Beyond The Clouds, 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, Cédric Tedeschi

1 Context and Motivations

The success of Cloud Computing has driven the advent of Utility Computing (UC). However, Cloud Computing is a victim of its own success: In order to answer the escalating demand for computing resources, Cloud Computing providers must build data centers (DCs) of ever-increasing size. As a consequence, besides facing the well-known issues of large-scale platforms management, large-scale DCs have now to deal with energy considerations that limit the number of physical resources that one location can host.

Instead of investigating alternative solutions that could tackle the aforementioned concerns, the current trend consists in deploying larger and larger DCs in few strategic locations presenting energy advantages. For example, Western North Carolina, USA, is an attractive area due to its abundant capacity of coal and nuclear power brought about the departure of the textile and furniture industry [23]. More recently, several proposals suggested 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 electricity consumption [25].

1.1 Inherent Limitations 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 pizzas at the edge of the polar circle, for instance, leads to important overheads since most of the users are *a priori* 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]

The two first points could be partially tackled by hybrid or federated Cloud solutions [9], that aim at extending the resources available on one Cloud with those of another one; however, the third one requires a disruptive change in the way UC resources are managed.

Another issue is that, according to some projections of a recent IEEE report [26], the network traffic continues to double roughly every year. Consequently, 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 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 a complementary solution to hybrid platforms in order to reduce the overhead of network exchanges. However, operating multiple small DCs breaks somehow the idea of mutualization in terms of physical resources and administration simplicity, making this approach questionable.

1.2 Ubiquitous and Oversized Network Backbones

One way to partially solve the mutualization concern enlightened by the defenders of large-scale DCs is to directly deploy the concept of micro/nano DCs upon the Internet backbone. People are (and will be) more and more surrounded by computing resources, especially those in charge of interconnecting all IT equipments. Even though these small and medium-sized facilities include resources that are barely used [8, 12], they can hardly be removed (*e.g.* routers). Considering this important aspect, we claim that a new generation of UC platforms can be delivered by leveraging existing network centers, 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 able to mutual-

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

ize resources that are mandatory to operate network/data centers while delivering widely distributed UC platforms able to better match the geographical dispersal of users. Figure 1 allows to better capture the advantages of such a proposal. It shows a snapshot of the network weather map of RENATER[2], the backbone dedicated to universities and research institutes in France. It reveals several important points:

- As mentioned before, most of the resources are underutilized (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 was deployed and is renewed to match the demand: The density of points of presence (PoP, *i.e.* a small or medium-sized network center) 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 was designed to avoid disconnections, since 95% of the PoPs can be reached by at least two distinct routes.

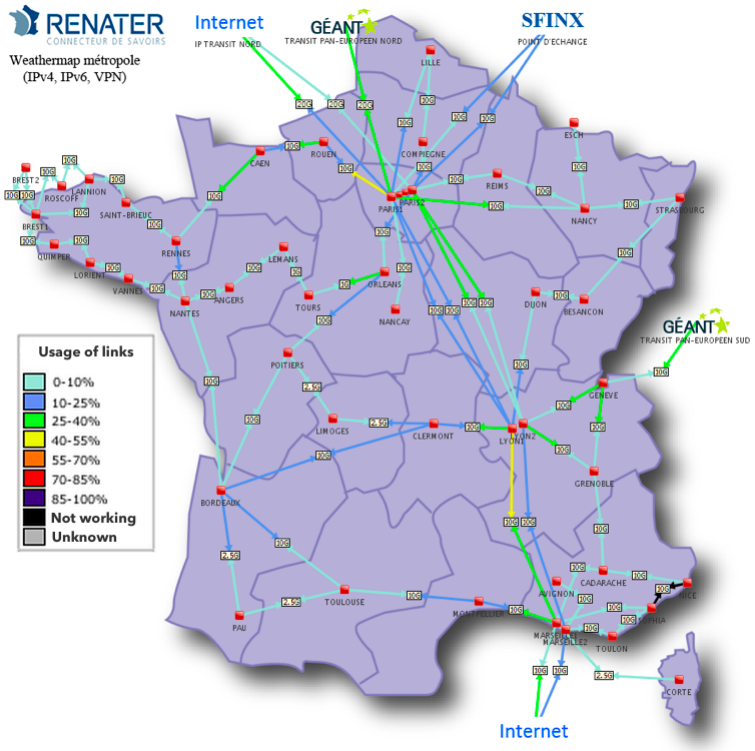

Fig. 1 The RENATER Weather Map on May 2013, the 27th, around 4PM. Each red square corresponds to a particular point of presence (PoP) of the network. The map is available in real-time at: http://www.renater.fr/raccourci

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

1.3 Locality-based Utility Computing

This chapter aims at introducing locality-based UC (LUC) infrastructures, a new generation of UC platforms that solves inherent limitations of the Cloud Computing paradigm 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 provide appropriate infrastructures, *i.e.*, 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.* those 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 PoP with a number of servers dedicated to hosting virtual machines (VMs). Because it is natural to assume that the network traffic and UC demands are 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 altogether the energy footprint of the network, the impact on latency and the congestion phenomena that may occur on critical paths (for instance Paris and Marseille on RENATER).

From the software point of view, the main challenge is to design a comprehensive distributed system in charge of turning a complex and diverse network of resources into a collection of abstracted computing facilities that is both reliable and easy to operate.

The design of the *LUC Operating System* (OS), an advanced system being able to unify many UC resources distributed on distinct sites, would enable Internet service providers (ISPs) and other institutions in charge of operating a network backbone 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 in each center of their backbone.

In addition to considering *locality* as a primary concern, the novelty of the LUC OS proposal is to consider the VM as the basic object it manipulates. Unlike existing research on distributed operating systems designed around 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 virtual 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.

1.4 Chapter Outline

Section 2 describes the key objectives of a LUC OS and the associated challenges. Section 3 explains why our vision differs from current 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 many mechanisms. Trying to identify all of them and establishing how they interact is an on-going work (see Section 4). However, we have pointed out the following key 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. These resources are small or medium-sized computing facilities and may become highly volatile according to the network disconnections.
- **Reactivity:** To deal with the dynamicity of the infrastructure, a LUC OS should swiftly handle events that require performing particular operations, either on virtual or on physical resources. This has to be done with the objective of maximizing the system utilization while meeting the quality of service (QoS) expectations of VEs. Some examples of operations that should be performed as fast as possible include (i) the reconfiguration of VEs over distributed resources, sometimes spread across wide area networks, or (ii) the migration of VMs, while preserving their active connections.
- **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 keep working despite node failures and network disconnections (LUC OS robustness) and to (ii) provide snapshotting as well as high availability mechanisms for VEs (VM robustness).
- **Sustainability:** Although the LUC approach would reduce the energy footprint of UC services by minimizing the cost of the network, 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 making an optimal usage of each source of

energy. To achieve such an objective, the LUC OS should take account of 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.).

- **Security and Privacy:** Similarly to resiliency, security, and privacy issues affect the LUC OS itself and the VEs running on it. Regarding the LUC OS, the goals are to (i) create trust relationships between different locations, (ii) secure the peer-to-peer layers, (iii) include security and privacy decision and enforcement points in the LUC OS and (iv) make them collaborate through the secured peer-to-peer layers to provide end-to-end security and privacy. Regarding the VEs, users should be able to express their requirements in terms of security and privacy; these requirements would then be enforced by the LUC OS.

In addition to the aforementioned objectives, working on a virtual infrastructure requires to deal with the management of VM images. Managing VM images in a distributed way across a wide area network (WAN) is a real challenge that will require to adapt state-of-the-art techniques related to replication and deduplication. Also, the LUC OS must take into account VM images location, for instance (i) to allocate the right resources to a VE or (ii) to prefetch VM images, to improve deployment performance or VM relocations.

Finally, one last scientific and technical challenge is the lack of a global view of the infrastructure. Maintaining a global view would indeed limit the scalability of the LUC OS, which is inconsistent with our objective to manage large-scale geographically distributed systems. Therefore, we claim that the LUC OS should rely on decentralized and autonomous mechanisms, that can match and adapt to the volatile topology of the infrastructure. Several decentralized mechanisms are already used in production on large-scale systems; for instance, Amazon relies on the Dynamo service [19] to create distributed indexes and recover from data inconsistencies; moreover, Facebook uses Cassandra [30], a massive scale structured store that leverages peer-to-peer techniques. 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 trends 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 interface standardization for supporting cooperation and resource sharing, their propositions are incremental improvements of existing UC models. Recent investigations on hybrid Clouds and Cloud federation are comparable in some ways to previous works done on Grids, since the purpose of a Grid middleware is to interact with each resource management system composing the Grid [15, 47, 52].

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 with regard to previous works on Grid systems. More concretely, this project investigates new network technologies to provide end-users of hybrid/federated Clouds with the possibility to configure and virtually operate the network backbone interconnecting the different sites they use [38].

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) [14]. 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 points of presence of backbones and operated through a unified system, the LUC OS. As far as we know, the only system that investigated whether a widely distributed infrastructure can be operated by a single system, was the XtreemOS Project [37]. Although this project shared 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 kind of distributed OS, manipulating VEs instead of processes and considering locality as a primary concern. Referred to as a LUC OS, such a system will include most of the mechanisms that are common to current UC management systems [2, 4, 5, 6, 33, 36]. However, each of them will have to be rethought in order to leverage peer-to-peer algorithms. While largely unexplored for building operating systems, peer-to-peer/decentralized mechanisms have the potential to achieve the scalability required to manage LUC infrastructures. Using this technology for establishing the base mechanisms of a massive-scale LUC OS will be a major breakthrough from current static, centralized or hierarchical management solutions.

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

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 framework to manage 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 with one another to efficiently use the LUC resources. In our context, efficiency means that a good trade-off is found between users' expectations, reliability, reactivity and availability, while limiting the energy consumption of the system and providing scalability.

In DISCOVERY, each agent has two purposes: (i) maintaining a knowledge base on the composition of the LUC platform and (ii) ensuring the correct execution of VEs. 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 rely on dedicated mechanisms related to:

- The localization and monitoring of physical resources,
- The management of VEs,
- The management of VM images,
- Reliability,
- Security and privacy.

4.2 Resource Localization and Monitoring Mechanisms

Keeping in mind that DISCOVERY should be designed in a fully decentralized fashion, its mechanisms should be built on top of an overlay network able to abstract out changes that occur at the physical level. The specific requirements of this platform will lead to the development of a novel kind of overlay networks, based on locality and a minimalistic design. More concretely, the first step is to design, at the lowest level, an overlay layer intended to hide the details of the physical routes and computing utilities, while satisfying some basic requirements such as 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 especially on one key feature to implement a LUC OS: retrieving nodes that are geographically close to a given departure node.

Giving Nodes a Position

The initial configuration of the physical network can take an arbitrary shape. We choose to rely on the Vivaldi protocol [18]. Vivaldi is a distributed algorithm assigning coordinates in the plane to nodes of a distributed system. Each node is equipped with a *view* of the network, *i.e.*, a set of nodes it knows. This view is initially assumed as random. 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. To achieve this, each node periodically checks the round trip time between itself and another node (randomly chosen among nodes in its view) and adapts its distance (by changing its coordinates) with this node in the plane accordingly. See Figure 2 and Figure 3 for an illustration of 4 nodes (A, B, C and D) moving according to the Vivaldi protocol. A globally 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 by means of a simple gossip protocol.

Searching for Close Nodes

Once the map is achieved (each node knows 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. 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, 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 the closest ones to a given initial node.

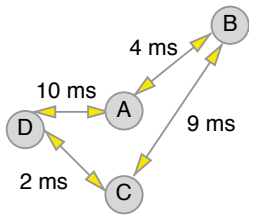

Fig. 2 Vivaldi plot before updating positions. Each node pings other nodes. Each node maintains a map of distance.

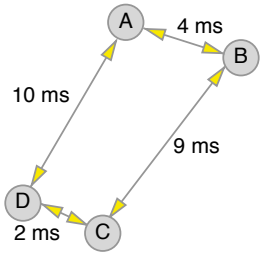

Fig. 3 Vivaldi plot after updating positions. The computed positions of other nodes have been updated.

[4] The term *spiral* is here a misuse of language, since the graph actually drawn in the plane might contain crossing edges. The only guarantee is that when following the path constructed, the nodes are always further from the initial node.

Let us consider that our initial point is a node called I. The first step is to find a node to build a two-node spiral with I. Such a node is sought in the view of I by selecting the node, say S, having the smallest distance with I. I then sends its view to S, I stores S as its successor in the spiral, and S adds I as its predecessor in the spiral. Then I forwards its view to S. S creates a new view by keeping the n nodes which are the closest to I in the views of I and S. This last view is then referred to as the *spiral view* and is intended to contain a set of nodes among which to find the next step of the spiral. Then S restarts the same process: Among the spiral view, it chooses the node with the smallest distance to I, say S', and adds it in the spiral – S becomes the predecessor of S' and S' becomes the successor of S. Then, the spiral view is sent to S' which updates it with the nodes it has in its own view. The process is repeated until we consider that enough nodes have been gathered (a parameter sent by the application).

Note that one risk is to be blocked by having a spiral view containing only nodes that are already in the spiral, leading to the impossibility to build the spiral further. 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 node s may not be included. The only property ensured is that one step forward on the built path always takes us further from the initial 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: 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 the 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 activity of the application, we envision a routing mechanism which will be improved by past routing requests. By means of the spiral mechanism, a node is able to contact its neighboring nodes to start routing a message.

This initial routing mechanism can be very expensive, as the number of hops can be linear in the size of the network. However, from previous communications, a node 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 are able to learn long links, which will significantly reduce the number of hops of future requests. We are currently studying the amount of requests needed to get close to a logarithmic routing complexity. More generally, we are working on the estimation if the activity of the application required to (i) guarantee the constant

efficiency of the overlay and to (ii) converge, starting 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: For instance, some VMs must be on the same node/site to cope with performance objectives while others should not be collocated 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 VMs if need be or simply to retrieve the location of a particular VE. Most of these mechanisms are provided by current UC platforms. However, as mentioned before, they should be revisited and leverage peer-to-peer mechanisms to correctly run on the infrastructure we target (*i.e.* in terms of scalability, resiliency and reliability).

As a first example, placing the VMs of a VE requires to be able to find available nodes that fulfill the VM needs (in terms of resource requirements 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, a simple navigation ensures that the request will encounter a bridge, leading to the exploration of further nodes. This navigation goes on until an adequate 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, to ensure the particular needs of a VM, it can be necessary to relocate other VMs. According to the predefined constraints of VEs, some VMs might be relocated on far nodes while others would prefer to be suspended. Such a mechanism has been deeply studied in the DVMS framework [3, 44]. DVMS (Distributed Virtual Machine Scheduler) is able to dynamically schedule a significant number of VMs throughout a large-scale distributed infrastructure while guaranteeing VM resource expectations.

A second example regards the networking configuration of VEs. Although it might look simple, assigning the right IP 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 single 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, software-defined networking becomes more and more important; investigating proposals such as the Open vSwitch project [42] looks promising to solve such an issue.

4.4 VM Images Management

In a LUC infrastructure, VM images could be deployed in any place from any other location. However, being in a decentralized, large-scale, heterogeneous and widely spread environment makes the management of VM images more difficult than with conventional centralized repositories. At coarse grain, (i) the management of the VM images should be consistent with regard to the location of each VM in the DISCOVERY infrastructure and (ii) each VM image should remain reachable or at least recoverable in case of failures. The envisioned mechanisms to manage VM images have been classified into two categories. First, some mechanisms are required to efficiently upload VM images and replicate them across many nodes, to ensure efficiency as well as reliability. Second, other mechanisms are needed to schedule VM image transfers. Advanced policies are important to improve the efficiency of each transfer that may occur either during the first deployment of a VM or during its relocations.

Regarding storage and replication mechanisms, an analysis of an IBM Cloud concludes that a fully distributed approach using peer-to-peer technology is not the best choice to manage VM images, since the number of instances of the same VM image is rather small [40]. However, central or hierarchical solutions are not suited for the infrastructure we target. Consequently, an improved peer-to-peer solution working with replicas and deduplication has to be investigated, 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 [29]. This 30% can become a 30% reduction in space, or a 30% increase in reliability or in transfer speed. Depending on the situation, we should decide to go from one scenario to another.

Regarding the scheduling mechanisms, a study showed that VM boot time can increase from 10 to 240 seconds when multiple VMs running I/O intensive tasks use the same storage system [50]. Some actions, like providing the image chunks needed to boot first [51], 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 [34].

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

4.5 Reliability Mechanisms

Although we can expect that the frequency of failures on LUC resources should be similar than in current UC platforms, it is noteworthy to mention that the expected mean time to repair failed equipments might be much higher since resources will

be highly distributed. For these reasons, specific mechanisms should be designed to manage failures transparently with a minimum downtime.

Ensuring the high availability of the DISCOVERY system requires being able to autonomously relocate and restart any service on a healthy node in case of failure. Moreover, a Cassandra-like framework [30] is required to avoid losing or corrupting information belonging to stateful services, since it provides a reliable and highly available back-end.

Regarding VEs reliability, a first level of fault tolerance can be provided by leveraging periodical VM snapshotting capabilities. In case of failure, a VE can be restarted from its latest snapshot. Performing VM snapshotting in a large-scale, heterogeneous, and widely spread environment is a challenging task. However, we believe that adapting ideas that were recently proposed in this field [39] 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 [41]. To implement VM replication in a WAN, solutions to optimize synchronizations between replicas [24, 45] should be investigated. Also, we think that a LUC infrastructure has a major advantage over other UC platforms, since it is tightly coupled with the network infrastructure. As such, we can expect *low* latencies between nodes, that would enable us to provide a strong consistency between replicas while achieving acceptable response time for the replicated services.

Reliability techniques will of course make use 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 navigation 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 [35]. 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 addressed 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, another challenge is to ensure that they are not compromised. Recent advances [16] might enable to tackle such concerns.

The third challenge will consist in (i) providing end-users with a way to define their own security and privacy policies and (ii) ensuring that these policies are enforced. The expression of these policies itself is a complex task, since it requires to improve the current trade-off between security (and privacy) and usability. To

ease the expression of these policies, we are currently designing a domain specific language to define high-level security and privacy requirements [13, 31]. These policies will be enforced in a decentralized manner, by distributed security and privacy decision and enforcement points (SPDEPs) during the lifetime of the VEs. Implementing such SPDEP mechanisms in a distributed fashion will require to conduct specific research, since currently there are only prospective proposals for classic UC infrastructures [10, 48]. Therefore, we need to investigate 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 development. 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 the actors model [7]. This enables us to easily switch between a push and a pull oriented approach depending on our needs.

Our preliminary studies showed that a common building block is mandatory to handle resiliency 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 Figure 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 peer-to-peer overlay as well as when neighbors change. Considering that several overlays may co-exist in 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 maintenance of the overlays. In the example, the first service relies on a *Peer Actor* implementing a Chord overlay [49], while the second service uses an additional *Peer Actor* implementing a CAN structure [46].

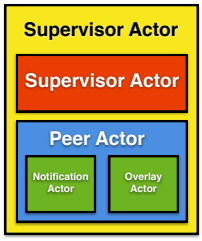

Fig. 4 The *Peer Actor* Model. The *Supervisor Actor* monitors all the actors it encapsulates while the *Peer Actor* acts as an interface between the services and the overlay.

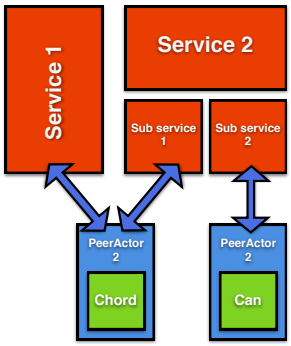

Fig. 5 A *Peer Actor* Instantiation. The first service relies on a *Peer Actor* implementing a Chord overlay while the second service uses an additional *PeerActor* implementing a CAN structure.

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.

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 infrastructure scale, failures are the norm rather than the exception; hence we decided that each actor would be monitored by a *Supervisor Actor* (see Figure 4). DISCOVERY services are under the supervision of the DISCOVERY agent: This design allows to precisely define a strategy that will be executed in case of service failures. This will be the way to introduce self-healing and self-organizing properties to the DISCOVERY system.

This building block has been fully implemented[5] by leveraging the Akka/Scala framework [1].

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 DVMS [44] proposal (see Section 4.3). The low-level overlay that is being currently implemented is a robust ring based on the Chord algorithm combined with the Vivaldi positioning 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 proof of concept (POC), we are performing several experiments on the Grid'5000 testbed[6] that comprises hundreds of nodes distributed on 10 computing sites that are geographically spread across France. To make experiments with DISCOVERY easier,

[5] Code is available at: `https://github.com/BeyondTheClouds`

[6] `https://www.grid5000.fr`

we developed a set of scripts that can deploy thousands of VMs throughout the whole infrastructure in a *one-click* fashion [11]. By deploying our POC on each node and by leveraging the VM deployment scripts, we can evaluate real scenario by injecting 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 elasticity capabilities 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 [43] is one of the promising systems for elastic Cloud applications. At the same time, a few projects have started investigating distributed/collaborative ways 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 systems, 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 that is 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 Internet, is a major concern for the society. Although we need to conduct deeper investigations, we clearly expect that by its design and the way to operate it, a LUC infrastructure will have a smaller impact with a better integration in the whole Internet ecosystem.

Moreover, the LUC proposal is an interesting way to deploy the data furnaces proposal [32]. Concretely, following the smart city recommendations (*i.e.* delivering efficient and sustainable ICT services), the construction of new districts in metropolises may take advantage of each LUC/Network PoP in order to heat buildings while operating UC resources remotely by means of 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 as 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 current model of UC is limited by intrinsic issues. Instead of following the current trend by trying to 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 by means of 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 mechanisms of the LUC

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

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 should be computed locally, while 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 Networking 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 are 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.

References

1. Build powerful concurrent & distributed applications more easily. `http://www.akka.io`.
2. CloudStack, Open Source Cloud Computing. `http://cloudstack.apache.org`.
3. Distributed VM Scheduler. `http://beyondtheclouds.github.io/DVMS/`.
4. Nimbus is Cloud Computing for Science. `http://www.nimbusproject.org`.
5. Open Source Data Center Virtualization. `http://www.opennebula.org`.
6. The Open Source, Open Standards Cloud. `http://www.openstack.org`.
7. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA, 1986.
8. O. Andrew. Data networks are lightly utilized, and will stay that way. *Review of Network Economics. 2(3)*, 2003.
9. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.
10. J. Bacon, D. Evans, D. M. Eyers, M. Migliavacca, P. Pietzuch, and B. 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.
11. D. Balouek, A. Lèbre, and F. Quesnel. Flauncher and DVMS – Deploying and Scheduling Thousands of Virtual Machines on Hundreds of Nodes Distributed Geographically. In the Sixth IEEE International Scalable Computing Challenge (collocated with CCGRID), Delft, The Netherlands., May 2013.
12. T. Benson, A. Akella, and D. 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.
13. M. Blanc, J. Briffaut, L. Clevy, D. Gros, J. Rouzaud-Cornabas, C. Toinard, and B. Venelle. Mandatory Protection within Clouds. In S. Nepal and M. Pathan, editors, *Security, Privacy and Trust in Cloud Systems*. Springer, 2013.
14. F. Bonomi, R. Milito, J. Zhu, and S. 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.

15. R. Buyya, R. Ranjan, and R. 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*, ICA3PP'10, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.
16. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, Dec. 2002.
17. K. Church, A. Greenberg, and J. Hamilton. On Delivering Embarrassingly Distributed Cloud Services. In *HotNets*, 2008.
18. F. Dabek, R. Cox, M. F. Kaashoek, and R. 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. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. 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. A. Ganguly, A. Agrawal, P. O. Boykin, and R. 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. J. V. H. Gary Cook. How dirty is your data ? Greenpeace International Report, 2013.
24. B. Gerofi and Y. 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. A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, Dec. 2008.
26. I. . E. W. Group. IEEE 802.3TM Industry Connections Ethernet Bandwidth Assessment, July 2012.
27. F. Hermenier, J. Lawall, and G. Muller. BtrPlace: A Flexible Consolidation Manager for Highly Available Applications. *IEEE Transactions on Dependable and Secure Computing*, 99(PrePrints), 2013.
28. C. Janiesch, M. Matzner, and O. 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. K. Jin and E. L. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 7:1–7:12, New York, NY, USA, 2009. ACM.
30. A. Lakshman and P. Malik. Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.
31. A. Lefray, E. Caron, J. Rouzaud-Cornabas, H. Y. Zhang, A. Bousquet, J. Briffaut, and C. Toinard. Security-Aware Models for Clouds. In *Poster Session of IEEE Symposium on High Performance Distributed Computing (HPDC)*, June 2013.
32. J. Liu, M. Goraczko, S. James, C. Belady, J. Lu, and K. 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.
33. S. Lowe. *Mastering VMware vSphere 5*. Wiley Publishing Inc., Oct. 2011.
34. M. Mao and M. 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.

35. K. W. Miller, J. Voas, and P. Laplante. In trust we trust. *Computer*, 43:85–87, October 2010.
36. R. Moreno-Vozmediano, R. Montero, and I. Llorente. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer*, 45(12):65–72, 2012.
37. C. 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.
38. P. Murray, A. Sefidcon, R. Steinert, V. Fusenig, and J. Carapinha. Cloud Networking: An Infrastructure Service Architecture for the Wide Area. HP Labs Tech Report - HPL-2012-111R1, July 2012.
39. B. Nicolae, J. Bresnahan, K. Keahey, and G. 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, USA, 2011. ACM.
40. C. Peng, M. Kim, Z. Zhang, and H. Lei. VDN: Virtual Machine Image Distribution Network for Cloud Data Centers. In *INFOCOM, 2012*, pages 181–189, March 2012.
41. D. Petrovic and A. 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.
42. B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending Networking into the Virtualization Layer. In *ACM HotNets*, Oct. 2009.
43. G. Pierre and C. Stratan. ConPaaS: A Platform for Hosting Elastic Cloud Applications. *IEEE Internet Computing*, 16(5):88–92, Sept. 2012.
44. F. Quesnel, A. Lèbre, and M. Südholt. Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS. *Concurrency and Computation: Practice and Experience*, June 2012.
45. S. Rajagopalan, B. Cully, R. O'Connor, and A. 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.
46. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, Aug. 2001. ACM.
47. 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.
48. R. Sandhu, R. Boppana, R. Krishnan, J. Reich, T. Wolff, and J. 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.
49. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.
50. T. Tan, R. Simmonds, B. Arlt, M. Arlitt, and B. Walker. Image Management in a Virtualized Data Center. *SIGMETRICS Perform. Eval. Rev.*, 36(2):4–9, 2008.
51. C. 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.
52. H. Zhao, Z. Yu, S. Tiwari, X. Mao, K. Lee, D. Wolinsky, X. Li, and R. 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.