An Overview of Embedding Strategies in Cloud Computing Backbone Networks

Ilhem Fajjari§†, Nadjib Aitsaadi‡, Guy Pujolle† and Hubert Zimmermann §
§Ginkgo Networks: 2 A Rue Danton, 92120 Montrouge, France
‡HIPERCOM – INRIA: Domaine de Voluceau, Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France
†UPMC - University of Paris 6: 4 Place Jussieu, 75005 Paris, France
ilhem.fajjari@lip6.fr, nadjib.aitsaadi@inria.fr, guy.pujolle@lip6.fr, hubert.zimmermann@ginkgo-networks.com

Abstract

This paper is an overview of virtual network embedding strategies within cloud infrastructure backbone networks. We will first of all summarise and define the main principles of cloud computing and its different supplied services. Cloud computing can be defined as the combination of three components: grid computing, cluster computing and virtualisation. We will propose a new cloud infrastructure architecture, named the Cloud Infrastructure using Virtualisation within Data centers and Backbone (CIVDB), which makes use of virtualisation in all the cloud’s equipment (i.e. routers and data centres). We will then describe, analyse, and compare the main virtual network mapping algorithms for cloud infrastructure networks found in existing literature: i) VNE-Least, ii) VNE-Cluster, iii) VNE-Subdividing, iv) VNE-Greedy and v) VNE-AC. We will evaluate the above strategies in terms of: i) virtual network request reject rate, ii) embedding cost of virtual network request, iii) embedding revenue of virtual network request and iv) average usage rate of physical resources.

Index Terms

Cloud computing, IaaS, Network virtualisation, Embedding problem.

I. INTRODUCTION

Cloud computing is new paradigm that provides an end-user with utility computing via the Internet using a Web browser. In existing literature, we have found more than twenty definitions of cloud computing [1]–[3]. We believe that the definition proposed in [3] is the most pertinent. It consists of combination of three concepts: i) cluster computing, ii) grid computing and iii) virtualisation. Cluster computing is defined in [4] [5] as: “a type of parallel and distributed system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource”. Moreover, grid computing is defined in [3] as: “a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users’ quality of service requirements”. Cloud computing is defined as grid computing in which each geographical node is a cluster or data centre (i.e. home computer system). All the geographically distributed data centres are interconnected via the provider’s substrate backbone network. It is worth noting that virtualisation is available within data centers and the backbone network. Therefore, thanks to virtualisation, many simultaneous and independent services can be delivered. In fact, all the resources (physical and software) can be virtualised.

As described in [6], a cloud can deliver many services that can be classified in three main groups. The first group, Software as a Service (SaaS), is a model of software deployment where an application is installed in the provider’s infrastructure (i.e. data centers). In doing so, an application is hosted as a service provided to customers via the Internet. In fact, end-users do not need to install and run an application on their own local computer. SaaS also reduces the cost and complexity of upgrading applications for companies. For instance, Google Apps and Salesforce Customer Relationships Management (CRM) are two clouds providing SaaS. The second group, Platform as a Service (PaaS), supplies the developers with a software platform (e.g. programming-language-level, APIs, etc.). The main objectives are: i) to facilitate the interaction between the infrastructure and cloud applications, and ii) to speed up the process (conception, implementation, etc.). For instance, Google’s App Engine is a PaaS that provides a python language and APIs to interact with Google’s cloud. Finally, the third group, Infrastructure
as a Service (IaaS), provides computational physical resources to cloud users, such as processing capacity, storage and routers. In this respect, customers are free to install and manage their own software stack (operating system, application, routing protocol, etc.) as super-users. It is worth noting that IaaS can be supplied thanks to virtualisation technologies (OS virtualisation, para-virtualisation and hardware-assisted virtualisation), which guarantee isolation between virtual instances sharing the same physical resources. For instance, Amazon’s Elastic EC2 is a well-known cloud offering IaaS.

In this paper, we will propose a new cloud infrastructure architecture called the Cloud Infrastructure using Virtualisation within Data centers and Backbone (CIVDB). It is based on a VICTOR architecture [7] that does not use a virtualisation technology in the backbone network. Then, we will give an overview of the related virtual network embedding strategies within a cloud offering IaaS. In fact, the customer formulates a virtual network backbone request which interconnects all its geographically distributed data centres. Moreover, the customer also determines the quality of service of a virtual network in terms of i) the routers’ processing power, ii) the routers’ memory and iii) the links’ bandwidth. In this regard, the cloud provider’s main objective is to lease the maximum number of virtual networks while ensuring QoS, in order to maximise revenue. Note that an embedding strategy has to map i) each virtual router in a substrate router and ii) each virtual link between two virtual routers in a substrate path between the substrate nodes hosting the virtual link’s extremity nodes. The problem we are investigating is NP-hard and, to the best of our knowledge, few mapping strategies have been proposed in existing literature [8]–[12]. In this research paper, we will describe and compare the virtual network strategies already found in literature. To do so, we implemented all the strategies and used extensive simulations to gauge and evaluate each one’s performance in terms of request reject rate, request revenue and request cost.

This paper is organised as follows. In the next Section we will explain our proposed architecture, CIVDB. In Section III, we will formulate the virtual network embedding problem within a cloud supplying IaaS. Then, in Section IV, we will describe the main related mapping strategies. In Section V, we will analyse the performance of related mapping algorithms based on extensive simulations. Finally, in Section VI, we will conclude the paper and outline the main challenges for future research.

II. CLOUD INFRASTRUCTURE USING VIRTUALISATION WITHIN DATA CENTERS AND BACKBONE (CIVDB) ARCHITECTURE

As illustrated in Fig. 1, the CIVDB consists of a set of geographically distributed data centres interconnected with a Substrate backbone Network \( \mathcal{SN} \). The latter is formed by a set of geographically distributed routers, interconnected with wired connections (e.g. fibre optic). In fact, the \( \mathcal{SN} \) contains two types of routers: i) access and ii) core. It is worth noting that each data centre is attached to only one access router, through which it can communicate with core routers.

Thanks to virtualisation technology, many independent applications can be hosted in data centres (i.e. SaaS). Moreover, an application can be deployed in many geographical sites and makes use of a Virtual Network \( \mathcal{VN} \), mapped in the \( \mathcal{SN} \), to link all the geographical sites (i.e. IaaS). In this respect, an end-user can install any routing protocol within the allocated \( \mathcal{VN} \) and be responsible for network administration. In other words, since virtualisation technology offers isolation, an end-user can only manage their \( \mathcal{VN} \) (i.e. instance) and cannot deteriorate the rest of the \( \mathcal{VN} \)'s hosted in the \( \mathcal{SN} \).

The CIVDB contains one or more Centralised Controllers (CC), as depicted in Fig. 1, in the aim of managing the cloud by i) monitoring the cloud infrastructure, ii) embedding applications in data centres, iii) mapping a virtual network in the substrate backbone network, etc.

Our architecture, CIVDB, is an improved version of the VICTOR architecture proposed in [7]. Indeed, in VICTOR, the backbone network consists of a set of Forwarding Elements (FE). Their role is limited to the simple forwarding of data according to the forwarding tables established and downloaded from Centralised Controllers (CC). It is worth noting that FEs do not implement any control plane or routing functions. VICTOR is confronted with a scalability problem since centralised controllers manage all the flows in the backbone network. In addition, VICTOR does not take advantage of virtualisation technology within the Substrate Network. For this reason, and in order to get around the VICTOR’s disadvantages, we proposed CIVDB. As it uses virtualisation, many independent \( \mathcal{VN} \) instances can be deployed at the same time in the \( \mathcal{SN} \).
III. FORMULATION OF THE VIRTUAL NETWORK EMBEDDING PROBLEM

In this section, we will formulate the Virtual Network (VN) embedding problem within the cloud’s Substrate backbone Network (SN). Indeed, the SN is modelled as an undirected graph denoted by $G^s(N^s, E^s)$, where $N^s$ and $E^s$ are the sets of physical nodes and their connected links respectively. Each physical router, $n^s_i \in N^s$, is characterised by its i) residual processing power, ii) residual memory, iii) type: access or core, and iv) geographic location. Likewise, each physical link, $e^s_x \in E^s$, is typified by its available residual bandwidth.

Similarly, the VN request is modelled as an undirected graph, denoted by $G^v(N^v, E^v)$, where $N^v$ and $E^v$ are the sets of virtual nodes and their virtual links respectively. Within the VN request, each virtual node, $n^v_i \in N^v$, is associated with the required processing power, memory, type, and geographic location if it is an access router. Moreover, each virtual link, $e^v_x \in E^v$, requests a quota of bandwidth.

It is worth noting that all the physical resources (i.e. bandwidth, processing power, memory) in $G^s$ are limited. In fact, $G^s$ is not able to host an infinite number of VN requests. Consequently, an intelligent and judicious mapping of the VN in $G^s$ is necessary in order to maximise the acceptance rate and the substrate provider’s profits.

Node mapping is constrained so that for each VN request, $G^v$, two virtual nodes cannot be assigned to the same substrate node. In addition, each virtual node must be assigned to only one physical node. The virtual node, $n^v_i$, can be mapped in the substrate node, $n^s_j$, if the available residual resources (i.e. power processing and memory) are at least equal to those required. Besides, if $n^v_i$ is an access router then it can only be embedded in substrate access routers located within a predefined geographic area.

Each virtual link, $e^v_x$ between $n^v_i$ and $n^v_j$, is assigned to an unsplitable substrate path, denoted by $P_{e^v_x}$, between $n^s_i$ and $n^s_j$. Note that $P_{e^v_x}$ is a set of substrate links. Moreover, the available residual bandwidth of all the physical links in $P_{e^v_x}$ must be at least equal to the request bandwidth. In fact, SN’s mainly make use of shortest-path-based routing protocols such as Open Shortest Path First (OSPF). Indeed, to employ sophisticated splittable routing algorithms, IP routers that are already deployed must be upgraded. However, this would significantly increase capital expenditure.

The objective is to generate, for each VN request, the best possible mapping while also minimising the embedding cost in terms of allocated resources in the SN. Note that for a specific VN request, no matter what the mapping, the resources allocated by the virtual nodes are identical. However, assigned resources for the virtual links depend on the substrate path length. As described above, the embedding problem can be formulated as a binary combinatorial
optimisation problem. It has been proved to be NP-hard [13]–[15]. In the next section, we will outline the related \(VN\) embedding algorithms found in existing literature.

IV. RELATED VIRTUAL NETWORK EMBEDDING STRATEGIES

Few research papers have studied the \(VN\) embedding problem. In this section, we will outline the main proposals found in existing literature.

In [8], the authors propose three \(VN\) assignment algorithms: VNE-Least, VNE-Cluster and VNE-Subdividing. The first method, VNE-Least, treats virtual node and virtual link mapping separately. Thus, substrate and virtual nodes are sorted according to their stress and degree of connectivity respectively. Then, VNE-Least assigns the virtual node with highest degree of connectivity to the least stressed substrate node recursively until all the virtual nodes are embedded. Thereafter, VNE-Least makes use of the shortest distance algorithm to connect the mapped virtual nodes. On the other hand, the VNE-Cluster and VNE-Subdividing algorithms take into account the substrate link load when selecting substrate nodes. To do so, the authors qualify each substrate node using a new metric evaluating both its stress and that of its directly connected links. Then, the substrate and virtual nodes are sorted according to the newly defined metric and the degree of connectivity respectively. Moreover, a new path distance is defined based on the substrate link stress. VNE-Cluster assigns the virtual nodes in a similar manner to VNE-Least, except that the nodes’ stress and path distance are replaced by the newly defined metrics. The final proposal, VNE-Subdividing, subdivides the \(VN\) request into star topologies so as not to deal with the request as one whole unit. Next, the stars are mapped sequentially using VNE-Cluster. However, the authors assume that the resources are unlimited in the \(SN\), which is not a realistic assumption. Thus, the \(VN\) request reject rate is not evaluated in the paper.

In [9], the authors proposed the VNE-Greedy virtual network embedding algorithm. In this system, the substrate and virtual nodes are sorted according to the available and requested resources respectively. Then, the virtual node with the highest resource request is assigned to the substrate node containing the largest available resource metric value recursively until all the virtual nodes are mapped. Next, the nodes are connected using the K-shortest paths algorithm. VNE-Greedy’s main drawback is the substrate path building algorithm. Indeed, the shortest path algorithm does not consider congested \(SN\) links, which implies an increase of hot-spots in the \(SN\) and an increase in request reject rate. Moreover, virtual node and virtual link mapping are not coordinated.

The authors of [10] model the \(VN\) as a directed graph with two types of nodes: access and core. The required \(VN\) resources are defined in terms of the expected traffic, which is expressed as an upper limit on allowed traffic between all access node pairs. The authors’ objective is to calculate the minimum request bandwidth for the \(VN\) according to the access nodes’ upper limit of traffic. Nonetheless, the weakness in the proposal lies in its lack of consideration for \(SN\) capacities (i.e. unlimited) and the use of static routing tables in the network. In addition, the proposal requires the use of a star \(VN\) topology, which is strongly binding.

In [11], the authors propose two \(VN\) embedding algorithms, named Deterministic-ViNE and Randomized-ViNE. Here, the substrate graph is augmented with meta-nodes and meta-edges to form a meta-graph. Each meta-node corresponds to one virtual node and each meta-edge is a link between a meta-node and the substrate nodes located in its required geographic area. Note that each virtual node is associated with a specified region where it could be hosted. Nevertheless, it is not realistic to expect end-users to specify all the virtual nodes’ locations. In fact, only the locations of access nodes can be fixed. The main drawback here is the nodes’ locations constraints. Indeed, when these are not defined, D-ViNE and R-ViNE cannot be executed since the meta-graph cannot be built.

In [12], we propose a new \(VN\) embedding algorithm based on the ant colony metaheuristic, named VNE-AC, which operates as follows. First, the \(VN\) request is divided into a set of solution components that are sorted and then solved sequentially. Note that solving a solution component is only equivalent to building a small part of the overall solution. Next, a set of parallel artificial ants are launched to iteratively explore the search space until a predetermined number of iterations is reached. During each iteration, each ant incrementally constructs the solution by moving from one solution component to another. To do so, the ant localises potential candidates in the \(SN\) for each solution component in the search space then selects just one by applying a stochastic local decision. It is worth noting that the decision is based on heuristic information and artificial pheromone trails, which respectively quantify the desirability of the transition \textit{a priori} and \textit{a posteriori}. Once each ant builds its full solution, the best
one (i.e. the one that most enhances the objective function) among all solutions generated by all ants is selected. Furthermore, the artificial pheromone is slightly evaporated into the surrounding environment. This helps the ants to discover new trajectories in the environment and not to be trapped in local optimums. Nevertheless, the artificial pheromone trail of each solution component is reinforced at visited points in the environment in relation to the best trajectory travelled by ants to build the whole solution. This helps the ants to improve and continually refine the best solution obtained. The process is repeated until the upper bound of iterations is reached. The overall best solution generated by all ants is considered to be the output solution. Unlike to [8] and [10], the available substrate resources (processing power, memory and bandwidth) are considered during the mapping process. On the other hand, contrary to [9] and [10], VNE-AC coordinates the virtual node and virtual link assignment in order to enhance the mapping efficiency. Finally, unlike [11], the geographic locations are set only for access routers, whereas the locations of core routers are not constrained. Thus, the problem is more complex since the solution space is increased.

In the next section, we will evaluate the performance of the above VN embedding methods based on extensive simulations.

V. COMPARISON OF VIRTUAL NETWORK EMBEDDING STRATEGIES

In this section, we will study the efficiency of the related embedding strategies described in the previous section: i) VNE-Least [8], ii) VNE-Cluster [8], iii) VNE-Subdividing [8], iv) VNE-Greedy [9] and v) VNE-AC [12]. To achieve this, we will first of all describe our discrete event VN embedding simulator. Then, we will define several metrics aimed at evaluating the performance of the prominent existing schemes mentioned above. Finally, building on the outputs of the above simulations, we will assess and comment the results obtained. Note that we cannot compare the above methods with strategies proposed in [10] [11] since they assume a constrained hypothesis, which cannot be considered in the above strategies. Indeed, in [10], mapping is based on the upper bounds of network flows, and in [11], the authors need the geographic coordinates of all routers (access and core).

A. Simulation environment

We implemented a discrete event VN embedding simulator, using the GT-ITM tool to generate random SN and VN topologies. Note that we modelled the arrival of VN requests by a Poisson Process, with rate $\lambda_A$. We also modelled VN lifetime by an exponential distribution with a mean $\mu_L$.

As stated in [9], we set the $SN$ size to 100. In this case, the ratio of access and core nodes was fixed at 20% and 80% respectively. Furthermore, we set the $VN$ size according to a discrete uniform distribution, using the values given in [2, 10]. Since virtual access nodes are defined by customers, we assumed that each virtual node could be access or core with a probability of 0.5. In both cases ($VN$ and $SN$), each pair of nodes is randomly connected with a probability of 0.5. The arrival rate, $\lambda_A$, and the average lifetime, $\mu_L$, of $VN$’s are fixed to 4 requests per 100 time units and 1000 time units respectively. We calibrated the capacity of substrate nodes and links (i.e. processing power, memory and bandwidth) according to a continuous uniform distribution, taking the values in [50, 100]. We also set the required virtual resources according to a continuous uniform distribution, using the values given in [10, 20].

In our simulator, we then implemented the following strategies: i) VNE-Least, ii) VNE-Cluster, iii) VNE-Subdividing, iv) VNE-Greedy and v) VNE-AC. Note that the parameters of VNE-AC are calibrated as in [12]. We set the number of $VN$ requests to 2000. All simulation results of pseudo-random strategies were calculated with a confidence level equal to 99.70%. Note that tiny confidence intervals are not shown in the following figures.

B. Performance metrics

In this section, we will define the performance metrics used to assess our proposal.

1) $Q(t)$: is the reject rate of $VN$ requests during the time period $[0, t]$.

2) $C(t)$: is the average provisional cost of requests in the $SN$ during the time period $[0, t]$. Formally,

$$C(t) = \frac{\sum_{G^v \in AR_t} C(G^v)}{|AR_t|}$$

(1)

where $C(G^v)$ quantifies the mapping cost in terms of resources (bandwidth, cpu, memory) as defined in [12], and $AR_t$ is the set of accepted requests during $[0, t]$. 
3) \( \overline{R(t)} \): measures the average revenue of \( \mathcal{VN} \) requests. Note that when \( \overline{R(t)} \) is high, the embedding strategy maps requests generating a larger revenue. It can be expressed as:

\[
\overline{R(t)} = \frac{\sum_{G^v \in AR_t} R(G^v)}{|AR_t|}
\]

where \( R(G^v) \) is the request revenue as defined in [8] [9] [11].

4) \( U(t) \): measures average usage rate of substrate links at time \( t \).

In the following section, we will present the results of our simulations and summarise the key observations.

C. Evaluation results

Table I compares the reject rate and total revenue of all (i.e. 2000) \( \mathcal{VN} \) requests. It shows that \textit{VNE-AC} significantly outperforms the related strategies. We can see that \textit{VNE-AC} denies only 4.56\% \( \pm \) 0.40\% of requests. It also shows that \textit{VNE-Cluster}, \textit{VNE-Least} and \textit{VNE-Subdividing} decline the most requests (69\%, 73.90\% and 67.45\% respectively). This is mainly because the latter strategies do not consider the residual resources in the \( \mathcal{SN} \). In addition, we can see that \textit{VNE-AC} is roughly three times better than \textit{VNE-Greedy} (4.56\% \( \pm \) 0.40 \( \rightarrow \) 12.95\%).

<table>
<thead>
<tr>
<th>Strategy</th>
<th>reject rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNE-AC</td>
<td>4.56 ( \pm ) 0.40</td>
</tr>
<tr>
<td>VNE-Greedy</td>
<td>12.95</td>
</tr>
<tr>
<td>VNE-Cluster</td>
<td>69</td>
</tr>
<tr>
<td>VNE-Least</td>
<td>73.90</td>
</tr>
<tr>
<td>VNE-Subdividing</td>
<td>67.45</td>
</tr>
</tbody>
</table>

![Table I](image)

Fig. 2. Reject rate - Q

Fig. 2 illustrates the reject rate of requests \( Q(t) \) during the simulation. It shows that \textit{VNE-AC} notably minimises the reject rate compared with the related strategies, and that it does so throughout all simulations. This can be
explained by the fact that the VNE-AC algorithm aims to maximise the residual resources in the SN. Thus, it avoids more critical resources and consequently prevents congestion in the SN. As illustrated in Fig. 2, it is worth noting that the request reject process starts late with VNE-AC (at 1420 time unit) compared with the related strategies in which the best one starts at 703 time unit. Moreover, the VNE-AC reject rate peak is the lowest in Fig. 2 and then decreases during the simulation. This proves that the resources are allocated efficiently. In fact, unlike other algorithms, VNE-AC is based on pseudo-randomised node selection. As proved in algorithm design randomisation, this approach leads to effective performance in computationally intractable problems.

As shown in Fig. 3, thanks to VNE-AC’s path-distance metric, the algorithm succeeds in making a trade-off between minimising the length of substrate paths and maximising residual bandwidth in the SN. We can see that VNE-AC allocates approximately the same bandwidth in the SN but that it assigns more requests. Note that VNE-Least and VNE-Subdividing consume lower levels of resources than VNE-AC. This can be explained by the fact that VNE-Least is based on the shortest path algorithm and that VNE-Subdividing favours the mapping of small VN requests in terms of required resources (as illustrated in Fig. 5), which rationally decreases the provisional cost. However, both strategies have the highest reject rates since they do not take into account the bottleneck effect in the SN. Moreover, the figure shows that VNE-Cluster consumes high level of resources due to its path metric that does not take into account bandwidth usage.

Fig. 3. Average usage rate of links - U

It is clear in Fig. 4 that VNE-AC produces the best average provisional cost thanks to its path metric and the large number of accepted requests. We can also see that the average provisional cost of requests obtained by VNE-AC is constant throughout the simulations, even though more VN requests are embedded in the SN. It is worth noting that VNE-Cluster leads to an expensive mapping cost caused by the long substrate paths allocated to virtual links.

Fig. 5 shows that VNE-AC performs better than the other approaches tested since it provides a higher overall revenue to the SN provider. We can see that the average revenue of VNE-AC is roughly constant, in spite of incoming requests. The rest of strategies provide lower revenue due to their higher level of rejects which degrade their efficiency. We can conclude that coordinated virtual node and virtual link mapping stages lead to efficient mapping and thus generate a higher income.
VII. CONCLUSIONS

Cloud computing is a new paradigm that is built on three basic principles: i) grid computing, ii) cluster computing and iii) virtualisation. In this paper, we addressed the challenging and complex virtual network embedding problem within a cloud infrastructure network. Indeed, we briefly outlined the definitions of cloud computing and its supplied services, before proposing a new cloud computing infrastructure called the Cloud Infrastructure using Virtualization within Data centers and Backbone (CIVDB). We saw that a few virtual network embedding strategies for cloud infrastructure networks have already been proposed in existing literature, namely: i) VNE-Least, ii) VNE-Cluster, iii) VNE-Subdividing, iv) VNE-Greedy and v) VNE-AC. We compared these approaches using extensive simulations measuring i) virtual network request reject rate, ii) embedding cost of the virtual network request, iii) embedding revenue of the virtual network request, and iv) average usage rate of physical resources.

As the virtual network embedding issue is a recent problem, we believe that other methods will be proposed in the near future as the research community increasingly focuses on issues relating to cloud computing. In addition, the migration of virtual routers in a substrate network, made possible thanks to virtualisation technology, is an interesting solution for tackling bottlenecks in the demand for cloud services. We believe that migration can be used as a complementary solution in order to maximise the provider's revenue.

REFERENCES

Fig. 5. Comparison of average revenue - $R(t)$


