Future Generation Computer Systems 51 (2015) 61–71

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing

Mihaela-Andreea Vasile[a], Florin Pop[a,*], Radu-Ioan Tutueanu[a], Valentin Cristea[a], Joanna Kołodziej[b]

[a] *Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania*

[b] *Institute of Computer Science, Cracow University of Technology, Poland*

HIGHLIGHTS

- We proposed a hybrid approach for tasks scheduling in Heterogeneous Distributed Computing.
- The proposed algorithm considers hierarchical clustering of the available resources into groups.
- We considered different scheduling strategies for independent tasks and scheduling for DAG scheduling.
- We analyze the performance of our proposed algorithm through simulation by using and extending CloudSim.

ARTICLE INFO

Article history:
Received 4 August 2014
Received in revised form
29 October 2014
Accepted 27 November 2014
Available online 9 December 2014

Keywords:
Resource provisioning
Task scheduling
Clustering
Heterogeneous distributed computing
Multimedia content-driven applications

ABSTRACT

Today, almost everyone is connected to the Internet and uses different Cloud solutions to store, deliver and process data. Cloud computing assembles large networks of virtualized services such as hardware and software resources. The new era in which ICT penetrated almost all domains (healthcare, aged-care, social assistance, surveillance, education, etc.) creates the need of new multimedia content-driven applications. These applications generate huge amount of data, require gathering, processing and then aggregation in a fault-tolerant, reliable and secure heterogeneous distributed system created by a mixture of Cloud systems (public/private), mobile devices networks, desktop-based clusters, etc. In this context dynamic resource provisioning for Big Data application scheduling became a challenge in modern systems. We proposed a resource-aware hybrid scheduling algorithm for different types of application: batch jobs and workflows. The proposed algorithm considers hierarchical clustering of the available resources into groups in the allocation phase. Task execution is performed in two phases: in the first, tasks are assigned to groups of resources and in the second phase, a classical scheduling algorithm is used for each group of resources. The proposed algorithm is suitable for Heterogeneous Distributed Computing, especially for modern High-Performance Computing (HPC) systems in which applications are modeled with various requirements (both IO and computational intensive), with accent on data from multimedia applications. We evaluate their performance in a realistic setting of CloudSim tool with respect to load-balancing, cost savings, dependency assurance for workflows and computational efficiency, and investigate the computing methods of these performance metrics at runtime.

1. Introduction

We are surrounded by multimedia devices, whether they are sources (smart-phones, cameras, drones) or devices that display multimedia information (computers, tablets, smart-phones, TVs), and also by new multimedia content-driven applications, needed in various domains (scientific research, education, healthcare, etc.). A massive amount of heterogeneous data (video, audio, photos), organized in different formats, is created and requires gathering, processing and then aggregation in a fault-tolerant, reliable and secure heterogeneous distributed system. Different Cloud solutions are used to store, deliver and process this data. Heterogeneous distributed systems (HDC) may be created as a mixture of Cloud systems (public or private), mobile devices networks,

* Corresponding author.
E-mail addresses: mihaela.vasile@cti.pub.ro (M.-A. Vasile), florin.pop@cs.pub.ro (F. Pop), radu.tutueanu@cti.pub.ro (R.-I. Tutueanu), valentin.cristea@cs.pub.ro (V. Cristea), jokolodziej@pk.edu.pl (J. Kołodziej).

http://dx.doi.org/10.1016/j.future.2014.11.019
0167-739X/© 2014 Elsevier B.V. All rights reserved.

desktop-based clusters, etc., and are able to handle the current challenges in managing large heterogeneous data sets in a limited amount of time [1]. In this context, dynamic resource provisioning for Big Data application scheduling became a challenge in modern High-Performance Computing (HPC) systems [2,3]. Unfortunately, most existing scheduling algorithms for tasks with dependencies in HDC systems do not consider reliability requirements of interdependent tasks and many times the execution fails because the tasks are allocated to unsuitable resources [4].

Modern HPC systems should be able to face variable demand loads in an efficient way when referring to the resources utilization [5] (use the minimum number of resources) or SLA achievement. A resource provisioning mechanism is an important element that should be used to manage utilization of available resources and to detect if there has been reached peak demand at a certain moment and the system should be extended by requesting additional resources (from a private Cloud for example), or if the system could be shrunk because there are just few requests and check if the QoS requirements of current applications will be still met [6].

Scheduling is a key problem in the context of Cloud computing, virtualization and Big Data because scheduling techniques also have to evolve along with HPC systems. Classic schedulers have been built for batch, homogeneous environments, having a static internal behavior with no or very few changes in the resources structure. The emergent modern systems are defined by highly heterogeneous environments with variable structure—new resources may be added if needed and removed when the work load decreases (on demand provisioning). The distributed computations as a divisible load scheduling problem for MapReduce is another new approach [7]. In this approach, a divisible load model of the computation, and two load partitioning algorithms are proposed. Novel scheduling approaches have to monitor the system structure and adjust the planning according to the real-time situation. This will also provide increased fault-tolerance because changes are always expected (either planned or not).

A workflow [8,9] describes the automation of a process, be it a business, scientific or general process and the set of rules (dependencies). In other words, workflows represent sets of elementary tasks and their dependencies, required for more complex purposes. So, the workflows are described by a DAG having nodes for each task and edges for each dependency. According with [10] the workflow execution should respond dynamically to interference of real-time monitoring and real-time execution, to support the experimentation process in HPC and in Big Data platforms. So, efficient scheduling of concurrent workflows becomes an important issue in HDC environments [11].

In this paper we present an extension of HySARC^2 scheduling algorithm for dependent tasks scheduling. The algorithm was introduced in [12] as a scheduling algorithm that improves workload on the resources available into the Cloud and satisfies tasks requirements. The HySARC^2 algorithm has three parts: (i) Analyze the available resources and group them into clusters (resource aware algorithm); (ii) Provision different groups of similar tasks to different clusters of resources; and (iii) Schedule the tasks in each cluster of resources. The initial version of HySARC^2 was applied for Bag-of-Tasks applications such as data mining algorithms or Monte Carlo simulations, having both IO and computational intensive phases. We extend the HySARC^2 applicability to applications modeled with workflows. We considered Modified Critical Path or Earliest Time First algorithms in our hybrid approach. The proposed scheduling algorithm allows a more efficient and exact structure of resources, because the tasks are classified before the scheduling phase and assigned to suitable groups of resources.

Our contributions in this paper can be summarized as follows:

- First, we proposed a hybrid approach for tasks scheduling in HDC considering both tasks and resources clustering. We acknowledge that there is no solution that could fit all kinds of tasks models. Therefore, our scheduling strategy is based on using different scheduling strategies, selected by taking into consideration both the heterogeneity of computing resources, and application tasks and/or flows. The effectiveness of the utilization of the computing resources is determined by the efficiency of the allocation strategy of the resources.
- We provided a model for adaptive and dynamic clustering considering the abstract modeling of HDC resources. The "distance" computed in this model highlights the similarity between resources, so we can group the resources in clusters and then use the formed clusters in HySARC^2 algorithm.
- We extended CloudSim [13] to consider this approach and we integrate four scheduling strategies: two for independent tasks (Earliest Deadline First and Shortest Job First) and two for DAG scheduling (Modified Critical Path and Earliest Time First).

The structure of the paper is the following. In Section 2 we analyze some existing scheduling solutions in the related work. Section 3 describes in detail the proposed approach: architecture, clustering and scheduling algorithms and the extensions for workflows. We analyze the total execution time and scalability in Section 4 as experimental methodology. Section 5 presents the integration in real Cloud platforms of our solution and in the last part, Section 6, are presented the conclusions and future research work.

2. Related work

The goal of HySARC^2 is to improve scheduling in a given Cloud environment, using adequate resources to satisfy both user requirements and service provider's interests, achieved by clustering and resource labeling. Therefore, topics of interest for this paper are clustering, resource provisioning, hybrid scheduling and algorithms for scheduling different types of tasks.

Cloud service providers are interested in optimizing available resources, in order to being able to satisfy as many user requirements as possible and as a result improving the profit. Efficient energy management is a challenging research issue in resource management in Cloud [14,15]. The HySARC^2 algorithm aims at efficient resource utilization: tasks assigned to suitable resources, having as effect energy saving because inadequate resources could be put in a hibernate state, in the limits of the Service Level Agreement (SLA) [16,17]. We will describe several solutions that take into consideration the resource allocation.

A Resource Aware Scheduling Algorithm which leverages two existing task scheduling algorithms, Min–min and Max–min, is described in [18]. Both algorithms use an estimation of tasks completion time and resource execution time. The presented algorithm alternates the two algorithms depending on input tasks.

An important feature for scheduling algorithms is to have a dynamic behavior according to real environment evolution. Such an algorithm is described in [19]. The algorithm is suitable for arbitrary constraints tasks, their dependencies may be organized as a graph, having as nodes the tasks and as edges the constraints. It consists in two parts: an initial scheduling phase and a rescheduling phase, in which tasks are separated in entry tasks and inner tasks, based on dependency of failing tasks. Depending on the type the node that fails, there may be used different scheduling algorithms: Highest Level First with Estimated Times, Modified Critical Path or Earliest Time First.

In [20] an algorithm having good results on the compromise cost-execution time is presented. The tests showed that the cost may descend with over 15% while the execution time satisfies users' requirements or the execution time may be shorter with average 20% and the costs would remain almost the same. The main steps of the algorithm are: (i) Reschedule tasks from previous rounds with highest priority; (ii) Compute tasks sub-deadlines: the latest completion time that cannot be exceeded; (iii) Compute execution time and cost for each task on each resource; (iv) Each task is distributed to the resource with lowest execution time and lowest cost; (v) Allow the user to view a graph with the relation time-cost and to choose desired compromise; (vi) Repeat for next scheduling round.

A heuristic genetic approach is described in [21,22], with a slight improvement of execution time. The proposed algorithm generates an initial schedule for tasks using a heuristic algorithm such as Min–min (described above in [18]); computes parameters such as make span for the generated allocation; select nodes (scheduled resources) and uses traditional genetic techniques to get the best population. Another strategy used as optimization method is co-allocation. Co-allocation provides a schedule for tasks with dependencies, having as main purpose the efficiency of the schedule, in terms of load balancing and minimum time for the execution of the tasks [23].[24] describes Aneka's deadline based algorithm for provisioning resources. Aneka is a cloud application platform that uses various resources, ranging from public cloud to desktops grids. The presented algorithm supports QoS-aware execution in hybrid clouds [25], with an emphasis on scientific applications. Its authors also present results from running the algorithm in a real hybrid cloud, with applications that only require a small amount of data. The algorithm reduces the application execution times and reveals the need for historical information regarding previous executions of the application to be scheduled.

The authors of paper [26] present an algorithm for handling DAG type of tasks named ICPDP (Improved Critical Path using Descendant Prediction). The paper first presents and analyzes heuristics for scheduling tasks with dependencies in the form of a DAG, for several applications with different requirements in terms of computing time and resources. The proposed algorithm is integrated in the DIOGENES project and experimental results have been derived from that, showing the improvement that ICPDP brings to DAG scheduling. Furthermore, based on directed acyclic graph (DAG) in [4] is proposed a reliability-aware scheduling algorithm for precedence constrained tasks, which can achieve high quality of reliability for applications.

In [27] is presented an energy-aware scheduling system based on heuristics, aiming to compute optimal solutions in terms of balance between make span and energy efficiency. The problem is formulated as a multi-objective optimization, for both performance and energy. The authors target heterogeneous parallel systems and use real consumption to develop their heuristics. Changing several parameters in their configuration, the proposed solution managed a notable reduction of energy consumption, while slightly increasing the make span as well.

As opposed to the previous paper, who analyzed scheduling on the workflow type of application, [28] analyzes scheduling of batch jobs. Batch machines are specialized on jobs belonging to a certain family and can process a number of those jobs simultaneously. The authors use a mathematical model and use repositioning of whole batches as opposed to singular jobs. This technique gave better results than the best overall heuristic algorithm.

SLA based scheduling has been a challenge for Cloud platforms. In [29] is described a SLA-aware PaaS Cloud platform called Cloudcompaas, aimed at providing high-level metrics and closer to end-user perception. The platform also provides a framework that enables corrections of QoS violations, using elasticity. It manages the complete resource lifecycle, being able to sustain heterogeneous utilization patterns. The simulations show that Cloudcompass can achieve minimum cost and maximum efficiency for several workload profiles tested.

There are several scheduling algorithms and strategies adopted in private Clouds. The scheduling in *OpenStack* [30] framework is accomplished by the nova-scheduler. The main scheduling phases in the process are: 1. filtering available resources according to users requirements; and 2. weighting phase—the filtered hosts are applied a cost function depending on the input tasks and then sorted from the best to the worst. When using *OpenNebula* framework [31], the default available scheduling is related to the allocation of VMs on hosts. It uses a Rank Policy for that purpose: the hosts not fulfilling VMs requirements (memory or CPU) are excluded; the remaining hosts are evaluated using a configurable rank function; VMs are allocated to hosts with higher rank [32].

The most used Cloud simulator is CloudSim [13,33]. In CloudSim there are available default scheduling policies both for VMs allocation on hosts and for tasks allocation on processing elements. The simulator offers space-shared and time-shared policies for VMs and tasks provision and those two available policies may be used in every combination having different effects in tasks execution. A similar tool is iCanCloud—a flexible and scalable Cloud infrastructure simulator [34].

3. Proposed model for HySARC^2 algorithm

This section formally states the model we are going to use: we define two entities, *Tasks* and *Resources*, and describe their properties. We describe the steps of HySARC^2 algorithm and the scheduling heuristics used. The clustering model used by HySARC^2 represents the concept for our hybrid approach.

3.1. Theoretical model

Task represents a sequence of operations and is described by four parameters and the task set has the following formal description:

$$\mathcal{T} = \left\{T | T = (P_1^T, P_2^T, P_3^T, P_4^T)\right\} \tag{1}$$

where

- P_1^T is the CPU processing time needed by the job to complete on any resource;
- P_2^T is the IO time representing the time needed to read the input data from disk before it can start processing;
- P_3^T is the preemption flag (preemptive or non-preemptive);
- P_4^T is the deadline.

The *Tasks* from a set can be independent, with no restrictions imposed by other tasks for scheduling and execution phases, or can be modeled as DAGs (Directed Acyclic Graphs), where nodes represent computation and edges represent communication, data flow, between nodes.

In our model the task graph is represented by a DAG (Directed Acyclic Graph) as:

$$\mathcal{G} = \mathcal{G}(V, E, w, ew_n) \tag{2}$$

where:

- V is a set of nodes. We will refer to the nodes using the n_1, $n_2, \ldots$ notation; A node n_i encodes a task T_i and the graph $\mathcal{G}$ encodes a task set $\mathcal{T}$;
- E is a set of directed edges (dependencies), noted as $e(n_i, n_j)$;
- $w : V \to R_+$ is a function that associates a weight $w(n_i)$ to each node $n_i \in V$; $w(n_i)$ represents the execution time of the task T_i, which is represented by the node n_i in V;

- ew_n is a function $ew_n : E \to R_+$ that associates a weight to a directed edge; if n_i and n_j are two nodes in V, then $ew_n(n_i, n_j)$ denotes the inter-tasks communication time between T_i and T_j (the time needed for data transmission between processors that execute tasks T_i and T_j).

When two nodes are scheduled on the same resource R, the cost of the connecting edge becomes zero. In this model a scheduling heuristic is considered efficient if the *makespan* (maximum execution time for any resource) is short and respects resource constrains, such as a limited number of processors, memory capacity, available disk space, etc. Many types of scheduling algorithms for DAGs are based on the list scheduling technique.

Each task has an assigned priority, and scheduling is done according to a list priority policy: select the node with the highest priority and assign it to a suitable machine. According to this policy, two attributes are used to assign priorities:

- *t-level* (top-level) for n_i is the weight of the longest path from the source node (the node that start the execution) to n_i:

$$t\text{-}level(n_i) = \max_{n_j \in pred(n_i)} \left\{ t\text{-}level(n_j) + w(n_j) + ew_n(n_j, n_i) \right\} \quad (3)$$

- *b-level* (bottom-level) for n_i is the weight of the longest path from n_i to the exit node (the node that finish the execution):

$$b\text{-}level(n_i) = w(n_i) + \max_{n_j \in succ(n_i)} \left\{ b\text{-}level(n_j) + ew_n(n_i, n_j) \right\}. \quad (4)$$

The time-complexity for computing *t-level* and *b-level* is $O(|V| + |E|)$, so there are no penalties for the scheduling algorithms.

We can define the *ALAP* (*As Late As Possible*) attribute for a node n_i to measure how far the node's start-time can be delayed without increasing the *makespan*. This attribute will have an important role for load balancing constrains because it shows if we can delay the execution start of a task T_i:

$$ALAP(n_i) = \min_{n_j \in succ(n_i)} \left\{ ALAP(n_j) - ew_n(n_i, n_j) \right\} - w(n_i). \quad (5)$$

For a graph $\mathcal{G}$, the critical path (*CP*) is the weight of the longest path and it offers an upper limit for the scheduling cost. Algorithms based on *CP* heuristics produce on average the best results. They take into consideration the critical path of the scheduled nodes at each step. However, these heuristics can result in a local optimum, failing to reach the optimal global solution [35]. The *t-level* and the *b-level* are bounded from above by the length of the critical path.

The *Resource* set has the following formal description:

$$\mathcal{R} = \left\{ R | R = (P_1^R, P_2^R) \right\} \quad (6)$$

where

- P_1^R is the precessing speed representing the time needed to execute an atomic operation;
- P_2^R is the IO speed representing the time needed by a machine to read/write a unit of data from the disk. In this model we consider both times to be equal. In practice the read time is less than write time.

Note that it is possible to develop a more complex model that still fits our needs. For example, other properties that can be considered as extension for resources are: parallel (single, uniform or unrelated processors) or dedicated processors, network topology. We are using throughout the paper the term "cluster" of resources to denote a set of heterogeneous machines (VMs) that are connected in a network and can be viewed as a single system (an instance in public or private Cloud). This is the model of a datacenter, which is specific to Big Data or HPC processing.

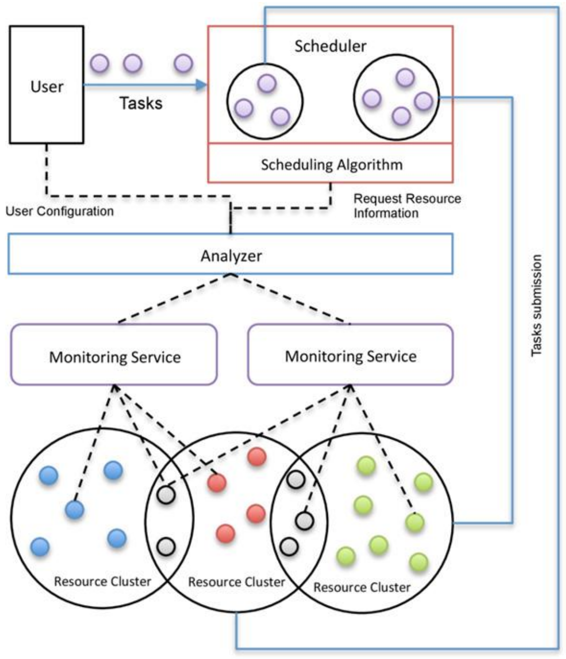

Fig. 1. Proposed Architecture used by HySARC^2.

3.2. HySARC^2 *architecture*

HySARC^2 design considers a framework with the following modules: Monitoring Service, Analyzer and Scheduler (Fig. 1). The general flow of HySARC^2 is as follows. We consider as input a set of tasks ($\mathcal{T}$ or $\mathcal{G}$). The resources from Cloud environment are monitored and grouped into clusters of resources. The tasks are allocated on the available resources according to different scheduling strategies used for each cluster of resources.

Monitoring Service module is designed as a background process. It finds the available resources and it is aware when a resource is added/removed to/from the system. The service monitors the resource characteristics and create the set (list) $\mathcal{R}$ of active resources. This set is used by the Analyzer module and Scheduler module without having to request information from the system each time an HySARC^2 algorithm instance is ran.

Analyzer module is used for clustering the resources and tasks according to user configuration or default predefined settings. We proposed the following behavior:

(a) the Analyzer supports user configuration. The user provides information about how many groups of tasks and resources should be created after the clustering phase. The default values are three clusters of resources (large, medium or small) and three clusters of tasks (CPU intensive, I/O intensive or mixed—both CPU and I/O intensive);
(b) next, the Analyzer gets the list of resources and properties from the Monitoring Service module, applies the clustering algorithm on the set of resources and labels each resource with the associated cluster (classification phase);
(c) the Analyzer receives the list of tasks and their properties from the user and applies the clustering algorithm. The tasks without dependencies are clustered as they are. For dependent task, we create clusters of DAGs by comparing them according with a "distance", which will be described in the next subsection;
(d) it provides to the Scheduler module the list of resource clusters and task clusters.

The **Scheduler** has the role of receiving input tasks and assign them to available resources. The Scheduler module workflow is:

1. The Scheduler receives the input tasks.
2. Next, the Scheduler sends the tasks to the Analyzer for clustering.
3. Further, the Scheduler receives from the Analyzer the clusters of tasks and available cluster of resources.
4. Finally, the Scheduler applies a hierarchical scheduling algorithm and send each task to the identified resource:
   A. the first step for scheduling a tasks cluster (a set of tasks) is assigning it to a cluster of resources.
   B. after that, the set of tasks is scheduled using a classical algorithm specific to selected cluster of resources.

The novelty of HySARC^2 algorithm is that different groups of resources are able to have different algorithms, more suitable for the resources and associated tasks properties, rather than to have a scheduling algorithm for all resources and tasks. This aspect deals with heterogeneous distributed computing environments, by finding the most suitable scheduling solution for a given context.

3.3. *Clustering Proposal for* HySARC^2

A clustering approach is going to be used for HySARC^2, the algorithm used being K-Means [36]. It is applied twice by the Analyzer module, once for the resources and once for the tasks. The abstract data input for K-Means algorithm is once, the available resources and second, the input tasks having different characteristics.

In order to apply K-Means algorithm, we must define the properties for tasks and resources taken into account by the clustering and also the "distance" between two elements in the set, as follows:

Tasks: estimated CPU processing time (P_1^T) and I/O operations time (P_2^T).
Resources: CPU processing power (P_1^R) and I/O operations speed (P_2^R).

The "distance" between two independent tasks or between two resources is necessary for identifying the "closest" cluster center. The "distance" highlights the similarity between entities. We define the same distance for independent tasks and resources as follows:

1. normalize the values of parameters along the entire set of entities (E denotes a task or a resource). We define the normalized value for property P_i^E, where $i = 1, 2$ as:

$$\tilde{P}_i^E = \frac{P_i^E}{\sum_K P_i^K} \tag{7}$$

2. the normalized parameters are considered as coordinates, so the distance between entity E_a and entity E_b having the properties P_1 and P_2 is the Euclidean distance:

$$distance(E_a, E_b) = \sqrt{\left(\tilde{P}_1^{E_a} - \tilde{P}_1^{E_b}\right)^2 + \left(\tilde{P}_2^{E_a} - \tilde{P}_2^{E_b}\right)^2}. \tag{8}$$

For a DAG we identify the graph structure to compute the "distance" used by K-Means, and take into consideration granularity and CCR (Communication to Computation Ratio) to compare two different DAGs.

Granularity g of a DAG $\mathcal{G}$ is defined as:

$$g(\mathcal{G}) := g(\mathcal{G}(V, E, w, ew_n)) = \min_{x\in V}\left\{\frac{w(x)}{\max\limits_{(x,i)\in E}\{ew_n(x,i)\}}\right\}. \tag{9}$$

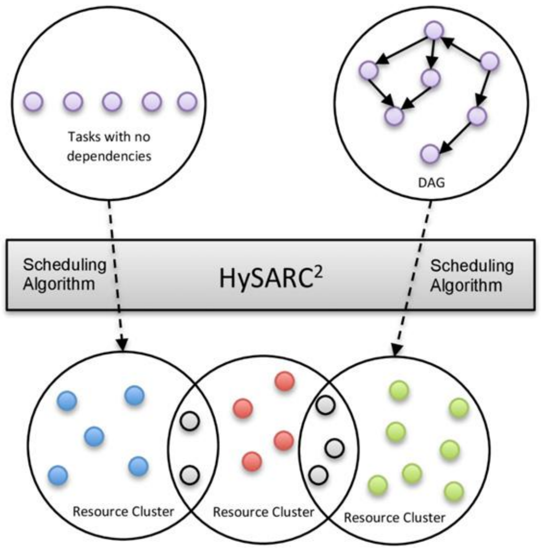

Fig. 2. Clustering phase for HySARC^2.

Depending on granularity, $\mathcal{G}$ can be coarse grained (the computation dominates the communication) or fine grained (the communication dominates the computation), so I/O bound term is equivalent with fine-grained and CPU-bound term is equivalent with coarse-grained.

CCR (Communication to Computation Ratio) can be defined as the average edge weight divided by the average node weight. We can consider also the max function for CCR computation, so we proposed the following two definitions:

$$CCR_s(\mathcal{G}) := CCR_s(\mathcal{G}(V, E, w, ew_n)) = \frac{|V| * \sum\limits_{(x,i)\in E} ew_n(x,i)}{|E| * \sum\limits_{x\in V} w(x)}, \tag{10}$$

$$CCR_m(\mathcal{G}) := CCR_m(\mathcal{G}(V, E, w, ew_n)) = \frac{\max\limits_{(x,i)\in E}\{ew_n(x,i)\}}{\max\limits_{x\in V}\{w(x)\}}. \tag{11}$$

CCR can help to judge the importance of communication in a task graph, which strongly determines the scheduling behavior. Based on CCR we classify task graphs in: $CCR < 1$—fine grained graph, $CCR \approx 1$—mixed, and $CCR > 1$—coarse grained graph.

Now, we can define the "distance" between two DAGs, $\mathcal{G}_a$ and $\mathcal{G}_b$ as:

$$distance(\mathcal{G}_a, \mathcal{G}_b) = |CCR(\mathcal{G}_a) - CCR(\mathcal{G}_b)|. \tag{12}$$

After the clustering phase is completed, we follow the actual scheduling of the resources for the input tasks, now grouped into clusters (see Fig. 2).

3.4. HySARC^2 *Scheduling algorithm*

The scheduling algorithm is applied by the Scheduler module in two steps:

STEP 1: associate groups of tasks with groups of resources, according to average parameters in tasks and resources groups. In other words, the clusters of tasks having a high average processing time required ("large" tasks) are assigned to resources with high computational capacity. The same reasoning applies to "small" tasks. For DAGs, coarse grained graphs are assigned to resources with high computational capacity and fine grained graph can be assigned to any type of clusters, considering that we have a homogeneous network infrastructure.

STEP 2: inside each group of resources run a specific scheduling algorithm in order to allocate the tasks/DAGs.

3.5. Scheduling algorithms used by HySARC^2

Independent Tasks approach. Given the fact that the tasks being processed by the scheduler are a set of independent tasks, we apply specific scheduling algorithms. The scheduler implements two scheduling algorithms for independent tasks, and alternates them in each cluster of resources, for analysis and comparison purposes:

1. *Shortest Job First* (SJF): associates with a task, its estimated CPU processing time ("small" job means having a low processing time), and as soon a resource is available, it assigns on it the shortest task in the waiting list. In order to achieve this more efficiently, the list of tasks is sorted ascending after the CPU processing time (see Algorithm 1).
2. *Earliest Deadline First* (EDF): associates with a task its deadline, and as soon a resource is available, it assigns on it the task with the nearest deadline in the waiting list. The tasks are kept into a priority list, the priorities are the inverse of the deadline, and the tasks with higher priority are scheduled sooner (see Algorithm 1).

Algorithm 1 Earliest Deadline First (EDF) / Shortest Job First (SJF)

```
1: procedure CLASSICAL_SCHEDULING($\mathcal{T}$, $\mathcal{R}$)
2:     sort tasks in $\mathcal{T}$:
       - descending after deadline ($P_4^T$) for EDF OR
       - ascending after CPU processing ($P_1^T$) time for SJF;
3:     while $\mathcal{T} \neq \emptyset$ do
4:         if $anyResourceAvailable(\mathcal{R}) == true$ then
5:             $R \leftarrow getRandomResourceAvailable(\mathcal{R})$;
6:             $T \leftarrow popTask(\mathcal{T})$;
7:             execute $T$ on $R$;
8:         end if
9:     end while
10: end procedure
```

We may observe that the difference between the two algorithms is how the tasks are being sorted in the waiting list. The list is being used as a stack.

Tasks with dependencies—DAG. There are several scheduling approaches based on list scheduling model:

1. *HLFET (Highest Level First with Estimated Times)* uses a hybrid approach of the list-based and level-based strategies. The algorithm schedules a task to a resource that allows the earliest start time [37].
2. *CCF (Cluster ready Children First)* is a dynamic scheduling algorithm based on lists. The graph is visited in topological order, and tasks are submitted as soon as scheduling decisions are taken. The algorithm assumes that when a task is submitted for execution it is inserted into the `RUNNING-QUEUE`. If a task is extracted from the `RUNNING-QUEUE`, all its successors are inserted into the `CHILDREN-QUEUE`. The running ends when the two queues are empty.
3. *Hybrid Re-mapper PS (Minimum Partial Completion Time Static Priority)* is a dynamic list scheduling algorithm specifically designed for heterogeneous environments. The set of tasks is partitioned into blocks so that tasks in a block do not have any data dependencies among them. Subsequently the blocks are executed one by one [38].
4. *MCP (Modified Critical Path)*—algorithm based on lists has two phases: the prioritization and selection of resources. Parameter used to prioritize nodes is ALAP (As Late As Possible) [39].
5. *ETF (Earliest Time First)*—algorithm based on keeping the processors as busy as possible. It computes, at each step, the earliest start times of all ready nodes and selects the one with the shortest start time.

We focus on Modified Critical Path (MCP) and Earliest Time First (ETF) because we are oriented towards high performance in the scheduling phase (see Algorithms 2 and 3).

Algorithm 2 Modified Critical Path (MCP) algorithm

```
1: procedure MCP($\mathcal{G}$, $\mathcal{R}$)
2:     for each $n_i \in \mathcal{G}$ do
3:         Compute the $ALAP(n_i)$;
4:     end for
5:     Create a node list $L = \emptyset$;
6:     for each node $n_i$ do
7:         Create a list $L_i = \emptyset$;
8:         $L_i = L_i \bigcup \{(n_i, ALAP(n_i))\}$;
9:         Sort $succs(n_i)$ in a descending order with respect to ALAP;
10:        for all $n_j \in succs(n_i)$ do
11:            $L_i = L_i \bigcup \{(n_j, ALAP(n_j))\}$;
12:        end for
13:        $L = L \bigcup L_i$;
14:    end for
15:    Sort these lists ($L$) in an ascending lexicographical order;
16:    repeat
17:        Extract $L_1$ the first node in the node list $L$;
18:        Schedule $L_1$ to a resource from $\mathcal{R}$ that allows the earliest execution;
19:                                          ▷ using the insertion approach;
20:        $L = L - L_1$;        ▷ Remove the node from the node list.
21:    until $L = \emptyset$;
22: end procedure
```

Algorithm 3 Earliest Time First (ETF) algorithm

```
1: procedure ETF($\mathcal{G}$, $\mathcal{R}$)
2:     for each $n_i \in \mathcal{G}$ do
3:         Calculate the static $b - level(n_i)$;
4:     end for
5:     $RN$ = entry nodes from $\mathcal{G}$;        ▷ the pool of ready nodes;
6:     repeat
7:         for each node $n_i \in RN$ do
8:             Calculate the earliest-start-time($n_i$) on each resource in $\mathcal{R}$;
9:         end for
10:        Pick the $(n_i, R)$ that gives the earliest time;
11:                                       ▷ using the non-insertion approach;
12:        Ties are broken by selecting the node $n_i$ with a higher static $b - level(n_i)$;
13:        Schedule the node to the corresponding resource;
14:        Add the newly ready nodes to the $RN$;
15:    until all nodes are scheduled
16: end procedure
```

4. Experimental methodology and results

One of the research challenges generally concerns the way in which multimedia information is gathered, analyzed, processed, represented and stored. We choose multimedia applications for

Table 1
Resources configurations used in test scenarios.

Test scenario	No. of VMs	No. of PEs
0	1	10
1	2	10
2	2	15
3	3	20
4	4	50
5	5	100
6	6	200
7	7	500
8	8	750

test case scenarios because they consist in both independent tasks and tasks with dependencies (workflows). More, the multimedia applications are both CPU intensive (they process a large amount of data) and I/O intensive (they access remote data). Several examples for these type of applications are: the case of large-scale video streams, like the one coming from a border surveillance outpost, a critical natural disaster or a nuclear accident like the one at Fukushima in 2011, a very large wide-distributed company who would like to deal with various multimedia content like documents, charts, contracts, video recordings of meetings, plans, etc., a modern museum that offers high quality multimedia mobile devices to all the visitors as audio/video guides, etc.

For the test cases, we considered tasks and resources with various requirements, as support for heterogeneous distributed computing modeling. The characteristics of the processing elements are:

- MIPS: 200, 400, 500, 800, 1000, 2000, 4000, 5000, 8000, 10000;
- RAM dimension: 512, 1024, 2048, 4096, 8192, 16384;
- we vary also the total storage value, but the values are not relevant, because we consider that when a task executes, all data are available on local storage.

Using CloudSim [13], we generate a maximum number of 1000 tasks, 1000 Processing Elements (PE) and 10 Virtual Machines (VM) with different number of cores. In each simulation we vary the number of virtual machines from 1 to 10 and the number of processing elements from 10 to 1000. The resources configurations used in our experimental simulations are presented in Table 1.

4.1. Tasks and resources clustering phase

We analyzed the Clustering Phase duration, along the above set of scenarios and for different grades of variability of the generated parameters (the parameters are in certain different grades similar or very different). In Fig. 3 we have the task clustering phase duration. In Fig. 4 we have the resource clustering phase duration. A large number of tasks produce an overhead and we can split the set of submitted tasks into multiple requests. For resource clustering we have similar times, so we can run periodically this procedure without any inconvenient. The overhead observed is justified because it slightly reduces the execution time.

4.2. Execution time

We analyze the average execution time of tasks along a combination of scenarios using a certain configuration (5 virtual machines and 100 processing elements): (i) we test by using or not the clustering algorithm; (ii) we also test with or without the default scheduling algorithm inside clusters; (iii) the tests are taken for three clusters of resources and three clusters of tasks or four clusters of resources and four clusters of tasks. In Fig. 5 we present the results for initial CloudSim Scheduling, only clustering, clustering and SJF algorithm. The conclusion is that the clustering phase

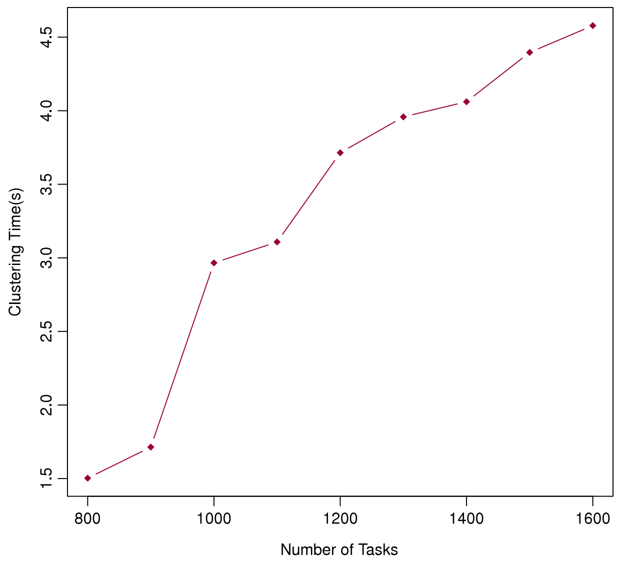

Fig. 3. Task clustering duration.

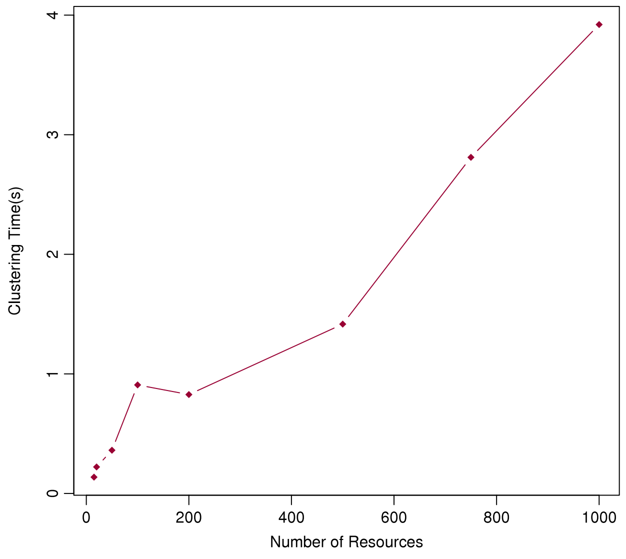

Fig. 4. Resource clustering duration.

Table 2
Workflows characteristics used in test scenarios.

DAG Id	No nodes	No edges	CCR_m	CCR_s	Granularity	Type
1	616	18 635	0.250	0.139	3.957	Coarse grained
2	628	19 491	0.245	0.140	4.000	Coarse grained
3	681	22 505	0.250	0.142	3.958	Coarse grained
4	686	22 817	0.998	0.998	0.991	Mixed
5	688	22 954	0.998	0.998	0.998	Mixed
6	707	24 118	1.007	1.007	0.988	Mixed
7	667	21 637	3.654	6.602	0.029	Fine grained
8	631	19 539	3.640	6.615	0.031	Fine grained
9	627	19 202	3.769	6.780	0.029	Fine grained

adds an overhead, but using a specialized scheduling algorithm we obtain a good improvement.

We built a DAG generator for testing the extension of the HySARC^2 for workflows. The DAG generator uses multiple parameters: minimum/maximum size for a node, minimum/maximum cost for an edge, the probability to have an edge between two random nodes. These parameters may be easily configured, so we can

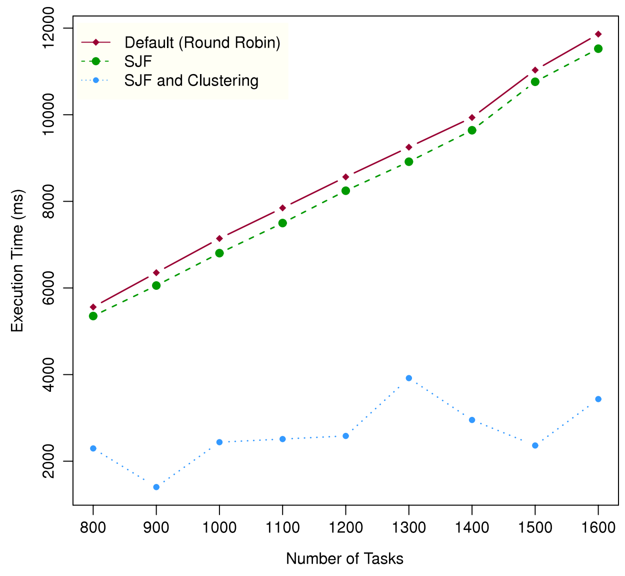

Fig. 5. Execution time comparison (simulation time/steps) for independent tasks.

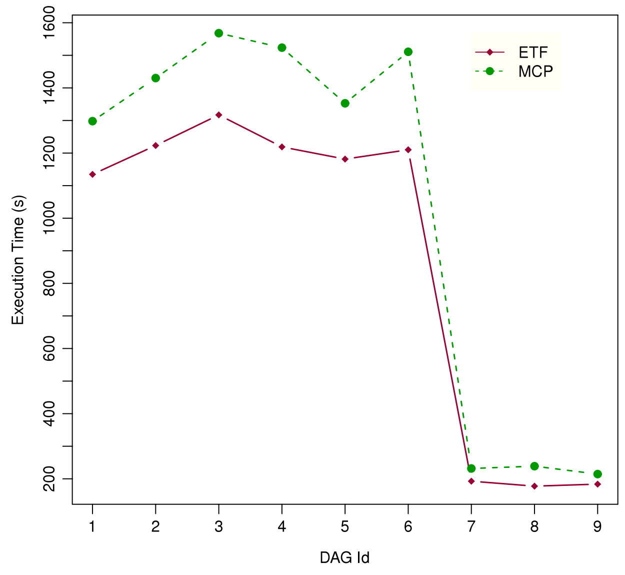

Fig. 6. DAG execution results on high performance IO resources.

generate highly heterogeneous workflows, with different granularity and CCR values.

We analyzed a set of nine graphs: IO intensive (fine grained, $CCR > 1$), CPU intensive (coarse grained, $CCR < 1$) and also balanced (mixed, $CCR \approx 1$). All workflows are presented in Table 2. We used both ETF and MCP algorithms for scheduling, on resources with different IO and CPU characteristics and compared the results for each DAG.

All generated workflows consider many processing tasks (similar to multimedia filtering applications)—around 675 tasks with different dependence constraints, depending on the data locality (see Table 2). The use of workflows allows us to consider the modularization of the task space, scalability, advance decisions regarding data movement, incorporation of periodic tasks execution, so the comparison between different workflows and classifying them into clusters offer a huge benefit for an enterprise system considering resource usage and costs.

We used different types of resources for DAG scheduling and execution, both high performance CPU and IO resources. The DAGs considered in our experiments are the ones described above, in

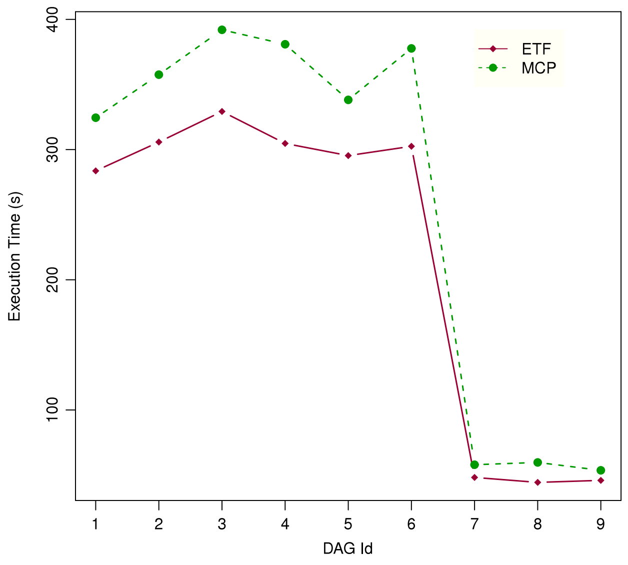

Fig. 7. DAG execution results on high performance CPU resources.

Table 2. As shown in the results of our experiments (see Figs. 6 and 7), the execution time has the same profile for both types of resources (High Performance IO resources or High Performance CPU resources). The difference is that the execution time decreases 4 times on high performance CPU resources. Also, the test scenarios 7, 8 and 9 are for fine grained task graphs and have better results than scenarios 1–7, this being justified by the uniform modeling of communication networks at a higher speed.

4.3. Scalability

We analyze the scalability of HySARC^2 compared to the default CloudSim scheduling and a classical scheduling algorithm (SJF or EDF). We observe the average execution time for scheduling and execution of 800 tasks, along the entire above set of scenarios for resources configuration (we consider a variable number of virtual machines and processing elements). Also, the resources have variable values for the generated parameters – computational and IO characteristics – therefore the simulated environment is a heterogeneous one. In Fig. 8 we have the average execution time for initial CloudSim scheduling, for clustering and one of the two scheduling algorithms (SJF or EDF). We can conclude that by adding HySARC^2 in a specific Cloud environment the scalability is preserved.

5. HySARC^2 integration in real Cloud platforms

HySARC^2 was developed as a distinct entity inside the CloudSim environment, therefore the proposed architecture is modular and could be integrated in a real Cloud platform. The main components of our architecture are highlighted as follows:

- The Monitoring Service maintains the information regarding the existing resources;
- The Analyzer performs the clustering phase previous to the actual scheduling;
- The Scheduler applies different scheduling algorithms specific to each cluster of tasks.

The integration of HySARC^2 in OpenStack consists in two main steps:

- integrate the Monitoring Service and
- integrate the Analyzer and the Scheduler.

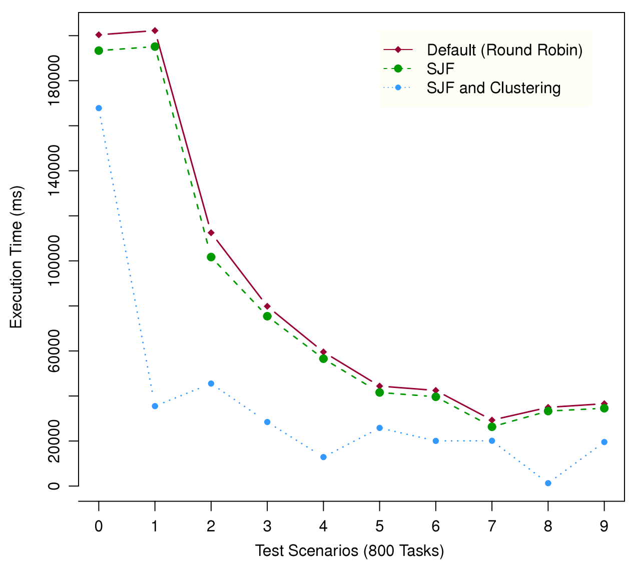

Fig. 8. Scalability. The analysis considers resources characteristics presented in Table 1 and a fixed number of independent tasks.

The Monitoring Service may be deployed as a daemon on the provider system, gathering information about existing resources at system start up, and receive notifications each time a modification occurs in the resource configuration (a resource is added, removed or updated).

In order to integrate the new scheduling logic, we can start from the implementation of the Nova Filter Scheduler. We should extend the Scheduler class in `nova.*` package and override schedule run instance and select destinations methods, by implementing our approach.

6. Conclusion

An extension to the HySARC^2 scheduling algorithm is proposed in this paper. Our approach considers clustering of the available resources and received tasks, before the phase of resource allocation. For tasks with no dependencies we proposed a metric to compare them in the clustering phase. For workflows, we compute CCR (Communication for Computation Ratio) and granularity. The proposed algorithm is based on traditional scheduling algorithms. We used Shortest Job First and Earliest Deadline First algorithms for independent tasks, and Earliest Time First and Modified Critical Path algorithms for DAGs. Clustering of resources and tasks along with computing DAG specific parameters bring efficiency to the scheduling, even these pre-processing steps of tasks and resources introduce a certain overhead. As shown in the experimental results, the overhead justifies itself because it slightly reduces the processing time.

Overall, this paper has the following contributions: it proposes a hybrid approach for tasks scheduling in HDC considering both tasks and resources clustering; our hybrid scheduling model is based on using different scheduling strategies, selected by taking into consideration both the heterogeneity of computing resources, and application tasks and/or flows; it provides a model for adaptive and dynamic clustering considering the abstract modeling of HDC resources; it extends CloudSim to consider these approaches and integrates four scheduling strategies.

For future work we will consider the scheduling algorithms that inspect the dynamic behavior of the resources and algorithms that allow tasks to be preempted according to a given priority and dynamically adapt the scheduling algorithm [40].

Acknowledgments

The research presented in this paper is supported by projects: "*SideSTEP—Scheduling Methods for Dynamic Distributed Systems: a self-* approach*", ID: PN-II-CT-RO-FR-2012-1-0084; *CyberWater* grant of the Romanian National Authority for Scientific Research (PN-II-PT-PCCA-2011-3.1-0602), CNDI-UEFISCDI, project number 47/2012; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow—PN-II-PT-PCCA-2013-4-0321; *clueFarm*: Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms—PN-II-PT-PCCA-2013-4-0870.

References

[1] D.C. Marinescu, Cloud Computing: Theory and Practice, first ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN: 0124046274, 2013, 9780124046276.
[2] A. Castiglione, M. Gribaudo, M. Iacono, F. Palmieri, Exploiting mean field analysis to model performances of Big Data architectures, Future Gener. Comput. Syst. (ISSN: 0167-739X) 37 (0) (2014) 203–211. http://dx.doi.org/10.1016/j.future.2013.07.016. URL: http://www.sciencedirect.com/science/article/pii/S0167739X13001611, special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for Big Data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications.
[3] E. Barbierato, M. Gribaudo, M. Iacono, Modeling Apache Hive based applications in Big Data architectures, in: Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools, ValueTools'13, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, ISBN: 978-1-936968-48-0, 2013, pp. 30–38. http://dx.doi.org/10.4108/icst.valuetools.2013.254398.
[4] X. Tang, K. Li, R. Li, B. Veeravalli, Reliability-aware scheduling strategy for heterogeneous distributed computing systems, J. Parallel Distrib. Comput. (ISSN: 0743-7315) 70 (9) (2010) 941–952. http://dx.doi.org/10.1016/j.jpdc.2010.05.002.
[5] V. Bharadwaj, D. Ghose, T.G. Robertazzi, Divisible load theory: a new paradigm for load scheduling in distributed systems, Cluster Comput. (ISSN: 1386-7857) 6 (1) (2003) 7–17. http://dx.doi.org/10.1023/A:1020958815308.
[6] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, Above the clouds: a Berkeley view of cloud computing, Dept. Electrical Eng. and Comput. Sciences, University of California.
[7] J. Berlińska, M. Drozdowski, Scheduling divisible MapReduce computations, J. Parallel Distrib. Comput. (ISSN: 0743-7315) 71 (3) (2011) 450–459. http://dx.doi.org/10.1016/j.jpdc.2010.12.004.
[8] M.M. Sharma, A. Bala, Survey paper on workflow scheduling algorithms used in cloud computing, Int. J. Inf. Comput. Technol. 4 (10) (2014) 997–1002.
[9] J. Yu, R. Buyya, A taxonomy of workflow management systems for grid computing, J. Grid Comput. 3 (3-4) (2005) 171–200.
[10] M. Mattoso, K. Ocaña, F. Horta, J. Dias, E. Ogasawara, V. Silva, D. de Oliveira, F. Costa, I. Araújo, User-steering of HPC workflows: state-of-the-art and future directions, in: Proceedings of the 2Nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, SWEET'13, ACM, New York, NY, USA, ISBN: 978-1-4503-2349-9, 2013, pp. 4:1–4:6. http://dx.doi.org/10.1145/2499896.2499900. URL: http://doi.acm.org/10.1145/2499896.2499900.
[11] H.-J. Jiang, K.-C. Huang, H.-Y. Chang, D.-S. Gu, P.-J. Shih, Scheduling concurrent workflows in HPC cloud through exploiting schedule gaps, in: Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing—Volume Part I, ICA3PP'11, Springer-Verlag, Berlin, Heidelberg, ISBN: 978-3-642-24649-4, 2011, pp. 282–293. URL: http://dl.acm.org/citation.cfm?id=2075416.2075442.
[12] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea, HySARC2: hybrid scheduling algorithm based on resource clustering in cloud environments, in: J. Kołodziej, B. Di Martino, D. Talia, K. Xiong (Eds.), Algorithms and Architectures for Parallel Processing, in: Lecture Notes in Computer Science, vol. 8285, Springer International Publishing, ISBN: 978-3-319-03858-2, 2013, pp. 416–425. http://dx.doi.org/10.1007/978-3-319-03859-9_36.
[13] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exp. (ISSN: 0038-0644) 41 (1) (2011) 23–50. http://dx.doi.org/10.1002/spe.995.
[14] S.-Y. Jing, S. Ali, K. She, Y. Zhong, State-of-the-art research study for green cloud computing, J. Supercomput. (ISSN: 0920-8542) 65 (1) (2013) 445–468. http://dx.doi.org/10.1007/s11227-011-0722-1.
[15] J. Kołodziej, F. Xhafa, Modern approaches to modeling user requirements on resource and task allocation in hierarchical computational grids, Int. J. Appl. Math. Comput. Sci. (ISSN: 1641-876X) 21 (2) (2011) 243–257. http://dx.doi.org/10.2478/v10006-011-0018-x.

[16] P. Leitner, W. Hummer, B. Satzger, C. Inzinger, S. Dustdar, Cost-efficient and application SLA-aware client side request scheduling in an infrastructure-as-a-service cloud, in: Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, CLOUD'12, IEEE Computer Society, Washington, DC, USA, ISBN: 978-0-7695-4755-8, 2012, pp. 213–220. http://dx.doi.org/10.1109/CLOUD.2012.21.
[17] L. Wu, S.K. Garg, R. Buyya, SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments, in: Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID'11, IEEE Computer Society, Washington, DC, USA, ISBN: 978-0-7695-4395-6, 2011, pp. 195–204. http://dx.doi.org/10.1109/CCGrid.2011.51.
[18] S. Parsa, R. Entezari-Maleki, RASA: A New Grid Task Scheduling Algorithm, JDCTA (2009) 91–99.
[19] A. Olteanu, F. Pop, C. Dobre, V. Cristea, A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems, Comput. Math. Appl. 69 (9) (2012) 1409–1423.
[20] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, Y. Yang, A compromised-time-cost scheduling algorithm in SwinDeW-C for instance-intensive cost-constrained workflows on a cloud computing platform, Int. J. High Perform. Comput. Appl. 24 (4) (2010) 445–456.
[21] K. Kaur, A. Chhabra, G. Singh, Heuristics based genetic algorithm for scheduling static tasks in homogeneous parallel system, Int. J. Comput. Sci. Secur. 4 (2) (2010) 183–198.
[22] J. Kołodziej, F. Xhafa, Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population, Future Gener. Comput. Syst. (ISSN: 0167-739X) 27 (8) (2011) 1035–1046. http://dx.doi.org/10.1016/j.future.2011.04.011.
[23] D. Moise, E. Moise, F. Pop, V. Cristea, Resource coallocation for scheduling tasks with dependencies, in grid, in: HiPerGRID Workshops Proceeding, Bucharest, Romania, ISSN: 2065-0701.
[24] C. Vecchiola, R.N. Calheiros, D. Karunamoorthy, R. Buyya, Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka, Future Gener. Comput. Syst. (ISSN: 0167-739X) 28 (1) (2012) 58–65. http://dx.doi.org/10.1016/j.future.2011.05.008.
[25] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Comput. 13 (5) (2009) 14–22.
[26] B. Simion, C. Leordeanu, F. Pop, V. Cristea, A hybrid algorithm for scheduling workflow applications in grid environments (ICPDP), in: Proc. of the 2007 OTM Confederated Int. Conf. on On the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS—Volume Part II, OTM'07, Springer-Verlag, Berlin, Heidelberg, ISBN: 3-540-76835-1, 2007, pp. 1331–1348. 978-3-540-76835-7. URL: http://dl.acm.org/citation.cfm?id=1784707.1784728.
[27] J.J. Durillo, V. Nae, R. Prodan, Multi-objective energy-efficient workflow scheduling using list-based heuristics, Future Gener. Comput. Syst. (ISSN: 0167-739X) 36 (0) (2014) 221–236. http://dx.doi.org/10.1016/j.future.2013.07.005. URL: http://www.sciencedirect.com/science/article/pii/S0167739X13001507 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.
[28] E. Cakici, S.J. Mason, J.W. Fowler, H.N. Geismar, Batch scheduling on parallel machines with dynamic job arrivals and incompatible job families, Int. J. Prod. Res. 51 (8) (2013) 2462–2477. http://dx.doi.org/10.1080/00207543.2012.748227.
[29] A. García García, I. Blanquer Espert, V. Hernández García, SLA-driven dynamic cloud resource management, Future Gener. Comput. Syst. (ISSN: 0167-739X) 31 (2014) 1–11. http://dx.doi.org/10.1016/j.future.2013.10.005.
[30] K. Jackson, OpenStack Cloud Computing Cookbook, Packt Publishing, ISBN: 1849517320, 2012, 9781849517324.
[31] D. Milojicic, I.M. Llorente, R.S. Montero, OpenNebula: a cloud management tool, IEEE Internet Comput.
[32] P. Sempolinski, D. Thain, A comparison and critique of Eucalyptus, OpenNebula and Nimbus, in: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, CLOUDCOM'10, IEEE Computer Society, Washington, DC, USA, ISBN: 978-0-7695-4302-4, 2010, pp. 417–426. http://dx.doi.org/10.1109/CloudCom.2010.42.
[33] R. Buyya, R. Ranjan, R.N. Calheiros, Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: challenges and opportunities, in: Int. Conf. on High Performance Computing & Simulation, 2009, HPCS'09, IEEE, 2009.
[34] A. Núñez, J.L. Vázquez-Poletti, A.C. Caminero, G.G. Castañé, J. Carretero, I.M. Llorente, iCanCloud: a flexible and scalable cloud infrastructure simulator, J. Grid Comput. (ISSN: 1570-7873) 10 (1) (2012) 185–209. http://dx.doi.org/10.1007/s10723-012-9208-5.
[35] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. (ISSN: 0360-0300) 31 (1999) 406–471.
[36] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, University of California Press, Berkeley, Calif., 1967, pp. 281–297. URL: http://projecteuclid.org/euclid.bsmsp/1200512992.
[37] S. Jin, G. Schiavone, D. Turgut, A performance study of multiprocessor task scheduling algorithms, J. Supercomput. (ISSN: 0920-8542) 43 (1) (2008) 77–97. http://dx.doi.org/10.1007/s11227-007-0139-z.
[38] A. Gandhi, H. Akkary, R. Rajwar, S.T. Srinivasan, K. Lai, Scalable load and store processing in latency tolerant processors, in: ISCA'05: Proceedings of the 32nd Annual International Symposium on Computer Architecture, IEEE Computer Society, Washington, DC, USA, ISBN: 0-7695-2270-X, 2005, pp. 446–457.
[39] S. Bansal, P. Kumar, K. Singh, An improved two-step algorithm for task and data parallel scheduling in distributed memory machines, Parallel Comput. (ISSN: 0167-8191) 32 (10) (2006) 759–774.
[40] N. Bessis, S. Sotiriadis, V. Cristea, F. Pop, Modelling requirements for enabling meta-scheduling in inter-clouds and inter-enterprises, in: Intelligent Networking and Collaborative Systems, INCoS, 2011, pp. 149–156.

Mihaela-Andreea Vasile, Computer Science diplomat engineer, is a master student on Parallel and Distributed Computing Systems program at University Politehnica of Bucharest, Faculty of Automatic Control and Computers, Computer Science Department. She is an active member of Distributed Systems Laboratory. Her research interests are in Cloud System and Big Data, especially in resource-aware scheduling, multi-criteria optimization, Cloud middleware tools, in-memory processing, Big Data applications design and implementation.

Florin Pop received his Ph.D. in Computer Science at the University Politehnica of Bucharest in 2008. He received his M.Sc. in Computer Science in 2004 and the Engineering degree in Computer Science in 2003, at the same University. He is Associate Professor within the Computer Science Department and also an active member of Distributed System Laboratory. His research interests are in scheduling and resource management, multi-criteria optimization methods, Grid middleware tools and applications development, prediction methods, self-organizing systems, contextualized services in distributed systems. He is the author or co-author of more than 150 publications (books, chapters, papers in internationals journals and well-established and ranked conferences). He served as guest-editor for *International Journal of Web and Grid Services* and he is Managing Editor for *International Journal of Grid and Utility Computing*. He was awarded with "Magna cum laude" distinction for his results during his Ph.D., one IBM Faculty Award in 2012 or the project "*CloudWay—Improving resource utilization for a smart Cloud infrastructure*", two Prizes for Excellence from IBM and Oracle (2008 and 2009), *Best young researcher in software services Award*, FP7 SPRERS Project, Strengthening the Participation of Romania at European R&D in Software Services in 2011 and two Best Paper Awards. He worked in several international (EGEE III, SEE-GRID-SCI, ERRIC) and national research projects in the distributed systems field as coordinator and member as well. He is a senior member of the IEEE, ACM and euroCRIS.

Radu-Ioan Tutueanu, Computer Science diplomat engineer, is a master student on Parallel and Distributed Computing Systems program at University Politehnica of Bucharest, Faculty of Automatic Control and Computers, Computer Science Department. He is an active member of Distributed Systems Laboratory. His research interests and orientation are in Cloud System and Real-Time System, especially in communications and agreements protocols, multi-criteria optimization, Cloud middleware tools, multimedia processing, Cloud applications design and implementation.

Valentin Cristea is a professor (since 1993) of the Computer Science and Engineering Department of the University Politehnica of Bucharest, and Ph.D. supervisor in the domain of Distributed Systems. His main fields of expertise are large scale distributed systems, cloud computing and e-services. He is co-Founder and Director of the National Center for Information Technology of UPB, and has a long history of experience in the development, management and/or coordination of international and national research projects. He led the UPB team in COOPER (FP6), datagrid@work (INRIA "Associate Teams" project), CoLaborator project for building a Center and collaborative environment for high-performance computing in Romania, distributed dependable systems project DEPSYS, and others. He co-supervised the UPB Team in European projects SEE-GRID-SCI (FP7) and EGEE (FP7). The research results have been published in more than 230 specialist papers in international journals or peer-reviewed proceedings, and more than 30 books and book chapters. He organized several scientific Workshops and Conferences. In 2003 and 2011 he received the IBM faculty award for research contributions in e-Service and Smart City domains. He is a member of the Romanian Academy of Technical Sciences.

Joanna Kołodziej (Ph.D. in Computer Science) graduated in Theoretical Mathematics from the Jagiellonian University in Cracow (Poland) in 1992, where she also obtained the Ph.D. in Theoretical Computer Science in 2004. In the period of 1992–1997 she worked as a project manager and senior CAD/CAM project manager in Petroleum Engineering (Bipronaft Cracow and INES Project Studio). She joined the University of Bielsko-Biala (Poland) in 1997 as an assistant professor and now works as an adjunct professor in computer science at the Department of Mathematics and Computer Science. She currently serves as a Director of Studies in Computer Science at the University of Bielsko-Biala and has served as an International Affairs Coordinator at the Faculty of the Mechanical Engineering and Computer Science, University of Bielsko-Biala in the period of 2008–2010. She is a full professional member of ACM and SIGEVO group. She is also a research fellow in Intelligent Information Systems Group at AHG University of Science and Technology, Cracow (Poland). The main topics of her research are evolutionary computation, mathematical modeling of stochastic processes, grid and cloud computing, intelligent networking, scalable computation, multi-agent systems, global optimization metaheuristics. She has published in international journals, books and conference proceedings of the research area. She is a guest editor of the '*Intelligent Decision Systems in Large-Scale Distributed Environments*' (Springer) book and 6 special issues of highly indexed journals in the domain, including 'Knowledge Engineering Review' (Cambridge Un. Press). She is serving as the editorial board member of several journals in her research area and as Managing Editor of International Journal of Space-Based and Situated Computing (Inderscience). She serves as a reviewer for the major journals in her research domain. She has served and is currently serving as General Co-Chair and General Program Co-Chair of several international conferences and workshops including PPSN 2010, ECMS 2011, CISIS 2011, 3PGCIC 2011, ICLS 2011, SCALSOL 2011, SCOPIN 2011. She also serves as a track chair and IPC member of many top conferences on evolutionary computing, AI, agent-based systems, high-performance and scalable computing, intelligent networking, grid and cloud computing including CEC 2008, GECCO 2010–2011, IEEE AINA 2011, IACS 2008–2009, ICAART 2009–2010, HPCC 2010, WICT 2011. She has been awarded for the best MSD Thesis in Theoretical Mathematics by Polish Mathematical Society in 1992 and for the best Ph.D. Thesis in Computer Science, Physics and Mathematics by The Foundation for Polish Science in 2004.