Beyond The Cloud, How Should Next Generation Utility Computing Infrastructures Be Designed?

Marin Bertier, Frédéric Desprez, Gilles Fedak, Adrien Lebre, Anne-Cécile Orgerie, Jonathan Pastor, Flavien Quesnel, Jonathan Rouzaud-Cornabas and Cédric Tedeschi

1 Context and Motivations

The success of Cloud Computing has driven the advent of Utility Computing. However Cloud Computing is a victim of its own success: In order to answer the ever increasing demand for computing resources, Cloud Computing providers must build data centers (DCs) of ever-increasing size. Besides facing the well-known issues of large-scale platforms management, large-scale DCs have to deal with energy considerations that limit the number of physical resources that one location can host. However, instead of investigating other solutions that can tackle aforementioned concerns, the current trend consists in deploying larger and larger DCs in few strategic locations that deliver energy advantages. For example, Western North Carolina, USA, is an attractive area due to its abundant capacity of coal and nuclear power following the departure of the region's textile and furniture manufacturing [23]. More recently, several proposal suggest building next generation DCs close to the polar circle in order to leverage free cooling techniques, considering that cooling is accounting for a big part of the consumed electricity [25].

1.1 Inherent limitation of large-scale DCs

Although building large scale DCs enables to cope with the actual demand , it is far from delivering sustainable and efficient UC infrastructures. In addition to requiring the construction and the deployment of a complete network infrastructure to reach each DC, it exacerbates the inherent limitations of the Cloud Computing model:

INRIA, France, e-mail: firstname.lastname@inria.fr

- The externalization of private applications/data often faces legal issues that restrain companies from outsourcing them on external infrastructures, especially when located in other countries.
- The overhead implied by the unavoidable use of the Internet to reach distant platforms is wasteful and costly in several situations: Deploying a broadcasting service of local events or an online service to order pizza at the edge of the polar circle leads to important overheads since it can be assumed that a vast majority of the users are located in the neighborhood of the event/the pizzeria.
- The connectivity to the application/data cannot be ensured by centralized dedicated centers, especially if they are located in a similar geographical zone. The only way to ensure disaster recovery is to leverage distinct sites.[1]

Although hybrid or federated Cloud solutions [8] that aim at extending the resources available on one Cloud with another one can partially tackle the two former points, the latter requires a disruptive change in the way UC resources are managed. Deploying a local events broadcasting service or an online service to order pizza at the edge of the polar circle for instance, leads to an important overhead in terms of energy footprint, network exchanges as well as latency since it can be assumed that a vast majority of the users are located in the neighborhood of the event/the pizzeria. According to some projections of a recent IEEE report [26], the network traffic continues to double roughly each year. Bringing the IT services closer to the end-users is becoming crucial to limit the energy impact of these exchanges and to save the bandwidth of some links. Similarly, this notion of locality is also critical for the adoption of the UC model by applications that need to deal with a large amount of data as getting them in and out actual UC infrastructures may significantly impact the global performance [20].

The concept of micro/nano DCs at the edge of the backbone [25] may be seen as the complementary solution to hybrid platforms in order to reduce the overhead of network exchanges. However, operating multiple small DCs breaks the idea of resources mutualization for energy saving, making this approach questionable. Moreover, the number of such micro/nano DCs will remain limited and the question of how federating a large number of such facilities is still not solved.

1.2 Ubiquitous and Oversized Network Backbones.

While larger and larger DCs are created, people are (and will be more and more) surrounded by computing resources, especially the ones in charge of interconnecting all IT equipments. Even though these small and medium-size facilities include resources that are barely used [7, 11], they can hardly be removed (*e.g.* routers). Considering this important aspect, we claim that a new generation of UC platorms can be delivered by deploying the concept of the micro/nano DCs directly inside the

[1] "Amazon outages – lessons learned", http://gigaom.com/cloud/amazon-outages-lessons-learned/ (valid on Nov 2013, the 30th).

internet backbone, starting from the core nodes of the backbone to the different network access points in charge of interconnecting public and private institutions. By such a mean, network and UC providers would be enable to mutualize resources that are mandatory to operate network/data centers while delivering widely distributed UC platforms that can better match the geographical dispersal of users.

Figure 1.2 illustrates the advantages of our proposal. It shows a snapshot of the network weather map of RENATER[2], the network backbone dedicated to universities and research institutions in France. It reveals several important points:

- As indicated, it clearly shows that most of the resources are under-used (only two links are used between 45% and 55%, a few between 25% and 40% and the majority below the threshold of 25%).
- The backbone has been deployed and is renewed according to demand: the density of points of presence (PoP) of the network as well as the bandwidth of each link are more important on the edge of large cities such as Paris, Lyon or Marseille.
- The backbone has been deployed in order to face disconnections, *i.e.* 95% of the PoPs can be reached by at least two distinct routes.

1.3 Locality Based Utility Computing

This chapter aims at introducing locality-based UC infrastructures, a new generation of UC platforms that resolves the lack of locality of current solutions relying on large-scale DCs. Although it involves radical changes in the way physical and virtual resources are managed, leveraging network centers is a promising way to deliver highly efficient and sustainable UC services,

From the physical point of view, network backbones such as the RENATER one provide appropriate infrastructures, that is, reliable and efficient enough to operate UC resources spread across the different PoPs. Ideally, UC resources would be able to directly take advantage of computation cycles available on network active devices, *i.e.* the one in charge of routing packets. However, leveraging network resources to make external computations may lead to important security concerns. Hence, we propose to extend each network center with a number of servers dedicated to host VMs. As we can expect that the distribution between network traffics and UC demands would be proportional, larger network centers will be completed by more UC resources than the smaller ones. Moreover by deploying UC services on relevant PoPs, a LUC infrastructure will be able to natively confine network exchanges to a minimal scope, minimizing both the energy footprint of the network and the impact on latency.

From the software point of view, the main challenge is to design a complete distributed system in charge of turning a complex and diverse network of resources

[2] http://www.renater.fr

into a collection of abstracted computing facilities that is both reliable and easy to operate.

By designing an advanced system that offers the possibility to operate a large number of UC resources spread throughout distinct sites *i.e.*, a *LUC Operating System*, in a unified manner, ISPs as well as academic and private institutions in charge of operating a network backbone will be able to build an extreme-scale LUC infrastructure with a limited additional cost. Instead of redeploying a complete installation, they will be able to leverage IT resources and specific devices such as computer room air conditioning units, inverters or redundant power supplies already present on each center of their backbone.

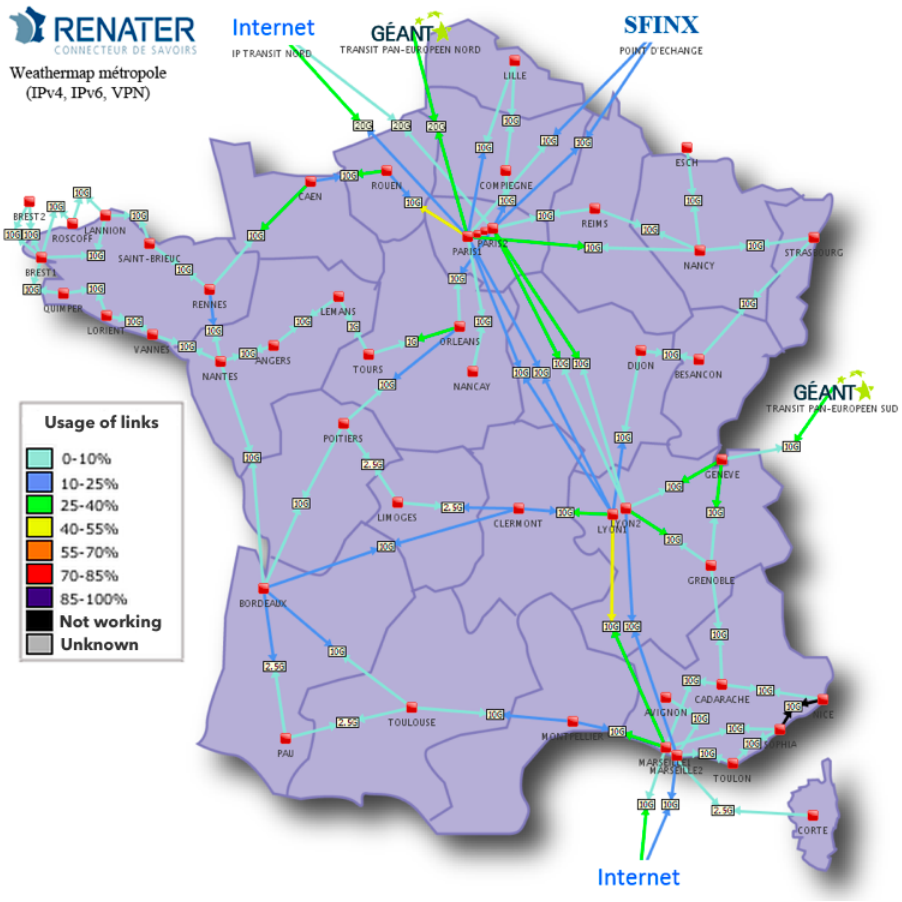

Fig. 1 The RENATER Weather Map on May 2013, the 27th, around 4PM.
Available in real-time at: http://www.renater.fr/raccourci

In addition to consider *Locality* as a primary concern, the novelty of the LUC OS proposal is to consider VM as the basic object it manipulates. Unlike existing research on distributed operating systems that have been designed on the process concept, a LUC OS will manipulate VMs throughout a federation of widely distributed physical machines. Virtualization technologies abstract out hardware heterogeneity, and allow transparent deployment, preemption, and migration of virtualized environments (VEs), *i.e.* a set of interconnected VMs. By dramatically increasing the flexibility of resource management, virtualization allows to leverage state-of-the-art results from other distributed systems areas such as autonomous and decentralized systems. Our goal is to build a system that allows end-users to launch VEs over a distributed infrastructure as simply as they launch processes on a local machine, *i.e.* without the burden of dealing with resources availability or location.

Section 2 describes the key objectives of a LUC OS and the associated challenges. Section 3 explains why our vision differs from actual and previous UC solutions. In Section 4, we present how such a unified system may be designed by delivering the premises of the DISCOVERY system, an agent-based system enabling distributed and cooperative management of virtual environments over a large-scale distributed infrastructure. Future work as well as opportunities are addressed in Section 5. Finally Section 6 concludes this chapter.

2 Overall Vision and Major Challenges

Similarly to traditional operating systems (OSes), a LUC OS will be composed of a significant number of mechanisms. Trying to identify all of them and establishing how they interact is an on-going work (see Section 4). However, having in mind the goal of delivering a unified system in charge of operating a complex and diverse infrastructure, and transform it into a LUC platform, we have identified the following objectives to be considered when designing a LUC OS:

- Scalability: a LUC OS must be able to manage hundreds of thousands of virtual machines (VMs) running on thousands of geographically distributed computing resources, including small and medium-sized computing facilities as well as any idle resource that their owner would make available. These resources might be highly volatile, especially if the LUC infrastructure allows to include resources hosted by end-users.
- Reactivity: To deal with the infrastructure's dynamicity, a LUC OS should swiftly handle events that require performing particular operations, either on virtual or on physical resources, with the objective of maximizing the system utilization while meeting QoS expectations of VEs. Reconfiguring VEs over distributed resources, sometimes spread across wide area networks, or moving VMs, while preserving their active connections, are examples of operations that should be performed as fast as possible.
- Resiliency: In addition to the inherent dynamicity of the infrastructure, failures and faults should be considered as the norm rather than the exception at such a

scale. The goal is therefore to transparently leverage the underlying infrastructure redundancy to (i) allow the LUC OS to continue working despite node failures and network disconnections and (ii) provide snapshotting as well as high availability mechanisms for VEs.
CT→AL **les points (i) et (ii) ne sont pas indépendants, j'ai l'impression que pour résoudre le point (i), on va utiliser ce que tu mentionnes dans le point (ii), du coup, c'est pas vraiment une énumération.**

- Sustainability: Although the LUC approach natively reduces the energy footprint of UC services by minimizing the network impact, it is important to go one step further by considering energy aspects at each level of a LUC OS and propose advanced mechanisms in charge of an optimal usage of each source of energy. To achieve such an objective, data related to the energy consumption of the VEs and the computing resources as well as the environmental conditions (computer room air conditioning unit, location of the site, etc.) should be taken into account by the system.
- Security and Privacy: Similarly to resiliency, security issues affect the LUC OS itself and the VEs running on it. For the LUC OS security, the goals are to (i) create trust relationships between different locations, (ii) secure the P2P layers, (iii) include security decision and enforcement points in the LUC OS and (iv) make them collaborate through the secured P2P layers to provide a secured infrastructure. For the VEs security, we need to provide users with a way to express their security requirements that will be enforced by collaborating several LUC OS security decision and enforcement points.
AL/JP **More on privacy? Only security is mentioned.**

In addition to the aforementioned objectives, targeting a distributed system where VM is the elementary granularity requires to deal with important issues regarding the management of the VM images. Managing VM images in a distributed way across a WAN is a real challenge that will require to adapt state-of-the-art techniques of replication and deduplication. Also, several mechanisms of a LUC OS must take into account VM images' location, for instance to allocate the right resources to a VE or to request VM images prefetching to improve deployment performance or VM relocations.

Amongst the numerous scientific and technical challenges that should be addressed, the lack of a global view of the system introduces a lot of complexity. In order to tackle it while addressing the above-mentioned challenges, we claim that internal mechanisms of a LUC OS should be based on decentralized mechanisms specifically designed for it. These techniques should provide mechanisms which are fully decentralized and autonomous, so to allow self-adapting control and monitoring of complex large-scale systems. Simple locality-based actions by each of the entities composing the system can lead to the global emergence of complex and sophisticated behaviors, such as the self-optimization of resource allocation, or the creation of decentralized directories. These techniques are starting to be used in well-known large systems. As an example, the Amazon website relies on its decentralized Dynamo service [19] to create largely distributed indexes and recover from data inconsistencies. Facebook's Cassandra massive scale structured store [29] also

leverages P2P techniques for its core operation. In a LUC OS, decentralized and self-organizing overlays will enable to maintain the information about the current state of both virtual and physical resources, their characteristics and availabilities. Such information is mandatory to build higher-level mechanisms ensuring the correct execution of VEs throughout the whole infrastructure.

3 Background

Several generations of UC infrastructures have been proposed and still co-exist [21]. However, neither Desktop, nor Grid, nor Cloud Computing platforms provide a satisfying UC model. Contrary to the current trend that promotes large offshore centralized DCs as the UC platform of choice, we claim that the only way to achieve sustainable and highly efficient UC services is to target a new infrastructure that better matches the Internet structure. Because it aims at gathering an unprecedented amount of widely distributed computing resources into a single platform providing UC services close to the end-users, a LUC infrastructure is fundamentally different from existing ones. Keeping in mind the aforementioned objectives, recycling UC resource management solutions developed in the past is doomed to failure.

As previously mentioned, our vision significantly differs from hybrid Cloud Computing solutions. Although these research activities address important concerns related to the use of federated cloud platforms such as the standardization of the interfaces for supporting cooperation and resource sharing over Cloud federations, their propositions are incremental improvements of the existing UC models. Recent hybrid and cloud federation investigations are comparable in some ways to previous works that have been done for Grids as the purpose of Grid middleware is to interact with each resource management system composing the Grid [14, 45, 49]. By taking into account network issues in addition to traditional computing and storage concerns in Cloud Computing systems, the European SAIL project[3] is probably the one which targets the biggest advances in regard to previous works on Grid systems. More concretely, this project investigates new network technologies to provide end-users of hybrid/federated Clouds the possibility to configure and virtually operate the network backbone interconnecting the different sites they use [37]. More recently, the *Fog Computing* concept has been proposed as a promising solution to applications and services that cannot be put into the cloud due to locality issues (mainly latency and mobility concerns) [13]. Although it might look similar to our vision as they propose to extend the Cloud Computing paradigm to the edge of the network, *Fog Computing* does not target a unified system but rather proposes to add a third party layer (*i.e.* the *Fog*) between cloud vendors and end-users. In our vision, UC resources (*i.e.* Cloud Computing ones) should be repacked in the different network hub of backbones and operated through a unified system, *i.e.* the LUC OS. As far as we know, the only system that investigated whether a widely distributed

[3] `http://www.sail-project.eu`

infrastructure can be operated by a single system, was the XtreemOS Project [36]. Although this project was sharing some of the goals of the LUC OS, it did not investigate how the geographical distribution of resources can be leveraged to deliver more efficient and sustainable UC infrastructures.

To sum up, we argue for the design and the implementation of a distributed OS, manipulating VEs instead of processes and considering locality as a primary concern. Referred to as a LUC Operating System, such a system will include most of the mechanisms that are common to current UC management systems [1, 3, 4, 5, 32, 35]. However, each of them will have to be rethought in order to leverage P2P algorithms. While largely unexplored for building operating systems, P2P/decentralized mechanisms have the potential to achieve the scalability required for LUC systems. Using all these technologies for establishing the foundation mechanisms of massive-scale distributed operating systems will be a major breakthrough from current static, centralized or hierarchical management solutions.

4 Premises of a LUC OS: The DISCOVERY Proposal

In this section, we propose to go one step further by discussing preliminary investigations around the design and the implementation of a first LUC OS proposal: the DISCOVERY system (DIStributed and COoperative Management of Virtual Environments autonomouslY). We draw the premises of the DISCOVERY system by emphasizing some of the challenges as well as some research directions to solve them. Finally, we give some details regarding the prototype that is under development and how we are going to evaluate it.

4.1 Overview

The DISCOVERY system relies on a multi-agent peer-to-peer system deployed on each physical resource composing the LUC infrastructure. Agents are autonomous entities that collaborate to efficiently use the LUC resources. In our context, efficiency means that a good trade-off is found between satisfying user's expectations, ensuring reliability, reactiveness as well as availability of the services while limiting the energy consumption of the system and providing scalability. We propose thus to leverage P2P techniques, that allow self-* properties, such as self-adaptation and self-repairing of overlays. To reduce the management complexity as well as the design and the implementation of the different mechanisms that are mandatory, we strongly support to use micro-kernel concepts. Such an approach should enable to design and implement services at higher level while leveraging P2P mechanisms at the lower ones. Furthermore, to address the different objectives and reduce the management complexity, we also underline that self-* properties should be present at every level of the system. We think that relying on a multi-agent peer-to-peer sys-

tem is the best solution to cope with the scale as well as the network disconnections that may create temporary partitions in a LUC platform.

In DISCOVERY, each agent has two purposes: (i) maintaining knowledge base on the LUC platform composition (ii) ensuring the correct execution of the VEs. Concretely, the knowledge base will consist of overlays that will be used for the autonomous management of the VEs life cycle. This includes the configuration, deployment and monitoring of VEs as well as the dynamic allocation or relocation of VMs to adapt to changes in VEs requirements and physical resources availability. To this end, agents will need to rely on dedicated mechanisms that can be classified as follows:

- Mechanisms related to physical resource localization and monitoring,
- Mechanisms related to VEs management,
- Mechanisms related to the VM images management,
- Mechanisms related to reliability,
- Mechanisms related to security and privacy.

4.2 Resource Localization and Monitoring Mechanisms

Keeping in mind that DISCOVERY should be designed in a fully distributed way, most of the mechanisms should build on top of overlays in order to abstract changes that occur at the physical levels. The specific requirements of this platform will lead to develop a particular kind of overlays, having minimalism in mind.

More concretely, the first step is to design, at the lowest level, a first overlay layer intended to abstract out the details of the physical routes and computing utilities, while satisfying the basic topology requirements needed – locality and availability. This overlay needs to enable the communications between any two nodes in the platform. While overlay computing has been extensively studied over the last decade, we emphasize here on minimalism, and one particular feature with regard to the envisioned platform described before: retrieving physically close nodes from a given starting node.

Giving nodes a position

The physical network's connections are considered as arbitrary. We choose to rely on the Vivaldi protocol [18]. Vivaldi is a distributed algorithms assigning coordinates in the plane to nodes of a distributed systems, each node being equipped with a *view* of the network, *i.e.*, a set of nodes it knows. This view is initially assumed as arbitrary. Coordinates obtained by a node reflects its *position* in the network, *i.e.*, close nodes in the network are given close coordinates in the plane. This is achieved by every node periodically checking the round trip time between itself and another node (from its view, a different one at each step) and adapting its distance (by changing its coordinates) with this node in the plane accordingly.

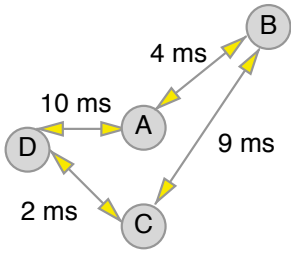

Fig. 2 Vivaldi plot before updating positions.
Each node ping other nodes. Each node maintains a map of distance.

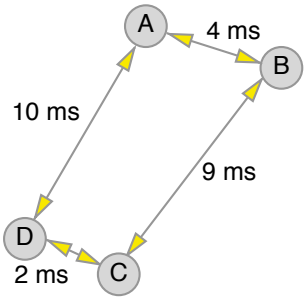

Fig. 3 Vivaldi plot after updating positions.
The computed positions of other nodes are periodically updated.

Note that a global accurate positioning of nodes can be obtained if nodes have a few long-distance nodes in their view [18]. These long distance links can be easily maintained through a simple gossip protocol.

Searching for close nodes

Once this map is achieved (each node its coordinates), we are able to decide whether two nodes are close by calculating their distance. However, the view of each node does not *a priori* contain its closest nodes. We need additional mechanisms to locate a set of nodes that are close to a given initial node – Vivaldi gives a *location* to a node, but not a neighborhood. To achieve this, we use a modified distributed version of the classic Dijkstra's algorithms used to find the shortest path between two nodes in a graph. The goal is to build a *spiral*[4] interconnecting the nodes in the plane that are closest to a given initial node. Let us consider our starting point is node s. The first step is to find a node to build a two-node spiral with s. Such a node is sought in the current view of s by selecting the node, say a, having the smallest distance with s. By contacting a, s then sends its view to a, s stores a as its successor in the spiral, and a adds s as its predecessor in the spiral. Then s forwards its view to a. a then creates a new view by keeping the n node which are closest to a in both s and a's views. This last view is then referred to as the *spiral view* and is intended to contain a set of nodes in which to find the next step of the spiral. Then a restarts the same process: Among the spiral view, it chooses the node with the smallest distance to s, say b, and adds it in the spiral – a becomes the predecessor of b and b becomes the successor of a. Then, the spiral view is sent to b which updates it with the nodes

[4] The term *spiral* is here a misuse of language, as there is no guarantee that the graph actually drawn in the plane does not contain crossing edges. The only guarantee is that when following the path constructed, the nodes are always further from the initial node.

it has in its own view. The process is repeated until we consider we have gathered enough nodes with regard to the requesting application.

One risk to be mitigated is not to blocked by a spiral view containing only nodes that are already in the spiral. However, this problem can be easily addressed by forcing the presence of a few long distance nodes whenever it is 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 starting point node *s* may not be included. The only property ensured is that one step forward the path built always takes us further from the starting point node.

To improve the *quality* of the spiral, *i.e.*, reduce the average distance between the nodes it comprises and the initial node, we add a learning mechanism coming with no extra communication cost, as it does not require any extra message. When a node is contacted for becoming the next node in one spiral, and receives the associated spiral view, it can also keep the nodes that are closest to itself, thus potentially increasing the quality of a future spiral construction.

Routing

In the context of a LUC infrastructure, one crucial feature is to be able to locate an existing VM. Having the same strategy consisting in improving the performance of the overlay based on the application's activity, we envision a routing mechanism which will be improved by past routing requests. Thanks to the spiral mechanism described above, a node is able to contact its neighboring nodes to start the path of a message.

This initial routing mechanism can be very expensive, as the number of hops can be linear in the size of the network. From previous communications, one is able to memorize long links to different locations of the network. Consequently,

From each routing request, the source of the request and each node on the path to the destination is able to learn long links, which will significantly reduce the number of hops of future requests. We currently study the amount of requests needed to approach a logarithmic routing complexity.

More generally, we work on the estimation of the application's activity required to (i) guarantee the constant efficiency of the overlay, (ii) converge, started from a random configuration, to a fully-efficient overlay network.

4.3 VEs Management Mechanisms

In the DISCOVERY system, we define a VE as a set of VMs that may have specific requirements in terms of hardware, software and also in terms of placement: some VMs must be on the same node/site in order to cope with performance objectives while others should not be collocated in oder to ensure high-availability criteria [27]. As operations on a VE may occur in any place from any location, each agent should provide the capability to configure and start a VE, to suspend/resume/stop it, to relocate some of its VM if need be or simply to retrieve the location of a particular VE. Most of these mechanisms are integrated on current UC platforms. However as mentioned, they should be revisited to correctly run on the infrastructure we target (*i.e.* in terms of scalability, resiliency and reliability). To this aim, the DISCOVERY system relies on the aforementioned P2P mechanisms. As a first example, placing the VMs of a VE requires to be able to find available nodes that fulfills the VM needs (in terms of resource needs as well as placement constraints). Such a placement can start locally, close to the client application requesting it, *i.e.*, in its local group. If no such node is found, simple navigation ensures that the request will encounter a bridge eventually, leading to the exploration of further nodes. This navigation goes on until one sufficiently available node is found. A similar process is performed by the mechanism in charge of dynamically controlling and adapting the placement of VEs during their lifetime. For instance, in order to ensure particular needs of a VM, it can be necessary to relocate other ones. According to the predefined constraints of VEs, some VMs might be relocated on far nodes while other would prefer to be suspended. Such a mechanism has been deeply studied and validated in the DVMS mechanism [2, 43]. DVMS (Distributed Virtual Machines Scheduler) enables to dynamically schedule a significant number of VMs throughout a large-scale distributed infrastructure while guaranteeing VM resource expectations.

A second example regards the configuration of the network elements of a VE. Although it might look simple, assigning the right IPs to each VM as well as maintaining the intra-connectivity of a VE becomes a bit more complex than in the case of a single network domain, *i.e.* a mono-site deployment. Keeping in mind that a LUC infrastructure is by definition spread WANwide, a VE can be hosted between distinct network domains during its lifetime. No solution has been chosen yet. Our first investigations led us to leverage techniques such as the IP over P2P project [22]. However, the definition of software network becomes more and more important. Investigating proposals such as the Open vSwitch project [41] looks a promising direction to solve such issue.

4.4 VM Images Management

In a LUC infrastructure, VM images could be deployed in any place from any other location, but being in a large-scale, heterogeneous and widely spread environment makes the management of VM images more difficult than more conventional cen-

tralized repositories. At coarse grain, the management of the VM images should be (i) consistent with regards to the location of each VM throughout the DISCOVERY infrastructure and (ii) reachable in case of failures. The envisioned mechanisms dedicated to the management of the VM images have been classified into two sub-classes. First, it will be mandatory to deliver appropriate mechanisms to efficiently upload and replicate VM images among a high number of nodes in order to ensure efficiency as well as reliability. Second, the DISCOVERY system should include specific mechanisms devoted to the scheduling of the VM image transfers. Advanced policies are important to improve the efficiency of each transfer that may occur either at the boot time or during VM relocations.

Regarding storage and replication mechanisms, an analysis of an IBM Cloud concludes that a fully distributed model using P2P technology is not the best choice to manage VM images as the number of instances of the same VM image is rather small [39]. However, central or hierarchical solutions are not suited for the infrastructure we target. Consequently, an augmented P2P solution working with replicas and deduplication will have to be investigated in order to provide more reliability, speed, and scalability to the system. For example, analyzing different VM images shows that at least 30% of the image is shared between different VMs. This 30% can become a 30% space reduction, a 30% increased reliability or a 30% of speed increase. Depending on the situation, we should decide to go from one scenario to another.

Regarding the scheduling mechanisms, it has been shown that a storage system with VMs being used by I/O intensive tasks can increase boot time from 10 to 240 seconds [47]. Some actions like providing the image chunks needed to boot first [48], defining a new image format, and pausing the rest of the I/O operations, can provide a performance boost and limit the overhead that is still observed in commercial Clouds [33].

More generally, the amount of data related to VM images is significant. Actions involving data should be aware of their implications on metrics like (but not limited to): energy efficiency, bandwidth, reliability, proximity, and hardware usage. The scheduler could also anticipate actions like moving images when the load is low or the energy is cheaper.

4.5 Reliability Mechanisms

In a LUC, failures will be much more frequent than in actual UC platforms. Furthermore, since resources could be highly distributed, the expected mean time to repair failed equipments might be much larger than in other UC platforms. For all these reasons, a set of dedicated mechanisms should be designed in order to provide fully transparent failure management with minimum downtime.

Ensuring the high availability of the DISCOVERY system requires being able to autonomously restart any service by relocating it on a healthy agent each time it is mandatory. To avoid losing or corrupting important information regarding the state

of the system, a Cassandra-like system [29] is required to provide a reliable and highly available back-end for stateful services.

Regarding VEs reliability, a first level of fault tolerance can be provided by leveraging VMs snapshotting capabilities. Periodical snapshots will allow restarting the VE from its last snapshot in the event of a failure. Performing VM snapshotting in a large-scale, heterogeneous, and widely spread environment is a challenging task. However, we believe that adapting recently proposed ideas [38] in this field would allow us to provide such a feature.

Snapshotting is not enough for services that should be made highly available, but a promising solution is to use VM replication [40]. To implement VM replication in a WAN, solutions to optimize synchronizations between replicas [24, 44] should be investigated. Also, we think that a LUC has the major advantage over other UC platforms, that it is tightly coupled with the network infrastructure. As such, we can expect *low* latencies between nodes and so, to be able to provide strong consistency between replicas while achieving acceptable response time for the replicated services.

Reliability techniques will of course make uses of the overlays for resource localization and monitoring. Replicated VMs should be hosted on nodes that have a low probability to fail simultaneously. Following the previously defined overlay structure, this can be done through a navigating scheme where at least one bridge is encountered. Monitoring a replica can then be done by having a *watcher* in the same local group as the replica.

4.6 Security and Privacy Mechanisms

To be successful, DISCOVERY needs to provide mechanisms and methods to construct trust relationships between resource providers. Trust relationships are known to be complex to build [34]. Providing strong authentication, assurance and certification mechanisms for providers and users is required but it is definitely not enough. Trust covers socio-economic aspects that must be covered but are out of the scope of this chapter. The challenge is to provide a trusted DISCOVERY base.

As overlays are fundamentals for all DISCOVERY mechanisms, the first challenge is to ensure that they are not compromise. Recent advances [16] might enable to tackle such concerns.

The second challenge will consist in providing end-users with a way to define their own security policies and to ensure that such policies are enforced. The expression of these requirements itself is a complex task. The challenge is to improve the current trade-off between security and usability. To ease the expression of these policies, we are currently defining a domain specific security language that defines high-level security requirements [12, 30]. These security policies will be enforced in a decentralize manner by distributed security decision and enforcement points (SDEPs) during the whole lifetime of the VE. Implementing such SDEP mechanisms in a distributed fashion will require to conduct specific research as they are

currently only prospective proposals for classic UC infrastructures [9, 46]. The challenge will consist in investigating whether such proposals can be adapted to the LUC infrastructure by leveraging appropriate overlays.

4.7 Toward a First Proof-of-Concept

The first prototype is under heavy implementation. It aims at delivering a simple mock-up for integration/collaboration purposes. Following the coarse-grained architecture described in the previous sections, we have started to identify all the components participating in the system, their relationships, as well as the resulting interfaces. Conducting such a work now is mandatory to move towards a more complete as well as more complex system.

To ensure a scalable and reliable design, we chose to rely on the use of high-level programming abstractions, more precisely, we are using distributed complex event programming [28] in association with actors model [6]. This enables to easily switch between a push and a pull oriented model according to our needs.

Our preliminary studies showed that a common building block is mandatory to handle resilience concerns in all components. Concretely, it corresponds to a mechanism in charge of throwing notifications that are triggered by the low level network overlay each time a node joins or leaves it. Such a mechanism makes the design and the development of higher building blocks easier as they do not have to provide specific portions of code to monitor infrastructure changes.

This building block has been designed around the *Peer Actor* concept (see Figure 4 and 5). The *Peer Actor* serves as an interface between higher services and the communication layer. It provides methods that enable to define the behaviors of a service when a resource joins or leaves a particular P2P overlay as well as when neighbors change. Considering that several overlays may co-exist through the DISCOVERY system, the association between a *Peer Actor* and its *Overlay Actor* is done at runtime and can be changed on the fly if need be. However, it is noteworthy that each *Peer Actor* takes part to one and only one overlay at the same time. In addition to the *Overlay Actor*, a *Peer Actor* is composed of a *Notification Actor* that processes events and notifies registered actors. As illustrated in Figure 5, a service can use more than one *Peer Actor* (and reciprocally). Mutualizing a *Peer Actor* enables for instance to reduce the network overhead implied by the overlays' maintenance. In the example, the first service relies on a *Peer Actor* implementing a Chord overlay while the second service uses an additional *Peer Actor* implementing a CAN structure.

By such a mean, higher-level services can take the advantage of the advanced communication layers without dealing with the burden of managing the different overlays. As an example, when a node disappears, all services that have been registered as dependent on such an event are notified. Service actors can thus react accordingly to the behavior that has been specified.

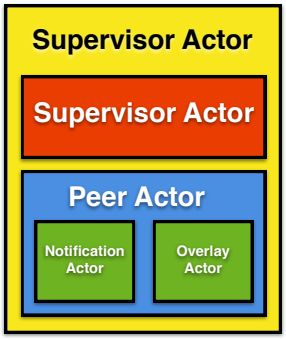

Fig. 4 The Peer Actor Model
The Supervisor actor monitors all the actors it encapsulates while the Peer actor acts as an interface between the services and the overlay.

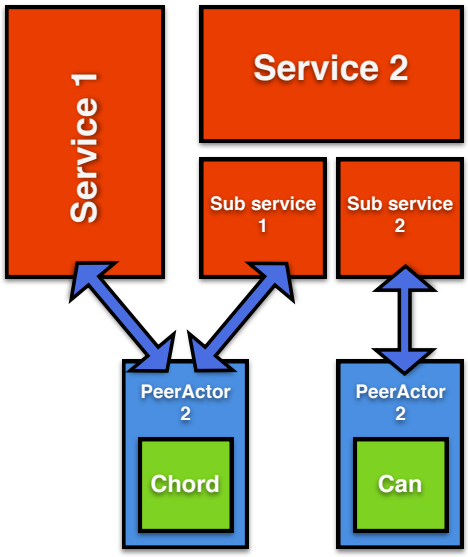

Fig. 5 A Peer Actor Instantiation
The first service relies on a *Peer Actor* implementing a Chord overlay while the second service uses an additional *PeerActor* implementing a CAN structure.

Regarding the design and the implementation of the DISCOVERY system, each service is executed inside its own actor and communicates by exchanging messages with the other ones. This ensures that each service is isolated from the others : When a service crashes and needs to be restarted, the execution of other services is not affected. As previously mentioned, we consider that at the LUC scale, failures are the norm rather than the exception, hence we decided that each actor will be monitored by a *Supervisor Actor* (see Figure 4). DISCOVERY services are under the supervision of the DISCOVERY agent: this design makes it possible to precisely define strategy that will be executed in case of service failures. This will be the way we introduce self-healing and self-organizing properties to the DISCOVERY system.

This building block has been fully implemented[5] by leveraging the SCALA/akka[6] framework.

As a proof of concept, we are implementing a first high level service in charge of dynamically scheduling VMs across a LUC infrastructure by leveraging the

[5] Code is available at: `https://github.com/BeyondTheClouds`
[6] `http://www.akka.io`

DVMS [43] proposal (see Section 4.3). The low-level overlay that is currently implemented, is a robust ring based on the Chord algorithm combined with Vivaldi positionning system: it enables services to select nodes that have low latency, so that collaboration will be more efficient.

To validate the behavior, the performance as well as the reliability of our POC, we rely first on the Simgrid [15] toolkit. Simgrid has been recently extended to integrate virtualization abstractions and accurate migration models. Simulations enable us to analyze particular situations and get several metrics that cannot be easily monitored on a real platform. Second, results obtained from simulations are then compared to real experiments on the Grid'5000 platform. Grid'5000 provides a testbed supporting experiments on various types of distributed systems (high-performance computing, grids, peer-to-peer systems, Cloud Computing, and others), on all layers of the software stack. The core testbed currently comprises 10 sites geographically spread across France. For the Discovery purpose, we developed a set of scripts that enables to deploy in a *one-click* fashion a large number of VMs throughout the whole infrastructure[10]. By deploying our POC on each node and by leveraging the VM deployment scripts, we can evaluate real scenario usages by conducting specific workloads in the different VMs. The validation of this first POC is almost completed. The resulting system will be the first to provide reactive, reliable and scalable reconfiguration mechanisms of virtual machines in a fully distributed and autonomous way. This new result will pave the way for a complete proposal of the DISCOVERY system.

5 Future Work/Opportunities

5.1 Geo-Diversification as a Key Element

The Cloud Computing paradigm is changing the way applications are designed. In order to benefit from the elasticity capability of Cloud systems, applications integrate or leverage mechanisms to provision resources, *i.e.* starting or stopping VMs, according to their fluctuating needs. The ConPaaS system [42] is one of the promising systems for elastic Cloud applications. At the same time, a few projects have started investigating distributed/collaborative way of hosting famous applications such as Wikipedia or Facebook-like systems by leveraging volunteer computing techniques. However, considering that resources provided by end-users were not reliable enough, only few contributions have been done yet. By providing a system that will enable to operate widely spread but more reliable resources closer to the end-users, the LUC proposal may strongly benefit to this research area. Investigating the benefit of locality provisioning (i.e. combining elasticity and distributed/collaborative hosting) is a promising direction for all web services that are embarrassingly distributed [17]. Image sharing system such as Google Picasa or Flickr are examples of applications where leveraging locality will enable to limit

network exchanges: Users could upload their images on a peer close to them and images would be transferred to other locations only when required (pulling vs. pushing model).

LUC infrastructures will allow envisioning a wider range of services that may answer specific SMEs requests such as data archiving or backup solutions while significantly reducing the network overhead as well as legal concerns. Moreover, it will make the deployment of UC services easier by relieving developers of the burden of dealing with multi-cloud vendors. Of course, this will require software engineering and middleware advances to easily take advantage of locality but proposing LUC OS solutions such as the DISCOVERY project is the mandatory step before investigating new APIs enabling applications to directly interact with the LUC OS internals.

5.2 Energy, a Primary Concern for Modern Societies

The energy footprint of current UC infrastructures and more generally of the whole Internet is a major concern for the society. By its design and the way to operate it, a LUC infrastructure will have a smaller impact. Moreover, the LUC proposal is an interesting way to deploy the data furnaces proposal [31]. Concretely, following the smart city recommendations (i.e. delivering efficient as well as sustainable ICT services), the construction of new districts in metropolises may take the advantage of each LUC/Network PoP in order to heat buildings while operating UC resources remotely thanks to a LUC OS. Finally, this idea might be extended by taking into account recent results about passive data centers, such as solar-powered micro-data centers[7]. The idea behind passive computing facilities is to limit has much as possible the energy footprint of major hubs and DSLAMS by taking advantage of renewable energies to power them and by using the heat they product as a source of energy. Combining such ideas with the LUC approach would allow reaching an unprecedented level of energy efficiency for UC platforms.

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 a lot of progress has already been made in cloud technologies, there are several concerns that limit the complete adoption of the Cloud Computing paradigm. In this paper, we outlined that in addition to these concerns, the actual model of UC is limited by intrinsic issues. Instead of the current trend trying to

[7] `http://parasol.cs.rutgers.edu`

cope with existing platforms and network interfaces, we proposed to take a different direction by promoting the design of a system that will be efficient and sustainable at the same time, putting knowledge and intelligence directly into the network backbone itself.

The innovative approach we introduced will definitely tackle and go beyond Cloud Computing limitations. Our objective is to pave the way for a new generation of Utility Computing that better matches the Internet structure thanks to advanced operating mechanisms. By offering the possibility to tightly couple UC servers and network backbones throughout distinct sites and operate them remotely, the LUC OS technology may lead to major changes in the design of UC infrastructures as well as in their environmental impact. The internal mechanism of the LUC OS should be topology dependent and resources efficient. The natural distribution of the nodes through the different points of presence should be an advantage, which allows to process a request according to its scale. Local requests have to be computed locally and large computations should benefit from a large number of nodes.

Finally, we believe that LUC investigations may contribute to fill the gap between the distributed computing community and the network one. This connection between these two communities has already started with the different activities around Software Defined Network and Network as a Service. This may result in the long view in a new community dealing with UC challenges where network and computational concerns are fully integrated. Such a new community may leverage the background of both areas to propose new systems that are more suitable to accommodate the needs of our modern societies.

We are of course aware that the design of a complete LUC OS and its adoption by companies and network providers require several big changes in the way UC infrastructures are managed and wide area networks operated. However we are convinced that such an approach will pave the way towards highly efficient as well as sustainable UC infrastructures, coping with heterogeneity, scale, and faults.

Acknowledgements If you want to include acknowledgments of assistance and the like at the end of an individual chapter please use the `acknowledgement` environment – it will automatically render Springer's preferred layout.

Appendix

When placed at the end of a chapter or contribution (as opposed to at the end of the book), the numbering of tables, figures, and equations in the appendix section continues on from that in the main text. Hence please *do not* use the `appendix` command when writing an appendix at the end of your chapter or contribution. If there is only one the appendix is designated "Appendix", or "Appendix 1", or "Appendix 2", etc. if there is more than one.

References

1. CloudStack, Open Source Cloud Computing.
2. Distributed VM Scheduler.
3. Nimbus is Cloud Computing for Science.
4. Open Source Data Center Virtualization.
5. The Open Source, Open Standards Cloud.
6. Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA, 1986.
7. Odlyzko Andrew. Data networks are lightly utilized, and will stay that way, 2003.
8. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4):50–58, April 2010.
9. Jean Bacon, David Evans, David M. Eyers, Matteo Migliavacca, Peter Pietzuch, and Brian Shand. Enforcing End-to-End Application Security in the Cloud (Big Ideas Paper). In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, pages 293–312, Berlin, Heidelberg, 2010. Springer-Verlag.
10. Daniel Balouek, Adrien Lèbre, and Flavien Quesnel. Flauncher and dvms – deploying and scheduling thousands of virtual machines on hundreds of nodes distributed geographically.
11. Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.
12. Mathieu Blanc, Jeremy Briffaut, Laurent Clevy, Damien Gros, Jonathan Rouzaud-Cornabas, Christian Toinard, and Benjamin Venelle. Mandatory Protection within Clouds. In Surya Nepal and Mukaddim Pathan, editors, *Security, Privacy and Trust in Cloud Systems*. Springer, 2013.
13. Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and its Role in the Internet of Things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.
14. Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.
15. Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, pages 126–131, 2008.
16. Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, December 2002.
17. Kenneth Church, Albert Greenberg, and James Hamilton. On Delivering Embarrassingly Distributed Cloud Services. In *HotNets*, 2008.
18. Frank Dabek, Russ Cox, M. Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 15–26, 2004.
19. Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220. ACM, 2007.
20. I. Foster. Globus Online: Accelerating and Democratizing Science Through Cloud-Based Services. *Internet Computing*, 2011.
21. I. Foster and C. Kesselman. *Advances in Parallel Computing - Volume 20: High Performance Computing: From Grids and Clouds to Exascale*, chapter The History of the Grid. IOS Press, 2011.

22. Arijit Ganguly, Abhishek Agrawal, P. Oscar Boykin, and Renato Figueiredo. IP over P2P: Enabling Self-Configuring Virtual IP Networks for Grid Computing. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 49–49, Washington, DC, USA, 2006. IEEE Computer Society.
23. Jodie Van Horn Gary Cook. How dirty is your data ?, 2013.
24. Balazs Gerofi and Yutaka Ishikawa. Enhancing TCP Throughput of Highly Available Virtual Machines via Speculative Communication. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*, VEE '12, pages 87–96, New York, NY, USA, 2012. ACM.
25. Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2008.
26. IEEE 802.3 Ethernet Working Group. IEEE 802.3TM Industry Connections Ethernet Bandwidth Assessment, July 2012.
27. Fabien Hermenier, Julia Lawall, and Gilles Muller. BtrPlace: A Flexible Consolidation Manager for Highly Available Applications. *IEEE Transactions on Dependable and Secure Computing*, 99(PrePrints), 2013.
28. Christian Janiesch, Martin Matzner, and Oliver Müller. A Blueprint for Event-Driven Business Activity Management. In *Proceedings of the 9th international conference on Business process management*, BPM'11, pages 17–28. Springer-Verlag, 2011.
29. Avinash Lakshman and Prashant Malik. Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.
30. Arnaud Lefray, Eddy Caron, Jonathan Rouzaud-Cornabas, Huaxi (Yulin) Zhang, Aline Bousquet, Jeremy Briffaut, and Christian Toinard. Security-Aware Models for Clouds. In *Poster Session of IEEE Symposium on High Performance Distributed Computing (HPDC)*, June 2013.
31. Jie Liu, Michel Goraczko, Sean James, Christian Belady, Jiakang Lu, and Kamin Whitehouse. The Data Furnace: Heating up with Cloud Computing. In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, HotCloud'11, pages 15–15, 2011.
32. Scott Lowe. *Mastering VMware vSphere 5*. Wiley Publishing Inc., Oct. 2011.
33. Ming Mao and Marty Humphrey. A Performance Study on the VM Startup Time in the Cloud. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 423–430. IEEE Computer Society, 2012.
34. Keith W. Miller, Jeffrey Voas, and Phil Laplante. In trust we trust. *Computer*, 43:85–87, October 2010.
35. R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer*, 45(12):65–72, 2012 2012.
36. Christine Morin. XtreemOS: A Grid Operating System Making your Computer Ready for Participating in Virtual Organizations. In *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, ISORC '07, pages 393–402. IEEE Computer Society, 2007.
37. Paul Murray, Azime Sefidcon, Rebecca Steinert, Volker Fusenig, and Jorge Carapinha. Cloud Networking: An Infrastructure Service Architecture for the Wide Area, July 2012.
38. Bogdan Nicolae, John Bresnahan, Kate Keahey, and Gabriel Antoniu. Going Back and Forth: Efficient Multideployment and Multisnapshotting on Clouds. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 147–158, New York, NY, USA, 2011. ACM.
39. Chunyi Peng, Minkyong Kim, Zhe Zhang, and Hui Lei. VDN: Virtual Machine Image Distribution Network for Cloud Data Centers. In *INFOCOM, 2012*, pages 181–189, March 2012.
40. Darko Petrovic and Andre Schiper. Implementing virtual machine replication: A case study using xen and kvm. In *Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pages 73–80, 2012.
41. Ben Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending Networking into the Virtualization Layer. In *ACM HotNets*, October 2009.

42. Guillaume Pierre and Corina Stratan. ConPaaS: A Platform for Hosting Elastic Cloud Applications. *IEEE Internet Computing*, 16(5):88–92, September 2012.
43. Flavien Quesnel, Adrien Lèbre, and Mario Südholt. Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS. *Concurrency and Computation: Practice and Experience*, June 2012.
44. Shriram Rajagopalan, Brendan Cully, Ryan O'Connor, and Andrew Warfield. SecondSite: Disaster Tolerance as a Service. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*, VEE '12, pages 97–108. ACM, 2012.
45. B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The Reservoir Model and Architecture for Open Federated Cloud Computing. *IBM J. Res. Dev.*, 53(4):535–545, July 2009.
46. Ravi Sandhu, Raj Boppana, Ram Krishnan, Jeff Reich, Todd Wolff, and Josh Zachry. Towards a Discipline of Mission-Aware Cloud Computing. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pages 13–18, 2010.
47. Tingxi Tan, Rob Simmonds, Bradley Arlt, Martin Arlitt, and Bruce Walker. Image Management in a Virtualized Data Center. *SIGMETRICS Perform. Eval. Rev.*, 36(2):4–9, 2008.
48. Chunqiang Tang. FVD: A High-Performance Virtual Machine Image Format for Cloud. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 18–18. USENIX Association, 2011.
49. Han Zhao, Ze Yu, Shivam Tiwari, Xing Mao, Kyungyong Lee, David Wolinsky, Xiaolin Li, and Renato Figueiredo. CloudBay: Enabling an Online Resource Market Place for Open Clouds. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, pages 135–142. IEEE Computer Society, 2012.