Tasks Allocation under the Publication-Subscription Scheme: Application to a Scientific Workflow Engine

Leila Abidi, Christophe Cérin *(Member, IEEE)*, Jonathan Lejeune and Yanik Ngoko
E-mail: {leila.abidi,christophe.cerin,jonathan.lejeune,yanik.ngoko}@lipn.univ-paris13.fr

Abstract—In a pay-on-demand computing system where the user should be in the center of our concerns, the system allocate (and thus the users pay for) resources that are not always truly needed. Since the users always want to get their money's worth, they reaquire that the booked resources are fully utilized and as much as possible. To date, researchers have pursued rule-based mechanisms to attempt to automate the matching between computing requirements and computing resources. However, most of these 'auto-scaling' mechanisms only support simple resource utilization metrics and do not specifically consider both user performance requirements and fairness in resource usage. In this paper, we present tasks allocation policies inside the RedisDG approach for executing scientific workflows under requirements on the load and the fairness of the computing nodes. One central concept in the paper is the Publication/subscription model that is used not only for the modeling of the allocation policies but also to offer a dynamicity intrinsic property for the RedisDG system. Considering a user that has reserved N computing nodes, the goal is intuitively to ensure that the tasks of the workflow are executed on the N computing nodes (fairness condition) such that the 'load' of the computing nodes is as high as possible (load condition). We accomplish our goal by designing algorithms for the allocation of tasks to nodes that depend on different strategies and reflecting different points of view for the performance metrics. We evaluate our approach according to the Montage representative workflow and show the savings in using our approach in the Grid5000 testbed context. This paper covers the full spectrum of the experimental scientific method from the problem specification, its formal modeling, its analysis, the semi-formal derivation of a solution, the implementation, the experimental part of the work and the feedback gained from the experiments a large scale.

Index Terms—Scientific workflow, Scheduling, Heuristics design for allocation, Resource management, Publish/subscribe, Experimental evaluation on large scale systems.

1 Introduction

1.1 The general problem

Resource utilization in computer systems is usually below 10% [1] due primarily but not only to resource over-provisioning. Efficient resource sharing in large scale infrastructure (clouds, grids...) is, as a consequence, challenging. Resource under-provisioning will inevitably scale-down performance and resource over-provisioning can result in idle time, thereby incurring unnecessary costs for the user. Allocating resources is thus one of the key issue to address when we consider *extreme large scale systems*, we mean the coupling of different sorts of large scale systems (clouds, grids, clusters) but also sensor networks and desktop grids aka volonteer computing where the 'nodes' are exposed to the Internet.

1.2 Our context is to execute a scientific workflow

In this paper, the problem of finding a 'good' allocation and resource utilization is considered as a balance between multiple objectives. The architectural context is an IaaS (Infrastructure as a service) that must offer a *workflow engine as a service*. The thesis we adopt in this paper is to consider that a scientific workflow engine shall be organized along three major elements:

1) a protocol for the interactions between the components of the workflow engine;
2) a execution model for the tasks of the workflow;
3) nodes.

1.3 Our principles for allocating resources

In our case, the specialization of the workflow engine is as follows:

1) The protocol for interactions is designed according to the Publication/Subscription model [2];
2) The problem of the choice of an execution model is reduced to a problem of allocating resources from a master to a set of workers; For that We devise heuristics that are based on multiple crieria;
3) nodes may join or leave the workflow system at any time to modeling a dynamic system; The Publication/subscription model helps in realizing this vision.

Recall that the Publish-Subscribe paradigm is an asynchronous mode for communicating between entities [2], [3]. 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.

• *M. Takoudjou is with the NEXEDI company as well as with Laboratoire de Recherche en Informatique de Paris Nord, university of Paris 13, France. M. Ngoko is with the Qarnot Computing company as well as with Laboratoire de Recherche en Informatique de Paris Nord, university of Paris 13, France.*

Manuscript received April 19, 2005; revised September 17, 2014.

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). This total decoupling between the production and the consumption of services increases the scalability by eliminating many sorts of explicit dependencies between participating entities. Eliminating dependencies reduces the coordination needs and consequently the synchronizations between entities. These advantages make the communicating infrastructure well suited to the management of distributed systems and simplify the development of a middleware for the coordination of components in our workflow engine context or for applications running in different domains and communicating through middleware solutions deployed on clouds.

Time decoupling is a key property that allows to make the systems more dynamic. We mean that with the Pub/Sub paradigm, "actors" may leave or enter into the system without interacting, therefore without disrupting the others actors. To enter into a system, one actor subscribes to one channel. To leave a system, one actor unsubscribes to a channel. In this way, we can implement, in a very easy way, a solution for what the cloud community calls *elasticity* i.e. how a cloud autonomously adapts its capacity to workload over time. In our context, there is no elasticity where new compute units are dynamically added/removed: the number of compute units is fixed in advance but all our framework is designed to support elasticity.

1.4 Our objectives

The integration of this workflow engine into a cloud is not introduced in this paper for the sake of completeness. However, to exemplify our work we can imagine a user, connected to a cloud and requesting a *scientific workflow engine* service. He pays for N computing units according to his budget, downloads his workflow description and executable codes, then he clicks on the "run" button. The cloud system deploys the infrastructure, activates the N computing units, executes the workflow. The cloud user needs to be sure that two objectives are fulfilled: (1) all the reserved computing units are used and (2) the processor load is as high as possible.

The maximization of theses two objectives is a hard because tasks can be allocated in two manners:

- either you distribute the tasks as much as possible on the different computing units but they are under used;
- or you consolidate tasks on few computing units and you get a better utilization on few resources but you potentially allocate and pay for unnecessary resources.

The contributions of the papers are twofold. First we provide with a series of new scheduling algorithm for a better usage of computing resources through strong policies for controlling the fairness and second we provide with extended experimentations, on the Grid5000 testbed, to analyze and to validate the proposed algorithms and the impact of the middleware on performance.

1.5 More information about the concepts used in the paper

Publication/Subscription is central in this paper. This concepts has been used in many contexts but never, to the best of our knowledge, in the context of designing a workflow engine or in the context of scheduling tasks.

A more typical example in using Pub/Sub is QoS Monitoring as a Service [4], where an application running on a private cloud is monitored and key performance indicators (cpu load, memory usage, disk-io operations. . .) are generated as publications. These publications are propagated to a third-party monitoring service that confront them against "service level agreement (SLA) publications" in order to detect violations of the SLA. Fluentd[1] and fluent-plugin-redisstore[2] are competitive Open source projects to implement this vision.

Other applications Pub-Sub include e-Health systems [5] where the authors describe an e-Health application scenario for monitoring patients with chronic diseases and show how and encryption schema can be used to provide confidentiality of the patients' personal and medical data, or the canonical example of stock trading [6] where authors evaluate their system in using a model of stock quotes application and through simulations.

In this paper we use the traditional form of Pub-Sub system and not more elaborated forms able to filter publications in order to limit the traffic [7], [8], [9], [10] or implementing privacy-preserving encrypted filtering which is a growing requirement for applications running on multiple untrusted private clouds providing the pub/sub service. This technique [5], [11] allows that publishers and subscribers do not share secret keys, such a requirement being against the loose-coupling of the model. Second, this technique allows brokers to route events by matching encrypted events against encrypted filters.

For the implementation, we use Redis[3] a well known open source project, because of its ability regarding the Pub-Sub service but also for its ability to store data. In our workflow management system we also need to record the codes to execute and and input and output files they handle along the computation. It is the first time to our knowledge that Redis is used in the context of a workflow engine. Despite the fact that only basic Redis functionalities are used, we will exhibit specific problems related to the fairness for our protocol. In this paper we do not control the fairness of a Pub-Sub substratum but the fairness of a protocol based on Pub-Sub hence no need for a more elaborated framework such as StreamHub [12] regarding the Pub-Sub paradigm.

Our workflow management system is of Desktop Grid [13] inspiration. We deal with Desktop Grid 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

1. http://www.fluentd.org
2. https://github.com/pokehanai/fluent-plugin-redis-store
3. http://redis.io

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 Pub-Sub 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 [13] have been developed using a centralized model. The most popular are BOINC [14], Condor [15], OurGrid [16] and Xtremweb [17]. In this paper we investigate the RedisDG [18], [19], [20] desktop grid middleware which is a light desktop grid middleware that has been formally designed. We also show in this paper that RedisDG is able to execute workflows, i.e., it can be considered as a workflow engine.

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

The goal and the difficulties are to define a desktop grid middleware able (the RedisDG protocol is the core component of our study) to support workflows that is light enough to be integrated easily into a cloud and using current Web technologies (in our case the Publish-Subscribe paradigm and the Redis framework).

1.6 Organization of the paper

The organization of the paper is as follows. In section **BLA BLA BLA** Section 5 is about the related works and section 6 concludes the paper.

2 RedisDG description

2.1 RedisDG modeling

We conducted in [21], [22] a formal modeling, based on our initial modeling of the publish-subscribe paradigm [23] as and adapted to Redis interactions. Indeed, Redis, our implementation language, has special properties regarding a publish instruction, for instance, faced to no registration. We invite the reader to review the bibliography for more technical details that are not the core of this paper.

2.2 RedisDG protocol

In this section, we remind the coordination algorithm of RedisDG system which is the core of our paper. It correspond 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; since the results returned by the workers can be manipulated or altered by malicious workers. The general objectives for the RedisDG protocol are:

- Using an asynchronous paradigm (publish-subscribe) that ensures, as much as possible, a complete decoupling between the coordination steps (for performance reasons);
- Ensuring the system resilience by duplicating tasks and actors. Even if the system is asynchronous and the tasks are duplicated, we need to ensure the progress of tasks execution. We also assume that actors are duplicated for resilience reasons;

In Figure 1, 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.

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 1 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 selects Workers according to SLA criteria.
7) The Workers, listening beforehand on the channel *TasksToDo* start executing the published tasks. The event ’execution in progress’ is published on the channel *TasksInProgress*.
8) 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

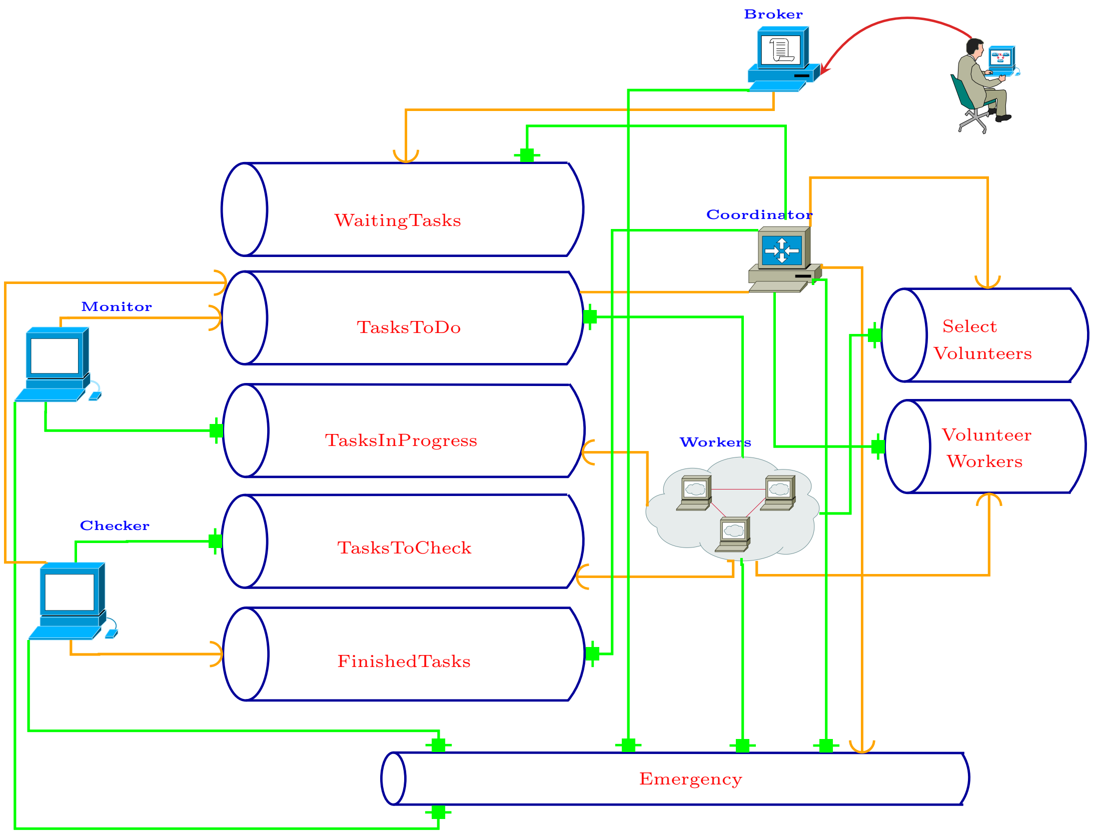

Fig. 1. Interactions between components of the RedisDG system

channel *TasksToDo*, in order to make the execution of the task done by other Workers.

9) Once the execution is completed, the Worker publishes the task on channel *TasksToCheck*.
10) The Checker verifies the result returned and publishes the corresponding task on the channel *FinishedTasks*.
11) The Coordinator checks dependencies between completed tasks and those waiting, and restarts the process in step (4).
12) 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.

Summarizing, it is important to understand all the interactions in this protocol because we will explain later on some pitfalls leading to observational behaviors on real infrastructures. These unexpected behaviors, not visible in the modeling steps, are related to the scheduling policies that we introduce now.

3 Scheduling mechanisms and performance metrics

3.1 Definitions

In this section we introduce the key ideas in order to authorize some post-mortem analysis of the executions that serve to estimate the fairness metric of an experiment.

3.1.1 Experiment definition

In order to define the notion of fairness we first need to define three basic sets attached to an experiment:

- Π being the set of scheduled tasks in the experiment
- W being the set of workers having participated in the experiment
- T being the set of instants representing the execution time of the experiment. 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 beginning and the end of an experiment.

Then we define two selecting methods over the tasks or over time.

TASKS SET PARTITIONING: In order to select specific tasks in Π according to a criteria we need to partition the set Π in disjoint subsets Π_i. Then $\Pi = \cup\Pi_i$. For example, if the criteria is the depth of the task in the task precedence graph, then $p \in \Pi_i$ if and only if $depth(p) = i$.

TIME WINDOWING: In order to select specific time window of an experiment, 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.

Thanks to this two previous selecting methods, we can now consider a 'spatial' view by zooming on specific region of the task graph and/or a 'temporal' view according to the experiment time by specifying a time window.

3.1.2 Execution definition

We now define the set E being the set of all task executions. A task execution $e \in E \subset \Pi \times W \times T \times T$ is a quadruplet (p_e, w_e, b_e, e_e) with p_e the task associated to the execution e, w_e is the worker executing e, b_e is the moment when task e_e starts its execution on worker w_e and e_e is the moment when p_e finishes its execution on worker w_e.

The subset $E_{\Pi_i} \subseteq E$, defines all execution $e \in E$ where the associate task p_e belongs to Π_i. Formally,

$$E_{\Pi_i} = \{e \mid e \in E \wedge p_e \in \Pi_i\}$$

The subset $E_{tw} \subseteq E$, defines all execution $e \in E$ 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,

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

Finally, 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:

$$T_{exe}(e, tw) = \begin{cases} e \in E_{tw} \Rightarrow min(e_e, e_{tw}) - max(b_e, b_{tw}) \\ e \notin E_{tw} \Rightarrow 0 \end{cases}$$

3.1.3 Fairness definition

We define the *fairness* as the standard deviation of the cumulative computing time for each worker on a given time window and a given partition of tasks. Formally this metric is defined by the function

$$Fairness(tw, \Pi_i) = \sigma\{CT(w, tw, \Pi_i) \mid w \in W\}$$

where the function CT is defined as

$$CT(w, tw, \Pi_i) = \sum_{e \in E_{tw} \cap E_{\Pi_i}} T_{exe}(e, tw)$$

Thanks to this general definition, we can define the global fairness of the experiment being the fairness of the full period of time for executing the whole tasks graph denoted $GlobalFairness$ defined simply as

$$GlobalFairness() = Fairness((t_0, t_{end}), \Pi)$$

The general objective for a 'good' fairness is to keep the standard deviation σ as low as possible under the requirements of the spacial view or the two temporal views.

3.1.4 Workflow execution time definition

TODO

3.2 Scheduling heuristics

In this section we introduce seven heuristics for scheduling the tasks of a workflow 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 compromise between the fairness and the response time. We also want to observe the impact of basic scheduling strategies on the overall behavior and properties of the RedisDG system.

These heuristics are designed to mix a more or less clairvoyant strategy for the time we spend in waiting information and a more or less clairvoyant strategy on the loads. When we do not use any information about the past we manage a non-clairvoyant strategy, when we use few information about the past (according to a threshold) we say that we manage a semi-clairvoyant strategy and when we authorize a long time windowing to collect information we say that the strategy is clairvoyant. In short, we manage and mix non-clairvoyant or semi-clairvoyant or clairvoyant strategies that we named *-strategy.

For instance, the first heuristic (H1) is both non-claivoyant for the time and the loads because we decide on the basis First Come First Serve (FCFS). The idea is to cover a large number of situations and to determine experimentally the best heuristic according to the pair of *-strategy for the time windowing and the loads.

HEURISTIC H1 (FCFS): in this case the coordinator publishes the independent tasks to all the workers and all the workers reply that they want to participate. There is no SLA and this strategy will generate many messages but the liveness of the execution is guaranteed.

HEURISTIC H2: in this case only free workers reply to the coordinator. This means that w does not serve any request. When the coordinator receives a reply from a worker w for task t, it first saves this will for t, but as t may already be allocated to some worker or even finished, and since we know that w is a free worker, the coordinator allocates to w a task that has not yet been executed among those for which it wished to participate. This strategy correspond to a semi-clairvoyant strategy for the time windowing and a non-clairvoyant strategy for the loads.

HEURISTIC H3: let λ_{max} be the first date for which a free worker has replied. Note that $lambda_{max}$ is a theoretical value that needs to be learned during the workflow execution. The coordinator waits until λ_{max} and chooses randomly between the first free workers. The random process should provide a better spatial distribution among the workers. This heuristic is clairvoyant for the time windowing and non-clairvoyant for the loads.

HEURISTIC H3': in this case, in supplement to H3, the worker is selected according to a probability. This probability is computed as $p' = 1 - p$ where p is the ratio between the accumulated execution time spent in the workflow and the accumulated execution time spent on the worker. The difference with the previous heuristic is on the random decision process. Here, we have a semi-clairvoyant strategy for the loads and a clairvoyant strategy for the time windowing.

HEURISTIC H3”: as for H3, the coordinator waits until λ_{max} and it chooses the worker with the minimal work. Here, we have a clairvoyant strategy for the loads (because it does not depends on a probability) and a clairvoyant strategy for the time windowing.

HEURISTIC H3”’: as for H3, the coordinator waits until λ_{max} and it chooses the worker with the minimal work, while considering the task it will process. In this case we are guessing that we have an estimate on the execution time of each task (for the Montage workflow, it may happens). This heuristic corresponds to the previous one in considering the estimated time to be zero. Here, we have a clairvoyant strategy for the loads (because it assume a knowledge on the execution time) and a clairvoyant strategy for the time windowing.

HEURISTIC H6: apprendre λ_{max} par un process de bandit (la politique e-greedy) et selectionner selon H5.

This heuristic is a combination of two approaches: a multi-armed bandit approach [] and an assignment problem approach []. Given n workers $w_1, w_2, \cdots, w_n$ the heuristic maintains a vector $c_1, c_2, \cdots, c_n$ of the accumulated execution time on each worker as an intermediate data structure.

Then we combine two approaches as follows:

1) In the *lower model*, the baseline scenario is the following: suppose that in the vector of tasks to do, we have m tasks and in the bunch of subscribers we have $m' > m$ subscribers. How to allocate the m tasks to subscribers in order to respect the fairness? To effectively deal with this problem, we propose to add an additional information that is the estimated time duration of each task. Then we can reduce the problem to an assignment problem that is resolved with the method of the Hungarian algorithm [24].
2) The lower model itself fits into an *upper model* that addresses the question of what value we need to choose for m'. Indeed, if we take m' small, we give a priority to fast workers to the detriment of the fairness. If we take m' large we get a fair solution. In general, the strategy is to give a discrete finite range of authorized values m'. So suppose we keep $m'_1, m'_2, \cdots, m'_k$ subscribers. Therefore the decision at the upper level for the scheduler is to know for each publish (we mean a collection of m independent tasks it received) what is the maximum number of subscribers that it chooses to apply the lower level. For it, we propose to use the model of the one-armed bandit.

All these heuristics will serve later on to graphically represent the overall performance by drawing a 2D plot with the fairness on the x-scale and the execution time of the workflow on the y-scale.

4 Experimental Analysis

In the previous experiments, because of the use of the PubSub mechanism and the Redis implementation, we have observed problems with fairness: the worker with the smallest latency with the Redis server has more chance to be served. We have introduced a ’brute force algorithm’ allocating the requests in a round robin-like manner to improve the situation.

We are now designing alternate solutions in putting more intelligence in the control of our RedisDG middleware. Of course, an alternate solution would be to modify the Redis implementation but we want to be independent of the publication-subscription system we use. Another issue is to keep the RedisDG protocol as it is (because it has been proven): the modifications of the implementation should be confined to the scheduling module.

4.1 Workflow used in the experiments

Workflow technologies are responsible for the scheduling of computational tasks on distributed resources, for the management of dependencies between taks and also for the staging of data in and outside the execution sites [25].

The Pegasus [26], [27] project encompassa set of technologies that contribute to the execution of applications based on workflows and in very different environment such as clusters located on a campus, grids and clouds. Pegasus automatically localize the input data et computing resources that are required by the application. It permits to scientists to focus on the building of an abstract view (the workflow) with no need to pay attention to the execution environment or on some low level requirements about the middleware (Condor). Pegasus also offer to the community an efficient coordination of the multiple distributed computing resources.

The MONTAGE [28] project has been created by NASA/IPAC *Infrared Science Archive* as an open-source toolbox to generate personalized mosiac of the sky from images in the *Flexible Image Transport System* (FITS) format. During the final production of the mosai, the geometry of the output is computed from the geometry of the input [29], [30]. The MONTAGE application has been represented as a workflow that can be executed on the Teragrid [31] infrastructure for instance.

On Figure 2, we show a MONTAGE workflow that is small (on ly 20 nodes) that has been generated from the workflow generator [32]. What is important to notice is the shape of the workflow and the corresponding independent part that could be executed in parallel.

In tabular 1, we show an instance of MONTAGE application, parametrized by a degree of 2 that we have executed with RedisDG. This example will be use in an intensive way in the paper. It exibit a workflow with 1446 tasks and 3722 dependency links between tasks. The execution of this quite large instance requires 9423 input files (including the intermediary files) et it generates 2889 files (including the intermediary files).

Basically, in order to validate our RedisDG workflow engine we have done experiments on the Grid’5000 [33] testbed in using (typically) from 30 to 200 nodes taken in the cities of Nancy, Grenoble and Lyon.

4.2 Testing heuristic H1

In this experiment we use heuristic H1 where each worker replies to a request for participation and the coordinator

Level	Task	Description	Number of tasks
1	mProject	The outputs of the jobs are the reprojected image and an 'area' image that consists of the fraction of the image that belongs in the final mosaic.	301
2	mDiffFit	computes a difference for each pair of overlapping images.	838
3	mConcatFit	fits the description of a data aggregation job	1
4	mBgModel	apply a correction to each image to obtain a good global fit	1
5	mBackground	apply a correction on the backgroung of the images	301
6	mImgtbl	aggregates metadata from all the images and creates a table that may be used by other jobs in the workflow	1
7	mAdd	co-adds all the reprojected images to generate the final mosaic in FITS format as well as an area image that may be used in further computation	1
8	mShrink	reduction in size of the FITS images by averaging blocks of pixels	1
9	mJPEG	convert to JPEG format	1

TABLE 1
Characteristics of a MONTAGE workflow with 1446 tasks

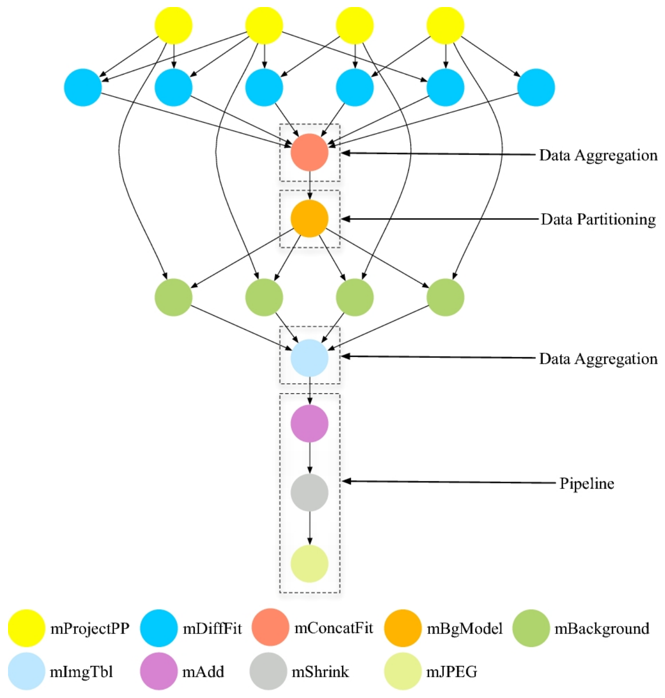

Fig. 2. An example of the MONTAGE workflow

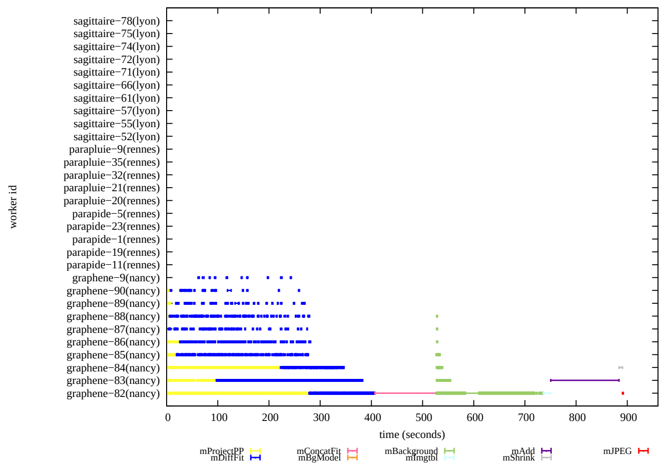

Fig. 3. Execution time of tasks 1 to 301

selects the winner on the basis of the First Come First Serve principle.

Figure 3 shows the execution time over 30 workers taken in Nancy (10), Rennes (10) and Lyon (10) with a Redis server in Nancy. These plots corresponds to the first level of the workflow depicted on Figure 2. Obviously, some workers, in particular `graphene-82`, respond better than others. This machine is physically located also on the Nancy site as well as the Redis server. This explains the observation.

We also conducted yet another experiment with 169 workers (170 minus 1 coordinator) on the same 3 sites and in the same proportion. We also observed the same phenomenon.

Another technical fact that we may consider in this case is the performance of the Redis server. In [22] we demonstrated, experimentally by a series of publish/subscribe calls, simmultaneous or not, that above 300 simultaneous publications the Redis server started to be overloaded but without loss of messages. At the begining of our current experiment, RedisDG generates $301 * 169 * 2 = 101738$ messages, almost simultaneously for the publication steps of the 301 independant tasts, received by 169 workers that all respond to the server, through yet another publication. This large amount of messages may also explain some performance degradation.

Following this series of experiments on a real platform, we conclude that the scheduling of tasks according to the H1 heuristic is not fair. All machines at the disposal of the application were not asked to participate in the computation. And even those involved didn't have equal loads of work. Thus, with this scheduling policy, some machines have more work than others.

Also, in our system, under heuristic H1, a worker responsible for carrying out a task, can always publish his volunteering to run other tasks and can be selected several times by the Coordinator. This is explained by the fact that, in the implementation, the worker has two *threads*: a thread responsible for the execution of a task and the other one responsible for detecting the publication of new tasks to do and also to announce the volunteering for the worker.

4.3 Testing heuristic H2

We have just observed that only 18% of the workers have participated into the effective execution of the 1446 tasks of the MONTAGE worklow over 169 workers. The new mecanism for selecting workers is as follows and according to heuristic H2. We wait during a 'time slot' for the arrival of a message for participating and we selected a worker that has not yet participated. When (almost) all the workers have participated at least one time, we restart a new round.

Then, we have restarted our MONTAGE application (the same workflow with 1446 tasks) but on 200 nodes of the GRID'5000 testbed. Figure 4 summarizes the behaviour of the execution. We observe that the shape is completely different. This is explained by the degree of paralelism that is now much more important. According to statistics conducted on Log files generated by RedisDG, we have founf that all of the workers, i.e. the 200, participated in the execution of the application (but not necessarily in an equal way). We also note that the total execution time has decreased from 16 minutes to 4 minutes, which is flattering for scheduling policy.

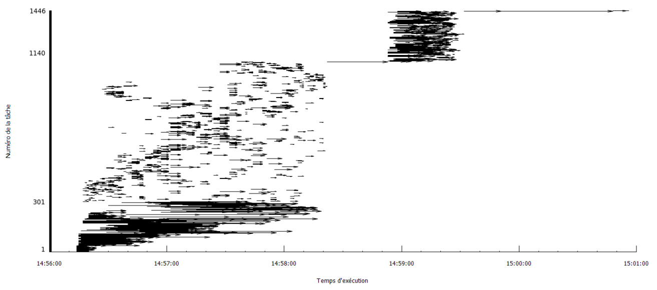

Fig. 4. Execution time of the MONTAGE Workflow (1446 tasks) on 200 workers

We have also restarted the same application on 340 workers. We observe on Figure 5 that the degree of parallelism is stil more important. . . but the execution time is also more important than the one with 200 machines. This is due to the selection of slow nodes for the execution of the last 4 sequential tasks. This problem is localized and we explain it because our SLA policy has been desactivated for this experiment.

We also observed loss of messages in response to the solicitations of the Coordinator. Starvation situations can happen but it is more an engineering problem (configuration of the Redis server; number of simultaneous requests that the coordinator and the Redis server may serve. . .) rather than a scientific issue. But it is certain that this will have an impact ultimately on the implementation of the RedisDG scheduler for a production environment.

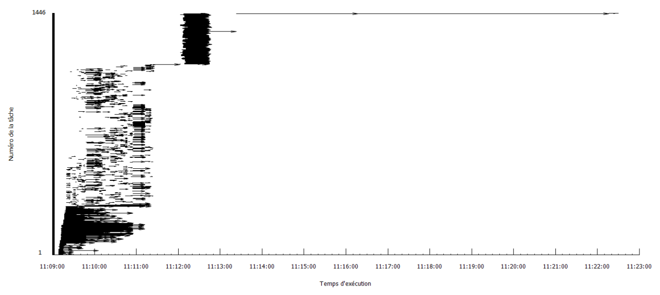

Fig. 5. Execution time of the MONTAGE workflow (1446 tasks) on 340 workers

5 Related works

5.1 Introduction

Applications in e-Science are becoming increasingly large-scale and complex. These applications are often in the form of workflows [29] such as Montage [?], Blast [34], CyberShake [35] with a large number of software components and modules. Since workflow managers are gaining in complexity and in importance, designers need a deep understanding of what is required by a workflow in terms of resources in order to improve the basic components such as the scheduler, the mechanism for provisioning with resources and the data manager [36].

Reader can find related works about the specification of desktop grid middleware and related issues in [13], [20], related work on the integration of applications in the SlapOS cloud in [37] for the BOINC use case. This paper aims at deploying the BOINC infrastructure on the fly, when a user needs it and without any system administrator intervention. The whole process is automated and constitutes the key difficulty. Reader can also find related works on workflow management systems in the synthesis from Valduriez [38] or in the work of Carole Goble [39]. These two papers are more related to cloud computing and data intensive scientific workflows in putting an emphasis on data management.

In what follows we put an emphasis only on the workflow scheduling problems, in general. Most of the works are about load balancing (some form of fairness), others are not related to DAG scheduling but informative. To summarize, in task/graph scheduling, heuristics from the literature are not adapted to our context because:

- they do not take into account our two objectives;
- they are 'clairvoyant'-like. They take a scheduling decision on the basis of a fixed number of workers. With the Pub-Sub paradigm in mind a 'more efficient' worker may join the system in the near future.

5.2 Scheduling and workflow management systems

In [40] 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 is not part of our assumption) and the DAG 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 [41] 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 [42] 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 [43] 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 [44] 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 [45] component is closely related to Swift and it provides high performance support for many-parallel-task computing (MPTC). The MTC model [46] consists of many, usually sequential, individual tasks that are executed on processor cores, without intertask communication. The tasks communicate only through the standard filesystem interfaces, and optimization are possible [47]. 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 Redis DG, 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 [45], authors notice that '*the native schedulers and application-launch mechanisms of today's supercomputers do not support a sufficiently fast task scheduling, startup, 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 [48]. 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 [49] and some other cloud management services such as enStratus [50], RightScale [51], and Scalr [52] 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 [53] 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 [54] 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 [55] 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 [56] 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.

5.3 A focus on workflow scheduling according to a Service Oriented view

In a series of works [57], [58], [59], [60] 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 [57] 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. They 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 author propose that this information be obtained by using historical data and applying some learning mechanisms.

In [60] 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 [61] 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 adaptions 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 [62] simulator.

In [63] 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.

6 Conclusion

The conclusion goes here.

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners. The authors would like to thank the NEXEDI company for its helpful comment about the implementation issues. This work has been done partly under the Wendelin grant from the ministry of industry in France (Investissement d'avenir).

References

[1] A. G. Greenberg, J. R. Hamilton, D. A. Maltz, and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," *Computer Communication Review*, vol. 39, pp. 68–73, 2009.

[2] 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.

[3] H. Abbes and J.-C. Dubacq, "Analysis of Peer-to-Peer Protocols Performance for Establishing a Decentralized Desktop Grid Middleware," in *Euro-Par Workshops*, ser. Lecture Notes in Computer Science, E. César, M. Alexander, A. Streit, J. L. Träff, C. Cérin, A. Knüpfer, D. Kranzlmüller, and S. Jha, Eds., vol. 5415. Springer, 2008, p. 235âĂŞ246.

[4] L. Romano, D. De Mari, Z. Jerzak, and C. Fetzer, "A novel approach to qos monitoring in the cloud," in *Proceedings of the 2011 First International Conference on Data Compression, Communications and Processing*, ser. CCP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 45–51. [Online]. Available: http://dx.doi.org/10.1109/CCP.2011.49

[5] M. Ion, G. Russello, and B. Crispo, "An implementation of event and filter confidentiality in pub/sub systems and its application to e-health," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 696–698. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866401

[6] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot: Content-based publish/subscribe over p2p networks," in *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, ser. Middleware '04. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 254–273. [Online]. Available: http://dl.acm.org/citation.cfm?id=1045658.1045677

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Trans. Comput. Syst.*, vol. 19, no. 3, pp. 332–383, Aug. 2001. [Online]. Available: http://doi.acm.org/10.1145/380749.380767

[8] R. Chand and P. Felber, "Scalable distribution of xml content with xnet," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 4, pp. 447–461, Apr. 2008. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2007.70816

[9] A. K. Y. Cheung and H.-A. Jacobsen, "Green resource allocation algorithms for publish/subscribe systems," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ser. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 812–823. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2011.82

[10] Y. Yoon, V. Muthusamy, and H.-A. Jacobsen, "Foundations for highly available content-based publish/subscribe overlays," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ser. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 800–811. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2011.93

[11] S. Choi, G. Ghinita, and E. Bertino, "A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations," in *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part I*, ser. DEXA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 368–384. [Online]. Available: http://dl.acm.org/citation.cfm?id=1881867.1881908

[12] R. Barazzutti, P. Felber, C. Fetzer, E. Onica, J.-F. Pineau, M. Pasin, E. Rivière, and S. Weigert, "Streamhub: A massively parallel architecture for high-performance content-based publish/subscribe," in *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, ser. DEBS '13. New York, NY, USA: ACM, 2013, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/2488222.2488260

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

[14] 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.

[15] 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.

[16] 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.

[17] 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. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2001.923246

[18] 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. [Online]. Available: http://doi.acm.org/10.1145/2480362.2480510

[19] 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. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38027-3_27

[20] 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. [Online]. Available: http://dx.doi.org/10.1109/SBAC-PAD.2014.50

[21] 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. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30767-6_7

[22] 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.

[23] 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. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SCC.2011.42

[24] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society of Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, March 1957.

[25] T. F. Jun Qin, Ed., *Scientific Workflows: Programming, Optimization, and Synthesis With ASKALON and AWDL.* Springer-Verlag Berlin and Heidelberg GmbH and Co. K, 14 aoÃżt 2012.

[26] http://pegasus.isi.edu/projects/pegasus.

[27] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, no. 0, p. âĂŞ, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X14002015

[28] http://montage.ipac.caltech.edu/index.html.

[29] 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.

[30] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12001732

[31] http://www.teragrid.org/.

[32] 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.

[33] http://www.grid5000.fr.

[34] S. F. Altschul, T. L. Madden, A. A. SchÃd'ffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and psiblast: a new generation of protein database search programs," *NUCLEIC ACIDS RESEARCH*, vol. 25, no. 17, pp. 3389–3402, 1997.

[35] R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, and K. Vahi, "CyberShake: A Physics-Based Seismic Hazard Model for Southern California," *Pure and Applied Geophysics*, vol. 168, pp. 367–381, 2011.

[36] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Gener. Comput. Syst.*, vol. 25, no. 5, pp. 528–540, may 2009. [Online]. Available: http://dx.doi.org/10.1016/j.future.2008.06.012

[37] 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. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2013.59

[38] 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. [Online]. Available: http://dx.doi.org/10.1007/s10723-015-9329-8

[39] 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. [Online]. Available: http://dx.doi.org/10.1002/cpe.3668

[40] 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. [Online]. Available: https://hal.inria.fr/hal-00926178

[41] G. Aupy, M. Shantharam, A. Benoit, Y. Robert, and P. Raghavan, "Co-scheduling algorithms for high-throughput workload execution," *Journal of Scheduling*, 2015. [Online]. Available: https://hal.inria.fr/hal-01252366

[42] 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.

[43] 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. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15277-1_21

[44] 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. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2011.05.005

[45] 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. [Online]. Available: http://dx.doi.org/10.1007/s10723-013-9259-2

[46] 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. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.138

[47] 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. [Online]. Available: http://arxiv.org/abs/0901.0134

[48] 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. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/UCC.2011.25

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

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

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

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

[53] 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. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2014.05.048

[54] 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. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063449

[55] 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. [Online]. Available: http://dx.doi.org/10.1109/SOCA.2013.44

[56] 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.

[57] 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. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01671-4_19

[58] ——, "Distributed scheduling policy in service oriented environments," in *11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC*

2009, Timisoara, Romania, September 26-29, 2009, S. M. Watt, V. Negru, T. Ida, T. Jebelean, D. Petcu, and D. Zaharie, Eds. IEEE Computer Society, 2009, pp. 205–212. [Online]. Available: http://dx.doi.org/10.1109/SYNASC.2009.24

[59] M. Frîncu and D. Petcu, "Osyris: a nature inspired workflow engine for service oriented environments," *Scalable Computing: Practice and Experience*, vol. 11, no. 1, 2010. [Online]. Available: http://www.scpe.org/index.php/scpe/article/view/642

[60] 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. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-6524-0_7

[61] 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. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TCC.2015.2394316

[62] 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. [Online]. Available: http://dx.doi.org/10.1002/spe.995

[63] 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. [Online]. Available: http://dx.doi.org/10.1109/CCGrid.2011.74