. Ahmad. A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids. *IEEE Trans on Parallel and Distributed Systems*, 20(3):346–360, 2009. 12, 51

[93] Ashfaq A. Khokhar, Viktor K. Prasanna, Muhammad E. Shaaban, and Cho-Li Wang. Heterogeneous computing: Challenges and opportunities. *IEEE Computer*, 26(6):18–27, June 1993. 18

[94] Jong-Kook Kim, Howard Jay Siegel, Anthony A. Maciejewski, and Rudolf Eigenmann. Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling. *IEEE Transactions on Parallel and Distributed Systems*, 19:1445–1457, 2008. 42, 104

[95] Jong-Kook Kim, Howard Jay Siegel, Anthony A. Maciejewski, and Rudolf Eigenmann. Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling. In *IEEE Transactions on Parallel and Distributed Systems* Kim et al. [94], pages 1445–1457. 12, 51

[96] Kyong Hoon Kim, R. Buyya, and Jong Kim. Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 541–548, May 2007. doi: 10.1109/CCGRID.2007.85. 12, 51

[97] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of virtual machines for real-time cloud services. *Concurr. Comput. : Pract. Exper.*, 23(13):1491–1505, September 2011. ISSN 1532-0626. doi: 10.1002/cpe.1712. URL http://dx.doi.org/10.1002/cpe.1712. 51

[98] J. Kolodziej, S. U. Khan, and F. Xhafa. Genetic algorithms for energy-aware scheduling in computational grids. In *Proceedings of the 6th IEEE International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing*, 3PGCIC, pages 17–24, Barcelona, ES, 2011. 11

[99] A.S. Krishna, B. Natarajan, A. Gokhale, D.C. Schmidt, N. Wang, and G. Thaker. Ccmperf: a benchmarking tool for corba component model implementations. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 140–147, 2004. doi: 10.1109/RTTAS.2004.1317258. 61

[100] M. Lammie, P. Brenner, and D. Thain. Scheduling grid workloads on multicore clusters to minimize energy and maximize performance. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 145–152, Oct 2009. doi: 10.1109/GRID.2009.5353071. 51

[101] Young Lee and Albert Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, pages 1–13, 2010. 43, 51, 77

[102] Young Choon Lee and Albert Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 92–99, 2009. 43

[103] Young Choon Lee and A.Y. Zomaya. Energy conscious scheduling for distributed computing systems under different operating conditions. *Parallel and Distributed Systems, IEEE Transactions on*, 22(8):1374–1381, Aug 2011. ISSN 1045-9219. doi: 10.1109/TPDS.2010.208. 12, 51

[104] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. Enacloud: An energy-saving application live placement approach for cloud computing environments. In *IEEE CLOUD'09*, pages 17–24, Bangalore, India, September 2009. IEEE. 13

[105] Xiaofei Liao, Liting Hu, and Hai Jin. Energy optimization schemes in cluster withÊvirtual machines. *Cluster Computing*, 13(2):113–126, 2010. ISSN 1386-7857. doi: 10.1007/s10586-009-0110-2. URL http://dx.doi.org/10.1007/s10586-009-0110-2. 55

[106] Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC06, New York, NY, USA, 2006. ACM. ISBN 0-7695-2700-0. doi: 10.1145/1188455.1188567. URL http://doi.acm.org/10.1145/1188455.1188567. 12

[107] Peder Lindberg, James Leingang, Daniel Lysaker, Samee Ullah Khan, and Juan 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*, 53, 2010. 42

[108] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111, Jun 1988. doi: 10.1109/DCS.1988.12507. 50, 55

[109] H. Liu, A. Abraham, and A. E Hassanien. Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. *Future Gener. Comput. Syst.*, 26: 1336–1343, 2010. 16

[110] Jose Luis Lucas-Simarro, Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. Scheduling strategies for optimal service deployment across multiple clouds. *Future Generation Computer Systems*, 29(6):1431 – 1441, 2013. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2012.01.007. URL http://www.sciencedirect.com/science/article/pii/S0167739X12000192. Including Special sections: High Performance Computing in the Cloud; Resource Discovery Mechanisms for P2P Systems. 12

[111] Ping Luo, Kevin Lü, and Zhongzhi Shi. A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.*, 67:695–714, 2007. 11, 12, 16, 17, 21, 22, 23, 27, 29, 30, 44, 119

[112] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the Eighth Heterogeneous Computing Workshop*, HCW '99, pages 30–44, Washington, DC, USA, 1999. ISBN 0-7695-0107-9. 11, 16

[113] P. Marshall, K. Keahey, and T. Freeman. Improving utilization of infrastructure clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 205–214, May 2011. doi: 10.1109/CCGrid.2011.56. 14

[114] Joseph McLean. Distributed.Net and United Devices Join Forces. http://www.distributed.net/images/d/d2/20001127_-_PR_-_United_Devices_Alliance.pdf, 2000. [Online; accessed 25-September-2012]. 50

[115] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: eliminating server idle power. *SIGPLAN Not.*, 44:205–216, 2009. 43

[116] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011. 59

[117] Racem Mellouli, Cherif Sadfi, Chengbin Chu, and Imed Kacem. Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. *European Journal of Operational Research*, 197(3):1150 – 1165, 2009. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2008.03.043. URL http://www.sciencedirect.com/science/article/pii/S037722170800307X. 3

[118] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E. G. Talbi, A. Y. Zomaya, and D. Tuyttens. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *J. Parallel Distrib. Comput.*, 71(11):1497–1508, November 2011. ISSN 0743-7315. doi: 10.1016/j.jpdc.2011.04.007. URL http://dx.doi.org/10.1016/j.jpdc.2011.04.007. 12, 51

[119] Microsoft. Windows Azure. http://www.windowsazure.com/, 2013. [Online; accessed 21-January-2013]. 67

[120] E. U. Munir, J. Li, S. Shi, Z. Zou, and Q. Rasool. A performance study of task scheduling heuristics in hc environment. In Hoai An Le Thi, Pascal Bouvry, and Tao Pham Dinh, editors, *Modelling, Computation and Optimization in Information Systems and Management Sciences*, volume 14 of *Communications in Computer and Information Science*, pages 214–223. Springer Berlin Heidelberg, 2008. 11, 16

[121] Aziz Murtazaev and Sangyoon. Oh. Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing. *IETE Technical Review*, 28(3): 212–231, 2011. doi: 10.4103/0256-4602.81230. 13

[122] Sergio Nesmachnow and Mauro Canabé. Gpu implementations of scheduling heuristics for heterogeneous computing environments. In *Proceedings of the XVII Congreso Argentino de Ciencias de la Computación*, pages 1563–1570, 2011. 12

[123] Sergio Nesmachnow, Enrique Alba, and Héctor Cancela. Scheduling in heterogeneous computing and grid environments using a parallel chc evolutionary algorithm. *Computational Intelligence*, 28(2):131–155, 2012. 16

[124] Sergio Nesmachnow, Héctor Cancela, and Enrique Alba. A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. *Appl. Soft Comput.*, 12 (2):626–639, 2012. 16

[125] B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*, pages 463–489, Los Alamitos, CA, USA, 1994. IEEE CS Press. 16

[126] William D. Norcott. IOzone Filesystem Benchmark. http://www.iozone.org/, 2006. [Online; accessed 01-May-2013]. 61

[127] A.-C. Orgerie, L. Lefevre, and J.-P. Gelas. Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, pages 171–178, Dec 2008. doi: 10.1109/ICPADS.2008.97. 51

[128] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, March 2014. ISSN 0360-0300. doi: 10.1145/2532637. URL http://doi.acm.org.proxy.bnl.lu/10.1145/2532637. 3

[129] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder. Heuristics for vector bin packing, 2011. URL http://research.microsoft.com/pubs/147927/VBPackingESA11.pdf. [Online; accessed 20-July-2013]. 84

[130] J.E. Pecero, P. Bouvry, H.J.F. Huacuja, and S.U. Khan. A multi-objective grasp algorithm for joint optimization of energy consumption and schedule length of precedence-constrained applications. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 510 –517, dec. 2011. doi: 10.1109/DASC.2011.97. 12, 51

[CCSA2013] JohnatanE. Pecero, CesarO. Diaz, Harold Castro, Mario Villamizar, German Sotelo, and Pascal Bouvry. Energy savings on a cloud-based opportunistic infrastructure. In AlessioR. Lomuscio, Surya Nepal, Fabio Patrizi, Boualem Benatallah, and Ivona Brandic, editors, *Service-Oriented Computing - ICSOC 2013 Workshops*, volume 8377 of *Lecture Notes in Computer Science*, pages 366–378. Springer International Publishing, 2014. ISBN 978-3-319-06858-9. doi: 10.1007/978-3-319-06859-6-32. URL http://dx.doi.org/10.1007/978-3-319-06859-6-32. 14, 57

[132] Frédéric Pinel, Johnatan E. Pecero, Pascal Bouvry, and Samee Ullah Khan. A review on task performance prediction in multi-core based systems. In *CIT*, pages 615–620. IEEE Computer Society, 2011. ISBN 978-0-7695-4388-8. 18

[133] Frederic Pinel, Bernabe Dorronsoro, Johnatan E. Pecero, Pascal Bouvry, and Samee U. Khan. A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids. *Cluster Computing*, pages 1–13, 2012. ISSN 1386-7857. doi: 10.1007/s10586-012-0207-x. 16

[134] L. Ponciano and F. Brasileiro. On the impact of energy-saving strategies in opportunistic grids. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 282–289, Oct 2010. doi: 10.1109/GRID.2010.5698003. 14, 55

[135] L. Ponciano and F. Brasileiro. On the impact of energy-saving strategies in opportunistic grids. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 282–289, Oct 2010. doi: 10.1109/GRID.2010.5698003. 51

[136] M. Pretorius, M. Ghassemian, and C. Ierotheou. An investigation into energy efficiency of data centre virtualisation. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on*, pages 157–163, Nov 2010. doi: 10.1109/3PGCIC.2010.28. 55

[137] Energy-Star Program. Report to congress on server and data center energy efficiency: Public law 109- 431, 2007. URL http://www.energystar.gov/ia/partners/prod_development/downloads\/EPA_Datacenter_Report_2

[138] Ariel Quezada-Pina, Andrei Tchernykh, José Luis González-García, Adan Hirales-Carbajal, Juan Manuel Ramírez-Alcaraz, Uwe Schwiegelshohn, Ramin Yahyapour, and Vanessa Miranda-López. Adaptive parallel job scheduling with resource admissible allocation on two-level hierarchical grids. *Future Generation Comp. Syst.*, 28(7):965–976, 2012. 17

[139] Ioan Raicu, Ian Foster, Mike Wilde, Zhao Zhang, Kamil Iskra, Peter Beckman, Yong Zhao, Alex Szalay, Alok Choudhary, Philip Little, Christopher Moretti, Amitabh Chaudhary, and Douglas Thain. Middleware support for many-task computing. *Cluster Computing*, 13(3):291–314, September 2010. ISSN 1386-7857. doi: 10.1007/s10586-010-0132-9. 16

[140] J.M. Ramírez-Alcaraz, A. Tchernykh, R. Yahyapour, U. Schwiegelshohn, A. Quezada-Pina, J.L. González-García, and A. Hirales-Carbajal. Job allocation strategies with user run time estimates for online scheduling in hierarchical grids. *J. Grid Comput.*, 9 (1):95–116, 2011. ISSN 1570-7873. 16

[141] John J. Rehr, Joshua J. Kas, Fernando D. Vila, Micah P. Prange, and Kevin Jorissen. Parameter-free calculations of x-ray spectra with feff9". *Phys. Chem. Chem. Phys.*, 12:5503–5513, 2010. doi: 10.1039/B926434E. URL http://dx.doi.org/10.1039/B926434E. 67

[142] Mersenne Research. GIMPS Great Internet Mersenne Prime Search. http://www.mersenne.org, 1996. [Online; accessed 20-September-2012]. 50

[143] Eduardo Rosales, Harold Castro, and Mario Villamizar. Unacloud: Opportunistic cloud computing infrastructure as a service. In *Cloud Computing 2011: The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 187–194. IARIA, 2011. ISBN 978-1-61208-153-3. 57, 59, 66, 78

[144] SabalcoreComputing. Sabalcore Home. http://www.sabalcore.com/, 2013. [Online; accessed 21-January-2013]. 67

[145] Luis F. G. Sarmenta, Sandra J. V. Chua, Paul Echevarria, Jose M. Mendoza, Rene-Russelle Santos, Stanley Tan, and Richard Lozada. Bayanihan computing .net: Grid computing with xml web services. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '02, pages 434–, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1582-7. URL http://dl.acm.org/citation.cfm?id=872748.873229. 50

[146] B. Schott and A. Emmen. Green methodologies in desktop-grid. In *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pages 671–676, Oct 2010. 55

[147] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour. Online scheduling in grids. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–10, 2008. doi: 10.1109/IPDPS.2008.4536273. 16

[148] Mohsen Sharifi, Hadi Salimi, and Mahsa Najafzadeh. Power-efficient distributed scheduling of virtual machines using workload-aware consolidation techniques. *The Journal of Supercomputing*, 61(1):46–66, 2011. ISSN 0920-8542. doi: 10.1007/s11227-011-0658-5. 51, 77

[149] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM J. Comput.*, 24(6):1313–1331, December 1995. ISSN 0097-5397. 17

[150] John F. Shoch and Jon A. Hupp. The "wormÓ programs-early experience with a distributed computation. *Commun. ACM*, 25(3):172–180, March 1982. ISSN 0001-0782. doi: 10.1145/358453.358455. URL http://doi.acm.org/10.1145/358453.358455. 50

[151] Oliver Sinnen, Leonel Augusto Sousa, and Frode Eika Sandnes. Toward a realistic task scheduling model. *IEEE Transactions on Parallel and Distributed Systems*, 17(3):263–275, 2006. ISSN 1045-9219. doi: http://doi.ieeecomputersociety.org/10.1109/TPDS.2006.40. 11

[152] Mark Snir, Steve Otto, Steven Huss-Lederman, David Walker, and jack Dongarra. *MPI The Complete Reference*. The MIT Press, Cambridge, MA, USA, 1998. ISBN 0-262-69215-5. 53

[153] Mark Stillwell, David Schanzenbach, Frédéric Vivien, and Henri Casanova. Resource allocation algorithms for virtualized service hosting platforms. *J. Parallel Distrib. Comput.*, 70(9):962–974, September 2010. 84

[154] V. S. Sunderam. Pvm: A framework for parallel distributed computing. *Concurrency: Pract. Exper.*, 2(4):315–339, November 1990. ISSN 1040-3108. doi: 10.1002/cpe.4330020404. URL http://dx.doi.org/10.1002/cpe.4330020404. 53

[155] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001. ISBN 0130888931. 16

[156] Andrei Tchernykh, Johnatan E. Pecero, Aritz Barrondo, and Elisa Schaeffer. Adaptive energy efficient scheduling in peer-to-peer desktop grids. *Future Generation Computer Systems*, (0):–, 2013. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2013.07.011. URL http://www.sciencedirect.com/science/article/pii/S0167739X13001568. 17

[157] Top500. Top 500 supercomputer sites. http://www.top500.org/, 2010. 49

[158] GiorgioLuigi Valentini, Walter Lassonde, SameeUllah Khan, Nasro Min-Allah, SajjadA. Madani, Juan Li, Limin Zhang, Lizhe Wang, Nasir Ghani, Joanna Kolodziej, Hongxiang Li, AlbertY. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Fredric Pinel, JohnatanE. Pecero, Dzmitry Kliazovich, and Pascal Bouvry. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1):3–15, 2013. ISSN 1386-7857. doi: 10.1007/s10586-011-0171-x. URL http://dx.doi.org/10.1007/s10586-011-0171-x. 12, 51

[159] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990. ISSN 0001-0782. doi: 10.1145/79173.79181. URL http://doi.acm.org.proxy.bnl.lu/10.1145/79173.79181. 53

[160] Christian Vecchiola, Xingchen Chu, and Rajkumar Buyya. Aneka: A software platform for .net-based cloud computing. *CoRR*, abs/0907.4622, 2009. 67

[161] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 243–264, New York, NY, USA, December 2008. Springer-Verlag New York, Inc. ISBN 3-540-89855-7. 13

[162] M. J. Villamizar Cano, H. E. Castro Barrera, D. Mendez Lopez, S. Restrepo Restrepo, and Rodriguez Rojas L. M. Bio-unagrid: Easing bioinformatics workflow execution using loni pipeline and a virtual desktop grid. In *Proceedings of the 3th International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies*, pages 4–10, Washington, DC, USA, 2011. XPS Xpert Publishing Services. ISBN 978-1-61208-137-3. URL http://www.iaria.org/conferences2011/BIOTECHNO11.html. 50

[163] Lizhe Wang, Samee U. Khan, Dan Chen, Joanna Kolodziej, Rajiv Ranjan, Cheng-zhong Xu, and Albert Zomaya. Energy-aware parallel task scheduling in a cluster. *Future Generation Computer Systems*, 29(7):1661 – 1670, 2013. ISSN 0167-739X. 12, 79

[164] Brian Ward. *The book of VMware: The complete Guide to VMware Workstation*. William Pollock, San Francisco, CA, USA, 1st edition, 2002. ISBN 1-886411-72-7. 54

[165] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. *USENIX SYMP. OPERATING*, pages 13–23, 1994. 12, 51

[166] Min-You Wu and Wei Shu. A high-performance mapping algorithm for heterogeneous computing systems. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, IPDPS'01, pages 74–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0990-8. 16

[167] Min-You Wu, Wei Shu, and Hong Zhang. Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In *Proceedings of the 9th Heterogeneous Computing Workshop*, HCW '00, pages 375–385, Washington, DC, USA, 2000. IEEE Computer Society. 11

[168] F. Xhafa and A. Abraham. Computational models and heuristic methods for grid scheduling problems. *Future Gener. Comput. Syst.*, 26:608–621, April 2010. 16

[169] A. YarKhan and J. Dongarra. Experiments with scheduling using simulated annealing in a grid environment. In *Proceedings of the Third Int Workshop on Grid Computing*, GRID '02, pages 232–242, London, UK, 2002. Springer-Verlag. 16

[170] Fan Zhang, Junwei Cao, Keqin Li, Samee U. Khan, and Kai Hwang. Multi-objective scheduling of many tasks in cloud platforms. *Future Generation Computer Systems*, (0): –, 2013. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2013.09.006. URL http://www.sciencedirect.com/science/article/pii/S0167739X13001854. 12

Publications

[ClusC2012] Harold Castro, Mario Villamizar, German Sotelo, Cesar O. Diaz, Johnatan E. Pecero and Pascal Bouvry. *Green flexible opportunistic computing with task consolidation and virtualization*. Journal of Cluster Computing. V16, n3. Springer US. issn:1386-7857. Pages 545-557. 2012.

[SUPE2013] Cesar O. Diaz, Johnatan E. Pecero, and Pascal Bouvry. *Scalable, low complexity, and fast greedy scheduling heuristics for highly heterogeneous distributed computing systems*. The Journal of Supercomputing. Springer US. issn:0920-8542. Pages 1-17. 2013. 23

[FGCS2014] Cesar O. Diaz, Johnatan E. Pecero, German Sotelo, Harold Castro, Andrei Tchernykh, and Pascal Bouvry. *Energy-Aware Resource Provisioning for HPC Applications on an Opportunistic PaaS Cloud*. Special Issue on New Trends in Data-Aware Scheduling and Resource Provisioning in Modern HPC Systems. Future Generation Computer Systems, FGCS. **Invited extended version**.

[COR2014] Cesar O. Diaz, Johnatan E. Pecero, Pascal Bouvry, and Andrei Tchernykh. *Fast and Scalable Heuristics for Independent Tasks Scheduling on Heterogeneous Computing Platforms*. Computer & Operations Research. ELSEVIER. **In Progress**.

[SCAL13] Harold Castro, Mario Villamizar, German Sotelo, Cesar O. Diaz, Johnatan E. Pecero, Pascal Bouvry and Samee U. Khan. *GFOG: Green and Flexible Opportunistic Grids*. Eds. Samee U. Khan, Albert Y. Zomaya and Lizhe Wang. **Scalable Computing and Communications, Theory and Practice**. Wiley&Sons. isbn:9781118162651. Pages 365-386. 2013

[ICA3PP2013] German A. Sotelo, Cesar O. Diaz, Mario Villamizar, Harold Castro, Johnatan E. Pecero, and Pascal Bouvry. *Building Platform as a Service for High Performance Computing over an Opportunistic Cloud Computing*. Eds. Kolodziej, Joanna and Martino, Beniamino and Talia, Domenico and Xiong, Kaiqi **Algorithms and Architectures for Parallel Processing**. Springer International Publishing Lecture Notes in Computer Science, LNCS V. 8285. isbn:978-3-319-03858-2. Pages 380-389. 2013 57

[CCSA2013] Johnatan E. Pecero, Cesar O. Diaz, Harold Castro, Mario Villamizar, German Sotelo, and Pascal Bouvry. *Energy Savings on a Cloud-Based Opportunistic Infrastructure* Eds. Alessio R. Lomuscio, Surya Nepal, Fabio Patrizi, Boualem Benatallah, and Ivona Brandic, **Service-Oriented Computing - ICSOC 2013 Workshops**. Springer

International Publishing Lecture Notes in Computer Science, LNCS V. 8377. isbn:978-3-319-06858-9. Pages 366-378. 2014 14, 57

[IAIT2012] Mateusz Guzek, Cesar Diaz, Johnatan E. Pecero, Pascal Bouvry, and Albert Y. Zomaya. *Impact of Voltage Levels Number for Energy-Aware Bi-objective DAG Scheduling for Multi-processors Systems*. Eds. Borworn Papasratorn, Nipon Charoenkitkarn, Kittichai Lavangnananda, Wichian Chutimaskul, and Vajirasak Vanijja. **Advances in Information Technology**. Springer Berlin Heidelberg. Communications in Computer and Information Science. isbn:978-3-642-35075-7. Pages 70-80.2012

[CCGrid2013] Cesar O. Diaz, Harold Castro, Mario Villamizar, Johnatan E. Pecero and Pascal Bouvry. *Energy-aware VM Allocation on an Opportunistic Cloud Infrastructure*. In proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). Pages 663-670. Delft, Netherlands. 2013 57

[CIT11] Harold Castro, German Sotelo, Cesar O. Diaz and Pascal Bouvry. *Green Flexible Opportunistic Computing with Virtualization*. In proceedings of the IEEE 11th International Conference on Computer and Information Technology. Pages 629-634. Paphos, Cyprus. 2011

[SCAL2011] Cesar O. Diaz, Mateusz Guzek, Johnatan E. Pecero and Pascal Bouvry. *Scalable and Energy-Efficient Scheduling Techniques for Large-Scale Systems*. In proceedings of the IEEE 11th International Conference on Computer and Information Technology. Pages 641-647. Paphos, Cyprus. 2011

[OPTIM2011] Cesar O. Diaz, Mateusz Guzek, Johnatan E. Pecero, Gregoire Danoy, Pascal Bouvry and Samee U. Khan. *Energy-aware fast scheduling heuristics in heterogeneous computing systems*. In proceedings of the International Conference on High Performance Computing and Simulation (HPCS). Pages 478-484. Istanbul, Turkey. 2011 11, 17, 18, 119

[CCGrid2014] Cesar O. Diaz, Johnatan E. Pecero, Pascal Bouvry, German Sotelo, Mario Villamizar and Harold Castro. *Performance Evaluation of an IaaS Opportunistic Cloud Computing* In proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2014. Pages 546 - 547. Chicago, IL, USA. 2014 59

[CCGrid2013-Poster] Cesar O. Diaz, Harold Castro, Mario Villamizar, Johnatan E. Pecero, and Pascal Bouvry. 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2013. May, 2013. Delft, Netherlands. Available in http://www.pds.ewi.tudelft.nl/fileadmin/pds/conferences/ccgrid2013/files/documents/poster

List of Figures

List of Tables

LIST OF TABLES

List of Algorithms