Communication-aware task placement for workflow scheduling on DaaS-based Cloud

Hadrien Croubois, Eddy Caron

Univ. Lyon, ENS de Lyon, Inria, CNRS, Université Claude-Bernard Lyon 1, LIP.

{firstname}.{lastname}@ens-lyon.fr

***Abstract*—Cloud platforms have emerged as a leading solution for computation. In the meantime, large computations have shifted from big parallel tasks to workflows of smaller tasks with data dependencies between them. Task placement is a major issue on Cloud platforms, especially considering the impact of data exchanges on cost and makespan. In this paper, we investigate the consequences of network contention regarding the use of existing scheduling policies on DaaS-based platforms (DaaS for Data as a Service). We show here that the legacy algorithms use inefficient network models. We then modify those algorithms using a new model inspired by DaaS-based Cloud platforms. Thus, we manage to statically pack tasks so that a batch scheduler could deploy many real-time submitted workflows on a dynamic Cloud platform. Simulations of Fork-Join workflows deployment using SimGrid show that our algorithm reduces computation time as well as deployment costs.**

***Index Terms*—workflow, Cloud, scheduling, clustering**

I. Introduction

Nowadays, computing platforms are used to execute increasingly complex operations composed of multiple interdependent tasks. Examples of such workflows are given in [1]. These can be submitted in real time by multiple users. Meanwhile, computing platforms have moved from institution-owned clusters to externalized Clouds. This new computing paradigm calls for new mechanisms to efficiently schedule tasks and provision resources.

Previous scheduling mechanisms were designed to statically schedule workflows on fixed computing clusters, and focused on communication cost. On Cloud-based platforms, recent approaches now focus on other issues such as Virtual Machine (VM) provisioning, resource sharing among users and failure handling.

As Cloud instances can migrate and the topology of the network is usually unknown, it is difficult to estimate communication cost and to tweak task placement accordingly. A usual solution is to deal with data locality and to migrate the environment.

Another argument against the use of task clustering algorithms is that it requires knowledge of all the jobs. In a dynamic context, the whole schedule would have to be recomputed for every new job submitted to the platform, leading to poor platform scalability. On the other end of the spectrum, batch scheduling algorithms are scalable in the context of many users submitting many independent jobs in real time.

Still, many Cloud solutions rely on Data as a Service (DaaS) tools for communication. Among such tools are Amazon S3, Dropbox, NFS and others. On an otherwise distributed infrastructure, DaaS are centralized elements that somehow constrain the level of parallelism. In fact, the capacity of a node to send and receive files from the rest of the network is limited by the bandwidth available between this node and the DaaS. Even with a distributed structured DaaS, availability of a specific piece of data to all nodes is bounded by the capacity of the nodes to access the data on different instances of the DaaS and consequently by the DaaS inner synchronization mechanisms. The whole DaaS can therefore be seen as a single entity potentially distributed among multiple machines.

To build an autonomous workflow manager for the Cloud, we focused on task packing. Our idea was to analyze each job workflow and to determine tasks packing for optimal placement without any prior knowledge of the platform current deployment. The resulting information could be useful to a batch scheduler downstream. Furthermore, if this workflow analysis could be done independently of the status of the platform, then it would not be necessary to recompute it with each subsequent change in the platform deployment, therefore ensuring good scalability.

In this paper, we focus on static clustering algorithms for workflows with data dependencies. We start by building workflow and platform models (Section III). Later, we show how legacy algorithms behave on cloud platform (Section IV), and propose an evolution based on a Cloud-inspired network topology (Section V). We then see how this new clustering method can help to pack tasks for efficient workflow deployment in the Cloud (Section VI).

II. Related work

Since the dawn of parallel and distributed systems, the scheduling issue has been considered in many context and with many different objectives, each relevant to some platform specificity. In this section we will give an overview of the existing approaches.

A. *Clustering algorithms*

The goal of clustering algorithms is to pack tasks into clusters prior to any execution. Given a global vision of the platform and the tasks to execute, a clustering is a mapping of tasks to the nodes of the platform, with all tasks being executed on the same node constituting a cluster. With this approach, expensive computation is needed to achieve a very efficient clustering, yet any change on the platform or the tasks to execute would lead to the whole clustering being recomputed.

In this category we find algorithm dealing with both homogeneous platforms [2]–[5] and heterogeneous platforms [6].

B. *Batch scheduling for Cloud*

The issue of the Cloud batch is the scheduling of many independent tasks and services, submitted dynamically, on heterogeneous platforms. One of the required key features is to have the addition of new tasks to the schedule be a very simple operation, thus ensuring the scalability of the system. Other elements, such as tasks priorities or energy consumption, can be used to make the schedule fit ones' objectives. On Cloud platforms, batch schedulers also adjust the platform size in order to meet all the deadlines while limiting the deployment cost [7].

While some tools handle graph deployment, the few that try to optimize placement based on data dependencies have poor network modeling [8].

C. *Workflow deployment on Cloud*

It is still an open problem to efficiently consider the constraints imposed by the real-time submission of workflows while handling the dynamism of Cloud platforms. All previous contributions focused on specific issues while leaving other aspects unconsidered.

A survey showed that most contribution did not consider the impact of data transfers [9]. Moreover, when this issue was considered, it was on simplified DAGs which failed to accurately model the whole extent of real applications.

We have identified two recent contributions which depict the current state of the art.

Mao *et al.* [10] dealt with a single workflow in which the tasks have different resources requirements.

Their algorithm not only packs tasks but also determines which type of nodes to deploy. However, this result does not account for multiple workflows. It also does not handle contexts in which communications go through a DaaS. We might be interested in investigating the underlying communication model in respect of the targeted platforms' behavior.

Malawski *et al.* [11] handled multiple dynamically submitted workflows composed of single-threaded tasks. In this work, the Cloud platform used for execution is dynamic but has an upper bound. Therefore, some workflows – with a lesser priority – can be dropped if the platform cannot be extended as required. Similarly to Mao *et al.*, the communication model used in this paper does not match our observations of DaaS-based platforms.

III. Workflow and platform modeling for communication-aware scheduling

Scheduling algorithms rely on specific definitions of resources and of the way tasks are executed on these resources. Before discussing the scheduling strategies themselves, we have to define the different objects we will be dealing with.

A. Generic network topology for DaaS-based Cloud platforms

It is commonly accepted that characterizing platform topology is a real issue on the Cloud. VM can migrate and resources such as routers can be shared with other users. Still, in an effort to build a scheduling policy both task and data transfer-oriented, we need to understand and predict the behavior of data transfers on this platform. As discussed in the introduction, communicating through a DaaS induces a centralization of the communications.

Our analysis is that this centralization will lead to contention at the links between a node and the DaaS. While the DaaS is a critical element, commonly located near the center of the network, the computing

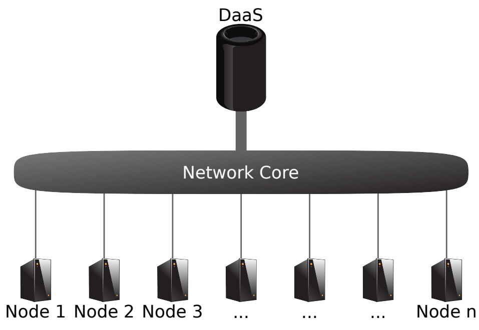

Fig. 1. A generic model of DaaS-based network topology.

stations can potentially be very far from it. This led us to a model where a node ability to place and retrieve data from the DaaS is constrained by the bandwidth between the node itself and the code of the network. This topology is illustrated in Fig. 1.

This model is generic enough for it to be independent of the actual Cloud deployment and of VM migrations, as long as the nodes bandwidth is not overestimated.

B. Data-centric representation of workflows

Workflows are usually represented as weighted graphs of tasks, with the weights on the nodes representing the computational cost of the tasks (in flops – floating point operations –) and the weights on the edges representing the cost of the communications (in bits of data transferred). However, as we moved to DaaS-based Cloud platforms, our representation of workflows needs to evolve accordingly.

While previous representations focused on the amount of data to be transferred between tasks, a more relevant approach would be to focus on data objects.

If we consider a fork-join distribution pattern, there is a major difference between sending n different pieces of data to n different agents and sending a single piece of data to the same n different agents. While in the second case we have a single upload from the initial task to the DaaS and n parallel downloads from the DaaS, in the first case we have a single task

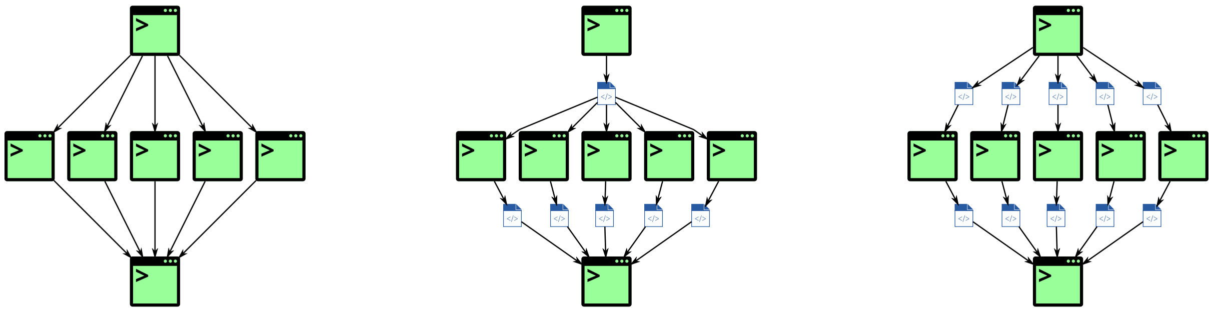
(a) Legacy representation (b) Our data-centric representation (single data upload) (c) Our data-centric representation (multiple data upload)

Fig. 2. Representation of a fork-join DAG with $n = 5$ independent jobs.

(the initial task) uploading the n different pieces of data to the DaaS at the same time, which would cause contention between the initial task and the core of the network where the DaaS is located.

Therefore, the first contribution of this paper is an extended "data-centric" representation of workflows which includes details about the different pieces of data produced and consumed by the tasks. Our representation (Table I) is an acyclic-oriented bipartite graph, with nodes from one side representing weighted tasks and nodes from the other representing pieces of data weighted by their size. Edges have no weight and only represent dependencies between pieces of data and their producers/consumers. Fig. 2 shows how two very different communication patterns have the same legacy representation. According to our network model, we expect bouts of network congestion when a single task uploads or downloads multiple files. Thus, we can expect some contention for the final task downloads in both cases (Fig. 2b and Fig. 2c). Yet, only the second case (Fig. 2c) should undergo upload contention for the initial task.

IV. Legacy communication-aware scheduling algorithms behavior on current platforms

Historical static scheduling algorithms such as DCP [5] were designed to take into account the impact of communication latency on workflows deployment. In this section we will see how they perform on DaaS-based platforms and analyze why they do not stand up to the task.

TABLE I
Workflow and Clustering notations.

Notation	Space	Description
$\mathcal{T}$		Set of all tasks
$\omega(t)$	$\mathcal{T} \mapsto \mathbb{R}$	Computational cost of t
$\mathcal{D}$		Set of all pieces of data
$d.src$	$\mathcal{T}$	Producer of data d
$d.dst$	$\mathcal{P}(\mathcal{T})$	Consumers of data d
$\omega(d)$	$\mathcal{D} \mapsto \mathbb{R}$	Communication cost of d
$\mathcal{C}$	$\mathcal{T} \mapsto$ VMs	Clustering

A. Contention on DaaS-based Cloud platforms

A common distribution pattern in workflows is the fork-join mechanism. In such context, an initial task is to send pieces of data to n independent tasks. Afterwards, those n tasks are sending back the results to a final task. We assume here that the n pieces sent by the initial tasks are different pieces of data of the same size.

When scheduling such a DAG using DCP, the algorithm takes action to achieve a high level of parallelism and thus places most tasks on distinct nodes. However, when executing the resulting task placement on a Cloud infrastructure, we see that communication

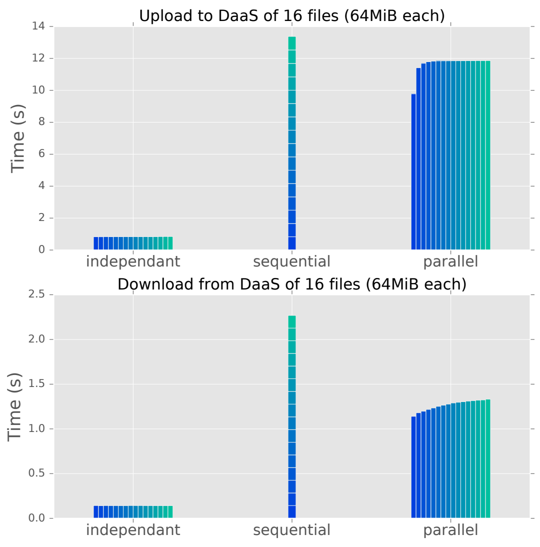

Fig. 3. Transfer times of 16 files from and to a DaaS: (left to right) predicted by DCP, experimental for sequential transfers, experimental for parallel transfers. Experiments were carried out using Grid'5000 testbed. Used nodes were on the Sagittaire cluster and the DaaS was a 10G chunk reserved on storage5k.

between the first task and the DaaS (upload of n files to the DaaS) and between the DaaS and the final task (download of n files from the DaaS) drastically suffers from contention. Fig. 3 shows that while DCP plans for all transfers to be simultaneous, the network congestion drastically reduces the performances of both upload to the DaaS and download from the DaaS. This results in transfer times about n times slower then predicted by DCP.

This gap between the communication model underlying DCP and the reality of current distributed platforms, in addition to the difficulty of handling the dynamic submission of workflows by users, explains why such scheduling algorithms are not efficient for tasks placement on Cloud platforms.

While other static scheduling algorithms have improved on DCP in many aspects, communication modelization is not among them. Algorithms like HEFT or CPOP have the benefit of handling heterogeneous platforms. However their design is based on the assumption of a limited number of ressources and thus do not match the cloud paradigm where new large instances are always available. Focusing on DCP helps us study the communication aspect of things without getting in the trouble of dealing with specificities that would not fit the cloud paradigm.

Algorithm 1 DCP static scheduling algorithm

```
$\mathcal{C} \leftarrow$ empty clustering    ▷ (one node per task)
compute $BL$ and $TL$ for each task using $\mathcal{C}$
**while** $\exists$ unmarked dependency between tasks **do**
    $(u, v) \leftarrow$ edge with the largest path length (most critical). Resolve ties by edge size (select largest).
    $\mathcal{C}' \leftarrow \mathcal{C}.mergeClusters(u, v)$
    compute $BL'$ and $TL'$ for each task using $\mathcal{C}'$
    **if** $DCPL(BL', TL') \leq DCPL(BL, TL)$ **then**
        $(\mathcal{C}, TL, BL) \leftarrow (\mathcal{C}', TL', BL')$
    **end if**
    mark $(u, v)$
**end while**
**return** $\mathcal{C}$
```

B. Dissecting the network model underlying the critical path computation

DCP relies on the computation of the critical path and on the zeroing of critical dependencies (see Algorithm 1). This critical path is determined through the computation of the *Bottom Level* (BL) and *Top Level* (TL) values for each task. Those values are used to highlight the constraints from the DAG input to the specified task and from this task to the DAG output. In fact, those BL and TL computations account for a worst-case scenario with respect to communication latency. This implies that the computation of those values is based on an implicit communication model.

$$c(u, v) = \begin{cases} 0 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ \omega(u \to v) & \text{otherwise} \end{cases} \tag{1a}$$

$$TL(v) = \begin{cases} 0 \text{ if } v \text{ has no predecessor} \\ \max_{u \in pred(v)}(TL(u) + \omega(u) + c(u, v), \\ \qquad avail_{TL}(\mathcal{C}, v)) \end{cases} \tag{1b}$$

$$BL(u) = \begin{cases} \omega(u) \text{ if } u \text{ has no successor} \\ \omega(u) + \max_{v \in succ(u)}(c(u, v) + BL(v), \\ \qquad avail_{BL}(\mathcal{C}, u)) \end{cases} \tag{1c}$$

The critical path computation used in DCP static scheduling algorithm is as shown in Eq. 1 (with u and v tasks in a DAG of tasks – workflow –, ω a weighting function on the tasks and data and $\mathcal{C}$ a linear clustering of the DAG)

From those values we can compute other metrics:

- Dynamic Critical Path Length (Makespan): $$DCPL = \max_{t \in \mathcal{T}} (TL(t) + \omega(t)) = \max_{t \in \mathcal{T}} (BL(t))$$
- Absolute Earliest Start Time: $$AEST(t) = TL(t)$$
- Absolute Latest Start Time: $$ALST(t) = DCPL - BL(t)$$
- Path Length (equal to $DCPL$ for critical tasks): $$PL(t) = TL(t) + BL(t)$$

We see that these formulas do not consider the possible impact simultaneous transfers could have on one another. In fact they disregard any form of contention. At first it looks like it considers a complete clique network where any pair of nodes can exchange data without being affected by the rest of the network, but it is in fact even stronger than that as any number of transfers between the same two nodes can take place at the same time without them having to share the bandwidth.

While such a topology could have made sense when dealing with small clusters, the gap with new distributed platforms is tremendous and explains the incapacity of DCP to efficiently predict communications. This leads to poor performances of the resulting task placements.

V. DCP EVOLUTION FOR DAAS-BASED CLOUD INFRASTRUCTURES

In the previous section, we discussed the behavior of DCP and its inadequacy to efficiently schedule workflows on DaaS-based Cloud platforms. In this section we will see how the models discussed in section III can be used to improve DCP ability to schedule workflows on modern platforms.

A. A communication model for Cloud infrastructures

Using the unmodified structure of the DCP algorithm (see Eq. 1), our objective is to use the platform and workflow models developed in Section III to modify the way communications affect workflow deployment.

In DCP equations (see Eq. 1), communications are modeled by the $c(u, v)$ formula. This is to be modified in order to match our communication model.

$$c(u,v) = 0, \qquad \text{if } \mathcal{C}(u) = \mathcal{C}(v)$$

$$c(u,v) = \sum_{\substack{d \in data \\ u \in d.src \\ v \notin d.dst \\ islocal_{\mathcal{C}}(d)=0}} \omega(d) \tag{2a}$$

$$+ \sum_{\substack{d \in data \\ u \in d.src \\ v \in d.dst}} \omega(d) + \max_{\substack{d \in data \\ u \in d.src \\ v \in d.dst}} \omega(d) \tag{2b}$$

$$+ \sum_{\substack{d \in data \\ u \notin d.src \\ v \in d.dst \\ islocal_{\mathcal{C}}(d,v)=0}} \omega(d) \tag{2c}$$

The modification described in Eq. 2 involves the computation of the worst case latency between tasks depending on their placement. If tasks u and v are placed on the same node, the communication cost between them is null (Eq. 2a). On the other hand, if u and v are placed on different nodes, we have to consider the upload time of all data produced by u and the download time of all data required by v. The worst case being when the tasks produced by u and required by v are the last to be uploaded by u and the first to be downloaded by v (Fig. 4).

In Eq. 2, the first sum (Eq. 2a) corresponds to the upload by task u of all data not required by v. The second line (Eq. 2b) corresponds to the interlaced upload by task u and download by task v of all data produced by u and consumed by v. Finally, the last sum (Eq. 2c) corresponds to the download by task v of all required data produced by tasks other than u. Those are also visible in Fig. 4.

In the third sum of Eq. 2 (Eq. 2c) it is not necessary to consider the download of data produced on the

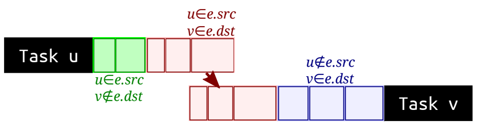

Fig. 4. Preview of the communications between two tasks for a data-based workflow on a DaaS-based platform (see Eq. 2).

same node. Similarly, in the first sum (Eq. 2a) we do not consider the upload of data which are consumed locally (producer and consumers all on the same node). The $islocal$ predicate is computed as following.

$$islocal_{\mathcal{C}}(d,u) = \begin{cases} 0, & \text{if } \mathcal{C}(d.src) = \mathcal{C}(u) \\ 1, & \text{otherwise} \end{cases} \tag{3a}$$

$$islocal_{\mathcal{C}}(d) = \prod_{v \in d.dst} islocal_{\mathcal{C}}(d,v) \tag{3b}$$

This evaluation of the communication-induced dependencies between two tasks corresponds to a worst-case scenario. It is most likely that a specific ordering of the communications could give us better results but, as always in static scheduling, we put ourselves in the worst-case scenario.

B. *Modified top- and bottom-level computations*

While DCP takes computation resources into account, we also need to consider node-based networking resources. If two tasks, with short computation time but with large amount of data to upload, are placed on the same node, the second task might be over before the data produced by the first task have been sent to the DaaS. The second task would therefore try to send data using an already busy uplink. Keeping track of the uplink and downlink availability is paramount when scheduling multiple tasks on the same node. The modified formulas for critical path computation, accounting for each node network availability, are described in Eq. 4 and graphically shown in Fig. 5.

Similarly to the way DCP deals with node availability, CPU (equivalent of node availability), uplink and downlink availability are updated during the linear scheduling of tasks. Each time a task is placed, the availability values of the concerned node, which are used to determine deployment timings, are updated in anticipation of the next task to be placed on this node. Initially, all availability values (which can be seen as time constraints on the task deployment) are initialized to 0.

$$c_{up}(u) = \sum_{\substack{d \in data \\ u \in d.src \\ islocal_{\mathcal{C}}(d)=0}} \omega(d) \tag{4a}$$

$$c_{down}(v) = \sum_{\substack{d \in data \\ v \in d.dst \\ islocal_{\mathcal{C}}(d,v)=0}} \omega(d) \tag{4b}$$

$$\begin{aligned} c_{total}(u,v) = & \sum_{\substack{d \in data \\ u \in d.src \\ v \notin d.dst \\ islocal_{\mathcal{C}}(d)=0}} \omega(d) + \sum_{\substack{d \in data \\ u \in d.src \\ v \in d.dst}} \omega(d) \\ & + \max_{\substack{d \in data \\ u \in d.src \\ v \in d.dst}} \omega(d) + \sum_{\substack{d \in data \\ u \notin d.src \\ v \in d.dst \\ islocal_{\mathcal{C}}(d,v)=0}} \omega(d) \end{aligned} \tag{4c}$$

$$\begin{aligned} TL(v) = \max_{u \in pred(v)}(& TL(u) + \omega(u) + c_{total}(u,v), \\ & avail^{up}_{TL}(\mathcal{C},u) + c_{total}(u,v), \\ & avail^{down}_{TL}(\mathcal{C},v) + c_{down}(v), \\ & avail^{cpu}_{TL}(\mathcal{C},v)) \end{aligned} \tag{4d}$$

$$\begin{aligned} BL(u) = \max_{v \in succ(u)}(& c_{total}(u,v) + BL(v), \\ & c_{total}(u,v) + avail^{down}_{BL}(\mathcal{C},v), \\ & c_{up}(u) + avail^{up}_{BL}(\mathcal{C},u), \\ & avail^{cpu}_{BL}(\mathcal{C},u)) + \omega(u) \end{aligned} \tag{4e}$$

As previously mentioned, we retained the structure of the DCP algorithm (Algorithm 1), to which we added our tailor-made formulas to compute the critical path. This gave us a generic task placement scheme which can deal with any DAG and which takes potential network congestion into account.

VI. Results and discussions

In the previous sections, we described a generic model for DaaS-based platforms as well as a variant of DCP that fits this model. In order to validate the relevance of this clustering algorithm to deploy DAGs on the Cloud, we are now going to compare it against a fully distributed scheme (all tasks are deployed on their own node, with no clustering) as well as against DCP.

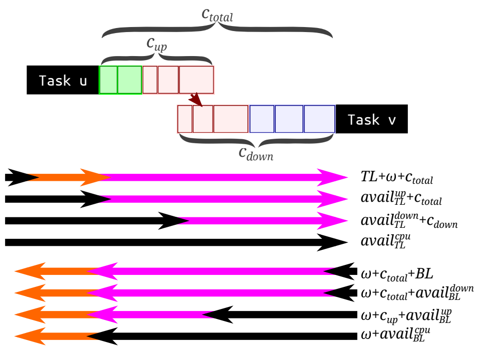
c_{total}
c_{up}
Task u
Task v
c_{down}
$TL+\omega+c_{total}$
$avail_{TL}^{up}+c_{total}$
$avail_{TL}^{down}+c_{down}$
$avail_{TL}^{cpu}$
$\omega+c_{total}+BL$
$\omega+c_{total}+avail_{BL}^{down}$
$\omega+c_{up}+avail_{BL}^{up}$
$\omega+avail_{BL}^{cpu}$

Fig. 5. Preview of the critical path computation taking node constraints into account in DaaS-based platforms (see Eq. 4).

This validation will also take into consideration our DAG description model. We will consider two fork-join cases which have the same description according to the legacy representation (see Fig. 2) but for which the new representation can help make more adequate decisions. True values were obtained on a simulated Cloud platform using SimGrid which has been shown to efficiently model concurrency and link sharing in large-scale networks [12].

A. Comparison of clustering heuristics

Fig. 6 shows both predicted and experimental Gantt charts obtained using different placement policies on different platforms. DCP leads to a very high level of communication parallelism to efficiently use many nodes and achieve a low makespan. However, using such a clustering on a simulated Cloud platform showed that congestion limits the actual data parallelism, leading to a much longer makespan.

We also see that results obtained using our model are very close to those simulated by SimGrid. This shows that our model makes clustering decisions based on a realistic approximation, leading to good results. The lower level of parallelism given by our algorithm not only reduces communication-induced latency but also limits the number of nodes to deploy. Further results also show that the different data distribution patterns in single-data and multiple-data fork-join DAGs lead to relevant clustering decisions.

B. Economical outcomes

While all deployments of a DAG, regardless of clustering, correspond to the same tasks being executed and therefore to the same amount of core-hours used, changing the clustering can affect the deployment cost. It is important to note that the main objective of static clustering algorithms such as DCP is to reduce the global makespan, assuming unlimited resources. In a Cloud context, this translates to an assignment of tasks to nodes, with a potentially very large number of nodes deployed. In a context where nodes are billed proportionally to the number of core-hours used, we could hope that, as the same amount of computation is achieved, the cost would roughly be the same, with any difference being caused by constraints on the reservation time (e.g. at least one hour).

However, these assumptions do not take into account the time during which a node is retrieving data prior to running a task or sending data after having run a task. During this time, we have to pay for the node even though we do not use its computing potential. Avoiding network congestion and ensuring smooth data transfers is a way of limiting this waste of CPU time. Table II shows the number of nodes used, the makespan, and the total deployment cost for our two models of fork-join DAGs using different scheduling algorithms. It is clearly visible that, in addition to reducing the makespan of the DAG, using the right clustering algorithm can help reduce deployment costs.

C. Computation to communication ratio

For our experiments, we used synthetic fork-join workflows where the computation time of each task was equivalent to the transfer time of one piece of data from/to the DaaS using the full bandwidth of the node. This ratio between computation and transfer

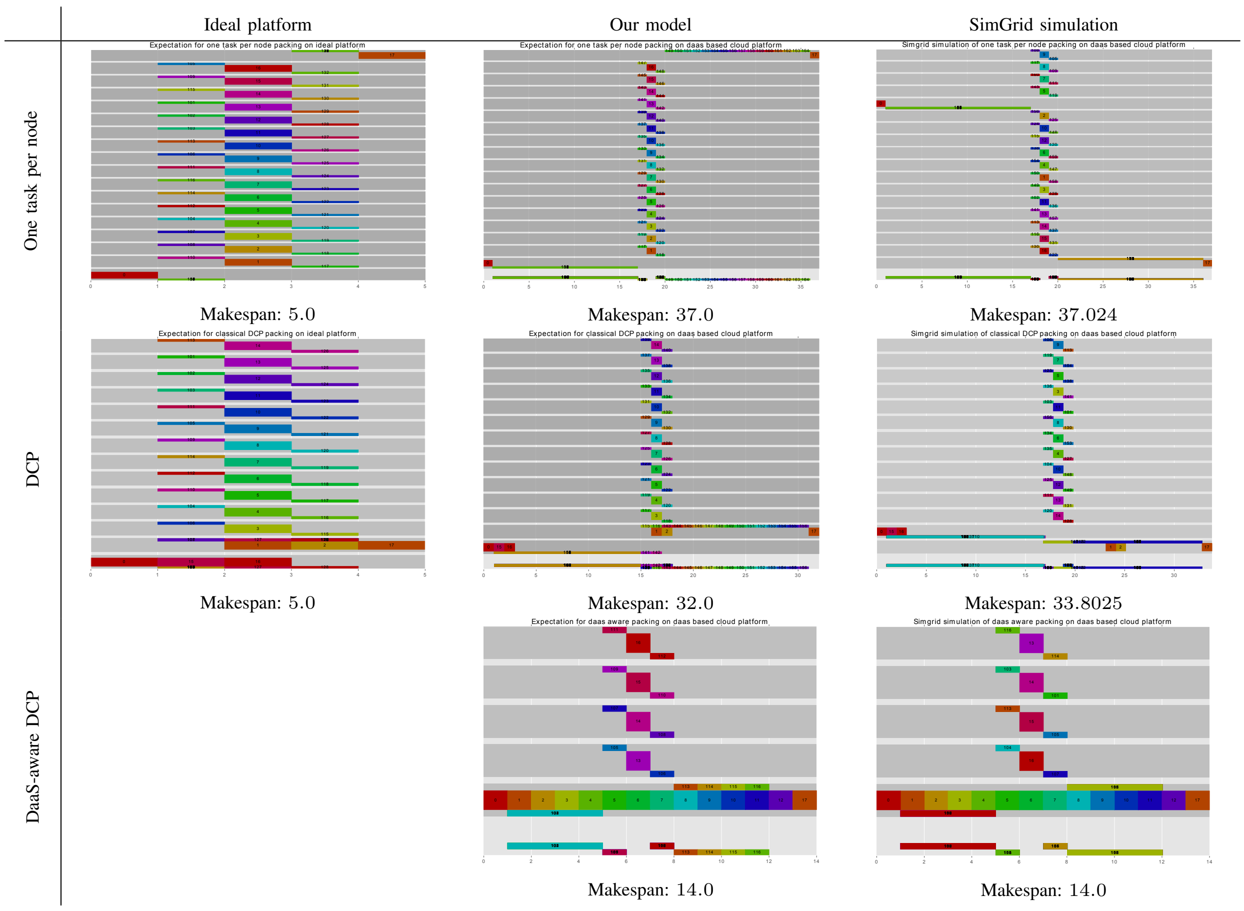

Fig. 6. Comparison of the different clustering policies (Gantt charts and their associated makespan) for multi-data fork-join DAG ($n = 16$). For each sub-figure, the X axis represents time (arbitrary units). Grey rows represent cores. Boxes in those rows are, from top to bottom, downlink utilization, tasks execution and uplink utilization.

times highlights congestion issues. In this case, single-node clustering achieves good results, as parallelism rapidly causes network congestion. With other ratios, we observe similar behaviors, with the DaaS-aware DCP being the fastest of all, only beaten on the cost aspect by the very slow single-node approach.

D. *Future work: cluster consolidation*

While the absence of dependencies between some tasks prevents both DCP and the DaaS-aware DCP from merging them, doing so could have a positive effect on the cost while maintaining the lowest possible makespan. This step, called clustering consolidation [10] is out of the scope of this paper. However, further work should explore how to adapt existing algorithms to our model.

VII. Conclusion

In this paper, we showed that the network topology is a key factor in predicting communication patterns and should therefore be considered by clustering algorithms. By designing a generic network model, we managed to improve the results of static scheduling in the context of DaaS-based Cloud platforms. In fact, the resulting clusters are both more efficient in terms of makespan (primary objective) and in terms of deployment cost compared to previous non-network-aware clustering algorithms.

We expect to use those results as the first component of an autonomous workflow manager. The next step is to integrate the computed task packing into a Cloud batch scheduler. With both components, we plan to

TABLE II

COST AND MAKESPAN DETAILS OF DIFFERENT CLUSTERING POLICIES FOR SINGLE-DATA AND MULTI-DATA FORK-JOIN DAG.

DAG	**Algorithm**	**#Nodes**	**Makespan** (t)	**Cost** (core×t)
Single Data Fork-join, as showcased in Fig. 2b, with $n = 16$	One task per node	18	22.024	67.204
	Single node	1	18.000	18.000
	DCP	14	18.024	56.168
	DaaS-aware DCP	2	13.012	20.012
Multiple Data Fork-join, as showcased in Fig. 2c, with $n = 16$	One task per node	18	37.024	82.204
	Single node	1	18.000	18.000
	DCP	14	33.803	70.156
	DaaS-aware DCP	5	14.000	26.048

contribute to or to build a tool that would efficiently deploy workflows on Cloud platforms and where the generic platform model could easily be reconfigured to adapt to changes in the platform paradigm.

ACKNOWLEDGMENTS

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). Platform simulations were carried out using SimGrid [12].

REFERENCES

[1] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[2] M.-Y. Wu and D. Gajski, "A programming aid for message-passing systems," in *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing*. Society for Industrial and Applied Mathematics, 1989, pp. 328–332.

[3] T. Yang and A. Gerasoulis, "List scheduling with and without communication delays," *Parallel Comput.*, vol. 19, pp. 1321–1344, 1993.

[4] ——, "Dsc: Scheduling parallel tasks on an unbounded number of processors," Tech. Rep., 1994.

[5] Y.-K. Kwok and I. Ahmad, "A static scheduling algorithm using dynamic critical path for assigning parallel algorithms onto multiprocessors," in *Proceedings of the 1994 International Conference on Parallel Processing - Volume 02*, ser. ICPP '94. IEEE Computer Society, 1994, pp. 155–159.

[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 260–274, 2002.

[7] Y. Yang, K. Liu, J. Chen, X. Liu, D. Yuan, and H. Jin, "An algorithm in swindew-c for scheduling transaction-intensive cost-constrained cloud workflows," in *Proc. of 4th IEEE International Conference on e-Science*, 2008, pp. 374–375.

[8] P. Varalakshmi, A. Ramaswamy, A. Balasubramanian, and P. Vijaykumar, *An Optimal Workflow Based Scheduling and Resource Allocation in Cloud.* Springer Berlin Heidelberg, 2011, pp. 411–420.

[9] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 237–256, 2009.

[10] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 2013, pp. 67–78.

[11] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12, vol. 22. IEEE Computer Society Press, 2012, pp. 1—-11.

[12] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.