HAL
archives-ouvertes.fr

Multi-criteria and satisfaction oriented scheduling for hybrid distributed computing infrastructures

Mircea Moca, Cristian Litan, Gheorghe Silaghi, Gilles Fedak

▶ **To cite this version:**

Mircea Moca, Cristian Litan, Gheorghe Silaghi, Gilles Fedak. Multi-criteria and satisfaction oriented scheduling for hybrid distributed computing infrastructures. Future Generation Computer Systems, Elsevier, 2016, 55, <10.1016/j.future.2015.03.022>. <hal-01239218>

HAL Id: hal-01239218
https://hal.inria.fr/hal-01239218

Submitted on 7 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

*Manuscript
Click here to view linked References

Multi-Criteria and Satisfaction Oriented Scheduling for Hybrid Distributed Computing Infrastructures

Mircea Moca[a], Cristian Litan[a], Gheorghe Cosmin Silaghi[a], Gilles Fedak[b],

[a] ***Babeș-Bolyai University***, *Cluj-Napoca, România*
[b] ***INRIA***, *Université de Lyon, France*

Abstract

Assembling and simultaneously using different types of distributed computing infrastructures (DCI) like Grids and Clouds is an increasingly common situation. Because infrastructures are characterized by different attributes such as price, performance, trust, greenness, the task scheduling problem becomes more complex and challenging. In this paper we present the design for a fault-tolerant and trust-aware scheduler, which allows to execute Bag-of-Tasks applications on elastic and hybrid DCI, following user-defined scheduling strategies. Our approach, named *Promethee scheduler*, combines a pull-based scheduler with multi-criteria Promethee decision making algorithm. Because multi-criteria scheduling leads to the multiplication of the possible scheduling strategies, we propose SOFT, a methodology that allows to find the optimal scheduling strategies given a set of application requirements. The validation of this method is performed with a simulator that fully implements the Promethee scheduler and recreates an hybrid DCI environment including Internet Desktop Grid, Cloud and Best Effort Grid based on real failure traces. A set of experiments shows that the Promethee scheduler is able to maximize user satisfaction expressed accordingly to three distinct criteria: price, expected completion time and trust, while maximizing the infrastructure useful employment from the resources owner point of view. Finally, we present an optimization which bounds the computation time of the Promethee algorithm, making realistic the possible integration of the scheduler to a wide range of resource management software.

Email address: mircea.moca@econ.ubbcluj.ro, cristian.litan@econ.ubbcluj.ro, gheorghe.silaghi@econ.ubbcluj.ro, gilles.fedak@inria.fr (Gilles Fedak)

Keywords:
Elastic computing infrastructures, Hybrid distributed computing infrastructures, Pull-based scheduling, Multi-criteria scheduling, Promethee scheduling

1. Introduction

The requirements of distributed computing applications in terms of processing and storing capacities is continuously increasing, pushed by the gigantic deluge of large data volume to process. Nowadays, scientific communities and industrial companies can choose among a large variety of distributed computing infrastructures (DCI) to execute their applications. Examples of such infrastructures are Desktop Grids or Volunteer Computing systems [8] which can gather a huge number of volunteer PCs at almost no cost, Grids [12] which assemble large number of distributed clusters and more recently, clouds [4] which can be accessed remotely, following a pay-as-you-go pricing model. All these infrastructures have very different characteristics in terms of computing capacity, cost, reliability, consumed power efficiency and more. Hence, combining these infrastructures in such a way that meets users' and applications' requirements raises significant scheduling challenges.

The first challenge concerns the design of the resource management middleware which allows the assemblage of hybrid DCIs. The difficulty relies in the number of desirable high level features that the middleware has to provide in order to cope with: *i*) distributed infrastructures that have various usage paradigms (reservation, on-demand, queue), and *ii*) computing resources that are heterogeneous, volatile, unreliable and sometimes not trustee. An architecture that has been proved to be efficient to gather hybrid and elastic infrastructures is the joint use of a *pull-based scheduler* with pilot jobs [37, 17, 35, 6]. The pull-based scheduler, often used in Desktop Grid computing systems [1, 10], relies on the principle that the computing resources pull tasks from a centralized scheduler. Pilot jobs consist in resource acquisition by the Promethee scheduler and the deployment on them of agents with direct access to the central pull-based scheduler, so that Promethee can work with the resources directly, rather than going through local job schedulers. This approach exhibits several desirable properties, such as scalability, fault resilience, ease of deployment and ability to cope with elastic infrastructures, these being the reasons why the architecture of *Promethee scheduler* that we propose in this paper, follows this principle.

The second challenge is to design task scheduling that are capable of efficiently using hybrid DCIs, and in particular, that takes into account the differences between the infrastructures. In particular, the drawback of a pull-scheduler is that it flattens the hybrid infrastructures and tends to consider all computing resources on an equal basis. Our earlier results [26, 27] proved that a *multi-criteria scheduling* method based on the Promethee decision model [11] can make a pull-based scheduler able to implement scheduling strategies aware of the computing resources characteristics. However, in this initial work, we tested the method on single infrastructure type at a time, without considering hybrid computing infrastructures, and we evaluated the method against two criteria: expected completion time (`ECT`) and usage price. In this paper, we propose the following extensions to the Promethee scheduler: *i*) we add a third criteria called *Expected Error Impact* (`EEI`), that reflects the confidence that a host returns correct results, *ii*) we evaluate the Promethee scheduler on hybrid environments, *iii*) we leverage the tunability of the Promethee scheduler so that applications developers can empirically configure the scheduler to put more emphasize on criteria that are important from their own perspective.

The third challenge regards the design of a new scheduling approach that maximizes satisfaction of both users and resource owners. In general, end users request to run their tasks quicker and at the cheapest costs, opposed to the infrastructure owners which need to capitalize their assets and minimize the operational costs. Thus, an overall scheduling approach should allow the resource owners to keep their business profitable and meantime, increase the end user satisfaction after the interaction with the global computing system.

The Promethee scheduler allows users to provide their own scheduling strategies in order to meet their applications requirements by configuring the relative importance of each criteria. However such configurable multi-criteria schedulers have two strong limitations: *i*) there is no guaranty that the user preferences expressed when configuring the scheduler actually translates in an execution that follows the favored criteria, and *ii*) the number of possible scheduling strategies explodes with the number of criteria and the number of application profiles, rapidly leading to an intractable situation by the user. We propose *Satisfaction Oriented FilTering* (SOFT), a new methodology that explores all the scheduling strategies provided by a Promethee multi-criteria scheduler to filter and select the most favorable ones according to the user execution profiles and the optimization of the infrastructure usage.

SOFT also allows to select a default scheduling strategy so that the scheduler attains a high and at the same time stable level of user satisfaction, regardless the diversity of user satisfaction profiles.

In this paper, we introduce the design of the fault-tolerant and trust-aware Promethee scheduler, which allows to execute Bag-of-Tasks applications on elastic and hybrid DCI, following user-defined scheduling strategies. We thoroughly present the algorithms of the multi-criteria decision making and the SOFT methodology. Finally, we extensively evaluate the Promethee scheduler using a simulator that recreates a combination of hybrid, elastic and unreliable environment containing Internet Desktop Grid, public Cloud using Spot Instance and Best Effort Grid. Simulation results not only show the effectiveness of the Promethee scheduler but also its ability to meet user application requirements. We also propose an optimized implementation of the Promethee algorithms and perform real world experiments to validate the approach.

The remainder of the paper is organized as follows. In section 2 we give the background for our work and define the key concepts, in section 3 we explain our scheduling approach and define the performance evaluation metrics. In section 4 we define SOFT, the methodology for optimal scheduling strategies selection. Then we present the architecture of the implemented experimental system in section 5. In section 6 we describe the experimental data, the setup and present the obtained results and findings. In section 7 we discuss related work and finally section 8 gives the concluding remarks and observations on this work.

2. Background

This section describes the multi-criteria scheduling on hybrid DCIs problem that we address in this work and defines the key concepts used in our discussion.

2.1. Description of the scheduling problem

In the considered context users submit their applications for execution on a system that aggregates the computing resources from at least three types of DCI: Internet Desktop Grids (IDG), Best Effort Grids (BEG) and Cloud. Each computing resource from the above mentioned infrastructures have different characteristics in terms of computing capacity, reliability, cost,

consumed power efficiency, and trust. For instance, Internet volunteer Desktop PCs could be considered as free of charge but insecure and unreliable, while a Cloud resource can be costly but far more secure and reliable.

Users usually expect good execution performance but they are also concerned about other issues like cost, confidence and environmental footprint of the infrastructure. Thus, a relevant concern for task scheduling is to attain the *best* usage of the infrastructures in order to meet users' expectations and, **at the same time**, insure a convenient capitalization of the resources for their owners.

2.2. Key concepts

Users submit bag of work-units to a centralized scheduler and expect (after a while) the corresponding results. For each work-unit the scheduler creates at least one task and inserts it into a BoT (Bag of Task). During the execution the scheduler aims at emptying this BoT by scheduling tasks to hosts belonging to various types of computing infrastructure.

We use a **pull-based** scheduling strategy. Hence our scheduler is a centralized component (master) based on the **pull communication model** for the interaction with hosts (workers). The reason for this design choice is the requirement for **elasticity** and **adaptability** to structure disruptions that characterize DCIs like IDG and BEG. This model allows a complete independence of all system components [23]. The pull model allows workers to have the contact initiative, which overcomes the real issue of connecting to volunteers residing behind firewalls [20] or other security components.

When a host h_{i_h} becomes available (either because it (re-)joins the system or after completing a task) it contacts the scheduler in order to receive a new task. This approach is efficient [36] since in IDGs, hosts contact the server quite seldom. More, if embedded into a real middleware, such a scheduling component becomes more scalable, since it does not keep track of the workers' state. Due to its advantages, many Desktop Grid systems (e.g BOINC[1], XtremWeb [10]), and Grid PilotJob framework (e.g FALKON [34], DIRAC[6]) and others rely on the pull model for the master-worker communication.

Due to the use of the pull model, the structure of the system and the scheduling process are driven by the behavior of participating hosts. As discussed above, there are two cases when a host pulls work: either when it (re-)joins the system, or after completing a task. We consider that a host may leave the system without preventing the scheduler. Such disruptions

may degrade the execution performance of a BoT and they are more likely to occur in IDG and BEG infrastructures.

Figure 1 depicts an overview of the considered scheduling context. As our discussion is focused on the scheduling method, we omitted to detail additional middleware components and layers which naturally occur in real systems. For example, by $Interface$ we designate specific mechanisms that allow hosts from different types of infrastructure to communicate with the scheduler. The details of this communication may differ from an infrastructure type to another. For instance it may be directly from a host to scheduler in IDG or via a mediator component for Cloud, as used in [6]. However, such mechanisms do not fall within our research scope.

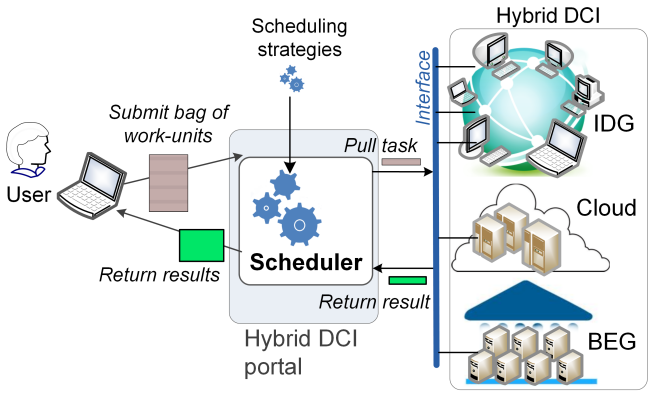


Figure 1: Overview on the Promethee scheduler architecture and its context.

Formalisms:

- A **work unit**, denoted with w_{i_w}, represents a unit of work that must be processed and for which an end user application expects a result from the scheduler. A work unit is characterized by:
  - a unique identifier $\texttt{ID}^W(w_{i_w})$,
  - the number of instructions $\texttt{NOI}(w_{i_w})$,
  - the arrival time $\texttt{TMA}(w_{i_w})$. This is the time when the scheduler received the work unit from a user application.
  - the completion time $\texttt{TMC}(w_{i_w})$. This is the time when the scheduler completes a work unit by receiving a correct result for a corresponding task.

- A task t_{i_t} is an *instantiation* $t_{i_w,r}$ of the work unit w_{i_w}, $r \in \mathbb{N}^*$. During the execution, $t_{i_w,r}$ is the r^{th} instance (replica) created for w_{i_w}, in the aim of completing it. A task t_{i_t} is characterized by:
  - a unique identifier $\texttt{ID}^T(t_{i_t})$,
  - the number of instructions $\texttt{NOI}(t_{i_t})$ (this value is inherited from its corresponding work-unit),
  - the creation time $\texttt{TC}(t_{i_t})$. This is the time when the scheduler created the task for an uncomplete work-unit,
  - the schedule time $\texttt{TMS}(t_{i_t})$,
  - the completion time $\texttt{TMC}(t_{i_t})$.
- A **pulling host** h_{i_h} *occasionally* contacts the scheduler to pull a task and is characterized by:
  - a unique identifier $\texttt{ID}^H(h_{i_h})$,
  - $\texttt{ID}^{DCI}$, a label indicating the identity of the computing infrastructure to which it belongs,
  - the computing capacity $\texttt{CPU}(h_{i_h})$, expressed in number of processed instructions per second. For the sake of simplicity, in this work only CPU capacity is considered and not other types of resource such as memory, bandwidth etc.
  - $\texttt{PRICE}(h_{i_h})$, which is the price (in monetary units) per second, *eventually* charged by host for processing tasks.

  After a host completes the execution of a task it contacts the scheduler to return the computed result and pull a new task.
- A set of scheduling **criteria** $\mathcal{C} = \{c_{i_c}, 1 \leq i_c \leq N_c\}$, provided either by the end user or by the infrastructure owner. For instance, a criterion c_{i_c} might be the expected completion time, the price charged for completing a task on a particular infrastructure or the expected error impact. Each criterion c_{i_c} will have assigned an importance weight $\omega_{i_c} > 0$, such as $\sum \omega_{i_c} = 1$.

The scheduler holds a set of work units $W = \{w_{i_w}, i_w \in \mathbb{N}\}$, and a set of tasks $T = \{t_{i_t}, i_t \in \mathbb{N}\}$. When the scheduler receives a work unit w_{i_w}, it

inserts into T a new task $t_{s,r}, r = 1$, as previously explained. When a task $t_{s,r}$ is scheduled, the scheduler inserts into T a new replica, $t_{s,r+1}$. This $r+1$ replica will have a lower priority for scheduling whilst the scheduler still waits to receive a result for the $t_{s,r}$ task.

3. The Promethee scheduling method

This section presents our approach of using Promethee[11] for task scheduling and the defined performance evaluation metrics.

3.1. The Promethee algorithm

When a host h pulls work, the scheduler uses the Promethee algorithm to rank the BoT based on the characteristics of the host. Then it selects the best ranked task and schedules it to the respective host.

Promethee[3] is a multi-criteria decision model based on **pairwise comparisons**, which outputs a **ranking** of the tasks. This method considers a set of criteria $C = \{c_{i_c}; i_c \in [1, N_c]\}$ to characterize tasks and a set of importance weights for criteria, $W = \omega_{i_c}(c_{i_c})$, so that $\sum_{i_c=1}^{N_c} \omega_{i_c}(\cdot) = 1$. First, the scheduler builds the matrix $A = \{a_{i_c,i_t}\}$ where each element a_{i_c,i_t} is computed by evaluating task t_{i_t} against criterion c_{i_c}. For instance, if price=2 monetary units/sec. and `CPU`=10 `NOI`/sec., the evaluation against this criterion for task t_1 having `NOI`=100 is 20; similarly, for a task t_2 with `NOI`=300 the evaluation is 60. Matrix A is the input for the Promethee algorithm and characterizes the BoT for a particular pulling host.

Based on the input matrix A, for each criterion c_{i_c}, the algorithm computes a preference matrix $G_{i_c} = \{g_{i1_t,i2_t}\}$. Each value $g_{i1_t,i2_t}$ shows the preference $P \in [0, 1]$ of task t_{i1_t} over t_{i2_t}. This value is computed using a preference function P that takes as input the actual deviation between the evaluations of t_{i1_t} and t_{i2_t} within each criterion c_{i_c}. If the criterion is min/max and the deviation is negative, the preference is $0/P(g_{i1_t,i2_t})$; it becomes $P(g_{i1_t,i2_t})/0$ when the deviation is positive. For short, the preference function brings the deviation value within the $[0, 1]$ interval. Resuming the previous example, if the algorithm is configured with a min target for price, then t_1 is preferred to t_2 with $g_{1,2} = P(40)$; otherwise, if the target is set to max, $g_{1,2} = 0$ since the price of t_1 is smaller than that of t_2.

For each criterion, the algorithm continues with the computation of Φ^+, which shows the power of a task to outrank all other tasks j. The computation aggregates $g_{i1_t,j}$ values (summation per line in matrix G). In a similar

way, Φ^- is computed to express the weakness of a task while being outranked by all other tasks j. The computation aggregates $g_{j,i1_t}$ values (summation per column) in matrix G. For each criterion, a net flow is computed as $\Phi^+ - \Phi^-$. Figure 2 depicts the calculation of the positive flow Φ^+ - summation by line of the preferences and for the negative flow Φ^- - summation by column, for the t_0 task.

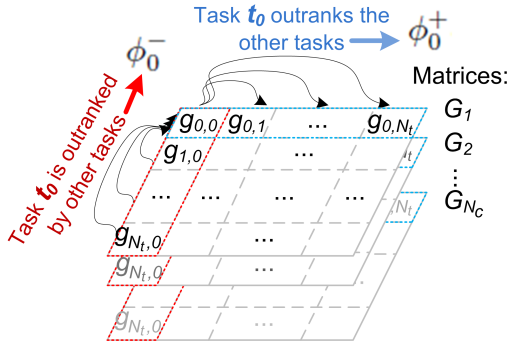


Figure 2: Computing positive and negative outranking flows for task t_0.

Finally the algorithm calculates an aggregated flow as a weighted sum on the net flows, using the importance weights ω assigned to criteria. This final aggregated flow is a list of scores for each task in the BoT. The task with the highest score is scheduled to the pulling host.

We notice that the Promethee algorithm uses a preference function P to calculate a preference among two tasks. The literature [11] suggests several definitions for the preference function, including *Linear*, *Level*, *Gaussian* and others. However, one is free to provide any other definition for this function. In section 6.3.2 we show how to choose an efficient preference function.

3.2. Evaluating tasks against a set of scheduling criteria

Before making a scheduling decision, the scheduler creates the evaluation matrix A, as previously described, by evaluating all tasks against each criterion in the considered set $\mathcal{C}$. In this section we describe the policies for evaluating tasks for a set of three criteria: the expected completion time, price and expected error impact. Although in our work we used three scheduling criteria, one can easily define and add her own new criteria to this scheduling method.

The **Expected Completion Time** (`ECT`) represents an estimation of the time interval needed by a pulling host to complete a particular task. This is calculated by dividing the number of instructions (`NOI`) of a task by the computing capacity (`CPU`) of the host.

The **Price** represents the estimated cost (expressed in monetary units) *eventually* charged by the pulling host for the completion of a task. This is computed as a product between the `ECT` and the price/time unit required by host. In our scheduler, if a host returns the result of a task later than its `ECT` the total cost for the execution of the BoT remains unchanged. By this approach we seek to mimic what happens in a real system where the SLA[1] can include agreed execution time limits and penalties if exceeded.

The **Expected Error Impact** (`EEI`) indicates the estimated impact of scheduling a particular task to the pulling host taking into account its reputation (error proneness) and the size of the task. For a less trusted pulling host the evaluation of a larger task would result into a higher error impact. The idea of using this criterion is to mitigate the behavior of scheduling large tasks to mistrusted hosts. In algorithm 1 we give the calculation steps for the $\texttt{EEI}(t_i)$ evaluation.

In order to create a realistic behavior of the hybrid DCI, in our experimental system we consider that hosts can produce erroneous results or even fail to return a result to the scheduler. In real systems this is either due to hardware/platform issues or due to malicious behavior of the hosts (i.e. in BOINC). Hence we use two parameters to model this behavior in each type of infrastructure: f the fraction of hosts that produce erroneous results and s, the probability with which they manifest this behavior. In order to use this information within the scheduling method we implemented a simple reputation model like the one described by Arenas et al. in [2]. The idea is that the scheduler computes and keeps track of hosts' reputation, based on the quality of the returned results during the execution. We assume that the scheduler has a result verification mechanism to check the validity of the results received from hosts. According to the implemented model, if a host returns an erroneous result for a large task, its reputation will be more severely affected compared to the erroneous result of a small task. Besides, the reputation score is more significantly affected by the quality of the *soonest completions*.

[1]Service Level Agreement

With respect to this rule, our scheduler computes a host's reputation based on a maximum of 5 last returned results. By this, the scheduler takes into account the recent behavior of hosts within the system.

Algorithm 1 The calculation of $\texttt{EEI}(t_i)$

1: Calculate the reputation of the pulling host, $R^{h_{i_h}}$.
2: Calculate the relative utility $\Upsilon(t_i) = \frac{NOI(t_i)}{NOI_{max}}$ $\{$*The utility of task t_i relative to its size*$\}$.
3: Based on $R^{h_{i_h}}$ and $\Upsilon(t_i)$ calculate $\texttt{EEI}(t_i)$ using a policy, like in table 1.

$R^{h_{i_h}}$ \ $\Upsilon(t_i)$	**(0, 0.5]**	**(0.5, 0.7]**	**(0.7, 1]**
(0, 0.5]	$\Upsilon(t_i) \cdot 2$	$\Upsilon(t_i) \cdot 4$	$\Upsilon(t_i) \cdot 8$
(0.5, 0.7]	$\Upsilon(t_i)/2$	$\Upsilon(t_i) \cdot 2$	$\Upsilon(t_i) \cdot 4$
(0.7, 1]	$\Upsilon(t_i)/4$	$\Upsilon(t_i)/2$	$\Upsilon(t_i)$

Table 1: `EEI` calculation scheme.

In table 1 we describe the `EEI` calculation scheme used in our implementation. This can be seen as a risk matrix, where `EEI` indicates the risk of scheduling a task of a certain size to a host with a particular reputation. The values in the table show that:

- the higher the relative task utility, the bigger the `EEI` value (for a particular reputation score) and
- the higher the host reputation, the smaller the `EEI` value (for a particular task relative utility).

3.3. Performance evaluation metrics

To evaluate the performance of the scheduling approach we use the following metrics: makespan, cost, execution efficiency, the user satisfaction Θ and the infrastructure utilization efficiency $\mathcal{E}$.

We define **makespan** (M) like Maheswaran et al. in [24], as the duration in time needed for the system to complete the execution of all work units belonging to the same application execution. This is computed by scheduler as the difference between the timestamp of the result received for the last

uncompleted task and the timestamp of the first task schedule. In all experiments the makespan reflects the execution at hosts plus the scheduling decision making time but not the network communication between scheduler and hosts. We consider that this value has no relevance for the evaluation of the scheduling method in the scheduling context defined in section 2.2.

The **cost** (C) indicates the total cost in monetary units accumulated for an entire application execution. Each result returned by a host to scheduler **may** increase the total cost of the whole execution. Exceptions from the cost cumulation are when a host returns the result after the estimated completion time. In a real system, this situation could be when a host from a Cloud infrastructure violates the SLA agreement.

The execution **efficiency** (E) is calculated for an application execution as a ratio between the number of instructions from successfully executed tasks and the number of instructions from the total scheduled tasks. Hosts that are error prone or those who fail, lead to a smaller execution efficiency since the tasks they received must be rescheduled.

In order to facilitate the analysis of the scheduling performance we add up the metrics defined from user perspective into the aggregated objective function Θ. This indicates the overall user satisfaction and it is defined in eq. 1. Given that, based on a set of parameters, a system designer can configure the scheduler in different ways using a set of strategies $\mathcal{S}$, Θ shows the relative satisfaction perceived by user U_{i_u} after the completion of her application, for a specific strategy s from $\mathcal{S}$. For instance, a high γ_m and a low γ_c shows that the user is more satisfied by a faster but costly execution, while a low γ_m and a high γ_c indicate that the user is more satisfied by a slower but cheaper execution of their application.

$$\Theta_s^{U_{i_u}}(M, C, E) = \gamma_m \cdot \frac{M_{max} - M_v^{U_{i_u}}}{M_{max} - M_{min}} + \gamma_c \cdot \frac{C_{max} - C_v^{U_{i_u}}}{C_{max} - C_{min}} + \gamma_e \cdot (1 - \frac{E_{max} - E_v^{U_{i_u}}}{E_{max} - E_{min}}) \quad (1)$$

where

- $M_{max}, C_{max}, E_{max}$ (and $M_{min}, C_{min}, E_{min}$) represent the maximum (and minimum) values of makespan, cost and execution efficiency measured for the BoT execution in all strategies within $\mathcal{S}$.

- γ_m, γ_c and γ_e denote weights of importance from the user perspective U_{i_u} over makespan, cost and execution efficiency. After the completion of their applications, end-users can compute the perceived satisfaction in their own ways. For this we define the set of unique user satisfaction profiles $\mathcal{L} = \{l_{i_l}, l_{i_l} = (\gamma_m, \gamma_c, \gamma_e)\}$ where $\gamma_m, \gamma_c, \gamma_e \geq 0$ and $\gamma_m + \gamma_c + \gamma_e = 1$.

From the resource owners' perspective we measure the relative **infrastructure utilization efficiency** $\mathcal{E}$, defined in eq. 2.

$$\mathcal{E}_s^{DCI_{i_{DCI}}} = \frac{NOI_s^{util}(DCI_{i_{DCI}})}{NOI_s^{tot}(DCI_{i_{DCI}})} \tag{2}$$

where, for a given strategy s from the set $\mathcal{S}_1$ (defined in section 4.2):

- $NOI_s^{tot}(DCI_{i_{DCI}})$ denotes the total number of operations executed on a particular infrastructure $DCI_{i_{DCI}}$ during the execution of a workload.
- $NOI_s^{util}(DCI_{i_{DCI}})$ denotes only the number of operations executed on a particular infrastructure $DCI_{i_{DCI}}$ that led to work-units completion.

Obviously, $NOI_{tot}(DCI_{i_{DCI}}) \geq NOI_{util}(DCI_{i_{DCI}})$ since hosts can fail before completing tasks, the computations can last longer than the expected completion time or the scheduler simply drops the erroneous results and schedule new replicas of the respective tasks.

We consider that the greater the relative efficiency of the resource utilization, the bigger the satisfaction of their owners.

4. Overall scheduling approach

This section presents our contribution in defining a methodology that allows one to select from a large number of defined scheduling strategies the optimal ones with regard to a set of application requirements.

4.1. The SOFT Methodology

In this section we give the Satisfaction Oriented FilTering (SOFT) methodology. This allows one to select the optimal scheduling strategies given her own set of criteria $\mathcal{C}$. Its essence derives from the practice of the performed

Algorithm 2 SOFT METHODOLOGY FOR OPTIMAL SCHEDULING STRATEGIES SELECTION.

1: Let c_{i_c} be a criterion from $\mathcal{C}$, the considered set of criteria.
2: **for all** $c_{i_c} \in \mathcal{C}$ **do**
3: Define the evaluation of a task t_i within this criterion {*used for building the evaluation matrix A, as discussed in section 3*}.
4: Define an appropriate metric and integrate it into the aggregating metric Θ {*the definition proposed in section 3.3 can be extended*}.
5: **end for**
6: Define $\mathcal{S}$, a set of scheduling strategies based on: the criteria set $\mathcal{C}$, min/max target for each criterion and combinations of importance weights assigned to criteria {*as in table 7 and 8*}.
7: Define L, a set of user satisfaction profiles {*as in table 9*}.
8: **for all** strategy s in $\mathcal{S}$ **do**
9: Run the system to complete a workload and record values for metrics defined at step 4.
10: **end for**
11: **for all** profile l in L **do**
12: **for all** strategy s in $\mathcal{S}$ **do**
13: Calculate Θ_s.
14: **end for**
15: **end for**
16: From $\mathcal{S}$ extract a subset $\mathcal{S}_1$ by applying a filtering method {*we proposed Filter1 with* $\mathbf{Win^{abs}}$/$\mathbf{Win^{freq}}$ *selection methods in section 4.2*}.
17: Define an efficiency metric $\mathcal{E}$ from the resource owners perspective. {*we proposed* $\mathcal{E}$ *in section 3.3*}
18: From $\mathcal{S}_1$ extract a subset $\mathcal{S}_2$ by applying a filtering method that employs $\mathcal{E}$ {*we proposed Filter2 in section 4.2*}.

experiments and analyses. So we describe a detailed process, from the definition of new criteria to the selection of a particular set of strategies that are optimal from both user and resource owners perspectives. Algorithm 2 defines the proposed methodology.

The methodology begins with the definition of criteria to be integrated into the scheduler and a corresponding metric for each (steps 1-5). On the constructed set of possible scheduling strategies (step 6) a selection method is applied in order to find those that provide high and stable user satisfaction levels (steps 7-16). On this resulting subset a second selection method is

applied in order to retain the strategies that are also the most efficient from the resource owners perspective (steps 17-18).

4.2. Finding optimal scheduling strategies

When using this scheduling method into a real system, its designer can configure the method through several key parameters which impact on the defined metrics. For instance, one may define a **large variety** of **scheduling strategies** by:

1. setting either *max* or *min* target to the task evaluation criteria presented in section 3.2, and
2. by defining different combinations of importance weights ω, assigned to those criteria.

To address this challenge, in this section we propose a methodology that supports a scheduler designer in finding optimal scheduling strategies given the formulated conditions. Our methodology is **generic** and **independent** from the **number and type of** considered **scheduling criteria**. Consequently, the advantage is that one can use it to configure the scheduler for her own set of criteria.

The methodology yields a set of optimal strategies considering all defined user satisfaction profiles at a time, but it can also be applied (with small adaptation) in a similar way for a particular profile only.

Figure 3 depicts the method, which at the highest level consists of two phases: **Filter1** - defined from the users' perspective and **Filter2** - defined from the resource owners perspective. From the set of all defined scheduling strategies $\mathcal{S}$, **Filter1** selects the strategies that conveniently satisfy the users, resulting $\mathcal{S}_1$. Next, **Filter2** selects from $\mathcal{S}_1$ the strategies that satisfy the resource owners (i.e. increasing the capitalization of their infrastructures) resulting $\mathcal{S}_2$. Finally, a scheduler designer can use the strategies from $\mathcal{S}_2$ to configure the scheduler.

From section 2.2 recall N_c, the number of scheduling criteria in the complete set of considered criteria $\mathcal{C}$ and ω_{i_c} the importance weight assigned to a criterion c_{i_c}. Based on $\mathcal{C}$ and the min/max target assigned to each criterion there are a number of 2^{N_c} possible *families* of scheduling strategies. Moreover, for each family $i \in \{1, ..., 2^{N_c}\}$, one may assign values for the weights $\omega_1, \omega_2, ...\omega_{N_c}$ such that $\sum_{j=1}^{N_c} \omega_j = 1, \omega_j \in [0, 1)$ in different combinations. Therefore, a scheduler designer faces the problem of optimizing the scheduling method on these two key parameters.

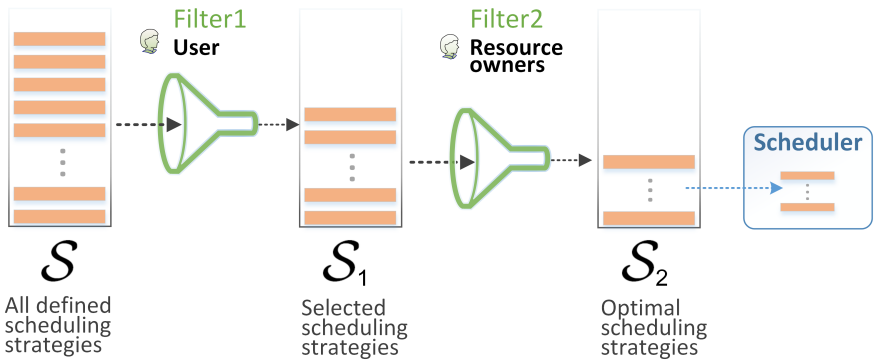


Figure 3: Method for selecting optimal scheduling strategies from user and resource owners perspectives.

Based on the optimization parameters we define a **scheduling strategy** s as a tuple $((b_1, \ldots, b_i, \ldots, b_{N_c}), (\omega_1, \omega_2, ..., \omega_{N_c}))$, with $b_i \in \{0, 1\}$, where 0/1 stands for min/max. Thus, the set $\mathcal{S}$ can be split into 2^{N_c} strategy families.

In the following we define the Filter1 and Filter2 phases. First, for each strategy $s \in \mathcal{S}$ and a sample L of N_L user profiles over $\mathcal{L}$ we compute the Θ metric and obtain a satisfaction sampling distribution characterized by the sample mean $\bar{u}_s = \frac{1}{N_L} \sum_{l=1}^{N_L} \Theta_l$ and the sample standard deviation $\sigma_s = \sqrt{\frac{1}{N_L - 1} \sum_{l=1}^{N_L} (\Theta_s - \bar{u}_s)^2}$.

Filter1 works in two steps:
Step1 constructs the set $\mathcal{S}'$ from strategies s that achieve both:

- ✓ a corresponding user satisfaction distribution of user satisfaction values so that $\frac{\bar{u}_s}{\sigma_s}$ is **above** some chosen percentile q_1, when sequentially considering all the $\frac{\bar{u}_s}{\sigma_s}$ distributions of values restricted to each strategy family and

- ✓ a flat positive satisfaction for all users, i.e. $\sigma_s = 0$ and $\bar{u}_s > 0$.

Step2 further selects from $\mathcal{S}'$ a set of strategies $\mathcal{S}_1$, by applying one of the following methods:

- ***Absolute winners*** **($\mathbf{Win^{abs}}$)**:
  Select the strategies that obtain the largest p values (e.g. $p = 2$) in the descending order of $\bar{u}_s$.

- ***Frequency winners*** **($\mathbf{Win^{freq}}$)**:
  Based on the distribution of $\bar{u}_s$ values, select the strategies $s \in \mathcal{S}'$ so that $\bar{u}_s$ is above some percentile q_2. Among these, identify those strategy families i which appear with the highest p frequencies (e.g. $p = 2$). Then, select the strategies that belong to the identified families and also for which weights $\omega_{(.)}$ occur with the highest frequency in the initially selected set of strategies.

Step1 provides a primary filtering of all strategies in $\mathcal{S}$, by selecting the ones that are among the first (i.e. fall into the q_1 percentile) in terms of *average* satisfaction value per unit of risk. The risk is represented by the heterogeneity of the considered user profiles, L. After this step we obtain the most homogeneous strategies with respect to the induced benefits. Step2 proposes two methods for further refining the strategies set: either $\mathbf{Win^{abs}}$ which trivially selects the strategy with the highest mean utility value, or $\mathbf{Win^{freq}}$ which takes into account the stochastic nature of user satisfaction realizations.

Filter2

From the $\mathcal{S}_1$ set, this phase further selects the strategies that perform well from the resource owners perspective. Hence, for each $s \in \mathcal{S}_1$ we compute the infrastructure utilization efficiency ϵ for each considered infrastructure $DCI_{i_{DCI}}$. Thus for N_{dci} infrastructures, we obtain a distribution of values $\{\epsilon_s^{(k)},\ k = 1, ..., N_{dci}\}$ with mean $\bar{\epsilon}_s = \frac{1}{N_{dci}} \sum_{k=1}^{N_{dci}} \epsilon_s^{(k)}$, representing the average infrastructure utilization efficiency of strategy s. Given these, we finally select the strategies s so that $\bar{\epsilon}_s$ is above a percentile q_3 within the distribution of the obtained values. These strategies form the $\mathcal{S}_2$ set which is further embedded into the scheduler.

In section 6.3.5 we present the results of applying this methodology for a setup with 3 scheduling criteria, 10 user satisfaction profiles (given in table 9) and 8·28=224 scheduling strategies.

5. Scheduling method implementation

In this section we describe the architecture of the scheduler implementation and other complementary components. The scheduler in general is inspired from the XtremWeb behavior, borrowing basic functional components like the pull-based and task replication mechanisms.

Figure 4 depicts a component-based representation of the implemented system architecture; its main components are: the Task Scheduler, the trace-based Hybrid DCI Simulator and the Visualizer.

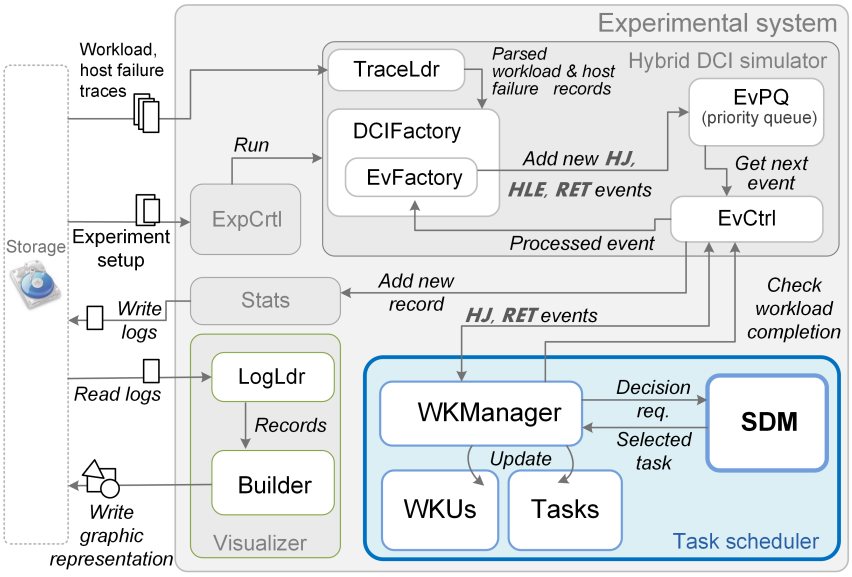


Figure 4: Experimental system architecture.

The `Task Scheduler` is a real implementation of the proposed scheduling approach. This component is responsible for selecting and scheduling tasks to pulling hosts. This component is called by the `EvCtrl` component when consuming a `HJ` (host join) or `RET` (return result) event from the queue. The effective decision making for the task selection is delegated to the `SDM` component, which, at its root implements the Promethee decision model, described in section 3.

The `Hybrid DCI Simulator` realistically reproduces the behavior of hosts from different types of computing infrastructure. It is based on events created from real failure traces. In this way we obtain hybrid IN/OUT behavior of

hosts originating from different types of DCI. After reading the experiment configuration, the `TraceLdr` component can load different subsets of hosts from any combination of DCIs and for particular time-windows from the total available trace-period. For each DCI type, the `DCIFactory` creates hosts accordingly in terms of CPU, price, error rate and energy efficiency. Before running an experiment, all hybrid DCI infrastructure events (`HJ` and `HLE` (host leave)) are created and added to the `EvPQ`. Also, based on workload information for each user, the `EvFactory` creates and adds `WKUA` (work unit arrival) events to the `EvPQ` queue.

The `EvCtrl` is the main component of the simulator, controlling the completion of BoTs. For this, after the creation of the hybrid DCI and workload, `EvCtrl` starts the execution of the workload by consuming events from `EvPQ` in a priority order, given by a time-stamp.

During the execution, based on the type of event currently being consumed the simulator creates new events, which we describe in table 2.

Event name	**Significance**	**Creation**
`HJ` - Host Join `HLE` - Host Leave	Indicates a host identity, its originating DCI and the time at which it joins/ leaves the system.	Created from real failure traces, before starting the BoTs execution.
`WKUA` - Work Unit Arrival	Indicates the identity of a work unit, the BoT to which it belongs and its arrival time at scheduler.	Created for different types of workload at the beginning of the experiment run.
`SCH` - Schedule	Marks the time when the scheduler schedules and sends a task to a pulling host.	Created for HJ/RET events if the BoT execution uncomplete.
`RET` - Return	Marks the time at which a host completes a task and returns the result to scheduler.	Created for SCH events.

Table 2: Rules for creating new events by the experimental system.

Algorithm 3 shows the order in which `EvCtrl` component *consumes* events. `EvCtrl` component reads events from `EvQueue` in their priority order. For some of them, new events are created or existent ones are deleted from the queue. For instance, when a host leaves during the execution of a

Algorithm 3 The event processing algorithm.

```
1: e = next event in EvQueue
2: if type of e is HJ then
3:    Call the SDM component to select a task t_{i_t} and schedule it to the
      pulling host.
4: else
5:    if type of e is HLE then
6:       for all event ∈ EvQueue do
7:          if type of event is not HJ or HLE then
8:             remove event from EvQueue
9:          end if
10:      end for
11:   end if
12: else
13:   if type of e is WKUA then
14:      w_{i_w} = the received work unit
15:      Create task t_{i_w,1}
16:      Add t_{i_w,1} to T
17:   end if
18: end if
```

task, the return result event (`HLE`) associated to the fallen node is deleted. Consequently, the scheduler will not receive the result for the respective task, causing a rescheduling of a new replica of the task.

During the execution of the system, the `EvCtrl` component calls the `Stats` component to log relevant meta-information about the tasks' completion process. We use this output for the characterization of the experimental data in section 6.1 and to retrieve efficiency metrics of the infrastructures' utilization in section 6.3.6.

The `Visualizer` creates custom representations of the meta-information collected by the `Stats` component during the system execution. Figure 4 was drawn from the information produced by the `Visualizer`. This facilitates the understanding of the different behaviors of the infrastructures in terms of availability of the hosts. The output of the `Visualizer` proved to be useful for driving our investigations, helping us to better understand the distribution of the tasks on different types of DCI.

6. Experiments and results

This section presents the experimental setup used for the validation of the scheduling approach presented in the previous sections. More precisely, we showed how to apply the scheduling methodology devised in section 4.2, in order to select the proper tuning of the scheduler, considering a hybrid computing infrastructure. We start by presenting the experimental data and setup then we discuss the obtained results.

6.1. Experimental data

With the aim of observing the behavior and measuring the performance of the proposed scheduling method, we developed and ran the system presented in section 5. The considered hybrid computing infrastructure was created by loading real failure traces [21, 19] as found in the public FTA (Failure Trace Archive) repository [15] for three types of infrastructure:

- **IDG**: For Internet Desktop Grid, the system loads BOINC failure traces (from the SETI@home project) characterized by highly volatile resources. The traces contain 691 hosts during a period of 18 months, starting from 2010.

- **Cloud**: For this environment the simulator loads Amazon EC2 spot instance traces containing 1754 hosts from 2011. Observing the traces, the resources are very stable.

- **BEG**: The Best Effort Grid is an infrastructure or a particular usage of an existing infrastructure (like OAR [5]) that provides unused computing resources without any guarantees that these remain available to user during the complete execution of his application [9]. For the creation of this environment, the simulator loads Grid5000 traces with host join and leave events for a period of 12 months (during 2011) from the following sites: Bordeaux, Grenoble, Lille and Lyon. The traces capture the activity of 2256 hosts. Inspecting the files we observe that the resources are quite stable, meaning that small groups of machines go off approximately at once for a small number of hours.

When the experimental system loads the failure traces, it assigns **CPU capacity** and **price** values to each host, according to the type of the originating DCI. The values are randomly chosen with uniform distribution from the ranges given in table 3. As order of magnitude, the values were set

DCI type	**CPU capacity** (number of executed instructions/ second)	**Price charged by host** (monetary units/ second)
IDG	$\{50, 100, 150, 200, 250, 300, 350, 400\}$	0
Cloud	250	0.001
	300	0.005
	350	0.0075
	400	0.01
BEG	$\{50, 100, 150\}$	0

Table 3: Host CPU capacity and price values for the considered types of DCI.

so that a host having the maximum considered CPU capacity completes the largest task ($10^7 instructions$) in almost 7 hours and the smallest task ($10^6 instructions$) in 40 minutes. While hosts from IDG and BEG compute for free (0 price), the hosts from Cloud require a price for their computations. The values in table 3 show that more powerful Cloud hosts are more expensive. In all experiments, CPU capacity and price are constant for a particular host throughout the execution of the BoT. Therefore we do not consider dynamic pricing models.

In real systems, hosts originating from different infrastructure types manifest unequal reliability levels. To mimic this aspect in our experiments we created **three types of host behavior** concerning the moment in time when a host returns the result for a completed task, relative to the estimated `ECT` (at scheduler):

- **In time** - when a host returns the result for a particular task at the expected completion time.
- **Delay** - when a host in order to complete and return the result for a task needs more than `ECT`. For this behavior, 15% of the results from a BoT are delayed with a factor between $(1, 1.5]$.
- **Never** - when a host do not yield the result for a particular task. For this behavior we consider that for 5% of the BoT, hosts never send a result to the scheduler. This is different from a host leaving the system (according to the failure traces). In that case, the same host may rejoin the system and pull for work again, while in this behavior the host is available, but do not finish the computation (due to a local error).

In all experiments we use all three behaviors, except the experiment from section 6.3.1, where we distinctly analyze the scheduler performance on each particular type of behavior.

From the publicly available failure traces we randomly selected subsets of hosts so that the aggregated CPU capacity per type of DCI is **roughly equal**. By this approach we obtain three types of computing infrastructure, which are comparable. They are balanced in terms of total computational capacity while having different degrees of heterogeneity in terms of availability, price and error-proneness.

For our purpose we kept static the individual characteristics of hosts for the duration of the experiments. It is out of the scope of this paper to investigate our scheduling approach with dynamic host models, like dynamic pricing of the computation.

In section 3.2 we shortly described our approach of considering two parameters f and s to implement the error proneness behavior of some hosts within the computing infrastructures. Recall f, the fraction of hosts that produce erroneous results and s, the probability with which they manifest this behavior. Our scheduler implements a simple reputation model[2] and computes the reputation of the pulling host, before the task evaluation phase (described in section 3.2). During the task evaluation phase, the scheduler uses this reputation score to calculate the expected error impact (the `EEI` criterion described in section 3.2). In our experiments we consider the f and s parameters as in table 4 - all values are expressed in % and have uniform distribution:

DCI type	**Value of f**	**Value of s**
IDG (BOINC: SETI@home)	40	[0,70]
Cloud (Amazon EC2)	2	[0,5]
BEG (Grid5000)	10	[0,20]

Table 4: Values for the f and s parameters.

Table 5 describes the **workload** for one user and one application execution.

6.2. Experimental setup

Each experiment is performed according to a particular setup, which is defined by a set of key parameters. For one setup, the value of a **target**

Name	Number of instructions (NOI)/work unit	Number of work units/ user/ application
WL	Randomly selected values from $[10^6, 10^7]$, uniformly distributed.	2000

Table 5: Description of the scheduler workload.

parameter may vary on a defined range and the other parameters take fixed values. This basic approach allows us to study the impact of the target parameter on the tracked metrics. For comparability, we used the same workload (presented in table 5) into the experimental system, for various particular setups.

For each experiment run, the system performs the following steps, in the given order:

1. Read the experiment setup from a configuration file.
2. Create the hybrid DCI by loading failure traces and instantiating `HJ` and `HLE` events for each type of DCI. In all experiments we use a hybrid DCI composed from Cloud, IDG and BEG, unless otherwise specified.
3. Create the workload for all users by instantiating the `WKUA` event.
4. Start the execution of the workload on the hybrid infrastructure. The running of an experiment ends when all work units within the given workload are complete.
5. Report the obtained results.

6.3. Results

This section presents the results of the experiments driven for the validation of the proposed multi-criteria scheduling method. The evaluation begins with simple scenarios focused on the specific mechanisms of the scheduling method. Next we propose a set of optimizations for the scheduler and show how to find optimal scheduling strategies by applying the filtering method defined in section 4.2.

6.3.1. Primary performance evaluation

In this section we validate the performance of the task scheduling method by comparing it with the First-come-first-serve (**FCFS**) approach, using **makespan** and **cost** as metrics.

The **FCFS** ($First - Come - First - Served$) method assumes that the scheduler randomly selects a task and assigns it to a pulling host.

Experimental setup: For each method, combination of DCIs and type of host behavior we measure and report the makespan for the completion of the workload.

In order to obtain a clear performance overview between the proposed scheduling method and FCFS, we depict the relative difference of makespan in figure 5. The values are calculated using eq. 3.

$$\Delta_M^i = \frac{M_{FCFS}^i - M_{Promethee}^i}{M_{Promethee}^i} \times 100 \tag{3}$$

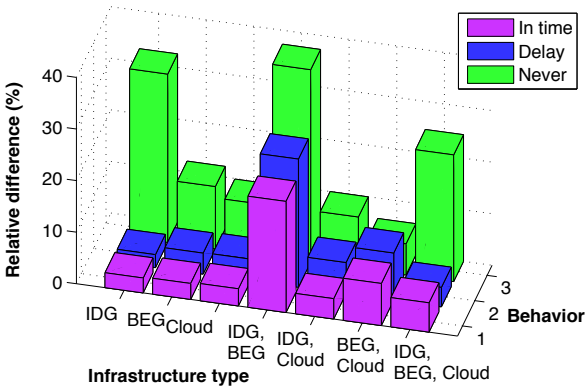


Figure 5: Performance improvement of the proposed scheduling approach and FCFS.

We observe that the Promethee-based scheduling outperforms FCFS, at different degrees in the considered types of DCI. While in Cloud environment the difference is 9-12%, a better performance (32% improvement) of the proposed scheduling method is shown for IDG, for the *never* behavior. Also, the proposed scheduling method shows a 38% improvement for the IDG-BEG hybrid infrastructure. As a general remark, our scheduling method performs significantly better on infrastructure combinations containing IDG (in which hosts are more likely to fail).

6.3.2. Evaluating Promethee scheduler overhead

When designing the scheduler, one can choose from several preference functions. These functions have different complexities and though different real execution costs (in terms of CPU cycles), on the machine running the

scheduler. In this section we evaluate the **overhead** of the scheduler for the following preference functions:

- $P_{Linear}(d_{i_c}) = \begin{cases} \frac{d_{i_c}}{\sigma} & \text{if } d_{i_c} \leq \sigma \\ 1 & otherwise \end{cases}$
- $P_{Level}(d_{i_c}) = \begin{cases} 0 & \text{if } d_{i_c} < q \\ \frac{1}{2} & \text{if } q \leq d_{i_c} < p \\ 1 & \text{if } d_{i_c} \geq p \end{cases}$
- $P_{Gaussian}(d_{i_c}) = \begin{cases} 1 - e^{-\frac{{d_{i_c}}^2}{2\sigma^2}} & \text{if } d_{i_c} > 0 \\ 0 & otherwise \end{cases}$

where:

- d_{i_c} is the deviation between the evaluations of two tasks within a criterion c_{i_c}: $d_{i_c}(t_1, t_2) = a_{i_c,1} - a_{i_c,2}$;
- σ is the standard deviation of the differences d_{i_c} for all pairs of tasks;
- q is the degree of indifference; any deviation below q leads to considering t_1 and t_2 equivalent;
- p is the degree of preference; any deviation greater than q gives a strict preference, either 0.5 (for deviations between q and p) or 1.

Experimental setup: for each type of considered preference functions (*Linear*, *Level* and *Gaussian*) we run the system to complete the same WL workload.

Figure 6 presents real execution time measurements of the decision-making algorithm, implemented by the `SDM` component (presented in section 5). While the values on the y axis represent duration in milliseconds for each scheduling decision making, the x axis shows the scheduling iterations needed for a workload completion. The results regard the *Linear*, *Level* and *Gaussian* preference functions. The graph clearly shows that *Gaussian* is significantly more CPU-consuming, compared to the *Linear* and *Level* functions. Consequently, although the *Linear* and *Gaussian* functions yield similar makespan values (as mentioned in section 6.3.3), when designing a real system one should use the *Linear* function due to its execution cost efficiency. In this chart, the execution time needed for making a scheduling decision decreases with the completion process, since the decision is computed on smaller sets of tasks.

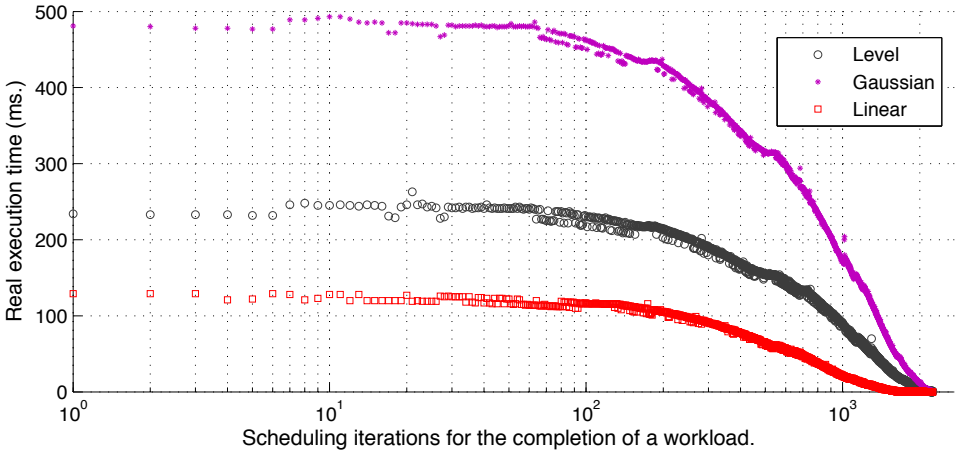


Figure 6: Real execution time values for the SDM component during the completion of a WL workload.

6.3.3. Tuning the scheduling method to increase performance

From section 3.1 we recall the preference function P, which may impact on the performance of the task selection algorithm. Therefore in this experiment we studied the achievements of the scheduler in terms of makespan when using the *Linear*, *Level* preference functions compared to FCFS. We also analyzed the consistency of the attained performance with respect to the utilized failure traces.

Experimental setup: in this experiment, for each method (*Linear*, *Level*, *Gaussian* and FCFS) the system is run 120 times to complete the same workload. For all methods the completion is carried by the scheduler with the same set of hosts but loading the failure traces at different start-points in time. By this we aim at proving that the relative performance of the proposed scheduling method is consistent with respect to the host failure behavior in time. In this experiment we use a hybrid DCI, composed from IDG, Cloud and BEG. First we compare our scheduler using *Level* and *Linear* preference functions described in section 6.3.3 and a FCFS scheduler.

Table 6 presents the descriptive statistics regarding the makespan distribution for the considered setup.

Figure 7 depicts the empirical cumulative distribution functions (CDFs)

[2]P-values are computed for the Kolmogorov-Smirnov test of normality under the null hypothesis is that the distribution of makespan is Gaussian.

Method	**Mean (STDEV)**	**Difference (%)**	**The p-values**[2]
Linear	367540,26 (104523,14)	0	0,190 ($> 10\%$)
Gaussian	371987,49 (105017,32)	+1,21	0,214 ($> 10\%$)
Level	414840,79 (110604,18)	+12,86	0,466 ($> 10\%$)
FCFS	432419,97 (118178,69)	+17,65	0,831 ($> 10\%$)

Table 6: Descriptive statistics of the makespan distributions for each method.

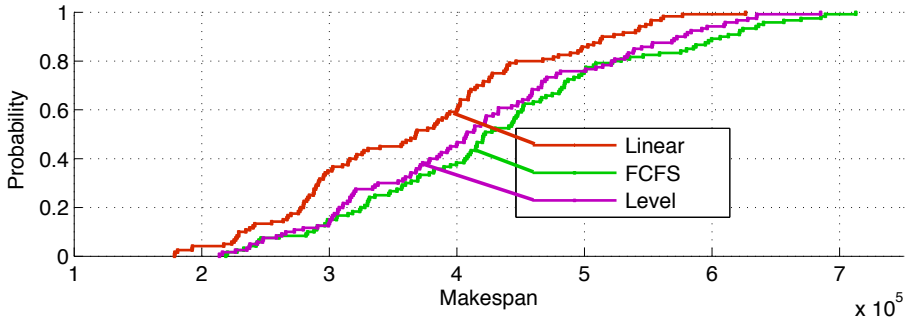


Figure 7: Stochastic dominance of the *Linear* method with respect to the $FCFS$ and *Level* methods.

of the execution times for the three methods. The Y-axis depicts the values of the CDF function $F(M)$ of the makespan M, showing the probability that the makespan records values lower than the scalar on the abscissa. We observe that the *Linear* function strictly dominates the other two methods, in the sense that $F_{Linear} > F_{Level}$ and $F_{Linear} > F_{FCFS}$ for all values on the abscissa. We also tested the dominance of the *Linear* method over the *Level* and $FCFS$ using the $t-test$ for the equality of means in two samples and the *Levene*'s test for equality of variances in two samples with 99% confidence level, and the results are the same. Statistical tests show a weak dominance of *Level* over $FCFS$, therefore we conclude that the designed scheduling method based on the Promethee decision model is superior to $FCFS$. We omitted the representation for the *Gaussian* function because its performance is very similar to the *Linear* function and they overlap.

6.3.4. Adjusting the Promethee window size

As stated in section 2.2, the scheduler makes a new scheduling decision for each pulling host. Hence, each time, the whole BoT is given to Promethee as input.

In this experiment we are interested in finding out whether for a particular application, the Promethee scheduler needs to compute the decision on the whole BoT in order to obtain a good makespan or it can cope with smaller subsets of the BoT, without loosing performance. For this we run the scheduler on a window of the BoT, denoted μ, defined as a fraction of tasks, randomly chosen from the BoT. By this we adjust the scheduler input size.

Experimental setup: For each value of μ in the [0.05,100] range we run the system to complete a WL workload and measure the makespan and the real execution time of the Promethee algorithm (on the machine running the scheduler).

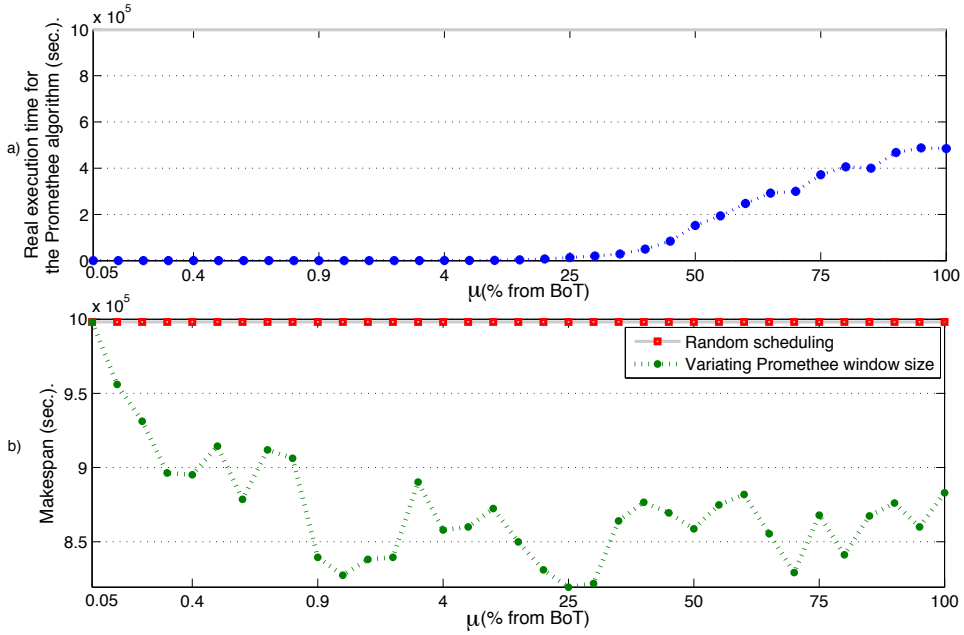


Figure 8: Real execution time of the Promethee algorithm and makespan values.

In figure 8 we present in comparison the impact of μ variation on both a) the real execution time of the Promethee algorithm and b) makespan. From the chart we observe that up to $\mu = 25\%$ the execution time is relatively flat,

then it linearly increases. Analysing the values we find that the algorithm has a good scalability since for $\mu = 50\%$ the execution time is almost double, then for $\mu = 100\%$ the execution time increased with a 1.3 factor only. At this time, we do not have information on the algorithm's behavior regarding scalability for a greater number of criteria. However, from the definition of the algorithm we deduct a high parallelization and distribution potential.

In figure 8 b) we observe that Promethee algorithm needs that the size of the window is above a threshold (i.e. 10%) in order to yield a low makespan. We explain this as the need of the algorithm of a minimum sample of the BoT in order to make relevant scheduling decisions. After this threshold we observe that the makespan oscillates within a small range.

6.3.5. Finding optimal strategies from user perspective - Filter1 phase

This experiment shows the results for applying the Filter1 phase previously defined in section 3.3. From $\mathcal{S}$ - the initial set of defined scheduling strategies we aim to select a subset of strategies that provide high levels of user satisfaction. For this we apply the two methods proposed in section 3.3. We recall that $\mathbf{Win^{abs}}$ method is conceived to admit strategies that achieve the highest levels of user satisfaction while $\mathbf{Win^{freq}}$ selects those strategies that yield high levels of user satisfactions, **but also stable** relative to the user satisfaction profile distribution L.

Experimental setup: for this experiment we used the following values:

- three scheduling criteria: `ECT`, price and `EEI`, so $N_c = 3$ and the resulting scheduling strategy families given in table 7;
- 28 combinations of importance weights $\omega_{(.)}$ given in table 8;
- 10 different user satisfaction profiles l, given in table 9.

For each defined scheduling strategy (based on tables 7 and 8) we ran the system to complete a WL workload and used Θ as evaluation metric.

Figure 9 shows the user satisfaction values delivered by the scheduler, for all scheduling strategies $s \in \mathcal{S}$ and user satisfaction profiles $l \in L$.

Strategies / Criterion	$s_{1,j}$	$s_{2,j}$	$s_{3,j}$	$s_{4,j}$	$s_{5,j}$	$s_{6,j}$	$s_{7,j}$	$s_{8,j}$
Price	max	max	max	max	min	min	min	min
ECT	min	min	max	max	min	max	max	min
EEI	min	max	min	max	max	max	min	min

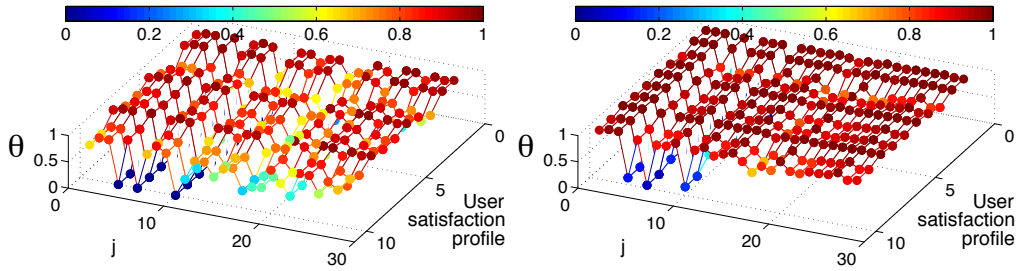
Table 7: $2^3 = 8$ possible and considered strategy families for the three scheduling criteria setup.

	Strategies													
Importance weights	$s_{i,1}$	$s_{i,2}$	$s_{i,3}$	$s_{i,4}$	$s_{i,5}$	$s_{i,6}$	$s_{i,7}$	$s_{i,8}$	$s_{i,9}$	$s_{i,10}$	$s_{i,11}$	$s_{i,12}$	$s_{i,13}$	$s_{i,14}$
$\omega_{\texttt{ECT}}$	1/3	2/3	2/3	0	1/3	0	1/3	0.5	0.5	0	0.1	0.45	0.45	0.2
$\omega_{\texttt{Price}}$	1/3	0	1/3	2/3	2/3	1/3	0	0.5	0	0.5	0.45	0.1	0.45	0.4
$\omega_{\texttt{EEI}}$	1/3	1/3	0	1/3	0	2/3	2/3	0	0.5	0.5	0.45	0.45	0.1	0.4
	$s_{i,15}$	$s_{i,16}$	$s_{i,17}$	$s_{i,18}$	$s_{i,19}$	$s_{i,20}$	$s_{i,21}$	$s_{i,22}$	$s_{i,23}$	$s_{i,24}$	$s_{i,25}$	$s_{i,26}$	$s_{i,27}$	$s_{i,28}$
$\omega_{\texttt{ECT}}$	0.4	0.4	0.1	0.1	0.2	0.7	0.2	0.7	0.1	0.1	0.3	0.6	0.3	0.6
$\omega_{\texttt{Price}}$	0.2	0.4	0.2	0.7	0.1	0.1	0.7	0.2	0.3	0.6	0.1	0.1	0.6	0.3
$\omega_{\texttt{EEI}}$	0.4	0.2	0.7	0.2	0.7	0.2	0.1	0.1	0.6	0.3	0.6	0.3	0.1	0.1

Table 8: The 28 considered combinations of importance weights $\omega_{(.)}$.

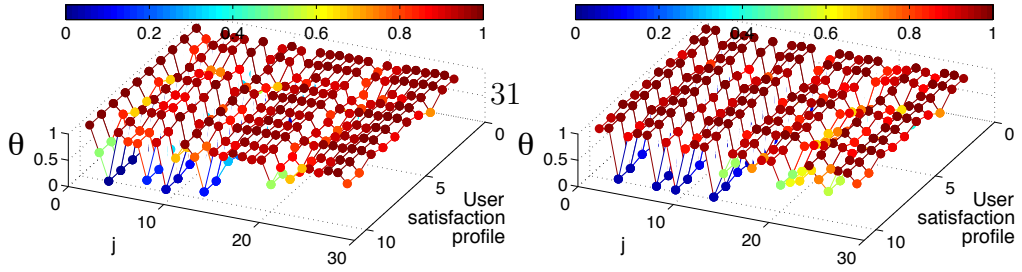
	User satisfaction profile									
Weights	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
γ_M	1	0	0	1/3	2/3	2/3	0	1/3	0	1/3
γ_C	0	1	0	1/3	1/3	0	2/3	2/3	1/3	0
γ_E	0	0	1	1/3	0	1/3	1/3	0	1/3	2/3

Table 9: 10 considered user satisfaction profiles (L) based on different combinations of weights in Θ.



(a) Strategies $s_{1,j}$

(b) Strategies $s_{2,j}$



(c) Strategies $s_{3,j}$

(d) Strategies $s_{4,j}$

Table 10 presents the results of applying both $\mathbf{Win^{abs}}$ and $\mathbf{Win^{freq}}$ methods defined in Step2 of the Filter1 phase (section 4.2). In this table we considered only the first two winning positions, so $p = 2$.

Percentile q_1	$\mathbf{Win^{abs}}$		$\mathbf{Win^{freq}}$		
	Strategy	**Place**	**Percentile** q_2	**Strategy**	**Place**
top 25%	$s_{3,1}$	1^{st}	top 25%	$s_{4,26}$ $s_{2,26}$	1^{st}
				$s_{3,26}$	2^{nd}
			top 15%	$s_{3,26}$	1^{st}
				$s_{2,26}$ $s_{3,20}$	2^{nd}
	$s_{2,20}$	2^{nd}	top 5%	$s_{3,1}$ $s_{3,26}$	1^{st}
				$s_{3,27}$ $s_{2,20}$	2^{nd}
top 15%	$s_{3,1}$	1^{st}	top 25%	$s_{3,26}$	1^{st}
				$s_{2,26}$	2^{nd}
			top 15%	$s_{3,26}$	1^{st}
	$s_{2,20}$	2^{nd}		$s_{2,26}$	2^{nd}
			top 5%	$s_{3,1}$ $s_{3,27}$	1^{st}
				$s_{2,20}$	2^{nd}

Table 10: Results of Filter1 phase: optimal scheduling strategies from a user satisfaction perspective. Recall that percentile q_1 is used in both methods while percentile q_2 is used in $\mathbf{Win^{freq}}$ method only.

It is obvious that for the same percentile q_1, the $\mathbf{Win^{abs}}$ method provided the same strategies, while with method $\mathbf{Win^{freq}}$ the set of winning strategies changed. However, the fact that increasing percentile q_1 does not change the *absolute winners* implies that these strategies are homogenous enough, in terms of the satisfaction induced for the considered L user profiles.

Moreover, we found that the set of *frequency winners* gets closer to the set of *absolute winners* as q_2 increased. However, the $\mathbf{Win^{freq}}$ method should not be applied for a very high percentile q_2 (e.g. above percentile 85%), since then the method starts to rely too much on very particular realizations of satisfaction values, within the current experiment.

Briefly put, the knowledge achieved from the presented results is that the ECT criterion should be always maximized, while price and EEI can be maximized or minimized, but never simultaneously minimized. The *winning* strategies with respect to the importance weights $\omega_{(.)}$ are for $j = \{1, 20, 26, 27\}$.

6.3.6. Finding optimal strategies from resource owners' perspective - Filter2 phase

In the previous experiment we evaluated the scheduling method using the Θ metric. We found a set of strategies that yield high levels of user satisfaction and is stable relative to the user satisfaction profile distribution. Hence, the evaluation of the scheduling method was purely from the user's perspective. Since in reality the **owners** of the utilized resources aim at maximizing their capitalization, we continue the evaluation of the scheduling method from their perspective.

Recall the idea that from $\mathcal{S}_1$ - the set of strategies that passed the Filter1 phase we further aim to select a subset of strategies that **also** maximize the efficiency $\mathcal{E}$ (previously defined at the end of section 3.3). By this, we ensure that the finally selected scheduling strategies perform best, also from resource owners perspectives.

Experimental setup: for the system executions presented in the previous experiment we calculate the efficiency $\mathcal{E}$ for each type of infrastructure. Then we seek for correlations between the calculated values and the winning strategies that passed Filter1.

First we report the calculated efficiency $\mathcal{E}$ for the strategies that satisfied users (passing Filter1). Then we compare these strategies with **the other strategies in the same family** by testing the compliancy with q_1=top 25% percentile and also with the median. In table 11 we present this test, where $\overline{\varepsilon} = \frac{1}{N_{dci}} \sum_{i_{dci}=1}^{N_{dci}} \varepsilon_{i_{dci}}$, representing the average efficiency calculated for the entire hybrid DCI; in our case $N_{dci} = 3$. In this table the strategies are ranked by their calculated $\overline{\varepsilon}$.

Based on this analysis we draw the following **observations**:

- The most efficient strategies are $s_{4,26}$ and $s_{2,20}$, consequently we consider them as similar candidates for the final winning strategy.
- The selected strategies comply with the top 25% test for all types of DCI, with a small exception - the $s_{4,26}$ strategy fails to this test for the Grid5000 infrastructure. Correlating this with table 12 we found that

Strategy	**Calculated efficiency and compliancy tests**						Average infrastructure utilization efficiency $\overline{\varepsilon}$
	ε_{BOINC}		$\varepsilon_{AmazonEC2}$		$\varepsilon_{Grid5000}$		
	top 25%	*median*	top 25%	*median*	top 25%	*median*	
$s_{4,26}$	0.636		0.929		0.573		
	pass	pass	pass	pass	fail	pass	0.713
$s_{2,20}$	0.648		0.896		0.512		
	pass	pass	pass	pass	pass	pass	0.685
$s_{3,26}$	0.547		0.917		0.568		
	fail	fail	fail	pass	pass	pass	0.677
$s_{3,27}$	0.576		0.897		0.547		
	fail	fail	fail	fail	fail	fail	0.673
$s_{3,1}$	0.560		0.901		0.534		
	fail	fail	fail	fail	fail	fail	0.665
$s_{2,26}$	0.623		0.881		0.442		
	fail	pass	fail	fail	fail	fail	0.649

Table 11: Calculated efficiencies and the top 25% percentile and median compliancy tests.

the failure is caused by a 0.001 difference in absolute value beneath the top 25% percentile threshold, which is irrelevant.

- Considering the overall compliance and the maximum average efficiency $\overline{E}$ recorded, we select the $s_{4,26}$ strategy as final winner. This may further be used by a system designer into the multi-criteria scheduler of a real system.

- Checking also with table 13, the second best strategy - $s_{2,20}$ yields a high efficiency on the BOINC infrastructure, being beaten only by strategies $s_{1,j}$. This strategy passes both top 25% and median tests on all infrastructure types, but yields a slightly lower average efficiency, namely 0.685.

		Infrastructure utilization efficiency $\mathcal{E}$		
Threshold	**Strategies**	ε_{BOINC}	$\varepsilon_{AmazonEC2}$	$\varepsilon_{Grid5000}$
top 25% percentile	$s_{2,j}$	0.593	0.890	0.503
Median	$s_{2,j}$	0.558	0.881	0.480
top 25% percentile	$s_{3,j}$	0.619	0.922	0.562
Median	$s_{3,j}$	0.601	0.915	0.549
top 25% percentile	$s_{4,j}$	0.612	0.927	0.574
Median	$s_{4,j}$	0.594	0.919	0.564

Table 12: Top 25% percentile and median threshold values for the $s_{2,j}$, $s_{3,j}$ and $s_{4,j}$ families.

In table 13 we report the **maximum** of the measured efficiency $\mathcal{E}$ over all strategies and infrastructure types.

Strategies / **DCI**	$s_{1,j}$	$s_{2,j}$	$s_{3,j}$	$s_{4,j}$	$s_{5,j}$	$s_{6,j}$	$s_{7,j}$	$s_{8,j}$
BOINC	0.653	0.648	0.648	0.636	0.506	0.489	0.534	0.515
Amazon EC2	0.932	0.927	0.928	0.932	0.885	0.870	0.892	0.889
Grid5000	0.580	0.615	0.575	0.607	0.441	0.505	0.520	0.471

Table 13: Maximum values for ε per scheduling strategy family and type of DCI.

Figure 10 shows the infrastructure utilization efficiencies obtained with the scheduling strategies selected during the Filter 1 phase. Due to the disruption patterns specific to each considered infrastructure type, the Amazon EC2 is the most efficiently employed, followed by BOINC and Grid5000. While for the Amazon EC2 all studied strategies attain a relatively similar utilization efficiency, for BOINC and Grid5000 the obtained efficiencies differ. Hence we observe that strategies like $s_{4,26}$, $s_{2,20}$, $s_{2,26}$ attain considerable higher efficiency levels on BOINC compared to Grid5000 and others like $s_{3,26}$, $s_{3,27}$ and $s_{3,1}$ obtain very similar efficiencies for BOINC and Grid5000. From these observations we conclude that one may use the latter strategies to configure the scheduling method, if the aim is to insure high and relatively similar satisfaction levels for the resource owners.

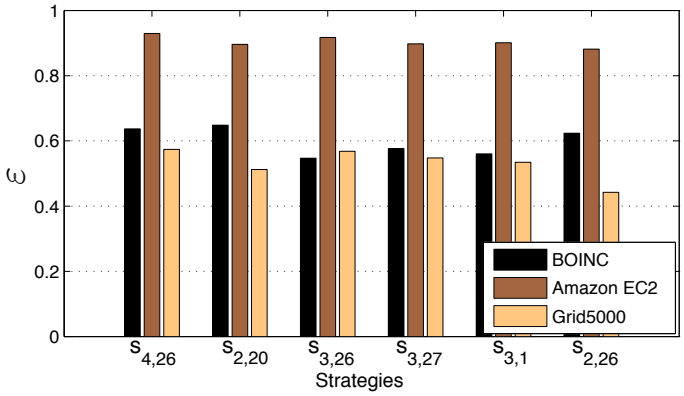
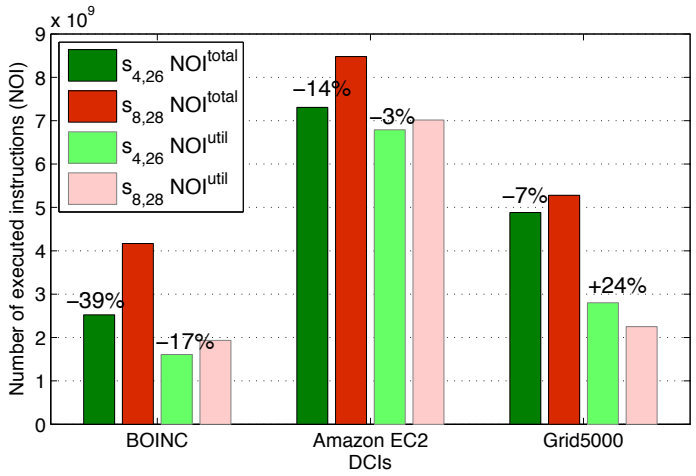
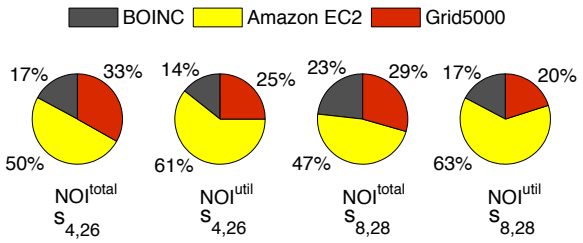


Figure 10: Obtained efficiencies per infrastructure type.

Figure 11 depicts a comparison of two scheduling strategies on the number of executed operations needed to complete a workload, on the considered hybrid DCI. Hence we confront the optimal $s_{4,26}$ and a pessimal $s_{8,28}$ scheduling strategies. We observe that they have quite similar behavior with respect to the distribution of tasks on the specific infrastructure types within the hybrid infrastructure. Inspecting figure 11a we observe that the optimal strategy utilizes more efficiently all infrastructure types composing the hybrid DCI.



(a) Differences in total and utile number of executed instructions.



(b) Distribution of the BoT on the different types of computing infrastructure.

Figure 11: Comparison on number of executed operations of optimal and pessimal scheduling strategies.

7. Related work

In this section we review other approaches for building middleware solutions to facilitate joint usage of computing resources, originating from different types of distributed computing infrastructure.

Addressing the need of simultaneously exploiting different computing infrastructures, for both research and business purposes, there are several attempts [9, 35, 25] to build middleware that aggregates resources from different types of infrastructure, in order to facilitate a better capitalization for their

owners. For instance, the RainCloud[3] project uses two types of clouds: a private Eucalyptus instance and Amazon EC2. The researchers working on this project highlight [31] the importance of carefully considering multiple criteria during the scheduling process for scientific workflow applications on Cloud. It is usually not sufficient to schedule the application just for minimum execution time, because this can drive the costs too high, especially on commercial Cloud systems. Hence, the user perspective must be considered within the scheduling process.

The European FP7 projects EDGeS[37][4] and EDGI[17][5] are representative examples of hybrid DCIs which mix Clouds, Grids and Desktop Grids. These projects have developed bridge technologies to allow bag-of-tasks, to flow from Grid infrastructures to Desktop Grid. The SpeQuloS [9] system, developed in the context of EDGI, uses private Cloud resources to provide quality of service to BoT applications executed on Desktop Grid. In contrast to SpeQuloS, our work allows a broader range of usage optimization of hybrid DCIs, such as cost minimization.

GridBot [35] puts together Superlink@Technion, Condor pools and Grid resources to execute both throughput and fast-turnaround oriented BoTs. It represents a first solution to combine several grids in a *monolithic* platform for large-scale execution of grids, leveraging on replication to handle the job failures on DCIs. GridBot scheduling is policy-based, considering only a fixed set of scheduling criteria. Our approach will be more generic, allowing one to include new criteria into the scheduler.

Iosup et al. [16] employ a performance analysis of BoTs scheduling on large-scale cluster infrastructures, given that multiple users submit simultaneously BoTs to the system and the scheduling is driven by several policies. Kim [18] presents multi-criteria scheduling of BoTs in order to reduce the power consumption at the infrastructure level, while preserving the agreed SLA over the whole bag-of-tasks. Muthuvelu et al. [29] emphasize the granularity of tasks composing the BoTs towards economic and efficient usage of grid resources, while satisfying the user QoS requirements. Their approach is directed by a bugetary constraint of users, who own a limited budget for the execution of the BoTs and they deal with time constraints, but neither

[3]The RainCloud project: http://imgi.uibk.ac.at/research/atmospheric-dynamics/projects/raincloud

[4]Enabling Desktop Grids for e-Science, http://www.edges-grid.eu/

[5]European Desktop Grid Infrastructure, http://edgi-project.eu

of them considers public Cloud.

Dealing with Cloud resources, Oprescu et al. [30] design a budget-constraint scheduler for BoTs, estimating the costs and the makespan for various scenarios before executing the user-selected schedule.

Mateescu et al.[25] propose a complex and innovative architecture for combining the benefits of the HPC, Grid and Cloud technologies. In their work, the authors aim at combining the best attributes of each technology, proposing a model for how these paradigms can work together and how scientific workloads can be managed in such a hybrid computing environment. A key concept proposed in their paper is the *Elastic Cluster*, as an extension of the notion introduced by the OpenNebula[28]. Hence, Mateescu et al. enrich the content of the concept by highlighting the necessity of key mechanisms that allows the different types of computing infrastructure work together. Such mechanisms facilitate the dynamic infrastructure management, workload management as well as the cooperation between cluster-level services and dynamic infrastructure management services.

There already exists a large literature on bi-objective and multi-criteria scheduling on Grid infrastructures. For instance [38] addresses the issue of multi-criteria scheduling workflows on the Grid while [22] adds the constraint of a multi-user environment. Multi-objective scheduling algorithms for multiple Clouds are slowly emerging as a focused research topic. In [13], authors present multi-objective scheduling algorithm, which aims at achieving application high-availability and fault-tolerance while reducing the application cost and keeping the resource load maximized. In [14] consider the issue of costs and elasticity in multiple clouds environment. While we did not consider a decentralised scheduler in this work, it is noteworthy that such direction [32] could be promising as well.

However, the majority of the above-mentioned work [30, 16, 29, 18] consider only a single type of DCI, while our work addressed the issues of using hybrid DCIs. In addition, because our evaluation includes IDG, which suffers from a high volatility of the computing resources, we also take into consideration fault tolerance in our scheduling strategies. Furthermore, we analyzed the scheduling problem from a dual perspective, mitigating the conflicting goals of end users and resource owners.

Other approaches [33] use a distributed strategy for scheduling, instead of a centralized one. In this work, the authors propose a decentralized and cooperative workflow scheduling in a dynamic and distributed Grid resource sharing environment. Through a set of experiments they compare the perfor-

mance of the proposed approach against a centralized coordination technique, showing that their approach is as efficient as the centralized technique with respect to achieving coordinated schedules. Also, in [7] the authors propose a decentralized scheduling model that uses aggregated information to route tasks to the most suitable execution nodes, and is easily extensible to provide very different scheduling policies.

8. Concluding remarks

In this paper we addressed the challenge of scheduling tasks in distributed computing infrastructures (DCI) like Grids and Clouds by proposing a multi-criteria scheduling method based on the Promethee algorithm. We presented the design for a fault-tolerant and trust-aware scheduler, which allows to execute Bag-of-Tasks applications on elastic and hybrid DCI, following user-defined scheduling strategies. Our approach, named *Promethee scheduler*, combines a pull-based scheduler with multi-criteria Promethee decision making algorithm. We explained how multi-criteria scheduling leads to a massive multiplication of the possible scheduling strategies. To address this challenge, we proposed SOFT, a methodology that allows to find the optimal scheduling strategies given a set of scheduling criteria and application requirements. The validation of this method was performed with a simulator that fully implements the Promethee scheduler and recreates an hybrid DCI environment including Internet Desktop Grid, Cloud and Best Effort Grid based on real failure traces. Through a set of experiments we showed that the Promethee scheduler is able to maximize user satisfaction expressed accordingly to three distinct criteria: price, expected completion time and trust, while maximizing the infrastructure useful employment from the resources owner viewpoint. We also presented an optimization which bounds the computation time of the Promethee algorithm so that it makes realistic the possible integration of the scheduler to a wide range of resource management software.

References

[1] D P Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th IEEE/ACM International GRID Workshop*, Pittsburgh, USA, 2004.

[2] Alvaro E. Arenas, Benjamin Aziz, and Gheorghe Cosmin Silaghi. Reputation management in collaborative computing systems. *Security and Communication Networks*, 3(6):546–564, 2010.

[3] J.P. Brans, P. Vincke, and B. Mareschal. How to select and how to rank projects: the Promethee method. *European Journal of Operational Research*, 2:228–238, 1986.

[4] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[5] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. A batch scheduler with high level components. In *Proc. of the 5th IEEE CCGRID Symp. - Volume 2*, pages 776–783. IEEE Computer Society, 2005.

[6] Eddy Caron, Vincent Garonne, and Andre Tsaregorodtsev. Definition, modelling and simulation of a grid computing scheduling system for high throughput computing. *Future Generation Computer Systems*, 23(8):968 – 976, 2007.

[7] Javier Celaya and Unai Arronategui. A task routing approach to large-scale scheduling. *Future Gener. Comput. Syst.*, 29(5):1097–1111, July 2013.

[8] Christophe Cérin and Gilles Fedak. *Desktop grid computing*. CRC Press, 2012.

[9] Simon Delamare, Gilles Fedak, Derrick Kondo, and Oleg Lodygensky. SpeQuloS: a QoS service for BoT applications using best effort distributed computing infrastructures. In *Proc. of the 21st Intl. Symp. on HPDC*, pages 173–186. ACM Press, 2012.

[10] G. Fedak, C. Germain, V. Néri, and F. Cappello. XtremWeb: A Generic Global Computing Platform. In *Proceedings of 1st IEEE CCGRID Symposium, Special Session Global Computing on Personal Devices*, pages 582–587, Brisbane, Australia, May 2001. IEEE/ACM, IEEE Press.

[11] J. Figueira, S. Greco, and M. Ehrgott. *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer, 2005.

[12] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Jounral of Supercomputer Applications*, 15, 2001.

[13] Marc E Frincu and Ciprian Craciun. Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 267–274. IEEE, 2011.

[14] Rui Han, Moustafa M Ghanem, Li Guo, Yike Guo, and Michelle Osmond. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Generation Computer Systems*, 32:82–98, 2014.

[15] INRIA. Failure trace archive. `http://fta.scem.uws.edu.au/`.

[16] Alexandru Iosup, Ozan Sonmez, Shanny Anoep, and Dick Epema. The performance of bags-of-tasks in large-scale distributed systems. In *Proc. of the 17th Intl. Symp. on HPDC*, pages 97–108. ACM, 2008.

[17] P. Kacsuk, J. Kovacs, Z. Farkas, A.Cs. Marosi, and Z. Balaton. Towards a powerful european dci based on desktop grids. *Journal of Grid Computing*, 9(2):219–239, 2011.

[18] Kyong Hoon Kim, Wan Yeon Lee, Jong Kim, and Rajkumar Buyya. Sla-based scheduling of bag-of-tasks applications on power-aware cluster systems. *IEICE Transactions on Information and Systems*, E93-D(12):3194–3201, 2010.

[19] D. Kondo, F. Araujo, P. Malecot, P. Domingues, M. Silva, L., G. Fedak, and F. Cappello. Characterizing Result Errors in Internet Desktop Grids. In *European Conference on Parallel and Distributed Computing EuroPar'07*, Rennes, France, August 2007.

[20] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, and Henri Casanova. Characterizing resource availability in enterprise desktop grids. *Future Generation Computer Systems*, 23(7):888–903, August 2007.

[21] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *Proc. of the 2010 10th IEEE/ACM Intl. Conf. on CCGRID*, pages 398–407. IEEE Computer Society, 2010.

[22] Krzysztof Kurowski, Ariel Oleksiak, and Jan Weglarz. Multicriteria, multi-user scheduling in grids with advance reservation. *Journal of Scheduling*, 13(5):493–508, 2010.

[23] O. Lodygensky, G. Fedak, F. Cappello, V. Neri, M. Livny, and D. Thain. XtremWeb & Condor sharing resources between Internet connected Condor pools. In *Proc. of the 3rd Intl. CCGRID Symp.*, pages 382–389. IEEE Computer Society, 2003.

[24] 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. IEEE Press.

[25] Gabriel Mateescu, Wolfgang Gentzsch, and Calvin J. Ribbens. Hybrid computing - where hpc meets grid and cloud computing. *Future Generation Computer Systems*, 27(5):440 – 453, 2011.

[26] M. Moca and G. Fedak. Using Promethee Methods for Multi-Criteria Pull-based scheduling on DCIs. In *8^{th} IEEE International Conference on eScience 2012*, pages 1–8. IEEE Press, 2012.

[27] Mircea Moca, Cristian Litan, GheorgheCosmin Silaghi, and Gilles Fedak. Advanced promethee-based scheduler enriched with user-oriented methods. In Jrn Altmann, Kurt Vanmechelen, and OmerF. Rana, editors, *Economics of Grids, Clouds, Systems, and Services*, volume 8193 of *Lecture Notes in Computer Science*, pages 161–172. Springer International Publishing, 2013.

[28] Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. Elastic management of cluster-based services in the cloud. In *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*, ACDC '09, pages 19–24. ACM, 2009.

[29] Nithiapidary Muthuvelu, Christian Vecchiola, Ian Chai, Eswaran Chikkannan, and Rajkumar Buyya. Task granularity policies for deploying bag-of-task applications on global grids. *Future Generation Computer Systems*, 29(1):170 – 181, 2013.

[30] Ana-Maria Oprescu, Thilo Kielmann, and Haralambie Leahu. Budget estimation and control for bag-of-tasks scheduling in clouds. *Parallel Processing Letters*, 21(2):219–243, 2011.

[31] Simon Ostermann and Radu Prodan. Impact of variable priced cloud resources on scientific workflow scheduling. In Christos Kaklamanis, Theodore Papatheodorou, and PaulG. Spirakis, editors, *Euro-Par 2012 Parallel Processing*, volume 7484 of *Lecture Notes in Computer Science*, pages 350–362. Springer, 2012.

[32] Mustafizur Rahman, Rajiv Ranjan, and Rajkumar Buyya. Cooperative and decentralized workflow scheduling in global grids. *Future Generation Computer Systems*, 26(5):753–768, 2010.

[33] Mustafizur Rahman, Rajiv Ranjan, and Rajkumar Buyya. Cooperative and decentralized workflow scheduling in global grids. *Future Gener. Comput. Syst.*, 26(5):753–768, May 2010.

[34] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde. Falkon: a fast and light-weight task execution framework. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2007.

[35] Mark Silberstein, Artyom Sharov, Dan Geiger, and Assaf Schuster. Gridbot: execution of bags of tasks in multiple grids. In *Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 11:1–11:12. ACM Press, 2009.

[36] A S Tanenbaum and M van Steen. *Distributed Systems Principles and Paradigms*. Pearson Education, 2007.

[37] Etienne Urbah, Peter Kacsuk, Zoltan Farkas, Gilles Fedak, Gabor Kecskemeti, Oleg Lodygensky, Attila Marosi, Zoltan Balaton, Gabriel Caillat, Gabor Gombas, Adam Kornafeld, Jozsef Kovacs, Haiwu He, and Robert Lovas. EDGeS: Bridging EGEE to BOINC and XtremWeb. *Journal of Grid Computing*, 7(3):335–354, 2009.

[38] Marek Wieczorek, Andreas Hoheisel, and Radu Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems*, 25(3):237–256, 2009.