Orchestrating Heterogeneous and Dynamic Computing Systems with Publish-Subscribe

Leila Abidi*, Christophe Cérin*, Jonathan Lejeune*, Yanik Ngoko* and Walid Saad*†

Université de Paris 13, LIPN and University of Tunis, LATICE†*
** 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France*
† ENSIT, 5 av. Taha Hussein, B.P. 56, Bab Mnara, Tunis, Tunisia
Email:{leila.abidi,christophe.cerin,jonathan.lejeune,yanik.ngoko,walid.saad}@lipn.univ-paris13.fr

***Abstract*—In this paper, we consider the problem of orchestrating a scientific workflow in a highly heterogeneous and dynamic environment. In such environments, we promote the development of opportunistic strategies that can automatically adjust the processing of tasks to changes in the environment, according to global objectives defined by the user. Our work shows how we can develop an opportunistic orchestrator in extending an existing desktop grid middleware (RedisDG) and supporting the Publish/Subscribe paradigm. The implementation of this paradigm introduces different challenging problems, among them the design of effective solutions for controlling the allocation of tasks. We solve this problem in designing algorithms that depend on different strategies and reflecting different points of view for the SLA metrics such as load, fairness, energy consumption of the computing nodes. A multi-criteria approach for the allocation is also introduced in this paper as well as an experimental study for the dynamicity of the RedisDG system. In this latter case, nodes can enter or leave the system in a controled manner at any time. Then we conduct experiments on the Grid5000 testbed in executing the MONTAGE workflow from the NASA and also the ADAPT workflow (Computational Fluid Dynamics) from our laboratory. Finally we analyze our experimental results and we sketch perspectives.**

***Index Terms*—Cloud applications and services, Scientific Workflow as a Service, Resource management and optimization in the cloud, Scheduling and allocation, Publish-Subscribe paradigm.**

1. Motivation and scientific positioning

In this paper, we deal with the execution of a scientific workflow in a highly dynamic and heterogeneous infrastructure. The workflow execution is a pretext for presenting a general framework and architecture for highly dynamic and heterogeneous infrastructures. The execution is also controled by the RedisDG system in order to make our ideas concrete.

The context we consider is of particular interest for the development of extreme edge and edge cloud. A hard question, that the dynamicity causes here, is that given a workflow to schedule, we do not have any a-priori knowledge on the resources that are available. To address it, we propose to implement a Publish/Subscribe-based mechanism for resource discovery and allocation. The mechanism is implemented in a prior system we developed for workflow orchestration in deskop grid environment: the RedisDG system.

The Publish-Subscribe paradigm is an asynchronous mode for communicating between entities [1]. Some users, namely the subscribers or clients or consumers, express and record their interests under the form of subscriptions, and are notified later by another event produced by other users, namely the producers. This communication mode is multipoint, anonymous and implicit. Thus, it allows spatial decoupling (the interacting entities do not know each other), and time decoupling (the interacting entities do not need to participate at the same time). The total decoupling between the production and the consumption of services increases the scalability by eliminating many sorts of explicit dependencies between participating entities.

Given a workflow to schedule, our 'RedisDG Publish-Subscribe' orchestrator proceeds as follows: first, tasks or subset of tasks that could be processed are selected (depending on tasks and data dependencies). Then, it publishes events for the execution of the tasks. Subscribers start to answer and at a *chosen date*, the orchestrator decides to deploy the tasks with the list of subscribers it knows. There are two challenges here: the first one is to decide on the *date* at which we consider that the sub-list of subscribers we have is enough. Indeed, the more we wait, the more we have subscribers, but the more we globally delay the processing of the workflow. The second challenge is to decide on the best allocation for the subscribers. In this paper, we propose a formalization of this problem and make an experimental evaluation of several heuristics based on the energy,

fairness and load of the computing nodes. Data-aware heuristics [2] for RedisDG are not considered in this paper because of the page limit.

We do believe that, as the resources are highly dynamic, the decisions on resources must be automated in an *opportunistic way*. Our intent is to build such a system that will guarantee that: the workflow is processed quickly and that resources are intensively and fairly used. Here, the fairness is important because the dynamic context we consider can be a volunteer computing setting where users have rewards proportionally to the computing time that they offer for the processing of jobs. Note that availability problems may also appear with the Amazon or other major cloud services providers context. The Spot instances of AWS is launched when your bid exceeds the current Spot market price, and will continue run until you choose to terminate it, or until the Spot market price exceeds your bid.

The contributions of the papers are twofold. First we provide with a generic architecture for dealing with highly dynamic and heterogeneous computing systems. Second, as a concrete contribution, we introduce an extended version of the RedisDG workflow engine equipped with a series of *new scheduling algorithms*, including a multicriteria one, for a better usage of *dynamic* computing resources. The initial scheduling decisions implemented in RedisDG was based only on the FCFC (First Come First Serve) policy. More efficient policies for controlling multiple SLA constraints (*load, fairness, energy*) are introduced in this paper. We also provide with intensive experimentation, on the Grid5000 testbed, to analyze and to validate the proposed algorithms and the impact of the middleware on performance. Experiments illustrate the different situations in which our new RedisDG system can be faced to. They also illustrate how our measures and solutions can be fine.

Indeed, we support the thesis that for building systems for heterogeneous and highly dynamic environments we need to be compliant with:

1) a publish-subscribe layer for the orchestration of the components of the system;
2) a set of opportunistic strategies for allocating tasks that are also based on the publish-subscribe layer;
3) a small number of software dependencies for the system and the ability to deploy the system and its applications on demand. This point is not specifically discussed in this paper but we promoted the 'easy to use', and systems that can be deployed without a system administrator[1].

This paper is organized as follows. After this introduction about the context of our work, we present in Section 2 our architecture for dealing with highly dynamic and heterogeneous systems. Section 3 introduces some related works on scheduling. Section 4 defines some concepts that will be used later in the paper and related to scheduling. Section 5 is devoted to scheduling mechanisms, and we introduce some heuristics. Section 6 is about the experimental evaluation. We start by introducing our use cases: the MONTAGE and the ADAPT workflows, and then we describe the different scenarios of experimentation. Section 7 concludes the paper and opens on future work.

2. Architecture

2.1. A classification of computing systems

In this subsection we introduce computing architectures, at the academic level, through recent advances and examples.

2.1.1. Cloud. We do not comment here the well known cloud services as they are provided by Amazon, Google, Microsoft... but we do rather make a focus on the Chameleon academic project[2]. This project is for a broad range of supported research that includes developing Platforms as a Service, creating new and optimizing existing Infrastructure as a Service components, investigating software-defined networking, and optimizing virtualization technologies.

To summarize, it is a project for doing research on Systems like the Grid5000 testbed in France. One objective is to offer basic services, on top of which we can run an experiment or build a new service for the purpose of studying them. To effectively support Computer Science experiments Chameleon offers bare metal reconfigurability on most of the hardware.

For instance Chameleon offers the OpenStack Load Balancer as a Service. The OpenStack networking component, Neutron, includes a Load Balancer as a Service (LBaaS). This service lets you configure a load balancer that runs outside of your instances and directs traffic to your instances. A common use case is when you want to use multiple instances to serve web pages and meet performance or reliability goals.

The Chameleon Infrastructure (CHI) is made of 65% of OpenStack, 10% of Grid5000 and 25% of "special sauce". In our case we also experiment on a very experimental testbed that introduces new challenges, for instance the deployment of the system under study. We do not deal, in this paper, with such consideration, as said in the introduction.

2.1.2. Grid. The DIET was initially designed as a software for grid-computing. We use it here to illustrate advanced grid middleware since DIET is now able to request for resources in clouds. It was designed for high-performance computing. It is currently developed by INRIA, ÃL'cole Normale SupÃl'rieure de Lyon, CNRS,

1. https://lipn.univ-paris13.fr/ abidi/CloudComDemo/Tuto-RedisDG-Vagrant.pdf

2. https://www.chameleoncloud.org/about/chameleon/

Claude Bernard University Lyon 1, and the SysFera company. Like NetSolve/GridSolve and Ninf, DIET is compliant with the GridRPC standard from the Open Grid Forum.

Basically, DIET's architecture is composed of:

- a client - the application that uses DIET to solve problems. Clients can connect to DIET from a web page or through an API or compiled program.
- a Master Agent (MA) that receives computation requests from clients. The MA then collects computation abilities from the servers and chooses one based on scheduling criteria. The reference of the chosen server is returned to the client. A client can be connected to an MA by a specific name server or a web page that stores the various MA locations. A multi-MA extension was developed by the University of Franche-ComtÃl'. Those Master Agents are connected by a communication graph. Several DIET platforms are shared by interconnecting their respective Master Agent (MA).
- a Local Agent (LA) that aims at transmitting requests and information between MAs and servers. The information stored on an LA is the list of requests and, for each of its subtrees, the number of servers that can solve a given problem and information about the data distributed in this subtree. Depending on the underlying network topology, a hierarchy of LAs may be deployed between an MA and the servers.
- a Server Daemon (SeD) that is the point of entry of a computational server. It manages a processor or a cluster. The information stored on a SeD is the list of the data available on a server (possibly with their distribution and the way to access them), the list of the problems than can be solved on it, and all the information concerning its load (e.g., CPU capacity, available memory).

For workflow management, DIET uses an additional entity called MA DAG. This entity can work in two modes: one in which it defines a complete scheduling of the workflow (ordering and mapping), and one in which it defines only an ordering for the workflow execution. Mapping is then done in the next step by the client, using the Master Agent to find the server where the workflow services should be run.

Recently in [3] Caron and all. introduced a differential evolution approach for cloud workflow placements under simultaneous optimization of multiple objectives. One objective is regarding the energy.

Data management is provided to allow persistent data to stay within the system for future re-use. This feature avoids unnecessary communication when dependencies exist between different requests.

As an extension to the DIET platform, the DIET team has added a new type of service daemon: SeD Cloud. It offers an access to the a library providing the Amazon EC2 interface. By using the SeD Cloud, the DIET user can access to virtual resources from an IaaS Cloud that implements the Amazon EC2 interface.

2.1.3. Internet of things.

2.1.4. Desktop Grid. We also deal with Desktop Grid [4] systems because they represent first an alternative to supercomputers and parallel machines. They offer computing power at low cost. Desktop grids (DGs) are built out of commodity PCs and use Internet as the communication layer. DGs also aim at exploiting the resources of idle machines over Internet. Second the increasing number of devices and new business opportunities put a pressure to make existing applications to support these new devices in order to remain competitive. Web and Cloud technologies now provide feasible means to put almost any desktop functionality "on the Internet". However, we believe that an effort should be done earlier in the design stage for the interactions between the components of the system to be built to get confidence in the Internet-centric system, especially when Publish/Subscribe is the privileged communicating paradigm.

Indeed, DGs have important features that explain the large number of international projects aiming to better exploit the computational potential. Many DGs systems [4] have been developed using a centralized model. The most popular are BOINC [5], Condor [6], OurGrid [7] and Xtremweb [8].

Let's talk about BOINC [5] to introduce a typical Desktop Grid infrastructure. BOINC handles a variety of factors that are unique to volunteer computing:

- Scale: The system must handle millions of nodes and tens of millions of jobs per day.
- Heterogeneity: Volunteer hosts are highly diverse in terms of software, hardware, and network connectivity.
- Trust and reliability: Incorrect computational results may be intentionally returned by malfunctioning or malicious participants.
- Communication access: Many systems are behind rewalls that allow only outgoing HTTP traffic.
- Availability: Volunteer hosts have sporadic presence and a high churn rate.
- Ease of use: The client software must be extremely simple to install and must work with no conguration or user intervention.

We recognize at least two important features for our concern: Heterogeneity and Availability. However, it is extremely hard to offer BOINC as a service, we mean a BOINC service on demand. By experience [9] we know that the 'cloudification of a system' is made easy if we count on a limited number of technologies and software dependencies. Taking resources inside major cloud providers is also hard with BOINC because BOINC is essentially based on the volunteering principle. At last,

BOINC scheduling mecanisms do not take into account multi-criteria approaches based on the energy and the load (for instance) to decide on the tasks allocation.

BOINC is based on the master/worker principle. A master/coordinator is in charge of distributing work to workers/slaves. If we have a look to the BOINC task server only, we found the following components:

1) Work generator – generates workunits. A workunit represents the inputs to a computation: the application a set of references input files, and sets of command-line arguments and environment variables. Each workunit has parameters such as compute, memory and storage requirements and a soft deadline for completion.
2) Scheduler – determines appropriate workunits for a client. BOINC scheduler implements a "local scheduling policy". This policy has several goals: a) To maximize resource usage (i.e. to keep all processors busy); b) To satisfy result deadlines; c) To respect the participantâĂŹs resource share allocation among projects; d) To maintain a minimal "variety" among projects.
3) Feeder – caches database information. To reduce database access and contention, the scheduler uses a cache of ready-to-send jobs in a shared-memory structure that is replenished by a single feeder program.
4) Transitioner – The transitioner implements the redundant computing logic: it generates new results as needed and identifies error conditions.
5) Validater – The validater examines sets of results and selects canonical results. It includes an application-specific result-comparison function.
6) Assimilater – The assimilater handles newly-found canonical results. Includes an application-specific function which typically parses the result and inserts it into a science database

2.2. Our architecture is inspired by Desktop Grid architecture

In this subsection we first introduce our proposition for an architecture dealing with highly dynamic and heterogeneous infrastructures, then we summarize recent advances for the RedisDG platform, our concrete implementation of the proposed architecture. At last, we introduce the coordination mechanism between the component of the architecture (the so called RedisDG protocol). Note that RedisDG can be seen either as the orchestration protocol or the concrete system for the execution of workflows.

2.2.1. Proposed architecture. The proposed architecture for dealing with heterogeneous and highly dynamic environments is depicted on Figure 1. The Physical Layer is composed of physical machines denoted M. They are notified by the Orchestration Layer of what they have to do and when. Machines can play the role of a Worker, a Coodinatoor, a Checker, a Monitor or a Broker, and as follows:

- Broker – This component "splits" the application task graph into chuncks and eventually duplicates the tasks. This component serves to analyze the task graph to pre-configure the execution in order to optimize it. For instance, root tasks may receive a special treatment.
- Coordinator – This component is in charge of allocation tasks to workers. It implements strategies to choose the "best worker" in a dynamic and heterogeneous context. It is the core component of the system where decisions are taken;
- Worker – This component implements the behavior of a worker i.e. it is in charge of the execution of a task. It can upload/download files according to the specification of the DAG. I is in charge of information to the coordinator, fo instance its availability, the energy consumption of the physical machine...
- Checker – This component ensures the results certification, for instance with a MD5 (hash function producing a 128-bit hash value) computation when tasks are duplicated.
- Monitor – This component monitor the machines to check if they are alive, for instance through a hearbeat mechanism. This component also learn about the newtwok bandwidth and the availability of nodes through machine learning algorithms (work in progress).

The application layer is composed, concretely, of an XML file that depicts the DAG and all the executable files and input files for the DAG. The user is in charge of providing this information to the system.

2.2.2. Recent advances for RedisDG. In this paper we add new features to the RedisDG [10], [11], [12] middleware which was initially designed as a light desktop grid middleware.

RedisDG has been formally designed and verified in [10]. In the grid computing and workflow engine fields, we are not accustomed to formally model our systems. We rather follow the traditional approach to design system which involves: the design according to ad-hoc methods, the realization and tests following simple scenarios to check if its behavior is satisfactory and if it is necessary to improve it or even design again. This is called an intuitive approach. Thus the alternative is modeling. For a specific scenario, a model can provide an accurate view of all feasible states that a system may have. Then, a simulation step is used to test whether the system behavior is satisfactory and highlights some problems. It does not completely replace experimentation which is necessary at the end of a satisfactory simulation but we get a high view of main system requirement

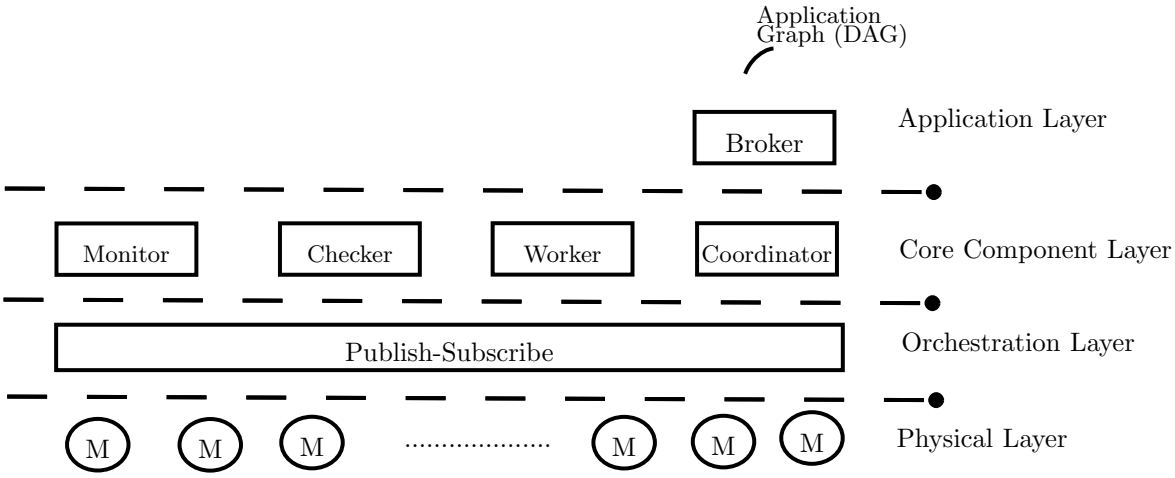


Figure 1. Architecture of our system and software components

and we can reason about specific or general properties in an automatic way.

RedisDG is based on Redis[3] for the Publish/Subscribe layer. Redis has special properties for a publish instruction faced to no registration that makes the design of scheduling decisions difficult. To get confidence into Redis we conducted in [13], [14] a formal modeling of the RedisDG protocol, based on our initial modeling of the Publish/Subscribe paradigm [15] and adapted to Redis interactions.

Recently, in [2] we proposed data-aware heuristics for the RedisDG system. Such heuristics are, for instance, based on the number of files requested by a task. The heuristics attempt to minimize the number of files transfers between computing nodes. We made experiments on the Grid5000 testbed in a context of containers and in a context of no containers. We demonstrated the effectiveness of our approaches.

2.2.3. RedisDG protocol. In this subsection, we recall the coordination algorithm of RedisDG system which is the core of our paper regarding the concrete implementation. It corresponds to the highest view possible. Some technical details are given in the experiments section. The algorithm is entirely based on the Publication-Subscription paradigm. To be short, the middleware offers the same features as the majority of desktop grid middleware such as Condor and BOINC. It manages scheduling strategies especially the dependencies between tasks, the execution of tasks and the verification/certification of the results.

In Figure 2, we depicts the steps of an application execution. In RedisDG, a task may have five states: *WaitingTasks, TasksToDo, TasksInProgress, TasksToCheck* and *FinishedTasks*. These states are managed by five actors: a broker, a coordinator, a worker, a monitor and a checker. Taken separately, the behavior of each component in the system may appear simple, but we are rather interested in the coordination of these components, which makes the problem more difficult to solve.

3. http://redis.io

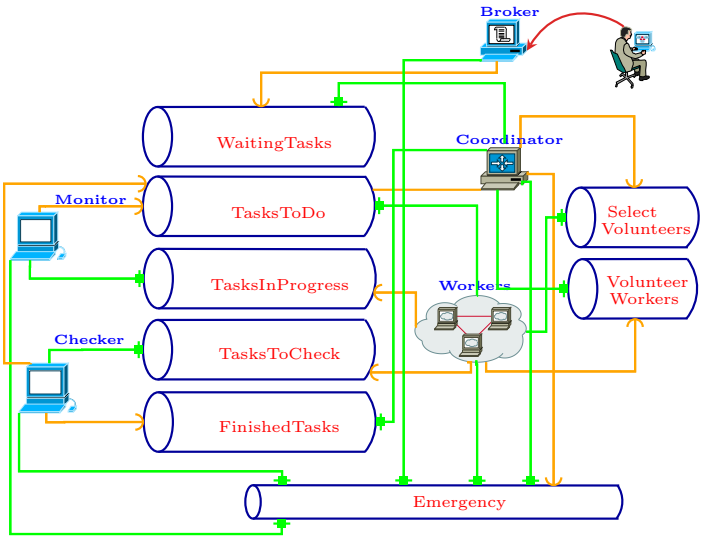


Figure 2. Interactions between components of the RedisDG system

The key idea is to allow the connection of dedicated components (coordinator, checker, ...) in a general coordination mechanism in order to avoid building a monolithic system. The behavior of our system as shown in Figure 2 is as follows:

1) Tasks batches submission. Each batch is a series-parallel graph of tasks to execute.
2) The Broker retrieves tasks and publishes them on the channel called *WaitingTasks*.
3) The Coordinator is listening on the channel *WaitingTasks*.

4) The Coordinator begins publishing independent tasks on the channel *TasksToDo*.
5) Workers announce their volunteering on the channel *VolunteerWorkers*.
6) The coordinator is aware of worker volunteering by listening the *VolunteerWorkers* channel.
7) The coordinator selects Workers according to several criteria (e.g. SLA).
8) The Workers, listening beforehand on the channel *TasksToDo* start executing the published tasks. The event 'execution in progress' is published on the channel *TasksInProgress*.
9) During the execution, each task is under the supervision of the Monitor whose role is to ensure the correct execution by checking if the node is alive. Otherwise the Monitor publishes again, tasks that do not arrive at the end of their execution. It publishes, on the channel *TasksToDo*, in order to make the execution of the task done by other Workers.
10) Once the execution is completed, the Worker publishes the task on channel *TasksToCheck* by indicating information about task execution (e.g. time execution, CPU consumption, etc.).
11) The Checker verifies the result returned and publishes the corresponding task on the channel *FinishedTasks*.
12) The Coordinator checks dependencies between completed tasks and those waiting, and restarts the process in step (4).
13) Once the application is completed (no more tasks), the Coordinator publishes a message on the channel *Emergency* to notify all the components by the end of the process.

3. Related works on scheduling

This section is an exhaustive survey of scheduling algorithms, mainly dedicated to workflow scheduling. It reveals stategies and methods. We also introduce some tools and we explain how our approach is different to all of these works.

Our work falls in the general category of online scheduling with unavailability constraints. In particular, we consider the particular case of opportunistic schedulers where the goal is to exploit the spatial diversity within the time of our compute nodes. In this special cases, there are several interesting proposals like the work of Meyer [16] and the general state of art of Dong and Akl [17]. We would like in particular to highly recommend the theoretical modeling proposed by Arnold Rosenberg [18]. The proposed model states various qualitative metrics that could be used to estimate the quality of an opportunistic scheduling of DAGs. Though we do not use these metrics, some of our heuristics exploit the underlying idea. Despite the large know-how in the opportunistic scheduling of workflow, we found few works that address the problem we consider. This is due to the particular usage we make of the Publish-Subscribe paradigm.

If we consider the building of a System, the closer work we found is the Google Cloud Pub/Sub[4]. The system offers for instance asynchronous messaging that allows for secure and highly available communication between independently written applications. Google Cloud Pub/Sub helps developers quickly integrate systems hosted on the Google Cloud Platform and externally. For example, a large queue of tasks can be efficiently distributed among multiple workers, such as Google Compute Engine instances or an order processing application can place an order on a topic, from which it can be processed by one or more workers.

Workflow management systems related works are presented in the synthesis from Valduriez [19] or in the work of Carole Goble [20]. These two papers are more related to cloud computing and data intensive scientific workflows in putting an emphasis on data management.

Many studies as those in [21], [22], [23], [24] may also serve as complimentary readings on workflow scheduling and they can be considered as conventional works. Many works validate the strategies through simulations. In our case our option is to run real world applications (MONTAGE and ADAPT) which requires a special effort for designing an experimental plan, to check the reproducibility of the experiments... Moreover, the context of these works are not adapted to our context because they do not take into account a dynamic view of the system in reacting to events when they arrive. Static information are supposed to be available (task graph, date of the events, task duration and costs...). They take a scheduling decision on the basis of a fixed number of workers. With the Publish-Subscribe paradigm in mind a 'more efficient' worker may join the system in the near future.

The context of these works are not adapted to our context because:

- they do not take into account a dynamic view of the system in reacting to events when they arrive;
- they are 'clairvoyant'-like. Static information are supposed to be available (task graph, date of the events, task duration and costs...). They take a scheduling decision on the basis of a fixed number of workers. With the Publish/Subscribe paradigm in mind a 'more efficient' worker may join the system in the near future.

In [25] authors consider the *pipelined workflow scheduling* where the execution of a large class of application can be orchestrated by utilizing task-, data-, pipelined-, and/or replicated-parallelism. Indeed, they focused on the scheduling of applications that continuously operate on a stream of data sets, which are processed by a given workflow, and hence the term pipelined. These data sets all have the same size (which

4. https://cloud.google.com/pubsub/overview

is not part of our assumption) and the DAG (Directed Acyclic Graph) model is used to describe the applications. Authors also dealt mainly with the throughput and latency performance metrics which are not under concern in our work. The main contribution of this survey is in structuring existing works by considering different levels of abstraction (workflow models, system models, performance models).

In [26] authors consider the problem of *co-scheduling* which means that we can execute several applications concurrently. They partition the original application set into a series of packs, which are executed one by one. The objective is to determine a partition into packs, and an assignment of processors to applications, that minimize the sum of the execution times of the packs. Authors assume that they know the execution profiles i.e. the execution time of each task on each processor, which is not part of our assumption.

In [27] authors investigated the problem of scheduling independent tasks under the paradigm of the master-worker. They consider heterogeneous situations where resources can have different speeds of computation and communication. The most interesting part of the work is to focus on the question of determining the optimal steady state scheduling strategy for each processor (the fraction of time spent computing and the fraction of time spent communicating with each neighbor). This question is quite different from the question of minimizing the total execution time, and the authors solve the problem is polynomial time. The paper demonstrate that we can observe the behavior of a system from a point of view that is not always focused on the execution time, as in our case.

In [28] authors considered the problem of dynamic load-balancing on hierarchical platforms. They focused more specifically on the *work-stealing* paradigm which is an online scheduling technique. They reviewed some existing variations and proposed two new algorithms, in particular the HWS algorithm which was analyzed in the case of fork-join task graphs. In their framework the authors considered that the graph is generated online during the execution. This is not our assumption. In our case we assume that new workers can potentially join the system in a dynamic way. However the analysis part in this work is interesting because it exemplifies the use of graph parameters such as the critical path. The execution time (mono criteria) is discussed in the paper as opposed to our work that offers a multi-criteria performance metric.

Swift [29] is an implicitly parallel programming language that allows the writing of scripts that manage program execution across distributed computing resources, including clusters, clouds, grids, and supercomputers. The JETS middleware [30] component is closely related to Swift and it provides high performance support for many-parallel-task computing (MPTC). The MTC model [31] consists of many, usually sequential, individual tasks that are executed on processor cores, without inter-task communication. The tasks communicate only through the standard file system interfaces, and optimization are possible [32]. We do not assume in our work the availability of a global file system. Data exchange between tasks are explicitly specified in the workflow description. With RedisDG, data exchange are implemented through Redis servers or by a 'scp-like' implementation, in a transparent way from the user point of view.

For the JETS middleware [30], authors notice that '*the native schedulers and application-launch mechanisms of today's supercomputers do not support a sufficiently fast task scheduling, start-up, and shutdown cycle to allow implementations of the many-task computing model to work efficiently, but the development of a specialized, single-user scheduler can allow many task applications to use a high fraction of the system compute resources*'. Thus, the paper is about a coupling between a 'system scheduler' and a 'user scheduler'. The key idea is as follows. First, the Swift script is compiled to the workflow language Karajan (internal representation), which contains a complete library of execution and data movement operations. Tasks resulting from this workflow are scheduled by well-studied, configurable algorithms and distributed to underlying service providers (external schedulers) including local execution, SSH, PBS, Globus, Condor or the Coasters provider [33]. The CoasterService uses task submission to deploy one or more allocations of *pilot jobs*, called Coaster Workers, in blocks of varying sizes and duration. Then the CoasterService schedules user tasks inside these blocks of available computation time and rapidly launches them via RPC-like communication over a TCP/IP socket. At least, we noticed that JETS currently operates according to a simple FIFO queuing approach. Authors plan to explore the addition of priority-based scheduling and backfill and to measure scheduler performance on workloads of varying size tasks. Our work is one step in that direction.

The AWS cloud system [34] and some other cloud management services such as enStratus [35], RightScale [36], and Scalr [37] offer schedule-based (or predetermined) and rule-based (dynamic) auto-scaling mechanisms. Schedule-based auto-scaling mechanisms allow users to add and remove capacity at a given time that is fixed in advance. Rule-based mechanisms allow users to define simple triggers by specifying instance scaling thresholds and actions, for instance to add/remove instance when the CPU utilization verifies a certain property making the framework dynamic. These mechanisms are simple and convenient when users understand their application workload and when the relationship between the scaling indicator and the performance goal is easy to determine. From a purely cloud point of view it is not realistic to let the user make actions at this level: more automation is needed because clouds are for non expert users in order to serve requests on-demand and in a self-service way.

The paper [38] presents the Maximum Effective Reduction (MER) algorithm, which optimizes the resource efficiency of a workflow schedule generated by any particular scheduling algorithm. MER takes as input a workflow schedule generated by an existing scheduling algorithm then, with the allowance of a limited increase in the original makespan, it consolidates tasks into a fewer number of resources than that used for the original schedule. To do this, MER essentially optimizes the trade-off between makespan increase and resource usage reduction. The paper introduce three building blocks, firstly the delay limit identification algorithm for finding the minimum makespan increase for the maximal resource reduction, second the task consolidation algorithm and third the resource consolidation algorithm. Finally, MER is evaluated in a simulated environment with three different scheduling algorithms and under four different workflow applications.

In [39] authors present an approach whereby the basic computing elements are virtual machines (VMs) of various sizes/costs, jobs are specified as workflows, users specify performance requirements by assigning (soft) deadlines to jobs. Then, the optimization problem is to ensure all jobs are finished within their deadlines at minimum financial cost. One key point is to dynamically allocating/deallocating VMs and scheduling tasks on the most cost-efficient instances. Another key point about a user intervention is that authors use deadlines that serve as the performance requirements specified by the users, and deadline misses are not strictly forbidden. Authors use deadline assignment techniques to calculate an optimized resource plan for each job and determine the number of instances using the Load Vector idea (intuitively, the vector is the number of the machines needed to finish the task on VM_m).

In [40] authors propose a resource-efficient workflow scheduling algorithm for business processes and Cloud-based computational resources. Through the integration into the Vienna Platform for Elastic Processes and an evaluation, they show the practical applicability and the benefits of the approach. Authors schedule workflows and not tasks inside workflows. This paper is related to cloud scheduling strategies and not, as in our case, to scheduling tasks that have yet been deployed in a physical environment/infrastructure. The scheduling algorithm for elastic processes is responsible for finding a workflow execution plan which makes sure that all workflows are carried out under the given constraints. These constraints could be defined in a Service Level Agreement (SLA). Authors also assume that: a) each Backend VM hosts exactly one service instance, i.e., it is not possible that different service types are instantiated at the same Backend VM and, b) all VMs offer the same capabilities in terms of computational resources and costs. Authors also specify the Scheduler and Reasoner, which are responsible, respectively for creating a detailed scheduling plan according to the workflow deadlines, and lease or release the required Cloud-based computational resources. The reasoner made use of the Java Library Apache Commons Math to solve the OLS (Ordinary Least Square - Linear Regression) problem.

The paper [41] introduces a new scheduling criterion, Quality-of-Data (QoD), which describes the requirements about the data that are worthy of the triggering of tasks in workflows. Based on the QoD notion, authors propose a novel service-oriented scheduler planner, for continuous data processing workflows, that is capable of enforcing QoD constraints and guide the scheduling to attain resource efficiency. QoD describes the minimum impact that new input data needs to have in order to trigger re-execution of processing steps in a workflow. This impact is measured in terms of data size, magnitude of values and update frequency. QoD can also be seen as a metric of triggering relaxation or optimist reuse of previous results. The core of the paper is a new scheduling algorithm for the Cloud that is guided by QoD, budget, and time constraints. The Markov Decision Process (MDP) technique is used to transform the problem. Authors explain that branch scheduling on the MDP representation is performed by starting from the most 'complex' branch to the 'least' complex one. In fact they exhibit an optimization problem they solve using a dynamic programming algorithm.

In a series of works [42], [43] Marc Frîncu and all. explore the dynamic and unpredictable nature of the grid systems to offer mechanisms for adaptation at any given moment. For instance, the author proposed in [42] a scheduling algorithm which minimizes each task's estimated execution time by considering the total waiting time of a task, the relocation to a faster resource once a threshold has been reached and the fact that it should not be physically relocated at each reassignment but only at a precise moment, through a simple marking, to reduce the network traffic. The key advantage of the proposed solution is to consider tasks of multiple workflows when they arrive and not batch after batch. One drawback is that the algorithm is based on user estimates for the value of the execution time and authors propose that this information be obtained by using historical data and applying some learning mechanisms.

In [43] the focus is cloud computing and the authors noticed that vendors do prefer to use their own scheduling policies and often choose their negotiation strategies. In the framework of workflow scheduling, the goal in that paper is the minimization of the overall user cost. The problem addressed in the paper is to access services provided by different cloud vendors, each with their own internal policies. Two major problems are dealt with: finding cloud resources and orchestrating services from different cloud vendors. The key idea in the paper is, once the workflow is submitted, an agent tries to schedule the tasks on the best available service through negotiation with other agents. It can be noticed that no static scheduling decisions are made and that tasks are scheduled one by one as they become ready for scheduling.

In [44] authors developed the concept of *dynamic dataflows* which utilize alternate tasks as additional control over the dataflow's cost and QoS. Dataflow systems allow users to compose applications as task graphs that consume and process continuous data, and execute on distributed commodity clusters and clouds. The key point in the paper is that authors addressed the problem of scheduling tasks when the input rates changes. The goal is to build dataflow systems with a greater concern for self-manageability. Indeed authors investigated autonomous runtime adaptations in response to fluctuations in both input data rates and cloud resource performance. Authors formulated the underlying optimization problem as a constrained utility maximization problem during the period for which the dataflow is executed. Then they first use meta-heuristics to solve it, for instance a genetic algorithm-based algorithm. Second they proposed greedy heuristics to find an approximate solution to the optimization problem. At last, they evaluated the proposed heuristics through a simulation study based (partly) on the popular CloudSim [45] simulator.

In [46] the authors addressed the lack of integrated support for data models, including streaming data, structured collections and files, that are limiting the ability of workflow engines to support emerging applications that are stream oriented. The key idea of the proposed framework is in its ability to transition from one data model to another one. The paper is more about architectural issues than scheduling issues. However the workflow framework evaluation is done on a private Eucalyptus cloud which is always challenging because of the complex nature of real systems.

In [21] authors reviewed the solutions for allocating suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions specified by users. This paper can be considered as studying conventional solutions for the workflow scheduling problem. The context is Grid computing and the authors first introduced workflow management systems that define, manage and execute workflows on computing resources. Our context is more related to volunteer computing and we want to design, as a whole, the interactions of components, especially the interactions between the workflow scheduling and data movement components. Moreover, there are two types of abstract workflow model, deterministic and non-deterministic. In a deterministic model, the dependencies of tasks and I/O data are known in advance, whereas in a non-deterministic model, they are only known at run time. In our case the dependencies are known in advance but nodes publish their volunteering. We may consider them as active and not passive, making a strong distinction with other works.

The heuristics recalled in [21] are based on the performance estimation for task execution and I/O data transmission. In our work, we do not make any assumption about performance estimation, apriori known. Dependency mode scheduling algorithms intends to provide a strategy to order and map workflow tasks on heterogeneous resources based on analyzing the dependencies of the entire task graph, in order to complete these interdependent tasks at earliest time. The strategy ranks the priorities of all tasks in a workflow application at one time. One issue with this strategy is to set weights on tasks. One idea is to set the weight of each task and edge to be equal to its estimation execution time and communication time. In our case we assume that the execution context is fluctuating (nodes may enter/leave the system at any time) making the estimates of weights a challenging problem.

Fair resource allocation problem address how to divide resources fairly to a set of users in a system where users have different resource demands. Previous works in computer science [47], [48] mainly focused on the allocation of a single resource and took place in the field of networking. For instance authors first transform selection of the best users and rates issued from a complex general optimization problem into a set of two formulations: a multi-user scheduling problem that maximizes total system throughput and a control-update problem that ensures long-term deterministic or probabilistic fairness constraints.

In [49] authors proposed a multi-resource allocation mechanism, which is the generalized form extending a single resource allocation to multiple types resources allocation, known as Dominant Resource Fairness(DRF) mechanism. DRF equalizes user's dominant share among all users, and it provides several attractive fairness properties: sharing incentive (allocation should be better than dividing each resource equally among all users), envy-freeness (no user would like to trace his allocation with any other user), Pareto efficiency (no user can increase his allocation without decreasing allocation of other users) and strategy-proof (a user cannot increase her allocation by lying about her requirements). DRF has quickly attracted a lot of attention and has been extended to many dimensions.

Dolev and al. in [50] proposed another alternative notion of fairness for multi-resource allocation, called Bottleneck-Based Fairness (BBF). Roughly speaking, an allocation of resources is considered fair if every user either gets all the resources he wishes for, or else gets at least his entitlement on some bottleneck resource, and therefore cannot complain about not receiving more. Recently, Wang et al. in [51] proposed a new allocation model BAA based on the notion of per-device bottleneck sets. Th context of he work is multi-tiered storage made up of heterogeneous devices. Clients bottlenecked on the same device receive throughput in proportion to their fair shares, while allocation ratios between clients in different bottleneck sets are chosen to maximize system utilization.

Gutman and Nisan [52] formalized both fairness notions in economic terms, extending them to apply to a larger family of utilities. The technical results are algo-

rithms for finding fair allocations corresponding to two fairness notions. First of all, regarding the notion suggested by Ghodsi and al., they presented a polynomial-time algorithm that computes an allocation for a general class of fairness notions, in which their notion is included. Second, they showed that a competitive market equilibrium achieves the desired notion of fairness.

In [53] Wong and al. generalized the DRF mechanism and plunged it into a unifying framework trying to capture the tradeoffs between fair allocation and system efficiency. Intuitions behind the analysis are explained in two visualizations of multi-resource allocation.

Recently, Wang and al. [**?**] proposed DRFH, a generalization of DRF to an environment with multiple servers. They also designed a simple heuristic that implements DRFH in real-world systems. Large-scale simulations driven by Google cluster traces showed that DRFH significantly outperforms the traditional slot-based scheduler, leading to much higher resource utilization with substantially shorter job completion times.

Parkes and al. [54] extended DRF in several ways and considered the zero demands of some resources required by a user, and in particular they studied the case of indivisible tasks. Authors also charted the boundaries of the possible in this setting, contributing to a new relaxed notion of fairness and providing both possibility and impossibility results.

In [55] Cole and al. revisited the classic problem of fair division from a mechanism design perspective and provided an elegant truthful mechanism that yields surprisingly good approximation guarantees for the widely used solution of Proportional Fairness. They proposed a new mechanism, called the Partial Allocation mechanism, that discards a carefully chosen fraction of the allocated resources in order to incentivize the agents to be truthful in reporting their valuations. This mechanism introduced a way to implement interesting truthful outcomes in settings where monetary payments are not an option.

4. Definitions related to the allocation of tasks

We now define the set of notions we deal with, we mean the set of definitions regarding the system and the performance criteria. These notions are of particular interest for the scheduling heuristics we will introduce later on.

4.1. System definition

The system is defined according to the following basics sets:

- W being the set of workers of the system
- T being the set of instants representing the execution time of the system. This set is totally ordered by the chronological order. Consequently, $t < t'$, if t is older than t'. We denote t_0 and t_{end} the instants which correspond respectively to the the starting and the end of the system.

A job j is defined as a set of task denoted Π_j. This set is partially ordered according to a precedence relation $\prec$. Consequently two tasks $p_a, p_b \in \Pi_j$ verify the property $p_a \prec p_b$ iff p_b has to wait the end of p_a to begin its execution.

An execution of a job j is the set E_j being the set of all its task executions. A task execution $e \in E_j$ where $E_j \subset \Pi_j \times \mathcal{P}(\mathcal{W}) \times T \times T$ is a quadruplet $(p_e, \{w_e^1, w_e^2, \cdots, w_e^n\}, b_e, e_e)$ with p_e the task associated to the execution e, $\{w_e^1, w_e^2, \cdots, w_e^n\}$ is the set of workers executing the different replicas of e, b_e is the moment when task p_e starts its execution of its first replica (including input data retrieving) on the worker set and e_e is the moment when p_e finishes its execution, i.e., the moment where the system elects the definitive result of task p_e. We denote for job j :

- $tb_j \in T$ the moment when the first task of job j starts its execution.
- $te_j \in T$ the moment when the last task of job j finishes its execution.

We define a time window $tw \in T \times T$ being the couple of instants (b_{tw}, e_{tw}), verifying the property $b_{tw} < e_{tw}$ and b_{tw} being the instant of tw beginning and e_{tw} being the instant of tw ending.

The set E_{tw}, defines all task execution e of any job where the associate task has been executed in the time window tw. A task execution belongs to a time window, if (1) the beginning of e ranges between b_{tw} and e_{tw}, or (2) the end of e ranges between b_{tw} and e_{tw} or (3) the beginning and the end of tw ranges between b_e and e_e. Formally,

$$E_{tw} = \{e \mid e \in E \wedge (b_{tw} \leq b_e < e_{tw} \vee b_{tw} \leq e_e < e_{tw} \vee b_e \leq b_{tw} < e_e)\}$$

We can define the function $T_{exe}(e, tw)$ to compute the execution time of an execution task e in a given time window tw:

$$T_{exe}(e, twin) = min(e_e, e_{twin}) - max(b_e, b_{twin})$$

4.2. Performance criteria definition

In our work we consider three criteria: the execution time, the fairness and the energy consumption. With this choice we would like to underline that we are building a general framework for which we can plug a wide variety of performance criteria. We will see later on that we also manage, in our concrete tool, a large variety of decision criteria.

Execution time criteria definition. The global execution time for a given job j is defined as $ExecutionTime(j) = te_j - tb_j$

Fairness criteria definition. We define the *fairness* as the standard deviation of the cumulative computing time, for each worker, on a given time window. Formally this metric is defined by the function $Fairness(tw) = \sigma\{CT(w,tw) \mid w \in W\}$ where the function CT is defined as $CT(w,tw) = \sum_{e \in E_{tw}} T_{exe}(e,tw)$.

We can now define the fairness for a given job j, denoted $JobFairness(j) = Fairness((tb_j, te_j))$ being the fairness of the full period of time to execute all tasks belonging to j.

As the same principle, we can define the global fairness of the system denoted $GlobalFairness = Fairness((t_0, t_{end}))$.

Consequently the higher the value, the less the system is fair.

Energy consumption criteria definition. We denote $CPULoad(w,x)$ the average CPU load during the execution of a task x on a worker w. We assume that each worker can express a value of energy consumption of a task instance from the CPU load thanks to a function $\mathcal{E}$. This function depends of the consumption model of the hardware, i.e. the CPU, installed on the worker w. Consequently, $EnergyConsumption(w,x) = \mathcal{E}\big(CPULoad(w,x)\big)$.

Then we define the energy consumption for a given job j which is the sum of energy consumption of each executed task replica. Formally, $JobEnergyConsumption(j) =$

$$\sum_{(p_e,W_e,b_e,e_e) \in E_j} \Big(\sum_{w \in W_e} EnergyConsumption(w,p_e) \Big)$$

Concerning, the energy consumption model, we assume, as for the concrete work done by the VIFIB/NEXEDI[5] team related to the SlapOS cloud[6], that we have a discrete model that, for a given CPU load, gives the power (Watt) that the processor card consumes. In our work we use the following discrete models representing the power consumed by 3 processor cards as reported by VIFIB:

- Shuttle Computer: :
$$\mathcal{E}(x) = \begin{cases} 21.5 + 1.06 * x \text{ when } x \leq 25\% \\ 48 + 0.29 * x \text{ otherwise} \end{cases}$$
- Intel NUC:[7]
$$\mathcal{E}(x) = \begin{cases} 8.5 + 0.46 * x \text{ when } x \leq 25\% \\ 20 + 0.08 * x \text{ otherwise} \end{cases}$$
- Rikomagic Linux device:[8]
$$\mathcal{E}(x) = \begin{cases} 2.2 + 0.04 * x \text{ when } x \leq 25\% \\ 3.2 + 0.008 * x \text{ otherwise} \end{cases}$$

5. http://www.vifib.com/press/news-CO2
6. https://www.slapos.org
7. http://www.intel.com/content/www/us/en/motherboards/desktop-motherboards/nuc.html
8. http://www.cloudsto.com/mk802iii-le-mini-linux-pc.html

4.2.1. Availability of workers. In the field of desktop grids, the studies on the availability of workers [**?**], [**?**] by Kondo et al. show that workers follows different statistical laws. Authors have identified:

1) six clusters of nodes with different statistical laws (Gamma (4) - Log-Normal (1) and Weibull (1)) ;
2) a global law for all the workers that follows a Weibull law of shape of 0.3787 and scale of 3.0932. We keep this model for our work despite the fact that this choice is questionable versus the choice of a model with six clusters because the variability in the distributions in the clusters seems important as quoted by the authors;

In our experiments the workers can follow two laws in playing with the scale parameter of the law. This allows to generate a model where the worker is present on the first time interval and then no longer available on the other intervals. The second model makes it possible to simulate a machine not really available at the beginning and at the end of the time interval and available on intermediate time intervals. When the worker execution starts, we choose at random one law for the availability of the worker.

In our RedisDG system, any worker is now running as follows. When the coordinator publishes a request to execute a task, the workers now publish, among the information that return to the coordinator, if they are available. For that, the workers look at the current time interval and according to a threshold, the workers decide if they are available or not. In our implementation, when a period of time ends, we restart a new generation for a Weibull law in order to not wait indefinitely. This process helps in avoiding deadlock problems because we have ensured that the threshold, which determines whether the machine is available or not, is always greater than the smallest of the probability values. So, at each new generation there will be at least one time interval for which the machine will be available.

4.3. Theoretical view of the global problem

We now describe the problem that we want to solve. Let us assume a bag of N jobs where each job corresponds to a set of tasks. Our allocation problem can be formulated as a decision problem with N rounds. In the round i, the scheduler publishes and decides on the best allocation of tasks of the i^{th} job.

The round i starts at date t_{pub} with the publication of a request for the processing of the set of tasks $J_i = \{T_{i,1}, \ldots, T_{i,n}\}$. These tasks could be processed by at most $|W_i|$ workers according to a finite set of strategies $S_1, \ldots, S_k$. A strategy defines how the scheduler deploys J_i on the workers (W_i). We assume a time interval $]t_{pub}, t_w]$ in which the scheduler could wait before deciding on the allocation of all tasks. At any date $t \in]t_{pub}, t_w]$, there is a finite set $W_i(t)$ of Workers that

subscribed to the published request. The goal for the scheduler is to choose, at the beginning of each round i, a maximal waiting time (or a closing date) $t_i^{opt} \in]t_{pub}, t_w]$ and a strategy S_i^{opt} for the processing of J_i such as to maximize a total reward R

$$\sum_{i=1}^{N} R(t_{pub}, S_i^{opt}, W_i(t_i^{opt}))$$

In this model the reward, we can expect from a strategy, could include several possible dimensions (response time, fairness, energy etc.). It also depends on the publishing time t_{pub} and the closing date t_i^{opt}. The closing date is used because it decides on the set of workers to which a task could be assigned. The publishing date is considered to handle the cases where the strategy that was chosen will start to deploy the tasks at date $t' = t_{pub} + \epsilon$ which is the first date coming after t_{pub} and where $W_i(t_{pub} + \epsilon)$ is not null. In this general model, we also make the following considerations:

1) At any date $t < t'$, $W(t)$ is unknown;
2) At the beginning of each round, the scheduler does not know the reward that any strategy and closing date could lead to.

This general decision problem is hard to solve. Indeed, we can formulate some of its variants as a multi-armed bandit problem [56]. For instance, let us assume that the time is discretized and that we have only one strategy. Then, the problem we have consists in choosing a date between t_{pub} and t_w such as to maximize the total reward on a finite set of rounds; this corresponds to the multi-armed bandit problem. In the same way, if we consider that t_i^{opt} is always fixed, we have again a bandit problem that consists of choosing among k strategies in N rounds.

Our paper will not provide a solution to the general online problem. Instead, our objective will be to formulate some interesting strategies that could be practically considered in this general problem. The interest in these strategies is in particular due to the fact that they are implementable in a real publish-subscribe system. In addition to the formulation of strategies, we will also propose a performance characterization that gives insights on the rewards we could expect depending on the strategy we choose.

5. Scheduling mechanisms

In this section we introduce six heuristics to schedule the tasks of a set of job on a set of workers and according to the RedisDG publish/subscribe framework. We introduce heuristics to progressively answer to our general open question: what is the best strategy to satisfy, based on a single given criteria like fairness, execution time or energy consumption or a combination of criteria.

5.1. Considered heuristics

HEURISTIC 1 (FCFS): Upon a task becomes independent, the coordinator publishes it to all the workers. This task is allocated on the first answering worker.

HEURISTIC 2: Upon a task becomes independent, the coordinator publishes it to all the workers. If a worker w is free it answers directly, otherwise it delays the answer to the coordinator until it becomes free. This ensure the liveness property because at least one worker answers eventually. When the coordinator receives a reply from a worker w for task t, it first saves this will for t. If t has been already allocated to another worker, the coordinator allocates to w another task t' that has not yet been executed among those for which w wished to execute.

HEURISTIC 3: Upon a task becomes independent, the coordinator publishes it to all the workers and starts a timer for λ time unit. All worker answer whatever their state ensuring thus the liveness. During λ, the coordinator saves all volunteer workers for a task. Upon the expiration of the timer (timeout), it chooses a worker among the set of volunteers according to a policy described in the section **??**. If no worker has answered, the coordinator activates again the timer.

Heuristics 3-bis captures the availability of nodes in the selection process as follows.

HEURISTICS 3-BIS: upon a task becomes independent, the coordinator publishes it to all the workers and starts a timer for λ time unit. All workers answer, whatever their states, ensuring thus the liveness and also publish if they are available or not. During λ, the coordinator saves all volunteer workers for a task. Upon the expiration of the timer (timeout), it chooses a worker among the set of available volunteers according to a policy described in the beginning of section **??** (see the discussion about methods and rewards). If no worker has answered or if all the workers that have answered are unavailable, the coordinator activates again the timer and it publishes again the task.

Our proposition for implementing a multicriteria approach for selecting workers is also based on the abovementionned Heuristics 3, and is as follows.

HEURISTICS 3-TER: upon a task becomes independent, the coordinator publishes it to all the workers and starts a timer for λ time unit. All workers answer whatever their state ensuring thus the liveness and also publish if they are available or not. During λ, the coordinator saves all volunteer workers for a task. Upon the expiration of the timer (timeout), it chooses a worker among the set of available volunteers according to a Pareto front computation and based on the load, energy and fairness rewards. If no worker has answered or if all the workers that have answered are unavailable, the coordinator activates again the timer and it publishes again the task. Among the points on the Pareto front we choose the 'first' as the winner (others policies are

also possible but they are left for future works: random choice or according to the value of one criteria...).

Note that Pareto (see [57] for an example in the context of cloud computing) efficiency is a balance of resource distribution such that one individualâĂŹs lot cannot be improved without impairing the lot of one or more other individuals.

HEURISTIC 4: The aim of this heuristic is to promote the communication between workers avoiding the master node solicitation and thus useless communication. Indeed, it is possible to transfer directly intermediate data between two workers which execute two dependant tasks. To achieve this goal, the procedure is identical to the heuristic 3 except that:

- all tasks are published by the master at the job submission time to make a priori allocation based on policies described in the section **??**. Thus, once all tasks are assigned it is possible to determine the communication scheme.
- workers subscribe to the $FinishedTask$ channel in order to know when their predecessors has finished leading to a data downloading.

The choice of λ parameter has an impact over performance. A too small value implies that only quickest workers will answer that degrades the fairness and a too large value induces latency degrading the workflow execution time.

Among these four heuristics, heuristic 4 is the most clairvoyant one because the scheduling takes into account the shape of the task graph. This point could be improved by considering not only direct successor tasks but potentially a larger set of successors. This would surely make more complex the decision of the elected volunteer and we are not convinced that this option would bring significant improvement.

6. Experimental evaluation

In this section we studied the impact of the scheduling over the different performance criteria described in section 4.2 by experimenting the execution of two workflows on a real platform.

6.1. Workflow used in the experiments

The MONTAGE [58] project has been created by NASA/IPAC *Infrared Science Archive* as an open-source toolbox to generate personalized mosaic of the sky from images in the *Flexible Image Transport System* (FITS) format. During the final production of the mosaic, the geometry of the output is computed from the geometry of the input [59], [60].

In our experiment, we used an instance of MONTAGE represented by 1446 tasks and 3722 dependency links between tasks. The 1446 tasks are dispatched on the different levels of the workflow as shown in the Figure **??**. The execution of this quite large instance requires 9423 input files (including the intermediary files) and it generates 2889 files (including the intermediary files).

ADAPT is a framework for solving problems in fluid mechanics (CFD). We have formulated a workflow view for the problem depicted in [61]. We consider an ADAPT workflow with 506 tasks. The ADAPT workflow, has the following properties, different from the MONTAGE workflow, when we execute it on a real platform:

1) The cost in time for executing the first two stages of the workflow (METIS step) is very low comparing to the other costs;
2) The cost of transferring data between nodes is also very low. The transferring data correspond to the exchange of parameters of the model; Heuristics taking into account this cost has no influence on the result and may degrade the overall performance if the time cost for deciding or realizing the 'optimized' transfers is high.
3) The dag is symmetric, 'horizontally' and 'vertically'. If we observe the dag from the top to the bottom and excepting the initial METIS step, we note a succession of exchange and merge steps that repeat each others. The execution time of any 'path' from the root node to the sink node is the same, meaning that all the paths are equivalent. A scheduling strategy based on the critical path of the graph without weights (the longest path in term of number of traversed nodes from the source to the sink) leads to unless work since all the paths are equivalent.

6.2. Experimental testbed and configuration

We implemented the RedisDG protocol in python 2.7. and the experiments were conducted on the national academic french platform Grid'5000. In the first set of experiments We used three distant clusters in the platform: Lyon, Sophia and Rennes, and for the second set of experiments we used different clusters in the Nancy site.

A site (or cluster), for the first set of experiments, is composed of 10 machines where, on each machine, a worker process runs. The coordinator process, the broker process and the data server were installed in a eleventh machine in the Lyon site. Consequently, we used 30 machines for workers processes and 1 master machine. For sake of simplicity we disabled the monitoring and the task replication mechanism (no monitor and checker processes). We also considered that a worker was reliable (no Byzantine failure) and was volunteer for any task.

6.2.1. First set of experiments. We went through the different scheduling strategies described in the section 5, reduced to the following 8 experiments :

- Heuristic 1

- Heuristic 2
- Heuristic 3 and 4., each associated with any strategy for choosing a volunteer : *Uniform, Green, OldestElected* (cf. section **??**). The average value of λ (time to wait after task publishing to choose a volunteer) was set to 5 seconds.

For each strategy, we launch sequentially three times the same job Montage described in section 6.1 in order to show the constant behavior of our protocol.

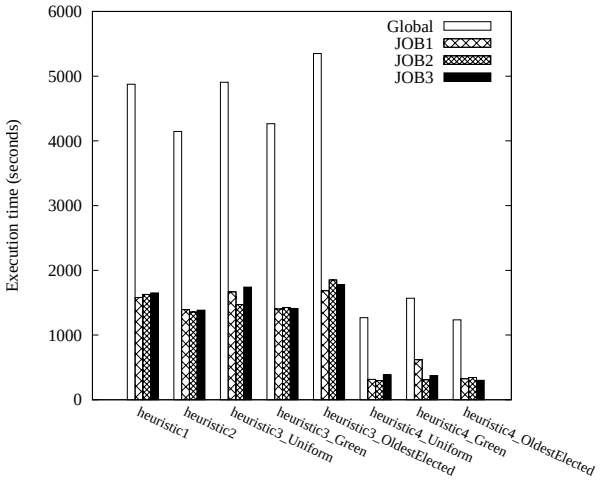
The figure 3 shows performance of the different scheduling strategies in terms of execution time (figure 3(a)), fairness (figure 3(b)) and energy consumption (figure 3(c)) (cf. section 4.2 for definitions). For each metric, we show specific performance of each job and a global performance for all the jobs. Thus, for the execution time and energy consumption metrics the global performance represents the cumulative time to execute the three jobs, whereas for the fairness metric, the global performance represents the $GlobalFairness$ function.

We can see on the figure 3 that the heuristic 1 has the worst value of fairness but has the smallest value of energy consumption. Actually the consequence of the heuristic 1 is that tasks are assigned on the closest workers from the coordinator node since it is a pure first come first serve policy. Thus only machines located in the Lyon cluster can obtain task to do before the remaining of the system which consequently works rarely. However the energy consumption is reduced than other heuristics since only one cluster is used and since the Lyon cluster consumes the least, according to its power model.

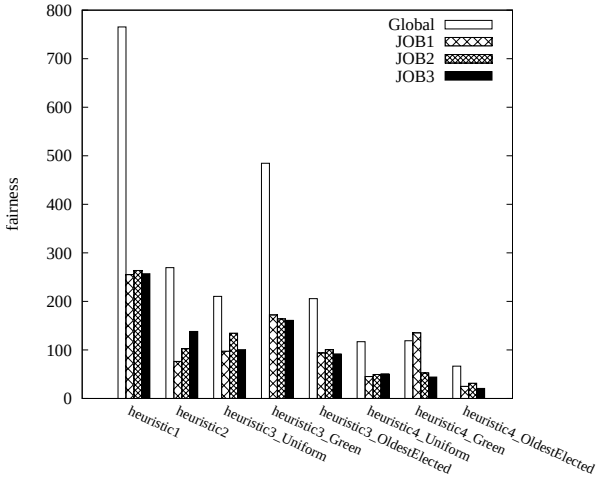
The heuristic 2 improves considerably the fairness because the tasks are affected on the first non working machine concerning eventually all workers of the system. Thus the parallelism is better, reducing the execution time. However, this has an negative impact over the energy consumption because more machines are used.

Let's analyze the heuristic 3 and its different policies of choice. As expected, the "green" policy gains in terms of energy consumption comparing to the other policies but it is the less fair because it concentrates task on the "greenest" cluster. The policies "Uniform" and "OldestElected" are equivalent in terms of fairness but better than the First Come First Serve strategy (heuristics 1 and 2). However, the execution time is degraded. Indeed, tasks are affected more uniformly on the workers set but the direct consequence is that the distant workers, from the data server, work more, inducing an overhead in term of data transfers.

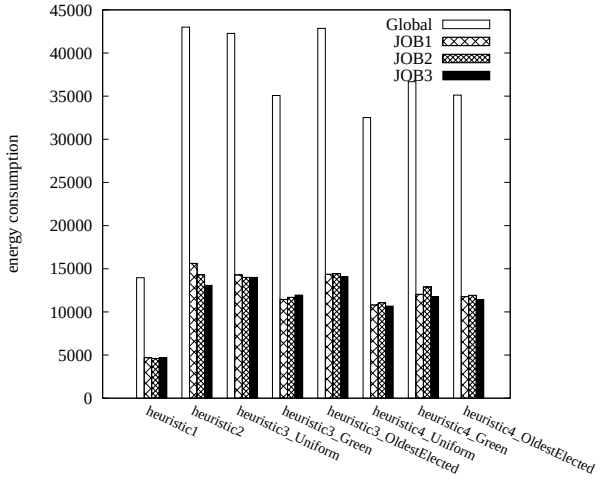
Finally the heuristic 4 reduces considerably the execution time because intermediate data are not uploaded on the data server but directly to the 'next' workers. This shows that data transfers should be included in the task assignment strategy for better performance. This data consideration has a consequence over the fairness criteria because we can see that the fairness is improved (except for the green policy). Indeed, with heuristic 4, the time to upload or download data becomes negligible



(a) Execution time



(b) Fairness



(c) Energy consumption

Figure 3. Experimental results of scheduling strategies

compared to the computational time of the task. Thus the time to execute a task becomes independent of its location which tends to have an equivalent execution time for any task. The "OldestElected" policy is the most fair policy for the heuristic 4 because the number of tasks per machine tends to be equal.

6.2.2. Second set of experiments. The number of nodes for executing the ADAPT workflow is 6 in the Nancy site: one for the coordinator and 5 workers. We also use 2 clusters of the Grid5000 testbed, both located in the Nancy site. During the execution, 3 workers have choosen (randomly) to behave like the first model for the availability and two workers have chosen to behave according to the second availability model.

The total execution time is 01h:07m:41s and it is given by the two measures of Start Time: 2016-12-25 01:17:17,999, and End Time: 2016-12-25 02:24:58,526. From the logs of the coordinator, we extracted the following trace:

```
2016-12-25 01:18:56,987 - RedisDG-coordinator - INFO - Coordinator publish task 5_1 to
workers
2016-12-25 01:18:56,989 - RedisDG-coordinator - INFO - worker worker_grimoire-7.nancy.
grid5000.fr_1482624969.4 is Falsevolunteer for 5_1 : state_task = SUBMITTED
2016-12-25 01:18:56,990 - RedisDG-coordinator - INFO - worker worker_graphene-49.nancy.
grid5000.fr_1482624969.46 is Truevolunteer for 5_1 : state_task = SUBMITTED
2016-12-25 01:18:56,990 - RedisDG-coordinator - INFO - worker worker_graphene-52.nancy.
grid5000.fr_1482624969.23 is Truevolunteer for 5_1 : state_task = SUBMITTED
2016-12-25 01:18:56,991 - RedisDG-coordinator - INFO - worker worker_graphite-2.nancy.
grid5000.fr_1482624969.27 is Falsevolunteer for 5_1 : state_task = SUBMITTED
2016-12-25 01:18:56,991 - RedisDG-coordinator - INFO - worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 is Truevolunteer for 5_1 : state_task = SUBMITTED
2016-12-25 01:19:03,118 - RedisDG-coordinator - INFO - Timeout for task 5_1
2016-12-25 01:19:03,118 - RedisDG-coordinator - INFO - volunteer for task 5_1, should
submit
2016-12-25 01:19:03,118 - RedisDG-coordinator - INFO - Choosing a volunteer for 5_1
2016-12-25 01:19:03,118 - RedisDG-coordinator - INFO - volunteer worker_griffon-25.nancy.
grid5000.fr_1482624968.45 choosed for 5_1
```

The coordinator publishes task 5.1. Two workers are unavailable (grimoire-7 and graphite-2). Three workers are available (graphene-49, graphene-52 and griffon-25). At time 01:19:03,118 the timer expires, thus a worker is selected (randomly). griffon-25 is elected as the worker in charge of task 5_1. From the logs of the griffon-25 worker we get:

```
2016-12-25 01:18:57,787 - RedisDG-worker - INFO - Worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 is finishing task T32_1
2016-12-25 01:18:57,790 - RedisDG-worker - INFO - I m available? True
2016-12-25 01:19:03,921 - RedisDG-worker - INFO - Task 5_1 has been selected for ['worker
_griffon-25.nancy.grid5000.fr_1482624968.45'] (state task = GIVEN)
2016-12-25 01:19:03,922 - RedisDG-worker - INFO - worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 selected for 5_1
2016-12-25 01:19:03,922 - RedisDG-worker - INFO - add task 5_1 in tasks_ready_to_run
2016-12-25 01:19:03,922 - RedisDG-worker - INFO - Worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 is executing task 5_1
2016-12-25 01:19:03,922 - RedisDG-worker - INFO - worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 : I start the execution of 5_1
```

This trace shows that at 01:18:57,787 griffon-25 has terminated the execution of task $T32_1$, and it is still available. At time 01:19:03,921 griffon-25 has been selected for the execution of task 5_1. Thus griffon-25 starts to execute this task.

Let us now introduce the logs of an execution of the ADAPT workflow according to the multicriteria approach. In this case we use 30 machines (1 for the coordinatoor and 29 workers) and 5 clusters of grid5000 (grisou, graphene, graphite, griffon, grimoire), all of them are located in the Nancy site. Note that this configuration[9] forms a very heterogeneous one.

We consider first the logs of the coordinator. We found the following trace:

```
2017-01-03 22:41:24,258 - RedisDG-coordinator - INFO - Choosing a volunteer for T11_1
2017-01-03 22:41:24,258 - RedisDG-coordinator - INFO - Scores of worker_grisou-17.nancy.
grid5000.fr_1483479622.02 is [1.0, 26.588, 4.8]
2017-01-03 22:41:24,259 - RedisDG-coordinator - INFO - Scores of worker_grisou-21.nancy.
grid5000.fr_1483479621.56 is [1.0, 22.772, 1.2]
2017-01-03 22:41:24,259 - RedisDG-coordinator - INFO - Scores of worker_graphene-18.nancy.
grid5000.fr_1483479622.4 is [1.0, 32.736000000000004, 10.6]
2017-01-03 22:41:24,259 - RedisDG-coordinator - INFO - Scores of worker_graphene-2.nancy.
grid5000.fr_1483479622.51 is [1.0, 67.343, 66.7]
2017-01-03 22:41:24,259 - RedisDG-coordinator - INFO - Scores for task T11_1 :


[[  1. 26.588   4.8  ]
 [  1.  22.772   1.2 ]
 [  1.  32.736  10.6 ]
 [  1.  67.343  66.7 ]]

'worker_grisou-17.nancy.grid5000.fr_1483479622.02': [1.0, 26.588, 4.8],
'worker_grisou-21.nancy.grid5000.fr_1483479621.56': [1.0, 22.772, 1.2],
'worker_graphene-2.nancy.grid5000.fr_1483479622.51': [1.0, 67.343, 66.7],
'worker_graphene-18.nancy.grid5000.fr_1483479622.4': [1.0, 32.736000000000004, 10.6]
2017-01-03 22:41:24,260 - RedisDG-coordinator - INFO - Scores after sort for task T11_1 :


[[  1.     26.588   4.8  ]
 [  1.     22.772   1.2  ]
 [  1.     32.736  10.6  ]
 [  1.     67.343  66.7  ]]

2017-01-03 22:41:24,260 - RedisDG-coordinator - INFO - Pareto-Frontier-Multi Result for
task
T11_1 is : [ 1. 26.588 4.8 ]
2017-01-03 22:41:24,260 - RedisDG-coordinator - INFO - volunteer worker_grisou-17.nancy.
grid5000.fr_1483479622.02 choosed for T11_1
2017-01-03 22:41:24,260 - RedisDG-coordinator - INFO - select the volunteer worker
worker_grisou-17.nancy.grid5000.fr_1483479622.02 for task T11_1 : state_task = GIVEN
```

The task requesting to be executed is task $T11_1$ and, at the time we need to execute it, only 4 workers are available. We notice that Pareto computation leads to the selection of `grisou-17` as the machine node that will execute task $T11_1$. In fact, two points are located on the Pareto front: $(1., 26.588, 4.8)$ and $(1., 22.772, 1.2)$. Our implementation selects the first one in the list.

To conclude this experimental section, we can say that all these examples illustrate the different situations in which our new RedisDG system can be faced to. They also illustrate how our measures and solutions can be fine. We could even envision a post-mortem analysis tool to fully exploit the logs.

7. Conclusion and future work

In this paper we have investigated the problem of orchestrating a scientific workflow in a highly heterogeneous and dynamic environment. We believe that this problem relates to the need for opportunistic strategies. Basically, in a dynamic and heterogeneous environment, the user can not make good decisions, even if he uses interactive scheduling. But if he relaxes the pressure (for example, he does not choose the number of workers), we have to find the 'best' decision in exploiting all opportunities.

RedisDG, seen as the protocol to manage the interactions between the components of the workflow engine, can also manage volatile computing nodes. Our system allows fine observations in this context and also according to multi-criteria decisions. Despite the efficiency, this strategy stagnates in environments with many objectives, i.e., possibly reaching a state where none result is good enough for all objectives simultaneously. This will

9. https://www.grid5000.fr/mediawiki/index.php/Nancy:Hardware

constitute our future work. Moreover, the selection of the best scenario for the deployment is also a challenge due the limited evaluation's strategies, their execution time and cost. We may point out that deploying a task inside Google cloud is maybe more or less costly than deploying inside the Amazon cloud, and this will constitute our second research direction.

Acknowledgment

Experiments conducted in this work are accomplished on the Grid'5000 testbed which is a scientific instrument supporting experiment-driven research in all areas of computer science, including high performance computing, distributed computing, networking and big data.

References

[1] P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[2] S. B. J. L. Y. N. Christophe Cerin, Leila Abidi and W. Saad, "Data management for the redisdg scientific workflow engine," in *The 6th IEEE International Symposium on Cloud and Service Computing, Nadi, Fiji, December 7-10, 2016*, 2016.

[3] D. Balouek-Thomert, A. K. Bhattacharya, E. Caron, K. Gadireddy, and L. Lefèvre, "Parallel differential evolution approach for cloud workflow placements under simultaneous optimization of multiple objectives," in *IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, BC, Canada, July 24-29, 2016*. IEEE, 2016, pp. 822–829.

[4] C. Cerin and G. Fedak, *Desktop Grid Computing*, 1st ed. Chapman and Hall-CRC, 2012.

[5] D. P. Anderson, "Boinc: A system for public-resource computing and storage," *Grid Computing, IEEE/ACM International Workshop on*, vol. 0, pp. 4–10, 2004.

[6] A. R. Butt, R. Zhang, and Y. C. Hu, "A self-organizing flock of condors," *J. Parallel Distrib. Comput.*, vol. 66, no. 1, pp. 145–161, 2006.

[7] N. Andrade, W. Cirne, F. V. Brasileiro, and P. Roisenberg, "Ourgrid: An approach to easily assemble grids with equitable resource sharing," *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*, vol. 2862, pp. 61–86, 2003.

[8] G. Fedak, C. Germain, V. Neri, and F. Cappello, "XtremWeb: a generic global computing system," *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pp. 582–587, 2001.

[9] C. Cérin and A. Takoudjou, "BOINC as a service for the slapos cloud: Tools and methods," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013*. IEEE, 2013, pp. 974–983.

[10] L. Abidi, J. Dubacq, C. Cérin, and M. Jemni, "A publication-subscription interaction schema for desktop grid computing," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013*, S. Y. Shin and J. C. Maldonado, Eds. ACM, 2013, pp. 771–778.

[11] L. Abidi, C. Cérin, and M. Jemni, "Desktop grid computing at the age of the web," in *Grid and Pervasive Computing - 8th International Conference, GPC 2013 and Colocated Workshops, Seoul, Korea, May 9-11, 2013. Proceedings*, ser. Lecture Notes in Computer Science, J. J. Park, H. R. Arabnia, C. Kim, W. Shi, and J. Gil, Eds., vol. 7861. Springer, 2013, pp. 253–261.

[12] W. Saad, L. Abidi, H. Abbes, C. Cérin, and M. Jemni, "Wide area bonjourgrid as a data desktop grid: Modeling and implementation on top of redis," in *26th IEEE International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2014, Paris, France, October 22-24, 2014*. IEEE Computer Society, 2014, pp. 286–293.

[13] L. Abidi, C. Cérin, and K. Klai, "Design, verification and prototyping the next generation of desktop grid middleware," in *Advances in Grid and Pervasive Computing - 7th International Conference, GPC 2012, Hong Kong, China, May 11-13, 2012. Proceedings*, ser. Lecture Notes in Computer Science, R. Li, J. Cao, and J. Bourgeois, Eds., vol. 7296. Springer, 2012, pp. 74–88.

[14] C. C. H. A. Leila Abidi, Walid Saad and M. Jemni., "Wide area bonjourgrid as a data desktop grid: modeling and implementation on top of redis." in *26th International Symposium on Computer Architecture and High Performance Computing*, october 2014.

[15] L. Abidi, C. Cérin, and S. Evangelista, "A petri-net model for the publish-subscribe paradigm and its application for the verification of the bonjourgrid middleware," in *IEEE International Conference on Services Computing, SCC 2011, Washington, DC, USA, 4-9 July, 2011*, H. Jacobsen, Y. Wang, and P. Hung, Eds. IEEE, 2011, pp. 496–503.

[16] L. A. V. C. Meyer, D. Scheftner, J. Vöckler, M. Mattoso, M. Wilde, and I. T. Foster, "An opportunistic algorithm for scheduling workflows on grids," in *High Performance Computing for Computational Science - VECPAR 2006, 7th International Conference, Rio de Janeiro, Brazil, June 10-13, 2006, Revised Selected and Invited Papers*, ser. Lecture Notes in Computer Science, M. J. Daydé, J. M. L. M. Palma, A. L. G. A. Coutinho, E. Pacitti, and J. C. Lopes, Eds., vol. 4395. Springer, 2006, pp. 1–12.

[17] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School of Computing, Queen's University, Kingston, Ontario, Tech. Rep., 2006.

[18] A. L. Rosenberg, "Scheduling dags opportunistically: The dream and the reality circa 2016," in *Euro-Par 2016: Parallel Processing - 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings*, ser. Lecture Notes in Computer Science, P. Dutot and D. Trystram, Eds., vol. 9833. Springer, 2016, pp. 22–33.

[19] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *J. Grid Comput.*, vol. 13, no. 4, pp. 457–493, 2015.

[20] Y. Simmhan, L. Ramakrishnan, G. Antoniu, and C. A. Goble, "Cloud computing for data-driven science and engineering," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 4, pp. 947–949, 2016.

[21] J. Yu, R. Buyya, and K. Ramamohanarao, *Workflow Scheduling Algorithms for Grid Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 173–214. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69277-5_7

[22] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneou s computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, Mar 2002.

[23] M. Rahman, S. Venugopal, and R. Buyya, "A dynamic critical path algorithm for scheduling scientific workflow appl ications on global grids," in *e-Science and Grid Computing, IEEE International Conference on*, Dec 2007, pp. 35–42.

[24] R. Sakellariou and H. Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18t h International*, April 2004, pp. 111–.

[25] A. Benoit, U. V. Catalyurek, Y. Robert, and E. Saule, "A Survey of Pipelined Workflow Scheduling: Models and Algorithms," *ACM Computing Surveys*, vol. 45, no. 4, 2013.

[26] G. Aupy, M. Shantharam, A. Benoit, Y. Robert, and P. Raghavan, "Co-scheduling algorithms for high-throughput workload execution," *Journal of Scheduling*, 2015.

[27] O. Beaumont, A. Legrand, and Y. Robert, "The master-slave paradigm with heterogeneous processors," *IEEE Trans. Parallel Distributed Systems*, vol. 14, no. 9, pp. 897–908, 2003.

[28] J. Quintin and F. Wagner, "Hierarchical work-stealing," in *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part I*, ser. Lecture Notes in Computer Science, P. D'Ambra, M. R. Guarracino, and D. Talia, Eds., vol. 6271. Springer, 2010, pp. 217–229.

[29] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. T. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[30] J. M. Wozniak, M. Wilde, and D. S. Katz, "JETS: language and system support for many-parallel-task workflows," *J. Grid Comput.*, vol. 11, no. 3, pp. 341–360, 2013.

[31] I. Raicu, I. T. Foster, and Y. Zhao, "Guest editors' introduction: Special section on many-task computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 897–898, 2011.

[32] Z. Zhang, A. Espinosa, K. Iskra, I. Raicu, I. T. Foster, and M. Wilde, "Design and evaluation of a collective IO model for loosely coupled petascale programming," *CoRR*, vol. abs/0901.0134, 2009.

[33] M. Hategan, J. M. Wozniak, and K. Maheshwari, "Coasters: Uniform resource provisioning and access for clouds and grids," in *IEEE 4th International Conference on Utility and Cloud Computing, UCC 2011, Melbourne, Australia, December 5-8, 2011*. IEEE Computer Society, 2011, pp. 114–121.

[34] "Amazon ec2. http://aws.amazon.com/ec2/."

[35] "enstratus. http://www.enstratus.com."

[36] "Rightscale. http://rightscale.com."

[37] "Scalr. https://www.scalr.net."

[38] Y. C. Lee, H. Han, and A. Y. Zomaya, "On resource efficiency of workflow schedules," in *Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10-12 June, 2014*, ser. Procedia Computer Science, D. Abramson, M. Lees, V. V. Krzhizhanovskaya, J. Dongarra, and P. M. A. Sloot, Eds., vol. 29. Elsevier, 2014, pp. 534–545.

[39] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Conference on High Performance Computing Networking, Storage and Analysis, SC 2011, Seattle, WA, USA, November 12-18, 2011*, S. Lathrop, J. Costa, and W. Kramer, Eds. ACM, 2011, pp. 49:1–49:12.

[40] P. Hoenisch, S. Schulte, and S. Dustdar, "Workflow scheduling and resource allocation for cloud-based execution of elastic processes," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications, Koloa, HI, USA, December 16-18, 2013*. IEEE Computer Society, 2013, pp. 1–8.

[41] S. Esteves and L. Veiga, "WaaS: Workflow-as-a-Service for the Cloud with Scheduling of Continuous and Data-Intensive Workflows," *The Computer Journal*, vol. 58, 2015.

[42] M. Frîncu, "Dynamic scheduling algorithm for heterogeneous environments with regular task input from multiple requests," in *Advances in Grid and Pervasive Computing, 4th International Conference, GPC 2009, Geneva, Switzerland, May 4-8, 2009. Proceedings*, ser. Lecture Notes in Computer Science, N. Abdennadher and D. Petcu, Eds., vol. 5529. Springer, 2009, pp. 199–210.

[43] M. E. Frîncu, "Scheduling service oriented workflows inside clouds using an adaptive agent based approach," in *Handbook of Cloud Computing.*, B. Furht and A. Escalante, Eds. Springer, 2010, pp. 159–182.

[44] A. G. Kumbhare, Y. L. Simmhan, M. Frîncu, and V. K. Prasanna, "Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure," *IEEE T. Cloud Computing*, vol. 3, no. 2, pp. 105–118, 2015.

[45] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.

[46] D. Zinn, Q. Hart, T. M. McPhillips, B. Ludäscher, Y. Simmhan, M. Giakkoupis, and V. K. Prasanna, "Towards reliable, performant workflows for streaming-applications on cloud platforms," in *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2011, Newport Beach, CA, USA, May 23-26, 2011*. IEEE Computer Society, 2011, pp. 235–244.

[47] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, Oct 2000.

[48] Y. Liu and E. Knightly, "Opportunistic fair scheduling over multiple wireless channels," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 2, March 2003, pp. 1106–1115 vol.2.

[49] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 323–336.

[50] D. Dolev, D. G. Feitelson, J. Y. Halpern, R. Kupferman, and N. Linial, "No justified complaints: On fair sharing of multiple resources," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: ACM, 2012, pp. 68–75.

[51] H. Wang and P. Varman, "Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, ser. FAST'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 229–242.

[52] A. Gutman and N. Nisan, "Fair allocation without trade," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, ser. AAMAS '12. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 719–728.

[53] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 1206–1214.

[54] D. C. Parkes, A. D. Procaccia, and N. Shah, "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities," in *Proceedings of the 13th ACM Conference on Electronic Commerce*, ser. EC '12. New York, NY, USA: ACM, 2012, pp. 808–825.

[55] R. Cole, V. Gkatzelis, and G. Goel, "Mechanism design for fair division: Allocating divisible items without payments," in *Proceedings of the Fourteenth ACM Conference on Electronic Commerce*, ser. EC '13. New York, NY, USA: ACM, 2013, pp. 251–268.

[56] N. Cesa-Bianchi, "Multi-armed bandit problem," in *Encyclopedia of Algorithms*, 2016, pp. 1356–1359.

[57] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under SLA constraints," in *MASCOTS 2010, 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Miami, Florida, USA, August 17-19, 2010*. IEEE Computer Society, 2010, pp. 257–266.

[58] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M. Su, T. A. Prince, and R. Williams, "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking," *Int. J. Comput. Sci. Eng.*, vol. 4, no. 2, pp. 73–87, Jul. 2009.

[59] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, Nov 2008, pp. 1–10.

[60] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013, special Section: Recent Developments in High Performance Computing and Security.

[61] I. Kissami, C. Cérin, F. Benkhaldoun, and G. Scarella, "Towards parallel CFD computation for the ADAPT framework," in *Algorithms and Architectures for Parallel Processing - 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings*, ser. Lecture Notes in Computer Science, J. Carretero, J. G. Blas, R. K. L. Ko, P. Mueller, and K. Nakano, Eds., vol. 10048. Springer, 2016, pp. 374–387.