Reliability Support in Virtual Infrastructures

Guilherme Koslovski*, Wai-Leong Yeow†, Cedric Westphal†, Tram Truong Huu‡,
Johan Montagnat§ and Pascale Vicat-Blanc¶
*INRIA - University of Lyon †DoCoMo USA Labs ‡University of Nice - I3S §CNRS - I3S ¶INRIA - LYaTiss
Email: guilherme.koslovski@ens-lyon.fr, wlyeow@ieee.org, cwestphal@docomolabs-usa.com, tram@polytech.unice.fr,
johan@i3s.unice.fr, pvb@lyatiss.com

***Abstract*—Through the recent emergence of joint resource and network virtualization, dynamic composition and provisioning of time-limited and isolated virtual infrastructures is now possible. One other benefit of infrastructure virtualization is the capability of transparent reliability provisioning (reliability becomes a *service* provided by the infrastructure). In this context, we discuss the motivations and gains of introducing customizable reliability of virtual infrastructures when executing large-scale distributed applications, and present a framework to specify, allocate and deploy virtualized infrastructure with reliability capabilities. An approach to efficiently specify and control the reliability at runtime is proposed. We illustrate these ideas by analyzing the introduction of reliability at the virtual-infrastructure level on a real application. Experimental results, obtained with an actual medical-imaging application running in virtual infrastructures provisioned in the experimental large-scale Grid'5000 platform, show the benefits of the virtualization of reliability.**

I. Introduction

Several proposals have been made to combine the virtualization of the computing resources with that of the network infrastructure (see for instance [1], [2], [3], [4]), delivering a provision model known as Infrastructure-as-a-Service or Cloud infrastructure services. This virtualization of the infrastructure encompasses computing, storage and networking resources, enabling the definition of confined execution environments, with a user-specified amount of virtual resources interconnected by a private virtual network. A key element of the virtualized infrastructure is to specify the reliability to be provided to the different tasks within the execution environment. Some tasks are critical and their failure would cause the system to collapse; the failure of some non-critical tasks could still significantly impact (for instance, delay) the completion of the overall effort. On the other hand, an element of a virtual infrastructure can be migrated to a different location in case of failure of the physical substrate.

One key goal is thus for the infrastructure user *to be able to specify the reliability associated with a task during the virtual infrastructure bootstrap*, and for the infrastructure provider to *transparently provide the reliability to the user* and to *effectively provision the desired reliability through the allocation of virtual back-up nodes* ready to take over in case of node failure through active state synchronization.

We present in this paper the key components to achieve this goal and to achieve reliability in the middleware. These components include a language to allow the specification of the reliability level for the different network elements; an interpretation mechanism to translate the reliability of the specified virtualized infrastructure into a provisioning of back-up resources; and an allocation mechanism to efficiently associate the reliable virtual infrastructure onto the physical resources.

These tools are designed so as to render the reliability *transparent*: an application could perform an assignment of tasks to the virtualized reliable resource, and receive in return the outcome of the tasks's execution independently of any physical node failure. The underlying physical resource providing the task might have changed, but the integrity of the virtual infrastructure is preserved. Reliability becomes a *service* provided by the infrastructure. Further, virtualizing reliability allows the use of the same node as back-up to multiple primary nodes, and thus strongly reduces the cost of providing reliability.

As a proof of concept, we implemented these tools over a large-scale distributed platform (Grid'5000 [5]) and evaluated the costs and benefits of reliability for an existing large-scale distributed application. In this example, an application highly sensitive to substrate failures, which is not able to be executed without reliability support in the presence of failures, was executed with *no modification* on its original code. A cost analysis using a simple pricing model shows that the overall cost to the application user for reserving additional resources for reliability is more than offset by the reduced execution time.

The paper is organized as follows. Section II motivates the introduction of reliability in virtual infrastructures and identifies system goals which will guide our design choices. In Section III, we discuss the issues associated with a specification language for reliability. Section IV discusses how to synchronize nodes to effect the desired reliability. In Section V, we discuss how to translate the language into a graph; Section VI then describes the tools used to allocate this graph onto the physical substrate. In Section VII, we describe an application we implemented as a proof of concept and show preliminary experimental results, as well as a cost analysis. Related works are reviewed in Section VIII. Section IX discusses our design and offers suggestions for future work.

II. Motivations and System Goals

Networking and computing infrastructures are subject to random failures of nodes and links. These failures are not rare in the case where the number of physical entities are large, especially in distributed systems. The reliability of a

system may be evaluated quantitatively and qualitatively. The Mean Time Between Failures (MTBF) is a statistical metric to determine the failure rate of the underlying infrastructure, which can be evaluated by the infrastructure's management system. Already, the impact of a node failure to a distributed application can be very different; a failed worker node amongst hundreds of others is less significant than the failure of a database server.

One approach is for the system designer to ensure reliability her/himself, by providing redundancy in the elements composing the system. However, this requires different sets of expertise: one is the expertise to design the system in order to deliver the intended application; another is to ensure that the components are integrated so as to support the desired reliability.

Furthermore, the actual reliability will depend on the physical resource upon which the system is deployed. If the application developer provides his/her own physical servers and switches, then s/he can specify the reliability of each individual element. If on the other hand the system is deployed using a virtualized infrastructure, the reliability characteristics of the physical resources might be unknown.

Since it is common for the application developer to delegate the elastic provisioning of resources to the infrastructure provider, in order to only use the proper amount of resource in the face of varying demands, the corresponding provisioning of the reliability must by the same token be delegated as well.

From the point of view of the application provider, it is easier and more flexible to specify a level of reliability and have the physical substrate provide it transparently as part of a service-level agreement. From the point of view of the physical network operator, reliability becomes a service that can be added and that can generate new revenue streams. Further, the infrastructure provider is free to manage reliability in light of his/her own constraints and optimization opportunities: a back-up might be associated to different resources from different independent applications. This multiplexing of the back-up resources provides economy of scale to the infrastructure provider.

These observations highlight a few requirements for a reliable virtualized infrastructure:

- the virtual-infrastructure user should be able to specify reliability in a flexible and expressive manner (this is discussed in more detail in Section III);
- the virtual-infrastructure provider should be able to implement reliability transparently for the user (Section IV);
- the virtual-infrastructure provider should be able to implement and allocate back-up resources efficiently (Section V and VI);
- both the virtual-infrastructure user and the physical-network provider need to see their business objectives satisfied (Section VII).

Fig. 1 summarizes the stages and requirements to provide reliable virtual infrastructures considering the application-provider specification. Our goal is to describe tools which allow to satisfy these requirements, and to deliver the efficient provisioning of reliability in a virtual infrastructure, as specified by the virtual infrastructure user.

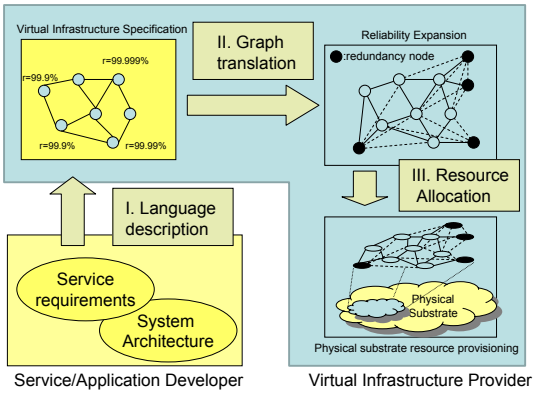


Fig. 1. The vertical integration of reliability from the user specification to the physical resource allocation.

An immediate consequence of the last requirement is that we will build upon tried-and-true existing components and technologies wherever possible, so as to provide an easier evolutionary path to implement our suggested designs.

The remainder of this paper is dedicated to translating this overarching goal into specific and practical design for the different required components.

III. Virtual Infrastructures Description

For the efficient provisioning of reliability, we use a *Virtual Private eXecution Virtual Infrastructure* (ViPXi) [1], as specified by the virtual-infrastructure user. A ViPXi is a time-limited interconnection of virtual computing resources interconnected by a virtual private network. By combining resource virtualization and network virtualization, the user of a ViPXi has the illusion s/he is using a private distributed system, while in reality s/he is using multiple systems that are part of a virtualized physical substrate.

ViPXis are dynamically-provisioned entities which can be defined and modeled to represent the application's requirements in terms of computing and communication. A descriptive language dedicated to virtual-infrastructures specification must be abstract enough and more adaptive than conventional resources-description languages and models [6], [7], [8]. In addition, it needs to combine the spatial and temporal aspects of virtual infrastructures.

New challenges coming from virtualization techniques have to be considered to complement the specification proposed by classical infrastructures. For example, the Open Virtualization Format (OVF) [9] proposes a mechanism to package and distribute software to be run in one or more virtual machines. In [10], this standard is extended to address the service-specification requirements in Cloud environments, including key performance indicators, service-elasticity rules and bounds, and required (public and private) network links.

Already, the Open Cloud Computing Interface Working Group (OCCI-WG) [11] is investigating a solution to interface

with Cloud Infrastructures exposed as services. The cloud infrastructures resources are described using a simple key-value-based descriptor format.

Unfortunately, none of the proposed languages meet all the specification requirements in terms of flexibility, expressiveness, reliability, and simplicity, required to achieve an optimal ViPXi specification and allocation [4], [12].

We extend the *Virtual eXecution Description Language* (VXDL: for more details, please refer to [13]) to enable the specification of reliable virtual infrastructures. VXDL is an XML-based language that allows the description of virtual infrastructures; more specifically, the identification and parameterization of virtual resources and groups of resources (according to their functionalities), as well as the network topology (based on the link-organization concept), using the same grammar. VXDL also introduces the internal virtual infrastructure timeline, which explores the elasticity of ViPXis, enabling application providers to specify the exact intervals when virtual resources must be provisioned.

The extension proposes identifying the required reliability level for a ViPXi. The application provider can set the reliability requirement individually for each virtual resource (nodes and links), or for a group of resources. This approach enables the composition of a ViPXi with different requirements in terms of reliability level. For instance, an application with a master-worker architecture, e.g. MapReduce, can require more reliability support for masters, and set the same reliability level for a group of workers. The example below illustrates the flexibility of the specification language. Part of a VXDL file, the example describes a group of 30 virtual nodes with a reliability specification of 99.9% (among other parameters).

```
<vxdl:vGroup id="workers" multiplicity="30">
 <vxdl:vNode id="worker">
  <vxdl:reliability>99.9%</vxdl:reliability>
  <vxdl:memory>
    <vxdl:simple>512</vxdl:simple>
    <vxdl:unit>MB</vxdl:unit>
  </vxdl:memory>
  <vxdl:cpu>
   <vxdl:cores>1</vxdl:cores>
   <vxdl:frequency>
     <vxdl:simple>1.0</vxdl:simple>
     <vxdl:unit>GHz</vxdl:unit>
   </vxdl:frequency>
  </vxdl:cpu>
 </vxdl:vNode>
</vxdl:vGroup>
```

IV. Providing Transparent Reliability

Recovering from failures has been well studied in the literature, and we can leverage existing solutions in our reliability design. This section specifies which solution we use from the available ones.

On a large-scale execution environment, the re-submission mechanism is one of the solutions used to make the application continue running when a failure is detected [14], [15]. The application's makespan is longer in this case especially when the submitted task's execution time is long. Another possible solution is to periodically save static snapshots of the entire ViPXi [16] to disk, while execution is in progress. The live snapshots are reloaded as a new submission if failures are encountered in the current execution. The application's makespan then depends on the re-submission interval and the snapshot interval, which may be long due to disk-access times.

These mechanisms, unfortunately, do not provide sufficient transparency against failures. Re-initiating or resuming applications at a later time to recover from failures will impact any time-sensitive applications. Therefore, a live protection mechanism such as Remus [17] or Kemari [18] is needed. In both Remus and Kemari, the memory state of a protected (critical) node is continuously "synchronized" with a replica (back-up node), as with checkpointing. When a failure in the protected node occurs, the back-up node can resume execution immediately, and the failover process can be made transparent to other nodes in the ViPXi. This live protection mechanism has another advantage over prior snapshotting mechanism: instead of the entire ViPXi, only the critical nodes need to be checkpointed.

The key difference between Remus and Kemari is that Kemari initiates a checkpoint only when external events occur, such as disk writing and network-packet sending, whereas Remus checkpoints at a regular interval. One important feature of Remus is that, at every checkpoint, the external output is buffered locally in the critical node until it is assured that the back-up node completes that checkpoint update. This ensures that any failover operation will be transparent to other unaffected nodes. Moreover, the protected node continues execution in parallel until the next checkpoint, thereby increasing system performance over classical lock-step checkpointing. Kemari, on the other hand, does not perform any buffering and relies on pausing the protected node to achieve the required transparency. We chose to use Remus over Kemari in our proof of concept as it provides a finer and customizable granularity between checkpoints, which can be as frequent as tens of milliseconds. As of Xen 4.0.0, Remus is included in the official Xen releases.

V. Translation of the Specification Language into a ViPXi Request

The VXDL parser [19] is a versatile tool that interprets and translates a ViPXi specification into a resource request to the physical infrastructure. Specifically, it analyzes the ViPXi specification, automatically fills in any missing components (e.g., default elements and values by some predefined templates), and translates the ViPXi into a graph representation for resource allocation (see next section). Furthermore, automated inclusion of back-up components into the graph for targeted reliabilities is added onto the VXDL parser. The procedure is described below.

A. Automatic Generation of Backup Nodes

A targeted reliability, in general, can be achieved with sufficient back-up nodes. A critical node with a low MTBF will require more back-up nodes on standby (synchronized through Remus) than another node with a higher MTBF for

the same reliability level, if physical failures are independent. As noted in [20], back-up nodes can be shared among different groups of critical nodes to minimize the total number of back-up nodes (and hence, minimal idle nodes). For example, a ViPXi has two groups of critical nodes with n_1 and n_2 critical nodes respectively, requires at least r_1 and r_2 reliability respectively, and k_1 and k_2 back-up nodes respectively. It is possible to share the back-up nodes for $n_1 + n_2$ nodes such that the total number of back-up nodes is lower than $k_1 + k_2$ provided that every back-up node is a standby for all other critical nodes. In [20], the Opportunistic Redundancy Pooling (ORP) mechanism imposes a sharing policy between groups of critical nodes such that it is possible to have $\min(k_1, k_2)$ back-up nodes so long as the reliabilities of every group is satisfied.

The VXDL parser uses ORP to evaluate the number of back-up nodes required. Since ORP assumes independent physical failures, it also generates additional physical-embedding constraints such that the physical locations of all shared back-up nodes and critical nodes validate that assumption. For example, virtual nodes may not be embedded onto the same physical host, or rack that is connected to the same switch, or power supply.

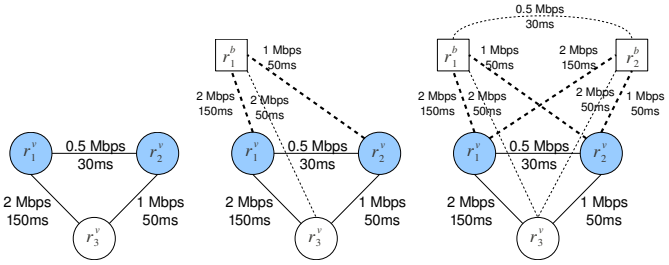
B. Backup links: consistent network topology

Failovers from the critical nodes to back-up nodes are expected to be transparent to the unaffected nodes of the ViPXi. While Remus guarantees the failover time in tens of milliseconds and consistency across the ViPXi through output buffering, consistency in the network topology has to be guaranteed through additional links to the back-up nodes. That is, failed critical nodes which are resumed at the back-up nodes must be connected to the rest of the ViPXi as described in the original specification.

To ensure failover transparency, the additional back-up links are pre-allocated (together with the ViPXi) rather than on demand after failures occurred. In the latter case, resources for back-up nodes cannot be guaranteed and, even if sufficient resources are available, undesired delays may be incurred during failover. Furthermore, active synchronization from the critical nodes to back-up nodes consume bandwidth, which has to be allocated as well.

Harary and Hayes [21] have devised methods to minimize the number of additional links required. Specifically, a new graph G' is constructed with $n+k$ nodes such that the original ViPXi is always a subgraph of G' when *any* k nodes are removed. Unfortunately, this class of solutions is infeasible in our system:

1) Guaranteeing that the ViPXi is a subgraph of G' only ensures that the graph after k node failures is isomorphic. Hence, recovering from failures may result in unaffected nodes being moved around in order to recover the ViPXi.
2) Exact solutions are found only for regular graphs such as rings, lines, square-grids and trees. For general graphs, heuristics are used [22], [23].



(a) Original ViPXi (b) Add one back-up (c) Add two back-ups

Fig. 2. The figures show the steps (from left to right) as each back-up node is added to the original ViPXi for reliability. Nodes r_1^v and r_2^v are critical nodes, and nodes r_1^b and r_2^b are back-up nodes. Backup links are reused for synchronization (in bold dotted lines), and the respective attributes are determined by the existing links in the original ViPXi.

3) The solution assumes unweighted links; adding weights on top of the solution introduces an additional layer of complexity.

As such, similar to the approach in [20], we add (i) links from nodes of the ViPXi to back-up nodes such that every back-up node is linked to neighbors of critical nodes, and (ii) links interconnecting the back-up nodes since two critical nodes which are neighbors of each other may fail simultaneously. We call the former set *first-order* back-up links and the latter set *second-order* back-up links. The second-order links are only required if two critical nodes are linked in the original ViPXi.

In addition to the results of [20], we reuse the first-order back-up links for synchronization between critical nodes and back-up nodes whenever possible. Algorithm 1 shows the procedure for the generation and reuse of first-order back-up links and their attributes: bandwidth (bw) and latency (lat). We omit details on the generation of second-order links and the remaining synchronization links that were not reused from the first-order links, since the procedure is similar to that of the first order.

Algorithm 1: Generating First-Order Backup links

```
R_b: set of back-up nodes.
C(b): set of critical nodes which uses b as a back-up node.
L_v: set of links in ViPXi.
for b ∈ R_b do
    for (i, j) ∈ L_v do
        if i ∈ C(b) then
            bw(b, j) ← bw(b, j) + bw(i, j)
            lat(b, j) ← min {lat(b, j), lat(i, j)}
            if j ∈ C(b) then
                Label (b, j) as synchronization link.
                Ensure bw and lat suffice for Remus.
```

Fig. 2 shows an example of how back-up links are generated. A new link between a back-up node and some other node is created if it is a neighbor of a critical node. Hence, in Fig. 2b, node r_1^b connects to all three nodes. Furthermore, the attributes of link (r_1^b, r_3^v) can function as links (r_1^v, r_3^v) or (r_2^v, r_3^v). Links (r_1^b, r_1^v) and (r_1^b, r_2^v) are reused for synchronization. With one more back-up node (as in Fig. 2c), the first-order back-up links of node r_2^b are the same as those of node r_1^b, and with a second-order back-up link between node r_1^b and r_2^b to function as the link (r_1^v, r_2^v) when both critical nodes fail.

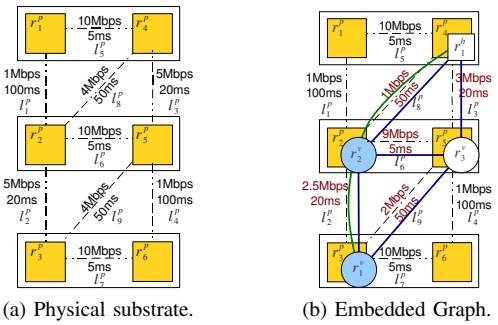
VI. ViPXi Allocation Algorithm

Translating a ViPXi to a graph representation results in a unified input to a resource allocation manager, regardless whether a ViPXi requires reliability support. This immediately translates the resource allocation problem into a graph embedding problem. However, reliability support demands tighter allocation constraints that are not present in ViPXi's that does not require any reliability guarantee. That is, virtual nodes should be mapped in a way that virtual node failures resulting from the physical substrate should be independent. Then, virtual nodes of the same ViPXi should not be allocated onto the same physical node. In a data center scenario, placing virtual nodes onto the same rack should be prohibited. In the subsequent sections, we describe the graph embedding problem, to the constraints.

A. Graph embedding and mapping constraints

Given a ViPXi graph $G_v(R_v, L_v, t)$ and a physical substrate graph $G_p(R_p, L_p, t)$ at time t where R_v and R_p are the set of virtual and physical nodes, respectively, and L_v and L_p are the set of virtual and physical links, respectively. Let P_p be the set of all simple physical paths between any two physical nodes. Further, denote by $Q_R(r, t)$ be the vector of capacities (storage, memory, CPU) of a node r (physical or virtual) at time t. Let $Q_L(l, t)$ be the vector of characteristics (capacity, latency) of link l, and $Q_P^p(p, t)$ be the vector of the same characteristics of a physical path p at time t. For a path $p = (l_1^p, l_2^p, \dots)$, the capacity of p is the minimum of all capacities of l_i^p in p and the latency of p is the sum of all latencies of l_i^p in p.

The embedding problem is then to obtain a map that maps virtual nodes R_v to physical nodes R_p, denoted by M_R, and virtual links L_v to physical paths P_p, denoted by M_L, such that the resource demands are satisfied, i.e., $Q_R(M_R(r_i^v)) \succeq Q_R(r_i^v)$ and $Q_P(M_L(l_i^v)) \succeq Q_L(l_i^v)$ for all virtual nodes and links in G_v. An example of an embedding of a ViPXi with one back-up node (Fig. 2b) onto a physical substrate is shown in Fig. 3. The physical substrate is composed of three racks that host two physical nodes each: r_1^p and r_4^p, r_2^p and r_5^p, and r_3^p and r_6^p, respectively. Suppose all physical node capacities are sufficient in this example, one mapping solution could be that in Fig. 3b. A virtual link can be mapped onto multiple links, e.g., in the case of l_2^p and l_8^p providing a virtual link between node r_1^v and back-up node r_1^b, provided that the minimum



(a) Physical substrate. (b) Embedded Graph.

Fig. 3. Example of embedding a graph in Fig. 2b onto a physical substrate represented by (a).

capacity and total latency are sufficient for the virtual link requirements. Physical links can host multiple virtual links, e.g., l_2^p. The remaining capacities of the links and nodes are then considered for embedding other virtual graphs that arrive at a later time t.

While the classical graph embedding problem enforces virtual nodes to be placed only onto unique physical nodes, it is insufficient to assure independent failures. In the example, nodes to be protected r_1^v and r_2^v, and back-up node r_1^b would need to be placed on different racks, hence creating additional mapping constraints to the graph embedding problem.

B. Mapping Solution

The graph embedding problem is well-known to be NP-hard [24]. It differs from that of a graph isomorphism problem (which is solvable in polynomial time) as virtual links can be mapped onto a series of physical links and thereby exploding the complexity. There has been numerous work on solving the graph embedding problem: isomorphism-based detection [25], [26], path-splitting methods [27], multicommodity flow modeling [28], and heuristics based on substrate characteristics [29]. These proposals aim at maximizing the resource usage or at minimizing the maximum link load. From the application perspective, the objective is to minimize the execution time and the cost of renting the infrastructure [12].

The additional allocation constraints between virtual nodes, however, does not make the problem less complex since solution space remains the same even though the search space may be reduced. To this end, we choose to use Lischka and Karl's [25] graph embedding method based on isomorphism detection. Furthermore, it is relatively straightforward to incorporate the additional allocation constraints using this method.

We briefly describe the graph embedding method as follows. It is essentially a depth-first search that looks at all possible node mappings and eliminate the choices based on feasibility of virtual links emanating from the node in consideration. Initially, all possible node mappings are generated and sorted in some order that optimize some objective (e.g. minimize cost). The first mapping is picked, and the subsequent possible mappings on the neighbors of that node is considered. At each search step, one such possible node mapping is examined and

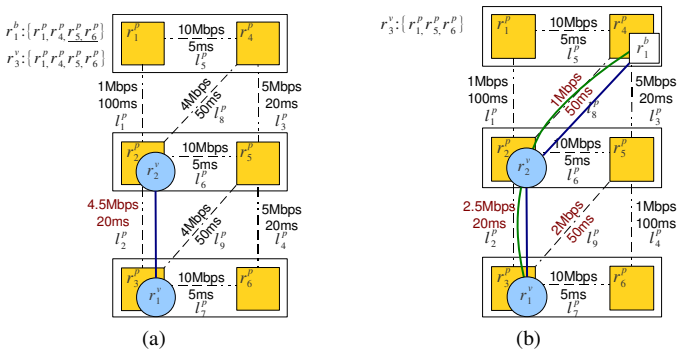


Fig. 4. Intermediate steps of mapping a graph in Fig. 2b onto a physical substrate represented by Fig. 3a. The list of the top left of each figure represents the available options for mapping the next step.

is considered only if links of that node to existing mappings are feasible. Refer to Fig. 4 for an illustration of the intermediate steps. Suppose a possible mapping considered in Fig. 4a and the next possible node mappings for neighbors of r_2^v are generated: for both r_1^b and r_3^v, possible mappings are to r_1^p, r_4^p, r_5^p and r_6^p. Fig. 4b supposes the mapping $r_1^b \rightarrow r_4^p$ is considered and subsequently examines it *in depth*. This mapping is feasible because the virtual links to existing node mappings are feasible, and links from r_1^b to other unmapped neighbors are still feasible in the remaining graph. The next set of possible for the remaining virtual nodes are generated and examined further *in depth*. If there are no feasible mappings for remaining nodes, the steps are backtracked and continued on the next available options.

There are other optimizations involved while considering possible mappings, particularly the ordering of possible mappings considered and pre-filtering possible mappings to avoid looking too deep into a search tree. We refer the reader to the paper [25] for further details. For our purpose of incorporating the additional placement constraints, a filtering step can be added to the list of generated possible mappings. For example, in Fig. 4a, r_5^p and r_6^p can be omitted from the possible mappings of r_1^b (underlined) because they are in the same rack as the existing map. Further optimizations may be made, e.g., sorting the order of possible mappings to be considered based on the additional constraints. We leave this to future work.

C. From Mapping to Allocation

The map provided by the allocation step is interpreted and instantiated using the HIPerNet framework[1]. The HIPerNet framework combines system and networking virtualization technologies with bandwidth sharing and advance reservation mechanisms to offer dynamic networking and computing infrastructures as services [1], [30]. At the lower level, the HIPerNet framework accesses and controls a part of the physical infrastructure that is virtualized and exposed. Enrolled physical resources are then registered to the HIPerNet registrar and can be allocated to ViPXis. Once the resources have been exposed, HIPerNet gets full control over it. This set of exposed virtualized resources composes the substrate hosting the ViPXis.

At run-time, the HIPerNet manager communicates with physical resources to deploy virtual nodes (configured respecting the users requirements), monitor their status and configure control tools to supervise the resource usage. In this fully-virtualized scenario, HIPerNet interacts with multiple resource providers to plan, monitor and control them. Functions such as fault management, load balancing, bandwidth management and performance control are handled taking both network- and resource-virtualization techniques into account.

[1]HIPerNet was designed in the context of the HIPCAL project http://hipcal.lri.fr/

VII. Evaluation Through a Use Case Application

We now apply these to an existing large-scale distributed application, named *bronze standard*, for proof of concept purpose. The bronze standard [31] technique tackles the difficult problem of validating procedures for medical-image analysis. As there is usually no reference, or *gold standard*, to validate the result of the computation in the field of medical-image processing, it is very difficult to objectively assess the results' quality. The statistical analysis of images enables the quantitative measurement of computation errors. The bronze standard technique statistically quantifies the maximal error resulting from widely used *image registration algorithms*. The larger the sample image database and the number of registration algorithms to compare with, the most accurate the method. This procedure is therefore very scalable and it requires to compose a complex application workflow including different registration-computation services with data transfer inter-dependencies.

The bronze standard application can be represented as a workflow of computational processes with I/O data dependencies, as illustrated in Fig. 5. In the experiments reported below, this workflow is enacted with the data-intensive grid-interfaced MOTEUR workflow manager [32] designed to optimize the execution of data-parallel flows. A clinical database of 59 pairs of patient images to be registered by the different algorithms involved in the workflow is used. Each service depicted in Fig. 5 is instantiated as an independent computing task that is delegated to one of the computing nodes. For each run, the processing of the complete image database thus results in the generation of 354 computing tasks (with a computation time of 30 seconds to 5 minutes each on a state-of-the-art PC). The data volume transferred for each task is in the order of 30 MB. The makespan of the application's execution is in the order of 20 minutes in the absence of failures.

The reliability mechanisms of HIPerNet presented in this work, are based on a modified version of Remus. To enable Remus protection, all VMs file-system were deployed on a network file system (NFS) server. The first benchmarks performed with Remus demonstrated that communication between the NFS server (source) and VMs (destination) should be limited to a maximum transfer rate of 100Mbps. Otherwise, Remus cannot keep a stable copy of the critical VM for the default

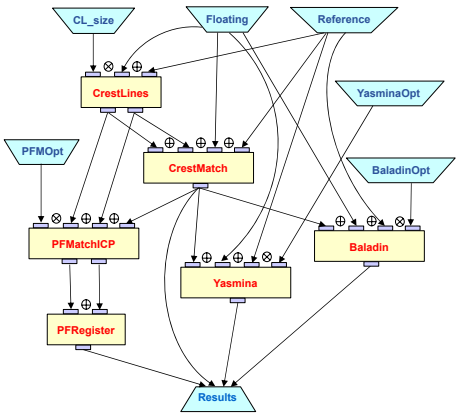


Fig. 5. Bronze Standard workflow.

checkpoint intervals of 200ms. In this initial work, the reliability mechanisms are applied to virtual nodes. The protection of network communication and data persistent on NFS are out of scope and they are not discussed. The experiments are carried out using ViPXis managed by HIPerNet within the Grid'5000 testbed [5]. Grid'5000 enables a user to request, reconfigure, and access physical machines belonging to nine sites distributed in France. In our experiments, we use 100 physical nodes to compose a pool of virtualized physical resources. The number of physical nodes exceeds that of the number of virtual resources specified (see next section) because virtual machines of *the same application* cannot be co-located in the same physical node to prevent correlated failures and additional virtual back-up nodes are needed to protect the critical nodes.

Faults are simulated by shutting down physical machines respecting the Mean Time Between Failures (MTBF) parameter. The MTBF of each node in each experiment is 60000s, 30000s and 15000s. Assuming the largest makespan to be 30 minutes, the failure probability of each node is then 0.03, 0.06 and 0.12, respectively. The initial MTBF value (60000s) is based on failure rate of servers (with a probability between 0.02 and 0.04) identified by [33]. The other lower MTBF values represent worse failure rates which could be attributed to a variety of reasons. Some examples are improper cooling in racks, irregular maintenance and inadequate protection from power interruptions.

A. ViPXi composition

The optimal ViPXi specification to *bronze standard* is based on [12], where 31 virtual resources are configured with 512MB of RAM, and 1GHz of CPU, and 10Mbps of bandwidth requirement for each virtual link between the database and the workers. Virtual nodes require exclusivity on physical nodes. As shown in Fig. 6, the HIPerNet engine deploys and manages virtual machines on these computers on demand (dark arrows).

The MOTEUR workflow engine, as a client of the HIPerNet framework, was hosted on one physical host, outside of the ViPXi. MOTEUR produces VXDL descriptions including the reliability requirements that are requested to the HIPerNet engine (blue connection). After receiving all virtual machines allocated to the ViPXi, MOTEUR connects to the computing nodes (worker nodes) to invoke the application services (red connections). The computing nodes connect to the database host to copy the input data and send the computational results, and the final results are sent to MOTEUR (green connections).

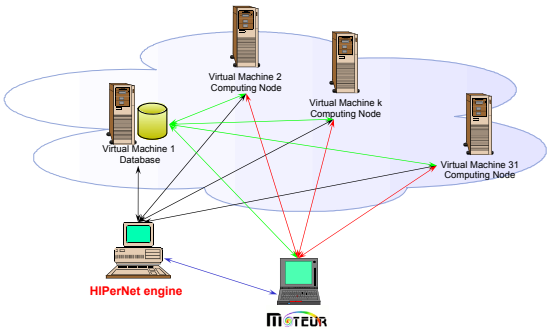


Fig. 6. Experimental infrastructure.

B. Cost model

From an infrastructure provider point of view, the major challenge is to account (financially or not) for resources usage according to specific criteria (*e.g.* fair share among users, digressive price, reliability level etc). Although a quasi-unlimited amount of computing resources may be allocated, a trade-off has to be found between (i) the allocated infrastructure cost, (ii) the expected performance, and (iii) the optimal performance achievable, which depends on the level of parallelization of the application.

Considering this scenario, we introduce a simple cost model for the pricing of a ViPXi with reliability support. The substrate provider estimates the provisioning cost of an extended ViPXi (already with back-up resources identified). We consider different prices for active and back-up resources.

We define a price function $\Psi_R(r,t)$ which sets the price for an amount of resource r at time t. Similarly, $\Psi_L(l,t)$ would set a price for the bandwidth. The total price for the use of the resource is thus, over the lifetime $[0,T]$ of the virtual infrastructure G:

$$P_G(T) = \int_0^T \left(\Sigma_{i\in R_v}\Psi_R(r_i^v(t),t) + \Sigma_{j\in L_v}\Psi_L(l_j^v(t),t)\right) dt$$

Introducing link and node redundancy to increase the reliability corresponds to an additive cost to the user which has to be evaluated. The cost function is extended to calculate the total price ($P_{G'}(T)$) for the extended graph ($G'_v(R'_v, L'_v, T)$), including reliable resources (R_b, L_b). The price of reliability (P_B) of a virtual infrastructure is given by

$$P_B(T) = \int_0^T \left(\Sigma_{i\in R_b}\Psi_R(r_i^b(t),t) + \Sigma_{j\in L_b}\Psi_L(l_j^b(t),t)\right) dt$$

and the total price of a reliable virtual infrastructure is of course $P_{G'}(T) = P_G(T) + P_B(T)$.

For a first order assessment of the performance of our model, we consider a pricing model based on the published prices of Amazon EC2[2] for Europe. A detailed economics analysis is outside of the scope of this document and we set $\Psi_R(r,t)$ to correspond to a fixed price per hour use for one of two types of nodes, and one of two types of contract: basic node (with 1.7GB RAM) and high performance node (with 7.5GB RAM); short term lease, and long term lease. Those prices are given on Table I. EC2 does not charge any link cost in between nodes of its data center, and since the data transfers in our application can be fulfilled by typical ethernet links, we do not include any specific link pricing in our basic cost analysis. For EC2-like infrastructures, there is a cost and delay associated with uploading the medical images to process up to the data center over the Internet, however this is independent of the reliability and outside of the scope of our paper.

TABLE I
AMAZON EC2 EUROPE PRICES FOR VMS (PER HOUR OR PART THEREOF).

VM Specifications	1.7GB RAM	7.5GB RAM
Short term lease	$0.095	$0.38
Long term lease	$0.031	$0.031

We consider prices for the user, but an analysis of the costs to the provider would yield similar results. Our intent is to provide some rough estimates to illustrate the trade-off between resource and reliability.

C. Experimental results

The application makespan when the application is executed on a substrate without simulated failures is 1205s $\pm$ 40s, serving as the base-line. For these values, the regular cost of this ViPXi without reliability support is $2.95 (short term lease), serving as base cost to analysis.

The first experiment examines the protection of the database node. In this case, the database is the unique component protected, and faults are submitted in accordance with MTBF definition. Table II summarizes the execution of this scenario. The application makespan increases proportionally to the number of failures detected on database node. Comparing with the base-line, the application makespan increases by +16%, +26% and +40% with regard to the MTBF values, 60000s, 30000s and 15000s, respectively.

In our experimental set-up, we provided reliability by backing-up the database 1:1, and the price for all values of the MTBF would be $3.04. However, while 1:1 replication made our proof-of-concept implementation feasible[3], it does not keep the required reliability at the specified level. To calculate the theoretical price of each ViPXi with the proper reliability support, we compute the number of back-up nodes required to provide the reliability level of 99.99% as a function of the MTBF, computed according to [20]. For this scenario, the cost of database protection with reliability level 99.99% increases the ViPXi cost by about 6%, 10%, and 13% for MTBF 60000s, 30000s, and 15000s, respectively (see table IV). If we assume that the back-up nodes are selected from a pool of nodes reserved for this purpose by the physical substrate operator with a long-term lease, then the price of reliability amounts to an increase of 2%, 3%, and 4% for MTBF 60000s, 30000s, and 15000s, respectively (again, see table IV).

Each workflow service has a pre- and post-processing stage where the input data is copied to worker node and the results are sent to the database. The more failures happen during these two stages, the more the application makespan increases. In table III, we present the data transfer time (in seconds) of this scenario. The data transfer time increase dominates when there are more failures detected on database node.

TABLE II
EXECUTION TIME AND % INCREASE OVER BASELINE FOR CRITICAL DATABASE PROTECTION ONLY (COLUMN DB), AND FOR COMPUTING NODES PROTECTION (COLUMN CN).

MTBF	DB	Increase	CN	Increase
∞	1205s		1205s	
60000s	1401s	16.26%	1208s	0.2%
30000s	1524s	26.47%	1225s	1.7%
15000s	1688s	40.08%	1244s	3.2%

TABLE III
TOTAL DATA TRANSFER TIME OF SIX APPLICATION SERVICES RUNNING WITH CRITICAL DATABASE PROTECTION SCENARIO.

MTBF	Total data transfer time
∞	165.02s $\pm$ 44.30s
60000s	190.20s $\pm$ 96.75s
30000s	292.96s $\pm$ 115.38s
15000s	299.61s $\pm$ 128.26s

The second experiment analyzes the protection of workers nodes. The MTBF varies in accordance with the failure model presented above. After a MTBF, a random physical machine will be crashed. The back-up virtual machine is automatically started and continues running the same workflow task. As presented in table II, the application makespan slightly increases with regard to the number failures detected on the infrastructure. The delay on the back-up node activation is compensated by other parallel executions. Providing reliability for workers nodes (99.9%) dramatically decreases the time to complete the application, from execution time for the 15000s MTBF of 1688s down to 1244s in Table II, a gain of almost 40%. Table V shows the price increase due to reliability for the different values of the MTBF, assuming that the back-up nodes are drawn from the same (short term lease) pool as the rest of the virtual infrastructure, or from a long term lease pool set aside by the physical substrate operator.

In both cases, database protection and workers protection, the application ran normally, with faults being transparent to the application provider.

[2]Amazon EC2: http://aws.amazon.com/ec2/

[3]The current Remus implementation for Xen 3.4 is limited to a 1:1 protection. This limitation also resulted in simpler allocation constraints than that described in Section VI.

TABLE IV
PRICE WITH RELIABILITY FOR DATABASE PROTECTION (RELIABILITY LEVEL 99.9%) AND FRACTION OF PRICE CORRESPONDING TO RELIABILITY WITH BACK-UP PROVISIONED ON SHORT TERM LEASES OR LONG TERM LEASES.

			Short term		Long term	
MTBF	p_{FAIL}	n_{r_b}	$P_{G'}$	$P_B/P_{G'}$	$P_{G'}$	$P_B/P_{G'}$
60000s	0.03	2	$3.13	6%	$3.01	2%
30000s	0.06	3	$3.23	10%	$3.04	3%
15000s	0.12	4	$3.33	13%	$3.07	4%

TABLE V
PRICE OF RELIABILITY FOR COMPUTING NODE PROTECTION (RELIABILITY LEVEL 99.9%) AND FRACTION OF PRICE CORRESPONDING TO RELIABILITY WITH BACK-UP PROVISIONED ON SHORT TERM LEASES OR LONG TERM LEASES.

			Short term		Long term	
MTBF	p_{FAIL}	n_{r_b}	$P_{G'}$	$P_B/P_{G'}$	$P_{G'}$	$P_B/P_{G'}$
60000s	0.03	5	$3.42	16.1%	$3.10	5.3%
30000s	0.06	8	$3.71	25.8%	$3.19	8.4%
15000s	0.12	12	$4.09	38.7%	$3.32	12.6%

We also performed the experiments using the task resubmission mechanism (application level) to compare with the ViPXi reliability service. In general, after a failure occurs on a worker node, a new worker node must be provisioned, and the task executed on the failed node has to be relaunched on the new worker node. We minimize the activation time of a back-up node to zero by reserving, deploying and configuring back-up nodes prior to the execution of the Bronze Standard. Hence, the only difference from the previous experiments is the time needed to rework the tasks on the failed worker nodes.

The number of back-up nodes for task resubmission mechanism is set to be the same as that in the previous scenario (i.e., 5, 8, and 12 for MTBF of 60000s, 30000s, 15000s, respectively) so that the cost and amount of resources used are equivalent. Our experimental results show that the application makespan increases significantly in comparing with the virtual infrastructure reliability service, +13.08%, +19.67% and +22.19% with respect to 60000s, 30000s and 15000s of the MTBF, as presented in table VI. The makespan gap would have been more if back-up nodes were not pre-allocated and configured. We do not present results otherwise since the time required for reservation, deployment and configuration may vary with the configuration and total utilization of the grid.

TABLE VI
APPLICATION MAKESPAN WITH RESUBMISSION MECHANISM AND PERCENTAGE INCREASED WHEN COMPARED WITH VIPXI RELIABILITY SERVICE.

MTBF	Reliability	Resubmission	Increase
60000s	1208s	1366s	+13.08%
30000s	1225s	1466s	+19.67%
15000s	1244s	1520s	+22.19%

VIII. RELATED WORK

Providing reliability on virtualized environments is an issue that has been studied in the recent years. Within virtual nodes, hypervisors such as Xen provides the capability to store live snapshots of the virtual machines to reliable storage, which can be resumed on other physical nodes if failures occur. Remus [17] and Kemari [18] improves on static snapshots by periodically updating live snapshots to replica nodes that are on standby. To checkpoint the entire virtual infrastructure as a whole, VNsnap [16] has been developed. VNsnap captures the entire virtual infrastructure's execution, communication and storage states, which can be resumed in other sites to recover from failures. From another perspective, proactive migration of virtual machines to other healthy nodes is considered [34] upon early warnings of impending failures.

Fault tolerance is provided some contexts, such as data centers [35], [36]. However, it is achieved through specific engineering of the network nodes and links overprovisioned for redundancy.

The allocation of virtual infrastructures has been already explored in previous works. Some algorithms focus on problem formulation considering nodes requirements together with network configuration [29], [27], [28]. In Emulab, a network is modeled characterizing the bandwidth capacity of each link, and the substrate nodes are not shared among multiples virtual infrastructures. Ricci et al. [37] developed the software *assign*, which explores the resources homogeneity of Emulab and introduces the definition of *vclasses and pclasses* (equivalence classes) that limits the search space of an allocation. While our framework requires a resource allocation mechanism, none of the above took into account reliability for embedding the virtual resource request onto the physical topology.

[20] provides mechanisms to pool back-up nodes to achieve some desired level of reliability. However, it is mostly a theoretical work and does not provide any vertical integration from the user specification to the physical substrate allocation.

In [38], Menth et al. focus on providing link reliability in wide-area network, by considering the most likely link failure combinations, and providing back-up links for these failures. This is distinct from this work, where reliability is applied to links but also node failures within a virtual infrastructure.

IX. CONCLUSIONS

We have presented a framework that introduces transparent reliability support into virtualized infrastructures. The transparency allows virtual infrastructure users to focus on application development and scale reliability requirements at deployment. The physical substrate operator can provide reliability as a service, and implement reliability transparently from the point of view of the service operator.

Our framework contains a specification language which describes the reliability parameters in a flexible and expressive manner; an algorithm to translate virtual infrastructure specifications to physical resources; an resource mapping algorithm to allocate them; and a synchronization mechanism that preserves virtual machine states in cases of physical node

failures. To provide an easier evolutionary path to implement the framework, some of these components are built upon tried-and-true existing technologies.

We implemented the framework on top of the HIPerNet framework, deployed over the Grid'5000 infrastructure, and demonstrated that it effectively supports reliability and enables the transparent execution of fault-sensitive distributed applications. In particular, our implementation points to a reduced completion time for the application for a slight increase of the resource cost.

Further work includes the implementation of a $n : k$ reliability ratio within our testbed, in order to fully benefit from the virtualization of reliability. This also involves implementing the sharing of redundant node across different virtual infrastructures in order to minimize the number of such redundant nodes, as described in Section V. For the cost benefit, we presented a simple yet promising back-of-the-envelope analysis. We would like to refine the model to better distinguish the economical trade-offs for each of the stakeholders: service customer, service provider and virtual infrastructure provider, in particular when the virtual infrastructure is hosted across different administrative domains.

ACKNOWLEDGMENTS

This work has been funded by the ANR CIS HIPCAL grant, the FP-7 SAIL project, the French ministry of Education and Research, INRIA, CNRS, via ACI GRID's Grid'5000 project and Aladdin ADT. The authors would like to thank Romaric Guillier for his help in the development of HIPerNet.

REFERENCES

[1] F. Anhalt, G. Koslovski, and P. Vicat-Blanc Primet, "Specifying and provisioning Virtual Infrastructures with HIPerNET," *Int. J. Netw. Manag.*, vol. 20, no. 3, pp. 129–148, May/Jun. 2010.

[2] "GENI System Overview," The GENI Project Office, September 2008.

[3] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The reservoir model and architecture for open federated cloud computing," *IBM J. Res. Dev.*, 2009.

[4] G. Koslovski, T. Truong Huu, J. Montagnat, and P. Vicat-Blanc Primet, "Executing distributed applications on virtualized infrastructures specified with the VXDL language and managed by the HIPerNET framework," in *Proc. ICST CLOUDCOMP*, Oct. 2009.

[5] F. Cappello, P. Primet et al., "Grid'5000: A Large Scale and Highly Reconfigurable Grid Experimental Testbed," in *Proc. IEEE Grid*, Nov. 2005.

[6] "Common Information Model (CIM) Standards." [Online]. Available: http://www.dmtf.org/standards/cim/

[7] J. van der Ham, P. Grosso, R. van der Pol, A. Toonk, and C. de Laat, "Using the network description language in optical networks," in *Proc. IFIP/IEEE IM*, May 2007.

[8] "Network Mark-up Language Working Group (NML-WG)," 2007. [Online]. Available: https://forge.gridforum.org/projects/nml-wg

[9] "Open Virtualization Format Specification (OVF)." 2009. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf

[10] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero, "Service specification in cloud environments based on extensions to open standards," in *Proc. ICST COMSWARE*, Jun. 2009.

[11] "Open Cloud Computing Interface Working Group (OCCI-WG)," 2009. [Online]. Available: http://forge.gridforum.org/sf/projects/occi-wg

[12] T. Truong Huu and J. Montagnat, "Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure," in *Proc. Cloud'10*, May 2010.

[13] G. Koslovski, P. Vicat-Blanc Primet, and A. S. Charão, "VXDL: Virtual Resources and Interconnection Networks Description Language," in *GridNets 2008*, Oct. 2008.

[14] D. Lingrand, J. Montagnat, and T. Glatard, "Modeling user submission strategies on production grids," in *Proc. ACM HPDC*, Jun. 2009.

[15] D. Lingrand, J. Montagnat, J. Martyniak, and D. Colling, "Analyzing the EGEE production grid workload: application to jobs submission optimization," in *Proc. JSSPP Workshop*, May 2009.

[16] A. Kangarlou, P. Eugster, and D. Xu, "VNsnap: Taking Snapshots of Virtual Networked Environments with Minimal Downtime," in *Proc. IEEE/IFIP DSN*, Jun. 2009.

[17] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication," in *Proc. USENIX NSDI*, Apr. 2008.

[18] Y. Tamura, K. Sato, S. Kihara, and S. Moriai, "Kemari: VM Synchronization for Fault Tolerance," in *USENIX '08 Poster Session*, Jun. 2008.

[19] G. Koslovski and P. Vicat-Blanc Primet, "VXDL Parser APPcode: IDDN.FR.001.260009.000. S.P.2009.000.10800," 2009.

[20] W.-L. Yeow, C. Westphal, and U. C. Kozat, "Designing and embedding reliable virtual infrastructures," in *Proc. VISA Workshop*, Sep. 2010.

[21] F. Harary and J. P. Hayes, "Node fault tolerance in graphs," *Networks*, vol. 27, no. 1, pp. 19–23, 1996.

[22] S. Dutt and N. R. Mahapatra, "Node-covering, error-correcting codes and multiprocessors with very high average fault tolerance," *IEEE Trans. Comput.*, vol. 46, no. 9, pp. 997–1015, 1997.

[23] M. Ajtai, N. Alon, J. Bruck, R. Cypher, C. Ho, M. Naor, and E. Szemeredi, "Fault tolerant graphs, perfect hash functions and disjoint paths," in *Proc. IEEE FOCS*, Oct. 1992.

[24] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: State of the art and research challenges," *IEEE Commun. Mag.*, Jul. 2009.

[25] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. VISA Workshop*, Aug. 2009.

[26] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, Oct. 2004.

[27] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Apr. 2008.

[28] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. IEEE INFOCOM*, Apr. 2009.

[29] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proc. IEEE INFOCOM*, Apr. 2006.

[30] P. Vicat-Blanc Primet, J.-P. Gelas, O. Mornard, G. Koslovski, V. Roca, L. Giraud, J. Montagnat, and T. T. Huu, "A scalable security model for enabling dynamic virtual private execution infrastructures on the internet," in *Proc. IEEE CCGrid*, May 2009.

[31] T. Glatard, X. Pennec, and J. Montagnat, "Performance evaluation of grid-enabled registration algorithms using bronze-standards," in *Proc. MICCAI*, Oct. 2006.

[32] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR," *Int. J. High Perform. C.*, Aug. 2008.

[33] D. Atwood and J. G. Miner, "Reducing data center cost with an air economizer," Intel White Paper, Tech. Rep., 2008. [Online]. Available: http://www.intel.com/it/pdf/Reducing_Data_Center_Cost_with_an_Air_Economizer.pdf

[34] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive Fault Tolerance for HPC with Xen Virtualization," in *Proc. ACM ICS*, Jun. 2008.

[35] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM '09*, 2009.

[36] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," in *ACM SIGCOMM '09*, 2009.

[37] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *SIGCOMM Computer Comm. Rev.*, 2003.

[38] M. Menth, M. Duelli, R. Martin, and J. Milbrandt, "Resilience analysis of packet-switched communication networks," *IEEE/ACM Trans. Netw.*, Dec. 2009.