Decentralised Multiple Workflow Scheduling via a Chemically-coordinated Shared Space

Héctor Fernandez, Marko Obrovac, Cédric Tedeschi

▶ To cite this version:

Héctor Fernandez, Marko Obrovac, Cédric Tedeschi. Decentralised Multiple Workflow Scheduling via a Chemically-coordinated Shared Space. [Research Report] RR-7925, INRIA. 2012, pp.14. <hal-00690357>

HAL Id: hal-00690357
https://hal.inria.fr/hal-00690357

Submitted on 23 Apr 2012

Decentralised Multiple Workflow Scheduling via a Chemically-coordinated Shared Space

Héctor Fernández, Marko Obrovac, Cédric Tedeschi

RESEARCH REPORT
N° 7925
April 2012
Project-Teams Myriads

ISSN 0249-6399 ISRN INRIA/RR--7925--FR+ENG

Decentralised Multiple Workflow Scheduling via a Chemically-coordinated Shared Space

Héctor Fernández, Marko Obrovac, Cédric Tedeschi

Project-Teams Myriads

Research Report n° 7925 — April 2012 — 14 pages

Abstract:
Community clouds have arisen as a promising infrastructure to face the ever-growing demand for computational power needed by applications. In such a community, different cloud providers federate their resources, allowing users to run applications across multiple sites through a global workflow scheduling system. However, current workflow scheduling approaches exhibit limited scalability, as they rely mostly on centralised scheduling mechanisms, and thus cannot fully leverage possible cooperation among cloud providers.

To overcome these limitations, we propose a fully decentralised workflow scheduling system which uses a chemistry-inspired model to coordinate decision making in a community cloud platform. Our system implements a distributed shared space inside which resources are associated to tasks, enabling a decentralised decomposition of workflows into tasks. They are then independently mapped onto resources. In particular, in spite of decentralisation, the system is able to select the momentarily most appropriate resource for a given task, independently of the cloud provider the resource is located on. The framework's expectations in terms of scalability are captured through simulation experiments.

Key-words: Community Clouds, Workflow Scheduling, P2P, Unconventional Programming Models, Chemical Programming Model

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Ordonnancement décentralisé de workflows multiples à travers un espace partagé chimiquement coordoné

Résumé :

Les nuages de cacul communautaires (*community clouds*) apparaissent aujourd'hui une infrastructure prometteuse pour faire face à la demande en puissance de calcul toujours grandissante des applications. Dans une telle plate-forme, différents fournisseurs de *clouds* agrègent leurs ressources, permettant ainsi aux utilisateurs d'exécuter leurs applications sur une plate-forme étendue de façon transparente, à travers un système d'ordonnancement global. Nous considérons des applications à bases de graphes de tâches (ou *workflows*).

Cependant, les systèmes d'ordonnancement de *workflows* actuels ont un passage à l'échelle limité, car ils s'appuient sur des ordonnanceurs centralisés, et ne peuvent pas correctement exploiter cette plate-forme collaborative.

Pour palier ces inconvénients, nous proposons un système d'ordonnancement totalement décentralisé, qui s'appuie sur une métaphore chimique pour la prise de décision coordonnée. Ce système met en œuvre un espace virtuellement partagé dans lequel les tâches des workflows sont associées aux ressources de la plate-forme de façon décentralisée. Malgré cette décentralisation, la plate-forme proposée est capable de sélectionner globalement la meilleure ressource courante pour une tâche qui est prête à être exécutée. Le passage à l'échelle du système est capturé à travers des expériences de simulation.

Mots-clés : Community Clouds, Ordonnancement de workflows, P2P, Modèle de programmation non conventionnel, Modèle de programmation chimique

Contents

1 Introduction

Applications tend more and more to take the shape of compositions of independent, loosely-coupled services bounded at run time. This is particularly true for scientific applications that are now commonly built as *workflows* of services, *i.e.* temporal compositions thereof. The success of the *myExperiment* platform[1] for service and workflow sharing is a clear sign of the shift in application conception.

Independently, over the last few years, the concept of cloud computing has emerged. Its elasticity property makes the cloud an appealing tool for large-scale scientific computations. Clouds have recently seen their usage extended and consolidated as computing environments for scientific research. For instance, the Magellan project [1] aims at providing a cloud-based platform for scientists. However, the computing power demanded by scientific applications can reach such proportions that some resource providers alone are not able to face them. This led to collaborations amongst different cloud providers, giving birth to an infrastructure called a *community cloud*. It allows different cloud providers to share their resources, making it possible for these workflows to be deployed and executed. The idea of sharing resources, however, carries with it new challenges such as the interoperability between different cloud providers, elasticity, security, as well as economical issues [2, 4].

In this paper, we focus on the crucial feature of scheduling, *i.e.*, the process deciding on which resource which task of the workflow has to be run. Two primary concerns in such platforms are *cooperation* (between different resource/cloud providers) and *decentralisation* (in regards to the envisioned scale of community clouds). Traditional centralised schedulers ought not to be put in use, as they would inevitably suffer from significant reliability and scalability limitations.

[1] http://www.myexperiment.org/

It is thus essential to promote decentralised and autonomic solutions which embrace all the participants in a community in order to select the appropriate resource for a task. However, while decentralisation offers a robust solution, it is not able to handle the cooperation between different resource providers. The platform should be enhanced with coordination mechanisms enabling efficient *task-to-resource* mapping.

Lately, *natural* metaphors, and in particular chemistry-inspired analogies, have been identified as a promising source of inspiration for developing new approaches for autonomic service coordination [17]. More precisely, we consider the *chemical programming paradigm* — a high-level execution model, where a computation is envisioned as a set of reactions taking place concurrently and autonomously between molecules of information moving and colliding non-deterministically. With such an execution model, the expression of coordination is done via a set of user-defined rules consuming molecules and producing new ones, freeing it of artificial sequentiality and structuring, making the paradigm a good candidate for specifying autonomic and decentralised coordination. Recently, it has been shown that this paradigm provides adequate abstractions to enact workflows [9] and execute them in a decentralised way [6].

Contribution. In this paper, we are proposing a fully decentralised workflow scheduling framework including two layers. The top layer is a chemically-coordinated shared space where workflows are decomposed into tasks, which are mapped to resources following simple chemical rules. The bottom layer implements this shared space in a fully decentralised way, based on a peer-to-peer overlay network allowing the efficient storage and retrieval of molecules. Altogether, the system proposed here, and evaluated through different simulation experiments, is a decentralised framework allowing for an efficient dynamic multiple workflow scheduling.

Outline. Section 2 introduces the chemical programming paradigm. Section 3 describes our decentralised workflow scheduling system and its chemistry-inspired coordination model. Section 4 evaluates the performance and network overhead of the framework. Section 5 presents the related works, and Section 6 draws some conclusions.

2 Chemical Programming Model

Natural analogies, and more specifically bio-chemical metaphors, have recently regained *momentum* in the construction of programming models coping with the requirements of the Internet of Services [17]. Initially proposed to adequately express highly parallel programs, the chemical programming paradigm exhibits properties required in emerging service platforms and naturally expresses autonomic coordination.

According to the chemical metaphor, molecules of data float in a chemical solution, and, on collision, react according to reaction rules (program) producing new molecules (resulting data). These reactions take place in an implicitly parallel, autonomous, and non-deterministic way until no more reactions are possible — a state referred to as *inertia*. The computation is carried out according to local conditions without any central coordination, ordering or serialisation. This programming style allows writing programs cleared of any artificial sequentiality and to concentrate on the functional aspects of the problem being solved. The simple presence of a molecule is enough to trigger a reaction requiring it.

HOCL (*Higher-Order Chemical Language*) [3] is a language following such a paradigm. In HOCL, the solution is a multiset containing molecules, and rules define reactions between molecules which rewrite it. The multiset is the solely data structure provided. For the sake of illustration, let us consider the following chemical program which extracts the maximum value from a set of integers:

$$\textbf{replace } x :: \mathit{int}, y :: \mathit{int} \textbf{ by } x \textbf{ if } x \geq y \textbf{ in } \ \langle 2, 4, 5, 7, 9 \rangle$$

The rule of the program specifies that any pair of integers inside the solution can react, consuming these two molecules and creating a new one with the highest value of the two, in case the condition holds. While the result of the computation is deterministic, the execution itself is not; HOCL simply ensures the mutual exclusion of reactions by the atomic capture of the reactants. In our example, a possible execution is the following:

$$\langle 2,4,5,7,9\rangle \rightarrow \langle 4,5,9\rangle \rightarrow \langle 5,9\rangle \rightarrow \langle 9\rangle$$

Because of the implicit parallelism of the execution model, reactions consuming 2 and 4, and 7 and 9 (and producing 4 and 9) were carried out simultaneously and independently from each other. Once there is only one number left in the multiset (here, 9) the rule cannot be applied any more, and inertia is reached. Note that, even though in imperative programming languages this symbolises the end of the execution, the chemical programming paradigm does not have that notion — inertia is simply a stable state. This means that in case new molecules (here, integers) are dynamically inserted into the multiset, an *imbalance* arises. It is going to be detected and a new execution cycle triggered. Enabling persistent coordination amongst scheduling entities requests for such a concept.

Going even further, HOCL treats rules as regular molecules. This characteristic opens doors to autonomic computing: by consuming and producing rules, a program is able to modify its execution flow on-the-spot. The newly established expressiveness has been at the origin of new approaches to build autonomic service platforms [6, 9, 10]. However, in these works, resources, on which the tasks of the workflows are run, are abstracted out, *i.e.* scheduling considerations were left aside.

Throughout the paper we are using the HOCL notation to express rules responsible for managing incoming workflows.

3 Decentralised Shared Space for Workflow Scheduling

We now present a fully decentralised just-in-time multiple workflow scheduler, of which the abstract organisation is shown on Figure 1: the scheduling process is shared by a set of *chemical engines* running on every resource machine, that constitute the *entry points* for the workflows which will be locally decomposed first into *levels* then into tasks for later execution. As illustrated by Figure 2, the proposed system is a two-layer architecture. It takes its roots in a generalised system for decentralised execution of chemical programs [11], but is adapted here for scheduling purposes. We now detail these two layers.

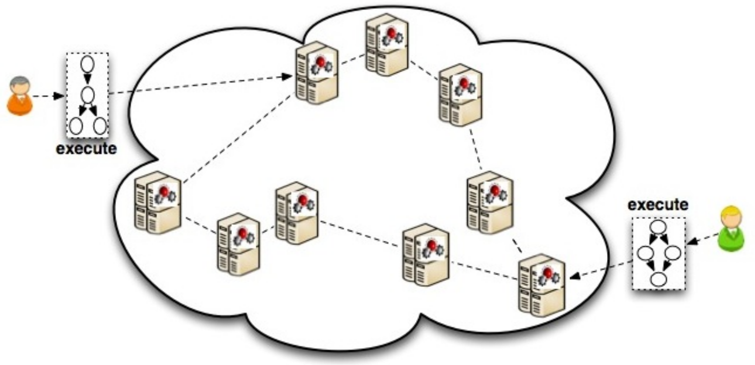

Figure 1: Proposed architecture overview.

Communication Layer. In order to abstract out the underlying network topology and to deal with the potential unlimited growth of resources, chemical engines are connected in a structured peer-to-peer overlay network [13, 14], illustrated in the lower part of Figure 2. Next, we assume that the chemical engines, referred to as *nodes* throughout the paper, communicate through a ring-shaped overlay network. However, indeed, a DHT with a different topology could be used.

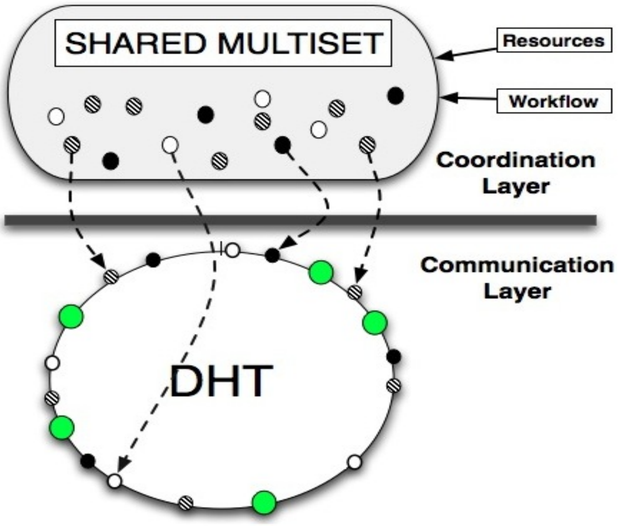

Figure 2: Two-layer architecture.

Coordination Layer. Due to the use of the DHT, chemical engines can share their local data — molecules — with other participants in a scalable fashion (DHTs provide a distribution and retrieval mechanism the complexity of which typically grows logarithmically with the number of nodes). This way, an actual *shared multiset* is created on top of the DHT, to which nodes expose their molecules — levels, tasks and resources, as shown in the upper side of Figure 2. Thus, nodes are able to retrieve and consume molecules they do not hold, giving birth to a decentralised *scheduling space.* Moreover, each chemical engine includes workflow-independent rules in charge of workflow decomposition and task scheduling, acting by consuming molecules within the shared multiset. These rules are named *generic scheduling rules* and are explained more in depth in Section 3.3.

In the remainder of the section we firstly describe the structure of the molecules of the scheduling space and their placement in the underlying DHT-layer. Then, we detail the decentralised scheduling process carried out as reactions take place in the upper layer.

3.1 Molecule Types

There are three types of molecules in our system: molecules representing workflow levels, task molecules and resource molecules. When a molecule is produced, it is assigned a unique identifier based on a cryptographic hash function (typically provided as part of the DHT, such as SHA1). The molecule is then placed in the shared multiset, *i.e.* in the DHT ring in the bottom layer, by routing it to the appropriate node based on its hash identifier. The molecules and their placement in the DHT are depicted in Figure 3.

3.1.1 Level Molecules.

Upon its entry in the system, a workflow is decomposed into levels by the entry node, producing *level molecules* (white dots in Figure 3). A level of a workflow comprises all of the tasks at the same distance from the exit task of a workflow's graph. Level molecules take the form $\textsc{Level} : idLevel : \langle task_1, \ldots, task_n \rangle$, where $idLevel$ identifies the level of the workflow, and $task_1, \ldots, task_n$ are the tasks located in it. Each level molecule is then sent to its appropriate destination node according to its hashed value.

3.1.2 Task Molecules.

Once it is the turn of a level to be processed, the node storing its molecule cuts it into a set of *task molecules* (black dots in Figure 3), one per task. A task molecule takes the form $\textsc{Task} : idTask : \langle command : res\_desc \rangle : \langle \textsc{Dest} : destTaskId, \ldots \rangle$, where $idTask$ is the task's identifier, $command$ denotes the actual service to invoke, res_desc is the description of the resource requirements needed to execute the task, and the $\langle \textsc{Dest} : destTaskId, \ldots \rangle$ sub-solution specifies to which tasks (given their task identifiers as $destTaskId$) the output of the task has to be sent. Upon a level's decomposition, task molecules are similarly stored in the DHT.

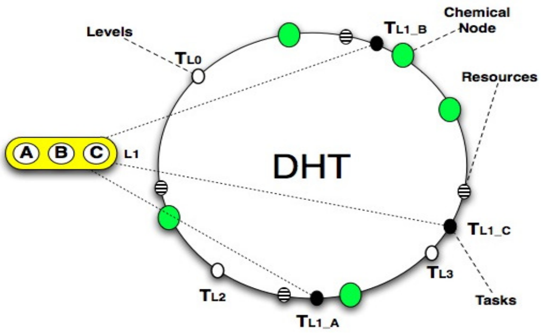

Figure 3: Scheduling molecules.

3.1.3 Resource Molecules.

Physical resources are represented by molecules of the form $\textsc{Res} : idRes : \langle feature_1, \ldots, feature_n \rangle$, where $idRes$ is the identifier of the resource and $feature_1, \ldots, feature_n$ are its current characteristics, such as the number of processors, the CPU load, or the memory usage. Since the features of a resource variate in time, every so often nodes *republish* resource molecules that replace old ones. Determining the right interval of republishing falls out of the scope of this paper. However, unlike level and task molecules, resource molecules are not uniformly hashed Instead, they are kept on the originating node (as suggested in Figure 3). Doing so keeps the network cost of republishing at zero. Furthermore, it allows the system to be up-to-date, since when a resource machine crashes the respective molecule consequently disappears from the shared multiset, preventing other participants to try to schedule a task on this resource.

3.2 Meta-Molecules and Resource Retrieval

In order to implement efficient scheduling policies, resources have to be ranked to efficiently select a resource satisfying a task's constraints. For this purpose, we introduce a second, order-preserving, DHT layer, physically matching the original one (node ids as well as the key space size are preserved), to store *meta-molecules* — *pointers* to resource molecules, in an order-preserving manner, which means that a meta-molecule's hash identifier is no more cryptographically hashed but based on its molecule's value. When a node (re)publishes its resource molecule, it creates a meta-molecule and stores it in the second DHT layer. As an illustration, consider two resource molecules $M_1 = \textsc{Res} : cpu(80\%)$ and $M_2 = \textsc{Res} : cpu(50\%)$. Supposing the ordering criterion for resource molecules is processor utilisation, M_1's meta-molecule's hash identifier would be greater than that of M_2's meta-molecule, since the resource represented by M_1 is more utilised than M_2's resource.

The second, order-preserving DHT layer is used when trying to map a task to a resource. When a node is searching for an appropriate resource to execute its task on, it simply consults the second DHT layer for resource meta-molecules. Moreover, the node is able to precisely locate specific matching resources in this layer, by issuing specialised queries such as $cpu < 80\%$. The support for such *range queries* has been extensively studied in literature, and typically require $O(\log^2(n))$ messages to complete, as discussed for instance in [5]. A meta-molecule *testifies* to the existence of a particular resource molecule; when a node obtains a meta-molecule, it is able to consequently obtain the original resource molecule the meta-molecule derives from.

3.3 Workflow Scheduling Process

Despite the wide variety of existing scheduling heuristics, none of them have been shown to deliver an efficient scheduling algorithm for both types of scientific workflows: data intensive and computation intensive. The selection of an appropriate workflow scheduling algorithm depends of several parameters estimated by users, such as communication costs and task completion times, making the schedulers prone to *judgement errors.* To this respect, the framework proposed in this paper aims at providing a decentralised and *just-in-time* task-to-resource mapping, on top of which any workflow scheduling heuristic can be implemented.

3.3.1 Inter-layer Execution Model.

The execution of rules is event-driven: a rule is triggered when a node receives a molecule or a workflow. Three entities can provoke a rule's execution: a workflow, a level molecule and a task molecule. Upon the receipt of a workflow or a level molecule, a node locally triggers rules to decompose the workflow into levels or levels into tasks, respectively. Afterwards, the levels and tasks generated are hashed and stored in the DHT.

On the other hand, when a node receives a task molecule, it has to find a resource to map it to. Thus, the node constructs a range query in which it lists the requirements a resource has to satisfy and then lets the DHT's range search mechanism (second layer) find it a match. If a matching resource meta-molecule has been found, the node proceeds to the next step: retrieving the resource molecule itself containing information for the actual execution of the task. Finally, the node sends its task to the resource node for execution.

3.3.2 Workflow Structure.

Workflows received by nodes are described using the *chemical workflow definition* (Algorithm 1). The general shape of this workflow representation is as follows: the main solution is composed of as many *task molecules* as there are tasks (or services) participating in the workflow. This chemical representation of workflows has been shown to be appropriate to express the decentralised execution of a wide variety of workflows patterns [6].

Algorithm 1 Chemical workflow representation.

```
$\langle$ // Multiset (Solution)
    $\textsc{Task} : 1 : \langle command_1 : res\_desc_1 \rangle : \langle \textsc{Dest} : 2, \textsc{Dest} : 3, \ldots \rangle$, // $\textsc{Task}_1$ definition
    $\textsc{Task} : 2 : \langle command_2 : res\_desc_2 \rangle : \langle \textsc{Dest} : 4, \ldots \rangle$,
    $\textsc{Task} : 3 : \langle command_3 : res\_desc_3 \rangle : \langle \textsc{Dest} : 4, \ldots \rangle$,
    $\textsc{Task} : 4 : \langle command_4 : res\_desc_4 \rangle : \langle \rangle$
$\rangle$
```

3.3.3 Workflow Scheduling Flow.

We now describe the workflow scheduling process, which is illustrated by Figure 4. Chemical rules used in the process are given in Algorithm 2.

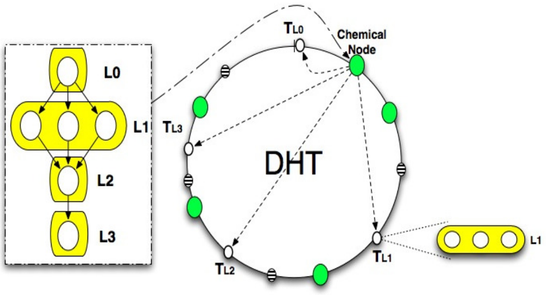

Figure 4: Workflow decomposition.

A workflow initially enters the platform by being sent to a given chemical node, its *entry point*, which receives the workflow and decomposes it in levels, producing level molecules. This step mimics the first phase of algorithms such LMT or HEFT [15]. Upon the receipt of a workflow, a node triggers the *workflowDecomp* rule which reorganises the tasks represented as sub-solutions of the workflow representation into levels. It consumes the molecules representing the different tasks in the workflow, and produces one molecule per level.

Algorithm 2 Generic rules.

```
                    — WORKFLOW DECOMPOSITION —
3.01  let workflowDecomp = replace ⟨ $\text{Task}_1$, $\text{Task}_2$, ..., $\text{Task}_n$ ⟩
              by Level:1:⟨ $\text{Task}_1$ ⟩, Level:2:⟨ $\text{Task}_2$ ⟩, Level:N:⟨ $\text{Task}_n$ ⟩
3.02

                    — LEVEL DECOMPOSITION —
3.03  let levelDecomp = replace-one Level:num:READY:⟨ $\text{Task}_1$, ... ,$\text{Task}_n$ ⟩
3.04                by $\text{Task}_1$, ... ,$\text{Task}_n$

                    — TASK TO RESOURCE MAPPING —
3.05  let mapTaskRes = replace $\text{Task}_i$, $\text{Res}_j$
3.06                by system.deploy( $\text{Task}_i$, $\text{Res}_j$ )
3.07                if ( $\text{Task}_i$.isCompatibleWith( $\text{Res}_j$ ) )
```

Tasks have to be scheduled level by level, calling for coordination between nodes involved in scheduling a given workflow, in order to allow the system to pass from scheduling tasks of one level to the next one, and so on until reaching the last level and delivering the final result to the requesting user. More precisely, the system has to assure tasks from level i are not being scheduled for execution before or during the scheduling of tasks from level $i-1$. To distinguish between levels which can be scheduled and those which have to wait on scheduling we use two types of molecules: $\text{Level} : num : READY$ and $\text{Level} : num$, respectively. The initial workflow decomposition through the activation of the rule *workflowDecomp* produces only $\text{Level} : num$ molecules to indicate that none of the levels can be scheduled at that time. However, as the scheduling goes on, these molecules will, one by one, turn into $\text{Level} : num : READY$ molecules, indicating that the tasks of the previous levels have been completed and that the tasks of the next level can be scheduled for execution.

To extract tasks from level molecules, a node uses the *levelDecomp* rule. This rule consumes a level molecule with its state set to $READY$, and produces as many task molecules as there are tasks in the given level (Algorithm 2, line 3.03). The task molecules are then hashed and stored in the DHT.

Upon receipt of a task molecule, a node uses the *mapTaskRes* rule to map the task to the best suitable resource it can find (Algorithm 2, line 3.05). Using the resources' requirements indicated in the task molecule, the second DHT layer is scanned by a range query reflecting the **if**-clause of the *mapTaskRes* rule. If a matching resource molecule is found, the rule produces a molecule that deploys the given task onto the resource found during molecule capture (denoted by the special *system.deploy()* molecule in Algorithm 2). Note that the resource molecule is consumed in the reaction, preventing other tasks to be scheduled before it has been republished. Once a node receives the notification from the system that its task has been completed, it notifies the node holding the level molecule for this task, which, in turn, keeps track of completed tasks. When all of its tasks have been completed, the node holding the (currently active) level molecule retrieves the inactive molecule of the next level. It then deletes it and creates a new, active level molecule and stores it with the same hash identifier. This act allows the next level to be decomposed and its tasks to be scheduled. This process carries on until the tasks of the last level have finished their execution. At the end, its node collects all of the results and transfers them to the entry node, which delivers them to the client which submitted the workflow for execution.

Multiple Workflow Scheduling Example. Let us consider two workflows, A and B, where task molecules of level *L1* (in both workflows) are awaiting scheduling, as illustrated on Figure 5. Task molecules A, B and C from *workflowA* and D and E from *workflowB* are ready to be executed. There are four available resources. This leads to a situation where not all of the tasks can be scheduled concurrently. Each of the five nodes holding task molecules tries to grab a matching resource molecule (due to the execution of the *mapTaskRes* rule). Supposing the nodes holding B, C, and D successfully grab their respective resource molecules, they execute a reaction following the *mapTaskRes* rule by deploying their tasks onto the matching resources. If the only available resource left does not meet the needs of neither A nor E, the nodes holding them wait for a small predefined amount of time and retry fetching a resource molecule.

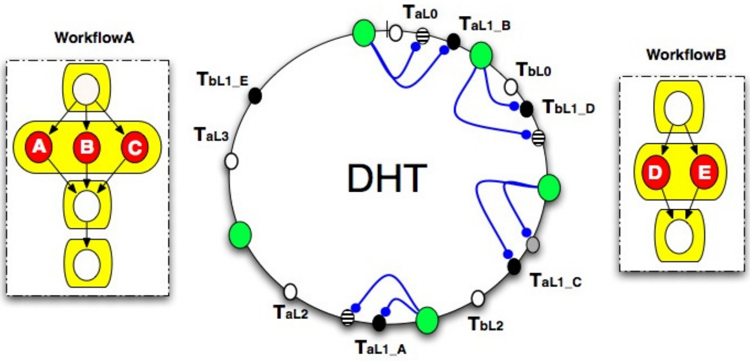

Figure 5: Multiple workflow mapping.

4 Evaluation

4.1 Simulation Set-up

To better capture the behaviour and expected performance of the proposed system, a Python-based simulator was built. It simulates a two-layered, DHT-structured network of nodes offering storage and computing power. The nodes also store the meta-information about available resources (the meta-molecules). Workflows sent to this network are processed and scheduled following the decentralised coordination model described in Section 3. The simulator operates in discrete time steps.

Goals and Assumptions. Our goal is to generally show the overhead, in terms of latency and network load, of the scheduling process itself, as we do not intend to provide a new scheduling algorithm, but a framework for decentralised coordination for large-scale scheduling. Accordingly, some assumptions were made. A workflow's depth (the number of levels) was randomly chosen between 3 and 10, with each level containing an arbitrarily assigned number of tasks (between 1 and 15). Furthermore, a task's duration was voluntarily kept low (between 1 and 10 time steps) as it facilitates the evaluation of the framework's overhead. Finally, the computing power of nodes and the capacity of links in the network were virtually unbounded. While such an assumption could sound unrealistic, our simulation did not intend to provide real accuracy in actual settings, as only a real-world deployment could do that. Our validation is oriented towards providing insight into the scalability of the framework.

4.2 Results

Execution Time. We first tried to capture the scalability in terms of time overhead, when both the number of nodes and workflows increase. Results are depicted on Figure 6. The first conclusion is that increasing the number of nodes has an impact on the time taken to schedule workflows. However, this overhead is limited, since the cost of the routing process grows logarithmically with the number of nodes (except for resource retrieval which requires $\log^2(n)$ messages to deal with range queries). Another conclusion that can be drawn by looking at all of the curves together is that when the number of workflows increases, the time to solve them does not increase much. This is a consequence of fully decentralising the scheduling process, which enables a high degree of parallelism. However, one can argue that, as Figure 6 suggests, having fewer nodes leads to better performance. This anomaly is inherent to the assumption that the network links and the computing power are unbounded. Hence, we need to have a look on the network overhead, which is depicted in Figures 7 and 8.

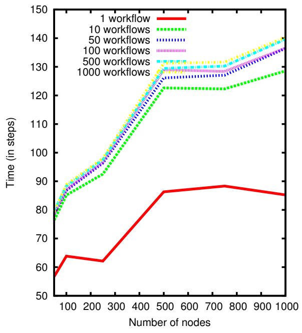

Figure 6: Execution time.

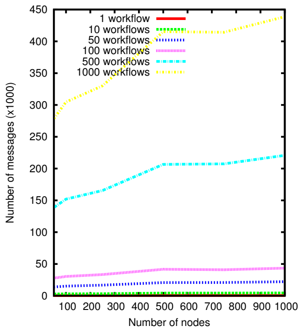

Figure 7: Network traffic.

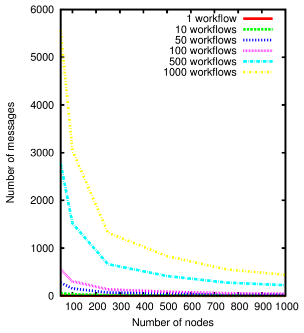

Figure 8: Traffic per node.

Network Overhead. The series of curves on Figure 7 shows the total number of messages generated in the same experiments as earlier. They first suggest that, due to the usage of logarithmic routing, increasing the number of nodes does not impact significantly the network. The curves also show that the number of messages increases with the number of workflows. In fact, the number of messages is directly proportional to the number of tasks to be scheduled, as the scheduling of a task relies on resource retrieval, which needs $O(log^2(n))$ messages to complete. However, this is partly inevitable as each task must be scheduled independently of others. Finally, Figure 8 depicts the evolution of the traffic load perceived by each node in the same conditions. While the traffic costs per node increase with the number of workflows, it is also drastically reduced when more nodes take part in the scheduling process. This behaviour is a highly desirable one, as we target large-scale platforms. This reinforces the scalability of the whole platform, since the system is able to spread the network load evenly as it increases in size.

5 Related Works

We here briefly review the most recent approaches dealing with decentralised scheduling intended for large-scale platforms.

Works such as [2] and [8] motivate the need of interlinking Grid systems through peering arrangements in order to enable resource sharing. Local schedulers are connected through *gateways* which are used to serve unsatisfied local requests. Ranjan *et al.* [12] designed a fully decentralised scheduler which uses a DHT split in regions, where requests for resources are posted.

Since each region is managed by one grid peer, the system is still susceptible to single-point-of-failure patterns. Finally, works presented in [7, 16] propose Gossip-based schemes to schedule computation-intensive jobs. Like in our system, there are no predefined schedulers — any entity can schedule a job.

In contrast to these approaches, our framework intends not only to decentralise the scheduling process, but also to maximise the efficiency of scheduling algorithms to be deployed, as the coordination layer, built upon a structured network globally ranking resources, enables a global knowledge of available resources in the platform, ensuring that each task will be run on an adequately and accurately chosen resource.

6 Conclusion

The evolution of clouds towards community clouds raises the need for large-scale, coordinated workflow mechanisms where the whole set of resources and jobs are associated in a transparent way. This calls for new mechanisms for large-scale coordination mechanisms. This paper proposes a fully decentralised coordination space relying on a chemical metaphor: workflows, jobs and resources are molecules to be consumed, *i.e.* matched. The underlying communication layer ensures a fully decentralised execution of this matching, by relying on a DHT. Moreover, the *DHT-driven* coordination enables a *just-in-time* scheduling technique capable of matching a task to its currently perfect resource candidate. Simulations were conducted, establishing further the feasibility and scalability of the approach.

A software prototype is being developed, interfacing the P2P-based layer with a chemical engine for coordination. It is planned to be deployed over actual platforms, and will constitute a platform for implementing different scheduling strategies in a fully decentralised fashion.

References

[1] The magellan research project. http://magellan.alcf.anl.gov/ (February 2012)

[2] Dias de Assunção, M., Buyya, R., Venugopal, S.: Intergrid: a case for internetworking islands of grids. Concurr. Comput. : Pract. Exper. 20, 997–1024 (2008)

[3] Banâtre, J., Fradet, P., Radenac, Y.: Generalised multisets for chemical programming. Mathematical Structures in Computer Science 16(4), 557–580 (2006)

[4] Caron, E., Desprez, F., Loureiro, D., Muresan, A.: Cloud computing resource management through a grid middleware: A case study with diet and eucalyptus. In: IEEE CLOUD. pp. 151–154 (2009)

[5] Chen, L., Candan, K., Tatemura, J., Agrawal, D., Cavendish, D.: On overlay schemes to support point-in-range queries for scalable grid resource discovery. In: 5th International Conference on Peer-to-Peer Computing. pp. 23–30. IEEE (2005)

[6] Fernández, H., Tedeschi, C., Priol, T.: A Chemistry-Inspired Workflow Management System for Scientific Applications in Clouds. In: 7th International Conference on e-Science. pp. 39–46. IEEE, Stockholm, Sweden (2011)

[7] Fiscato, M., Costa, P., Pierre, G.: On the feasibility of decentralized grid scheduling. In: SASO Workshops. pp. 225–229 (2008)

[8] Leal, K., Huedo, E., Llorente, I.M.: A decentralized model for scheduling independent tasks in federated grids. Future Generation Computer Systems 25 (2009)

[9] Napoli, C.D., Giordano, M., Pazat, J.L., Wang, C.: A Chemical Based Middleware for Workflow Instantiation and Execution. In: ServiceWave. pp. 100–111 (2010)

[10] Németh, Z., Pérez, C., Priol, T.: Distributed workflow coordination: molecules and reactions. In: IPDPS (2006)

[11] Obrovac, M., Tedeschi, C.: When Distributed Hash Tables Meet Chemical Programming for Autonomic Computing. In: 15th International Workshop on Nature Inspired Distributed Computing (NIDisC 2012), in conjunction with IPDPS 2012. IEEE, Shanghai, China (May, 21-25 2012), to appear.

[12] Ranjan, R., Rahman, M., Buyya, R.: A decentralized and cooperative workflow scheduling algorithm. In: 8th International Symposium on Cluster Computing and the Grid. CCGRID '08. pp. 1–8. IEEE (2008)

[13] Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science 2218, 329–350 (2001)

[14] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. 11(1), 17–32 (2003)

[15] Topcuoglu, H., Hariri, S., Wu, M.Y.: Task scheduling algorithms for heterogeneous processors. In: Heterogeneous Computing Workshop (HCW). pp. 3 –14 (1999)

[16] Vasilakos, X., Sacha, J., Pierre, G.: Decentralized As-Soon-As-Possible grid scheduling: A feasibility study. In: 2010 Proceedings of 19th Int. Conference on Computer Communications and Networks (ICCCN). pp. 1–6. IEEE (2010)

[17] Viroli, M., Zambonelli, F.: A biochemical approach to adaptive service ecosystems. Information Sciences pp. 1–17 (2009)

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399