# Locality-aware Cooperation for VM Scheduling in Distributed Clouds

Jonathan Pastor[1], Marin Bertier[2], Frédéric Desprez[4], Adrien Lebre[1], Flavien Quesnel[1] and Cédric Tedeschi[3]

[1] ASCOLA Research Group, Mines Nantes / Inria / LINA, Nantes, France
[2] ASAP Research Group, INSA / Inria / IRISA, Rennes, France
[3] Myriads Research Group, Université de Rennes 1 / Inria / IRISA, Rennes, France
[4] Avalon Research Group, LIP ENS Lyon UMR 5668, Lyon, France
firstname.lastname@inria.fr

**Abstract.** The promotion of distributed Cloud Computing infrastructures as the next platform to deliver the Utility Computing paradigm, leads to new virtual machines (VMs) scheduling algorithms leveraging peer-to-peer approaches. Although these proposals considerably improve the scalability, leading to the management of hundreds of thousands of VMs over thousands of physical machines (PMs), they do not consider the network overhead introduced by multi-site infrastructures. This overhead can have a dramatic impact on the performance if there is no mechanism favoring intra-site *v.s.* inter-site manipulations.
This paper introduces a new building block designed on top of a network with Vivaldi coordinates maximizing efficient collaborations between PMs. We combined such a mechanism with DVMS, a large-scale virtual machine scheduler and showed its benefit by discussing several experiments performed on four distinct sites of the Grid'5000 testbed. With our proposal and without changing the scheduling decision algorithm, the number of inter-site operations has been reduced by 72%. This result provides a glimpse of the promising future of using locality properties to improve the performance of massive distributed Cloud platforms.

**Keywords:** Cloud Computing, locality, peer-to-peer, network overlay, Vivaldi, DVMS, virtual machine scheduling

## 1 Introduction

Introduced a few years ago [6], the new trend to deliver Cloud Computing resources, in particular Infrastructure as a Service (IaaS) solutions, consists in leveraging several infrastructures distributed world-wide. If such distributed Cloud Computing platforms deliver undeniable advantages to address important challenges such as reliability, latency or even in somehow jurisdiction concerns, most mechanisms that were previously used to operate centralized IaaS platforms must be revisited to offer the same level of transparency for the end-users.

Keeping such an objective in mind, the use of the P2P paradigm has to be strongly investigated. This is particularly true for scheduling algorithms in charge of assigning virtual machines (VMs) on top of physical machines (PMs) according to their effective needs (and reciprocally usages), to preserve a good quality of service (QoS). Indeed and although major improvements have been done, centralized approaches [8] are neither scalable nor robust enough. Hierarchical solutions [4] that can be seen as good candidates face important limitations: First, finding an efficient partitioning of resources is a tedious task as matching a hierarchical overlay network on top of a distributed infrastructure is often not natural. Secondly, in addition to requiring complex failover mechanisms to face crashed leader/super peer and network disconnections, hierarchical structures have not been designed to react swiftly to physical topology changes such as node apparitions/removals and network performance degradations. P2P algorithms allow to address both concerns, *i.e.,* scalability as well as resiliency of infrastructures. However, and despite promising approaches have been proposed to address the scheduling problem of VMs in a P2P fashion [13, 3], they are still facing limitations coming from the overlay network they rely on. The approach proposed in [13] maps a ring overlay network on a distributed infrastructure which prevents making any distinction between close nodes and distant ones. Similarly, the approach described in [3], while adopting an orthogonal, gossip-based approach, still suffers from building a randomized overlay network, thus breaking the physical topology.

Considering that both the network latency and the bandwidth between peers have a strong impact on the reactivity criteria of the scheduling problem, *locality* properties of peers should be considered to favor efficient VM operations. In other words, to reduce as much as possible the time to switch from one schedule (*i.e.,* a mapping between VMs and PMs running in the infrastructure) to another one, it is crucial to make cooperation first between peers in the closest neighborhoods before contacting peers belonging to other sites. Moreover, it is noteworthy that this notion of locality is dynamic, and varies over time according to the network bandwidth/latency and disconnections.

The contribution of this paper is a new building block that enables to tackle the *locality* concern in distributed VM scheduling algorithms such as the two aforementioned ones. We estimate the locality through a cost function of the latency/bandwidth tuple between peers in the network, thus enabling each peer to select its closest neighbors. We rely on Vivaldi [2], a simple decentralized protocol allowing to map a network topology onto a logical space while preserving locality. On top of Vivaldi, a shortest path construction, similar to the well-known Dijkstra algorithm, is performed each time there is a need for cooperation between two nodes taking part in the schedule. We illustrate the advantage of this new building block by changing the overlay network in the DVMS proposal [13] for it. We selected DVMS as we have a good expertise of it and because it is, as far as we know, the only one that guarantees to find a solution if one exists [12].

The remainder of this article is structured as follows. In Section 2, we discuss some background regarding the DVMS proposal and the P2P technics to handle the locality aspects. Section 3 gives an overview of our proposal by introducing the short path algorithm on top of Vivaldi and the way we integrate it into

DVMS. In Section 4, we validate the proposal by analyzing its benefits with respect to the previous version of DVMS by discussing experiments conducted on Grid'5000. Related works are discussed in Section 5. Finally, we discuss perspectives and conclude this article in Section 6.

## 2 Background

### 2.1 DVMS

DVMS [12, 13] (Distributed Virtual Machine Scheduler) is a framework that schedules VMs cooperatively and dynamically in large-scale distributed systems. It is deployed as a set of agents that are organized following a ring topology and that cooperate with one another to guarantee that VM demands are satisfied during their executions. Concretely, when a node cannot guarantee the QoS for its hosted VMs or when it is under-utilized, it starts an iterative scheduling procedure (ISP) by querying its first neighbor to find a better placement; it thus becomes the initiator of the ISP. If the neighbor cannot satisfy the request, it is forwarded to the following free node until the ISP succeeds. When a viable mapping has been found, the leader (*i.e.,* the last peer that has taken part to the ISP) reconfigures the system by performing adequate VM migrations. Such an approach allows each ISP to send requests only to a minimal number of nodes and even though an ISP can reserve all nodes if the corresponding problem is particularly hard to solve (thus guaranteeing that a solution will always be found if it exits), experiments have shown that in most cases ISPs involve only few nodes. Moreover, the DVMS proposal allows several ISPs to occur independently at the same moment throughout the infrastructure; in other words, scheduling is performed on partitions of the system that are created dynamically, which significantly improves the reactivity of the system. To prevent conflicts that could occur if several ISPs performed concurrent operations on the same PMs or VMs, it should be emphasized that PMs are reserved for exclusive use by a single ISP.

An example involving three partitions is shown in Figure 1; in particular, we can see the growth of partition 1 between two steps. Explaining in detail the notion of "first out" is beyond the scope of this article but readers can consider that the "first out" relation enables to handle communications efficiently, as each node involved in a partition can forward a request directly to the first node on the outside of its partition [13].

We formally proved the correctness of DVMS using temporal logic, and we validated the first version of the prototype at large scale (by means of simulations involving up to 80k VMs and 8k nodes and with experiments on the Grid'5000 testbed involving up to 4.7k VMs and 470 nodes [12]).

As discussed earlier, one limitation of this approach is related to its ring topology that prevents it from taking into account the actual network topology. In other words, if the ISP strategy enables to limit the size of one partition to a minimal number of nodes, these nodes are selected without considering

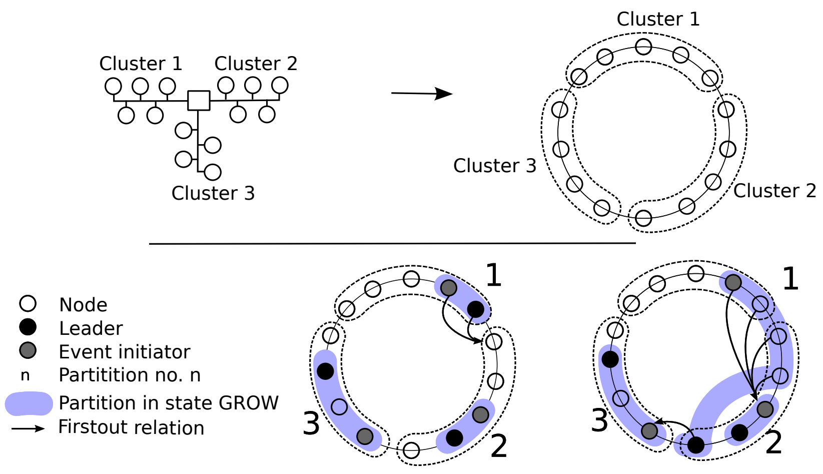

**Fig. 1.** Solving three problems simultaneously and independently with DVMS. The ring has been matched on top of three distinct clusters.

the network conditions at the time the ISP starts. This can lead to inefficient situations where VM migrations occur between two nodes that are far from each other, which lasts longer than a migration between two close nodes. Obviously the ring can be built to limit the distance between peers globally (*i.e.,* peers of the same region/area would be grouped together as illustrated in Figure 1). However, in such a case, at least two nodes of each group are directly connected to two far nodes. Note that an approach such as the one proposed in [5], which consists in deploying one ring per site and relying on a *super-ring* to interconnect few representatives of each local ring, would not solve many problems. Besides problems inherent to hierarchical and structured overlay networks, this solution would not provide a good answer to locality: When going out of the local ring, it would still not be possible to find the next closest ring.

## 2.2 Overlay Networks and Locality

As illustrated in the previous paragraph, one of the primary downsides of overlay networks lies in that they break the physical topology by connecting nodes that have no physical proximity. Besides hierarchical attempts in building locality-aware overlay networks [5, 17, 18], we can first mention the locality improvement mechanisms of the Pastry structured overlay network [15]. In order to reduce the latency of the routing process, each node is given the opportunity to choose the closest nodes to fill its routing table. Learning the existence of new nodes relies on a periodic exchange of parts of routing tables.

Similar mechanisms have been adopted within unstructured overlay networks to make their logical connections reflect the physical proximity of nodes, each node discovering its closest nodes through gossiping. Note that the proximity between two nodes can be estimated through any transitive metric, in particular the latency between the nodes [9].

These approaches need to constantly maintain the knowledge of close nodes in order to provide the *best* node possible at the cost of periodic communications (uncorrelated to the actual amount of requests to be processed by the overlay network).

The overlay network we propose in this paper differs in that it adopts a lazy approach consisting in searching close nodes only upon receipt of requests. This way, the quality of the response is proportional to the frequency of requests.

Our protocol relies on the Vivaldi protocol [2] to detect close nodes. Vivaldi places nodes in a multi-dimensional space. Each node is given coordinates inside this space reflecting its physical location. The protocol is based on simple message exchanges. Initially, each node is given a random position in the space and chooses (possibly arbitrarily) a small subset of nodes, composing its *view*. Then, each node starts estimating the round trip time between itself and another node chosen randomly in its view, and adapts its distance with this node in the space accordingly, coming closer to it or moving away from it. The nodes can repeat this step independently (each with another node from its view), to improve the accuracy of the positioning. A globally accurate positioning of nodes can be obtained very quickly (in a small number of such steps) if nodes have a few long-distance nodes in their view and if the network is not excessively dynamic. These long distance links can be easily maintained.

Recall that Vivaldi does not allow to directly know the nodes that are close in the network, but to be able to recognize them through their coordinates. Our overlay relies on the examination of Vivaldi coordinates of nodes discovered during the processing of requests sent to it.

## 3 Contributions

The aim of this paper is to revisit a distributed scheduling algorithm, the DVMS proposal, in order to take account of locality criteria. To this aim, we focus first on the overlay network, and second, we propose an abstraction that allows combining DVMS with a locality-aware overlay network without being intrusive in its source code.

### 3.1 Locality-aware Overlay Network

We here present our lazy locality-aware overlay network that underlies the VM scheduling platform we developed. It is made of two layers.

The lower layer is mainly an implementation of the Vivaldi protocol (which core mechanisms were described earlier) making nodes (that are initially interconnected arbitrarily) aware of their position in the infrastructure.

Based on these coordinates, the upper layer is responsible for building a locality-aware overlay dynamically. This layer takes its roots in the classic Dijkstra's shortest path algorithm to collect a set of close nodes starting from a given position.

**Searching for Close Nodes.** Once the Vivaldi map is achieved, and each node knows its coordinates, we are able to estimate how *close* two given nodes are by calculating their distance in the map. However, recall that the view of each node does not *a priori* contain its closest nodes. (In the following, we call this view the **network view**, to distinguish it from the **spiral view** to be introduced later.) Therefore, we need additional mechanisms to locate a set of nodes that are close to a given initial node. Vivaldi gives a *location* to each node, not a neighborhood.

We use a modified, distributed version of the classic Dijkstra's shortest path algorithm that leverages the Vivaldi map to build such a neighborhood. More specifically, its goal is to build a **spiral**[5] interconnecting the nodes in the plane that are the closest ones from a given initial node.

Let us consider that our initial (or root) point is the node $n_R$. The first step is to find a node to build a two-node spiral starting with $n_R$. This is done by selecting the node from $n_R$'s network view, say $n_i$, which exhibits the smallest distance with $n_R$. $n_i$ becomes the second node in the spiral. From this point on, $n_R$ remembers $n_i$ as its successor and $n_i$ remembers $n_R$ as its predecessor. $n_R$ also sends its network view to $n_i$, which, on receipt, creates its **spiral view** that contains the $N$ nodes closest to $n_R$ taken from both $n_R$ and $n_i$ network views. It will allow $n_i$ to find the next node to build the spiral. Assuming this closest node from $n_R$ in $n_i$'s spiral view is $n_j$, $n_j$ will be added in the spiral by becoming the successor of $n_i$. $n_j$ receives $n_i$'s spiral view and creates and fills its own spiral view with nodes closest to $n_R$ contained in both $n_i$'s spiral view and $n_j$'s network view. This algorithm is repeated until the amount of nodes requested by the application have been interconnected in the spiral.

Note that there is a risk to be blocked at some point, having a spiral view containing only nodes that are already in the spiral, hindering from extending it further. However, this problem can be easily addressed by introducing few long-distance nodes when the spiral view is created/updated.

**Learning.** Applying the protocol described above, the quality of the spiral is questionable in the sense that the nodes that are actually close to the root node $n_R$ may not be included.To improve the *quality* of the spiral, *i.e.,* to reduce the average distance from each of its nodes to the initial node, we rely on a learning mechanism coming with no extra communication cost: When a node is contacted to become the next node in one spiral, and when it receives the associated spiral view, it can also keep in its network view the nodes that are closer to itself, thus potentially increasing the quality of a future spiral construction. Such an improvement through learning is illustrated in Figure 2. Note that learning may also be used to constantly improve already built spirals. While providing obvious advantages, allowing it comes at the cost of changing links in the spirals dynamically, which may not match all applications' constraints.

[5] Our use of the term *spiral* is actually a misuse of language, since the graph drawn in the plane might contain crossing edges.

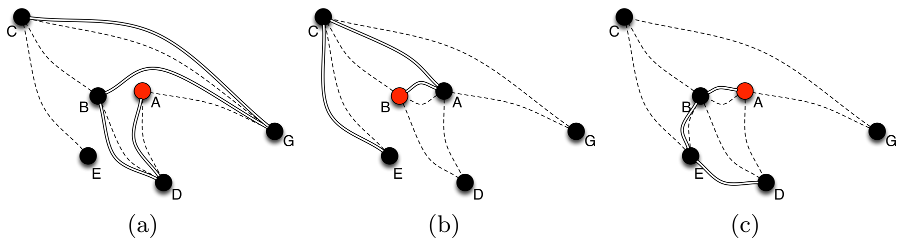

**Fig. 2.** Learning mechanism: (a) The initial view of each node is materialized by the dashed lines. Given these views, the spiral obtained from node A is represented by the double thick lines. In particular, this spiral allowed $A$ and $B$ to discover each other. (b) If $B$ starts building a spiral, it will start by contacting $A$. This spiral construction allows also E and B to discover each other. (c) If $A$ is requested to start another spiral, it will exhibit an increased locality awareness.

## 3.2 PeerActor: A Building Block to Abstract Overlay Networks

As described in Section 2.1, the DVMS proposal can be divided in two major components: (i) The ring overlay network and (ii) the protocol in charge of detecting and resolving scheduling issues. As our goal consists in taking into account locality criteria without changing the DVMS protocol, we designed a building block, *i.e., the Peer actor*, which enables us to revisit DVMS by abstracting the overlay network it relies on. At a coarse-grain level, the Peer actor can be seen as a generic layer for high level distributed services, providing network abstractions and robust communications between agents deployed on each node. By leveraging the Peer actor API, developers can focus on the service itself without dealing with node apparitions/removals and network disconnections.

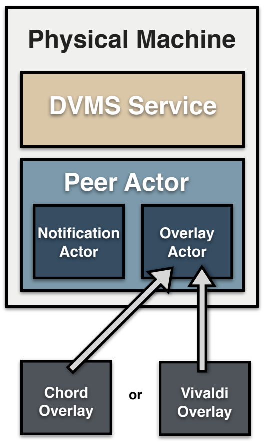

**Fig. 3.** DVMS on top of the Peer actor.

From the software point of view, the Peer actor relies on modern software frameworks (Scala and Akka) following the actor model rules. In such a model, each instance will collaborate by exclusively exchanging messages, and priority will be given to collaboration between close instances. Such a coding approach enables to tackle several issues of distributed systems such as deadlocks and race conditions.

As illustrated in Figure 3, the Peer actor contains two sub actors: The *Notification actor* and the *Overlay network actor*. The Notification actor enables services to subscribe to events that will be triggered by other services. The Overlay network actor is in charge of sending/receiving messages through the network. In order to compare both approaches, ring-based *v.s.* locality-aware, we developed two different Overlay network actors: The first one provides a

Chord-like overlay [16], while the second one delivers the locality-aware overlay described in Section 3.1.

## 4 Experiments

The main objective of the experiments we conducted was to estimate the impact of locality on the performance of a distributed scheduling algorithm. A significant portion of the reconfiguration time is spent in live migration of virtual machines, which depends of network parameters such as latency and bandwidth. One way to improve the performance of distributed scheduling algorithms is to promote collaborations between close resources, which can be reached by maximizing the ratio *nb intrasite migrations*/*nb migrations*.

### 4.1 Experimental Protocol

To compare our experiments, we implemented a dedicated injector that makes load changes of VMs during a predefined time. VMs are launched on PMs in a round-robin manner, *i.e.,* each PM hosts roughly the same number of VMs at the beginning. The experiment consists in repeatedly changing target CPU loads of VMs. Every $t$ seconds, the injector that is deployed on a dedicated node selects one VM and changes its CPU load according to a Gaussian distribution. $t$ is a random variable that follows an exponential distribution with rate parameter $\lambda$. The Gaussian distribution is defined by a mean ($\mu$) as well as a standard deviation ($\sigma$) that are given at the beginning of the experiment. The parameters are $\lambda = Nb\_VMs/300$ and $\mu = 70$, $\sigma = 30$. Concretely, the load of each VM starts from 0% and varies on average every 5 minutes in steps of 10 (with a significant part between 40% and 100% of CPU usage). The duration of each experiment was set to 3600 seconds.

For each experiment, we booked 40 compute servers spread over 4 geographical sites (10 servers per site) and 1 service server from the Grid'5000 testbed. The compute servers were used to run virtual machines and DVMS while the service node runs the aforementioned injector. Each compute node was equipped with 8 cores and hosted a number of virtual machines proportional to its number of CPU cores ($nbVM = 1.3 \times nb\ cores$), leading to a global number of 416 VMs. Although such a number is rather small regarding the latest experiments that have been performed on DVMS [12], our goal is not to validate once again the scalability criteria but to focus on the locality aspect of such an algorithm.

### 4.2 Results

**Maximization of Intra-Site Migrations.** Table 1 compares the ratio between intra-site migrations and the total number of migrations, using Chord or our locality-based overlay (LBO) network. The results show that the impact of locality is significant: Using LBO leads to an average number of 86.3% of intra-site migrations while using a Chord-based DVMS decreases this ratio to 49.6%.

| | Chord | LBO |
|---|---|---|
| average | 0.496 | 0.863 |
| minimum | 0.378 | 0.798 |
| maximum | 0.629 | 0.935 |

**Table 1.** Comparison of intra-site migrations ratio using DVMS/Chord and DVMS/LBO.

| | Grenoble | Luxembourg | Nancy | Rennes |
|---|---|---|---|---|
| Grenoble | 0.09 ms | 16.55 ms | 14.24 ms | 15.92 ms |
| Luxembourg | | 0.17 ms | 2.70 ms | 13.82 ms |
| Nancy | | | 0.27 ms | 11.42 ms |
| Rennes | | | | 0.23 ms |

**Table 2.** Latency measured between sites.

| | Chord | LBO |
|---|---|---|
| average size (servers) | 3.918 | 2.337 |
| average number of sites involved | 1.645 | 1.082 |
| average duration (msec) | 154.63 | 98.50 |

**Table 3.** Comparison of partitions metrics using DVMS/Chord and DVMS/LBO.

**Dynamic Clustering.** During our investigation of the results brought about LBO, we noticed that many of the inter-site migrations were performed between Luxembourg and Nancy sites. In Table 2, it is noticeable that Luxembourg and Nancy have a latency that is significantly below usual inter-site latencies (Nancy and Luxembourg are separated by only 100 kilometers), while Rennes and Grenoble have almost the same latency with all their respective remote sites. Indeed, servers located in Luxembourg and Nancy are more likely to collaborate with each other, while those located on Rennes and Grenoble will find collaborators regardless of their location. This explains why many of the inter-site migrations were performed between Luxembourg and Nancy. This means that LBO enabled DVMS to learn which site is more interesting to perform VM migration. Promoting low latency inter-site collaboration made many inter-site migrations acceptable compared to those executed by the Chord version.

**Reactivity.** Table 3 depicts metrics that allow for an objective comparison of the efficiency of both overlay networks.

Firstly, using the LBO decreases the number of servers that are involved in partitions by 41%, meaning that collaboration between close nodes has become more efficient.

Secondly, it is interesting that using the LBO leads to a partition duration 46% lower than that encountered with Chord. This result is consistent with the fact that with our locality-aware overlay, the number of sites that are involved in partitions becomes very close to one: Collaborating with closer nodes allows performing scheduling/reconfiguring phases much faster, since migration operations are shorter in time, thus increasing considerably the reactivity of the system.

## 5 Related Work

Many virtual infrastructure managers have been proposed to deal with specific concerns. In this section, we will focus on some of their limitations, especially regarding locality, scalability, and fault-tolerance.

The most common managers are the centralized ones, like Entropy [7, 8], since they are easy to deploy. They are generally designed to work on a cluster. In this context, they do not take account of the network topology, and they cannot manage VMs efficiently in a multi-site/multi-cluster deployment. Moreover, they are prone to fault-tolerance, scalability, and reactivity issues; to avoid these limitations, one possibility is to rely on more decentralized approaches, like hierarchical or distributed ones.

Hierarchical managers, like Snooze [4], may be more suited to handle locality. For instance, it is possible to setup (i) one manager per cluster, and (ii) one (fault-tolerant) super manager that monitors cluster managers and chooses on which cluster a new VM should start. The main problem with this approach is that, in the absence of cooperation between cluster managers, VMs cannot be migrated from one cluster to another, which is especially annoying if one cluster is overloaded. Moreover, the super manager is not necessarily aware of the network topology and therefore may not be able to interact efficiently with cluster managers if the latter are distributed among several sites. Furthermore, the super manager limits the scalability of this approach; to deal with this issue, researchers have designed distributed approaches.

Many distributed approaches have been proposed to manage VMs [1, 3, 10, 11, 14, 19]. Some of them are limited in terms of scalability since they (i) require a global view of the infrastructure to take a decision [14, 19] and/or (ii) rely on a centralized service node that is not fault-tolerant [11, 19]. Some approaches lead to a huge number of migrations [1, 11] without necessarily optimizing the chosen scheduling criterion [1]. Moreover, none of these approaches have been designed to take account of the network topology and therefore manage VMs efficiently in a multi-site deployment.

To summarize, a locality-aware distributed approach is required to (i) avoid issues related to scalability and single points of failures, and to (ii) manage VMs efficiently in heterogeneous network environments like those found in multi-site/multi-cluster deployments. The work presented in this paper targets such a challenge.

## 6 Conclusion

Cloud Computing has entered our everyday life at a very high speed and huge scale. From classic High Performance Computing simulations to the management of huge amounts of data coming from mobile devices and sensors, its impact can no longer be minimized. While promoted for a long time, delivering Cloud Computing capabilities by leveraging only few large-scale data centers does not enable to cope with the demand of Cloud resources anymore, and a new model consisting in leveraging several micro/nano data centers distributed WANwide is more and more investigated. The main challenge is thus to revisit most of the mechanisms that are common to current IaaS management systems to leverage more decentralized algorithms. Among the different contributions that have been proposed, a large number have focused on the scheduling issue of the VMs to achieve the scalability required but at the expense of the locality criteria. However, manipulating VMs WANwide degrades significantly the performance as well as the quality of the service of the whole system.

Hence, the first step toward such a highly distributed Cloud infrastructure is to take into account this notion of locality between Cloud Computing resources. In this paper, we showed how such locality criteria can be considered by delivering a new building block using P2P algorithms and a Vivaldi overlay network connected to the DVMS proposal, an efficient and flexible VM scheduler. Our first experiments over Grid'5000 showed that, connecting 4 different sites and scheduling VMs over them, we could gain up to 72% of inter-site operations. It is worth noting that one experimental observation we had during this work was that the proposed overlay network was actually able to reflect the underlying topology, and in particular to build a hierarchical overlay dynamically if the underlying topology is hierarchical.

Our future work will consist in refining the decision model used in scheduling mechanisms to enable them to consider the cost difference between intra-site and inter-site migrations, thus promoting intra-site migrations in multi-site partitions. More generally, the association between locality-based overlay networks and Peer Actors will become a building block for revisiting every single service composing IaaS systems. It will enable to deliver a new generation of Utility Computing as depicted by the Discovery Initiative[6].

**Acknowledgments.** Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER, and several Universities as well as other funding bodies (see `https://www.grid5000.fr`).

## References

1. Barbagallo, D., Di Nitto, E., Dubois, D., Mirandola, R.: A Bio-inspired Algorithm for Energy Optimization in a Self-Organizing Data Center. In: Self-Organizing Architectures, LNCS, vol. 6090, pp. 127–151. Springer (2010)
2. Dabek, F., Cox, R., Kaashoek, M.F., Morris, R.: Vivaldi: A Decentralized Network Coordinate System. In: 2004 conf. on Applications, technologies, architectures, and protocols for computer comm. pp. 15–26. SIGCOMM '04 (2004)

[6] `http://beyondtheclouds.github.io/`

3. Feller, E., Morin, C., Esnault, A.: A Case for Fully Decentralized Dynamic VM Consolidation in Clouds. In: CloudCom'12: 4th IEEE International Conference on Cloud Computing Technology and Science (Dec 2012)
4. Feller, E., Rilling, L., Morin, C.: Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds. In: CCGRID '12: 12th Int. Symp. on Cluster, Cloud and Grid Comp. pp. 482–489 (May 2012)
5. Garcés-Erice, L., Biersack, E.W., Ross, K.W., Felber, P., Urvoy-Keller, G.: Hierarchical Peer-To-Peer Systems. Parallel Processing Letters 13(4), 643–657 (2003)
6. Greenberg, A., Hamilton, J., Maltz, D.A., Patel, P.: The Cost of a Cloud: Research Problems in Data Center Networks. SIGCOMM Comput. Commun. Rev. 39(1), 68–73 (Dec 2008)
7. Hermenier, F., Demassey, S., Lorca, X.: Bin Repacking Scheduling in Virtualized Datacenters. In: CP '11: Proceedings of the 17th international conference on Principles and practice of constraint programming. pp. 27–41. Springer (2011)
8. Hermenier, F., Lawall, J., Muller, G.: BtrPlace: A Flexible Consolidation Manager for Highly Available Applications. IEEE Transactions on Dependable and Secure Computing 99(PrePrints) (2013)
9. Jelasity, M., Babaoglu, O.: T-Man: Gossip-based Overlay Topology Management. In: In Third International Workshop on Engineering Self-Organising Applications (ESOA'05). pp. 1–15. Springer-Verlag (2005)
10. Marzolla, M., Babaoglu, O., Panzieri, F.: Server consolidation in Clouds through gossiping. In: WoWMoM '11: Proceedings of the 12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks. pp. 1–6. IEEE Computer Society, Washington, DC, USA (Jun 2011)
11. Mastroianni, C., Meo, M., Papuzzo, G.: Self-economy in cloud data centers: statistical assignment and migration of virtual machines. In: Euro-Par '11: 17th International Conference on Parallel Processing. vol. 1. Springer (2011)
12. Quesnel, F., Lebre, A., Pastor, J., Sudholt, M., Balouek, D.: Advanced Validation of the DVMS Approach to Fully Distributed VM Scheduling. In: ISPA '13: 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. vol. 0, pp. 1249–1256 (Jul 2013)
13. Quesnel, F., Lèbre, A., Südholt, M.: Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS. Concurrency and Computation: Practice and Experience 25(12), 1643–1655 (Aug 2013)
14. Rouzaud Cornabas, J.: A Distributed and Collaborative Dynamic Load Balancer for Virtual Machine. In: Euro-Par 2010 Parallel Processing Workshops, Lecture Notes in Computer Science, vol. 6586, pp. 641–648. Springer (Aug 2011)
15. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware) (Nov 2001)
16. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: ACM SIGCOMM Computer Communication Review. vol. 31, pp. 149–160. ACM (2001)
17. Xu, Z., Mahalingam, M., Karlsson, M.: Turning Heterogeneity into an Advantage in Overlay Routing. In: INFOCOM (2003)
18. Xu, Z., Zhang, Z.: Building Low-Maintenance Expressways for P2P Systems. Tech. Rep. HPL-2002-41, Hewlett-Packard Labs (2002)
19. Yazir, Y.O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., Coady, Y.: Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In: Cloud '10: IEEE 3rd Int. Conf. on Cloud Computing. pp. 91–98. Los Alamitos, CA, USA (Jul 2010)