Data Centre Optimisation Enhanced by Software Defined Networking

Tommaso Cucinotta, Diego Lugones, Davide Cherubini, Eric Jul
Bell Laboratories, Alcatel-Lucent Ireland, Dublin, Ireland
{*name.surname*}@alcatel-lucent.com

***Abstract*—Contemporary Cloud Computing infrastructures are being challenged by an increasing demand for evolved cloud services characterised by heterogeneous performance requirements including real-time, data-intensive and highly dynamic workloads. The classical way to deal with dynamicity is to scale computing and network resources horizontally. However, these techniques must be coupled effectively with advanced routing and switching in a multi-path environment, mixed with a high degree of flexibility to support dynamic adaptation and live-migration of virtual machines (VMs). We propose a management strategy to jointly optimise computing and networking resources in cloud infrastructures, where Software Defined Networking (SDN) plays a key enabling role.**

***Keywords*-VM Placement, Data Centre Optimisation, Software Defined Networking**

I. Introduction

Information and communication technologies (ICT) are undergoing a continuous and steep evolution. The explosive increase in the availability of high-speed networks is causing a significant shift towards distributed computing models where processing and storage of data can be performed in cloud computing data centres on an on-demand basis. Moreover, many of the emerging cloud applications have precise requirements on performance and reliability while handling increasingly massive amounts of data. This requires novel techniques for dealing with advanced routing and switching that go beyond traditional network architectures and resource management strategies in cloud infrastructures. Indeed, it is critical not only to employ intelligent resource allocation mechanisms but also couple these with network architectures that are flexible and redundant. For this purpose, it is natural to see techniques typical of High-Performance-Computing (HPC) environments applied to cloud data centre design. For example, redundant network architectures designed with a multiplicity of paths among hosts may constitute an essential brick of a cloud infrastructure able to sustain huge data traffic, as needed by big-data workloads or due to instantiation and live-migration of VMs.

Unfortunately, apart from the heavily customised technologies in use within HPC data centres, common networking elements and management techniques struggle when dealing with such complex architectures, and they are completely inappropriate when it comes to flexibility, as required to meet the dynamic requirements of server virtualisation. However, recent developments in Software Defined Networking (SDN) seem to promise a radical change of the situation for better, where technologies such as OpenFlow seem to be the right solution to the above mentioned issues.

SDN technologies are known to lead to reduced operational costs through rapid and automated provisioning of network services to VMs instead of the current labour-intensive practice of manual provisioning. These benefits are achieved by separating the data and control functions and defining the appropriate programming interface between them. In contrast, most of today's routers and switches mix both functions according to proprietary vendor designs, making it hard to adjust network infrastructure when tens or hundreds of virtual machines are instantiated in the data centre. With SDN, it is possible to design logically centralised controller architectures using higher-order software programs having a full system *view*. This allows different levels of abstraction and simplicity of automation leading to the independent evolution and development of the control software and network hardware. There are several other advantages in terms of evaluation and potential virtualisation of network services, VM migration, large scale Layer 2 routing, security applications, etc. In this paper, however, we leverage the logically centralised system view to jointly optimise computing and networking resources to deliver high performance in a multi-tenant data centre.

Contributions: We present a resource management architecture for data centres based on: 1) an application model able to capture the rich resource requirements of future data-demanding cloud applications; 2) an optimisation model able to optimally map these requirements onto a cloud provider physical infrastructure, considering capacity constraints; 3) an SDN-based approach for the automatic and seamless configuration of the data centre network exploiting the output of the optimisation procedure. The approach is validated through the use of CloudNetSim [1], our OMNeT++-based simulator for cloud applications.

II. Related Work

Several works that address the problem of optimal allocation of services in Cloud systems have appeared in recent years [2], [3]. In [4], the authors examine this problem in a multi-provider hybrid Cloud setting against deadline-constrained applications. To this direction, a mixed integer optimisation problem is formulated with the objective to minimise the cost of outsourcing tasks from data centres to external Clouds, while maximising the internal utilisation of the data centres. In [5], authors address the problem of optimal VM allocation for maximising the revenue of Cloud providers by minimising the amount of consumed electricity. In [6], a resource allocation problem is formulated in which later tasks can reuse resources released by earlier tasks, and an approximation algorithm that can yield close to optimum solutions in polynomial time is presented. In [7], the same authors propose an allocation problem for vTelco applications, where arbitrary latency expressions are used to model the end-to-end latency requirements of services.

Another work focusing on placement of Telco services can be found in [8], where trade-offs between centralised versus distributed cloud architectures are investigated. In [9], the problem of optimal placement of VMs in distributed clouds for minimising latency is tackled. Complexity is reduced by recurring to a hierarchical split of the placement problem into the two reduced-complexity sub-problems of choosing the data centres in which to place, then choosing the specific racks and servers, and applying a partitioning of the application graph to be placed. In [10], a comparison is made of various algorithms for placing VMs in centralised vs distributed clouds, analysing also the impact on latency for accessing the placed services.

A probabilistic aspect to the allocation of distributed services can also be found in [11]. In prior work of ours [12], the problem of optimum allocation of real-time work-flows with probabilistic deadline guarantees was tackled. In that work, the main focus was on the probabilistic framework allowing the provider to overbook resources in the various time frames of each advance reservation request, knowing the probabilities of actual usage/activation of those services by the users. Also, in a more recent work [13], a probabilistic framework was used to model optimum allocation of horizontally scalable cloud services. Differently from these works, where modelling of the network is quite simplistic, in the present paper a complex data centre network topology with possibly multiple paths across hosts is considered, to the purpose of dealing with applications requiring significant amounts of data to transfer.

A way to estimate the probability that an end-to-end deadline is respected for a given composition of services is the one to build probabilistic models for the performance achieved by a composition of distributed services. For example, in [14] authors investigated on mathematical models to compute the probability density function of the response-time of service compositions under various compositional patterns. Also, in [15] authors modelled probabilistic interactions among multiple components of multi-tier cloud applications recurring to Markov Chains. However, in the present paper we focus on variability of the workload for services whose composition is deterministically fixed.

The use of SDN to facilitate the joint optimisation of VM allocation and selection of network paths, has been mentioned in [16]. The authors focus on a multi-path enabled data centre where traffic engineering techniques can be leveraged to route individual VM traffic. Still, there is no description nor analysis of the used SDN technology and how this optimisation is integrated with the network controller. To fill this gap, our work analyses different SDN technologies and network controllers that support efficient placement and routing of virtualised applications in a multi-tenant environment.

The major cause of the gap just mentioned is the state of maturity within the industry. These are the early days of SDN: less than 1% of all data centre systems are SDN-enabled. Currently, *OpenFlow* is the de-facto standard and new standards can be expected to emerge as industry matures. For example, the *Extensible Messaging and Presence Protocol* (XMPP), originally developed for instant messaging and online presence detection, has been proposed as an alternative to OpenFlow. Also, current standards for the data path, such as *Virtual eXtensible LAN* (VXLAN) [17] or similarly *MPLS over GRE*, are usable *"as-is"*, by "stretching" a Layer 2 network over a Layer 3 network and extending the number of isolated *broadcast* domains.

OpenFlow evolution inside the data centre *(intra data centre SDN)* is witnessed by several controller implementations available not only from industry, but also in the open-source world, such as *POX, Beacon, Floodlight, FlowER, OpenDaylight* and other new concepts and network abstractions such as *Flowvisor* [18] and *HyperFlow* [19] that allow researchers to develop isolated flow allocation strategies for multi-tenant large scale data centres. This rich set of tools simplifies significantly the integration of the optimisation solution proposed in this paper.

III. Proposed Approach

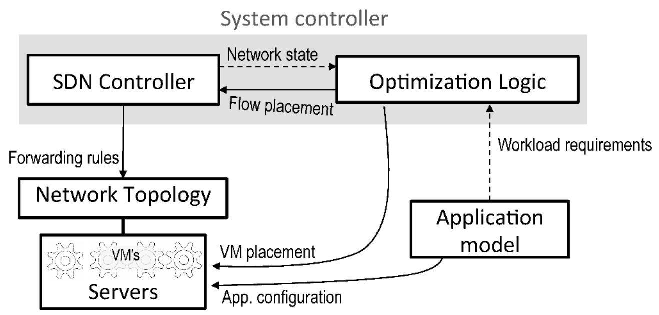


Figure 1: Overarching view of our proposed approach.

Our proposed architecture is visually depicted in Figure 1. At the heart of the system is an Optimisation Logic block responsible for deciding how to optimally accept and deploy applications on behalf of customers. Requests to deploy applications are specified in terms of a rich application model, constituted by a topology of inter-connected components with associated computing and networking requirements (the concept is largely inspired our prior work around the

ISONI Virtual Service Network specification developed in the IRMOS project [20]). The optimiser takes into account knowledge of the underlying physical topology, along with its foreseen status given the time-frame of the request to be accepted, and outputs placement decisions and routing tables that are used to automatically configure the underlying infrastructure.

A. Application Models

In our architecture we consider applications specified as generic graphs (or logical topologies), to be mapped onto the graph of the underlying physical infrastructure known to the cloud provider.

Typical application types that fall within the generic class of such DAG-like application models include, for example, interactive on-line multimedia applications and multi-tier web-based services. Typical the user interacts directly with a UI component handling details such as authentication and presentation logic. Users requests are usually forwarded towards additional components realising the core functionality, e.g., in a multi-tier web-based system, it is common to see, in addition to the web-server, an application server and a database back-end. These situations are to some extent modelled as simple linear work-flows where each tier needs certain computing and network bandwidth requirements. In case of compute-intensive operations, such as 3D rendering, physics simulations or sophisticated multimedia computations, it is common to distribute the work of one or more such components across multiple VMs to speed-up operations. This results generically into split-points and join-points within the application graph, thus the mentioned model is still able to capture this kind of scenarios, which are increasingly common and interesting in cloud environments due to the horizontal scalability capabilities of current IaaS platforms.

The application model may be associated with an end-to-end latency constraint (not shown here for the sake of brevity), as described in our prior works [12], [21].

B. Optimisation Logic

The physical resources of a cloud provider are generally considered as an interconnection of (potentially heterogeneous) networks that interconnect (potentially heterogeneous) computing nodes, in a redundant and multi-path environment. For example, various hosts in a rack are inter-connected through 1 GB/s links to a top-of-rack switch, which is inter-connected through 10 GB/s links to one or more higher-layer switches, in a reconfigurable network architecture which generally is capable of delivering packets among hosts through a few alternate paths. Formally, the network topology is characterised by the following elements.

- A set of N_H physical computing nodes, or hosts: $\mathcal{H} = \{1, \ldots, N_H\}$. Each host $h \in \mathcal{H}$ is characterised by
  - a maximum available/residual computing capacity $U_h \in \mathbb{R}^+$, which expresses the value of a given system-wide reference performance metrics;
  - a maximum available/residual storage capacity $M_h \in \mathbb{N}$, expressed in bytes.
- A set of interconnection switches $\mathcal{S}$.
- The overall set of interconnected (either computing or network) elements is denoted by $\mathcal{E} \triangleq \mathcal{H} \cup \mathcal{S}$.
- A set of physical links: $\mathcal{L} \subset \mathcal{E} \times \mathcal{E}$. Each link $l = (j_1, j_2) \in \mathcal{L}$ is characterised by the maximum/residual bandwidth $W_{j_1, j_2} \in \mathbb{R}^+$, expressed in bytes/s.
- For each hosts pair $(h_1, h_2) \in \mathcal{H} \times \mathcal{H}$, a set $\mathcal{P}_{h_1, h_2}$ of interconnection paths may be available and usable, where each path $p \in \mathcal{P}_{h_1, h_2}$ is associated with the sequence $\mathcal{P}_{h_1, h_2, p}$ of its $L_{h_1, h_2, p}$ links: $\mathcal{P}_{h_1, h_2, p} = \{(a_{h_1, h_2, p, 1}, b_{h_1, h_2, p, 1}), \ldots, (a_{h_1, h_2, p, L_{h_1, h_2, p}}, b_{h_1, h_2, p, L_{h_1, h_2, p}})\} \subset \mathcal{L}$.

The following notation is used to denote applications:

- Set of N_A applications: $\mathcal{A} = \{1, \ldots, N_A\}$.
- Each application $a \in \mathcal{A}$ is constituted by a topology of $m^{(a)}$ software components (encapsulated inside VMs): $\mathcal{A}^{(a)} \triangleq \left\{1, \ldots, m^{(a)}\right\}$, denoted also as $\left(\xi_1^{(a)}, \ldots, \xi_{m^{(a)}}^{(a)}\right)$;
  - each component $\xi_i^{(a)}$ has a computing workload of $C_i^{(a)}$, expressed in some reference performance metrics;
  - the topology is specified in terms of $m_L^{(a)}$ virtual links $\mathcal{L}^{(a)} \subset \mathcal{A}^{(a)} \times \mathcal{A}^{(a)}$;
  - each virtual link $l = (i_1, i_2) \in \mathcal{L}^{(a)}$ is associated with the networking requirements $b_{i_1, i_2}^{(a)}$ needed by $\mathcal{A}^{(a)}$ for handling the communications between $\xi_{i_1}^{(a)}$ and $\xi_{i_2}^{(a)}$, expressed in bytes/s.
- In each application topology, a special component is included representing the router $r \in \mathcal{E}$ of the data centre; then, the topology will include, within the virtual links $\mathcal{L}^{(a)}$, a virtual link between each other application component that needs to interact with the outside world and the router.

Now we can introduce the unknown allocation variables (the relationships among them is going to be clarified shortly):

- Booleans $x^{(a)}$ encoding whether or not application $\mathcal{A}^{(a)}$ is accepted by the provider for deployment within its infrastructure.
- Booleans $x_{i, h}^{(a)}$ encoding whether or not component $\xi_i^{(a)}$ of application $\mathcal{A}^{(a)}$ is deployed on physical host $h \in \mathcal{H}$.
- Derivative Booleans $x_{i_1, i_2, j_1, j_2}^{(a)}$ defined as the logical AND between $x_{i_1, j_1}^{(a)}$ and $x_{i_2, j_2}^{(a)}$.
- Booleans $z_{i_1, i_2, j_1, j_2, p}$ encoding whether or not $\xi_{i_1}^{(a)}$ and $\xi_{i_2}^{(a)}$ are placed respectively on $j_1 \in \mathcal{H}$ and $j_2 \in \mathcal{H}$ and the path $p \in \mathcal{P}_{j_1, j_2}$ available between said physical hosts is chosen for communications between

said application components. These variables allow for properly configuring the switches in the network if the application is accepted.

- Deviate Booleans $y^{(a)}_{i_1,\,i_2,\,j_1,\,j_2}$ encoding whether or not the communications among $\xi^{(a)}_{i_1}$ and $\xi^{(a)}_{i_2}$ will insist on the physical link connecting $(j_1,\,j_2) \in \mathcal{L}$.

For a given $a \in \mathcal{A}$, each component $i \in \mathcal{A}^{(a)}$ is to be placed on one physical host if the application is accepted ($x^{(a)} = 1$), or none of them if it is rejected. This constraint is expressed as: $\forall h \in \mathcal{H}$, $\sum_{j \in \mathcal{H}} x^{(a)}_{i,\,h} = x^{(a)}\ \forall a \in \mathcal{A},\ \forall i \in \mathcal{A}^{(a)}$.

For a given $a \in \mathcal{A}$, $i_1, i_2 \in \mathcal{A}^{(a)}$, $j_1, j_2 \in \mathcal{H}$, the logical AND constraint among variables $x^{(a)}_{i_1,\,i_2,\,j_1,\,j_2}$, $x^{(a)}_{i_1,\,j_1}$ and $x^{(a)}_{i_2,\,j_2}$ can be expressed as linear constraints (omitted). Also, the $z^{(a)}_{i_1,\,i_2,\,j_1,\,j_2,\,p}$ variables are clearly related to the $x^{(a)}_{i_1,\,i_2,\,j_1,\,j_2}$ as only one path needs to be chosen for each communication between $\xi^{(a)}_{i_1}$ and $\xi^{(a)}_{i_2}$. This is expressed as: $\forall a \in \mathcal{A}, \forall i_1,\, i_2 \in \mathcal{A}^{(a)}, \forall j_1,\, j_2 \in \mathcal{H}$, $\sum_{p \in \mathcal{P}_{j_1,\,j_2}} z^{(a)}_{i_1,\,i_2,\,j_1,\,j_2,\,p} = x^{(a)}_{i_1,\,i_2,\,j_1,\,j_2}$.

Furthermore, relationship among y and z variables can be made explicit as: $\forall a \in \mathcal{A} \forall i_1,\, i_2 \in \mathcal{A}^{(a)}\, \forall\, (j_1,\, j_2) \in \mathcal{L}\ :\ \sum_{h_1,\,h_2 \in \mathcal{H},\, p \in \mathcal{P}_{h_1,\,h_2} | (j_1,\,j_2) \in \mathcal{P}_{h_1,\,h_2,\,p}} z^{(a)}_{i_1,\,i_2,\,h_1,\,h_2,\,p} = y^{(a)}_{i_1,\,i_2,\,j_1,\,j_2}$.

Now we can formulate the problem as a Boolean Linear Programming (BLP) optimisation problem. The overall computing workload accepted onto each host, as well as the overall network workload admitted onto each physical link, must respect their respective maximum capacity values:

$$\sum_{a \in \mathcal{A}} \sum_{i \in \mathcal{A}^{(a)}} C^{(a)}_i x^{(a)}_{i,\,h} \quad \leq \quad U_h\ \forall h \in \mathcal{H} \tag{1}$$

$$\sum_{a \in \mathcal{A}} \sum_{i_1,\,i_2 \in \mathcal{A}^{(a)}} b^{(a)}_{i_1,\,i_2} y^{(a)}_{i_1,\,i_2,\,j_1,\,j_2} \quad \leq \quad W_{j_1,\,j_2}\ \forall\, (j_1,\, j_2) \in \mathcal{L} \tag{2}$$

Objective function: Among possible objectives of the optimisation carried out by the cloud provider, we have to introduce a suitable function to maximise. Normally, we want to maximise the overall gain of the provider, as coming from individual gain values $G^{(a)}$ associated to each application $a \in \mathcal{A}$. Also, we may want to maximise the saturation level of used resources, thus minimise the overall cost due to the use of new resources in the provider premises; therefore, for each new resource $h \in \mathcal{H}^{(new)} \subset \mathcal{H}$ or $(j_1,\, j_2) \in \mathcal{L}^{(new)} \subset \mathcal{L}$ that is not occupied yet in the data centre, the provider will incur a cost of K_h and $K_{j_1,\,j_2}$. For example, the provider might turn off parts of the data centre that are not immediately used, saving on the energy bill.

Alternatively, we may want to maximise the performance experienced by customer applications, namely to minimise the maximum saturation level among resources throughout the data centre; this implies minimising the extent to which physical resources are shared among different customers and VMs, reducing temporal interference.

The policy maximising gain while minimising cost of occupying new resources may be formalised as follows:

$$\max \sum_{a \in \mathcal{A}} G^{(a)} x^{(a)} - \sum_{h \in \mathcal{H}^{(new)}} K_h\, n_h - \sum_{(j_1,\,j_2) \in \mathcal{L}^{(new)}} K_{j_1,\,j_2}\, n_{j_1,\,j_2} \tag{3}$$

where n_h, $h \in \mathcal{H}^{(new)}$ are new derivative Boolean variables easily defined as logical OR of the $\left\{ x^{(a)}_{i,\,h} \right\}$ variables

$$\sum_{a \in \mathcal{A}} \sum_{i \in \mathcal{A}^{(a)}} x^{(a)}_{i,\,h} \quad \geq \quad n_h \tag{4}$$

$$\sum_{a \in \mathcal{A}} \sum_{i \in \mathcal{A}^{(a)}} x^{(a)}_{i,\,h} \quad \leq \quad \sum_{a \in \mathcal{A}} m^{(a)}\, n_h, \tag{5}$$

and $n_{j_1,\,j_2}$, $(j_1,\, j_2) \in \mathcal{L}^{(new)}$ are new derivative Boolean variables defined as logical OR of the $\left\{ y^{(a)}_{i_1,\,i_2,\,j_1,\,j_2} \right\}$ variables

$$\sum_{a \in \mathcal{A}} \sum_{(i_1,\,i_2) \in \mathcal{A}^{(a)}} y^{(a)}_{i,\,j_1,\,j_2} \quad \geq \quad n_{j_1,\,j_2} \tag{6}$$

$$\sum_{a \in \mathcal{A}} \sum_{i_1,\,i_2 \in \mathcal{A}^{(a)}} y^{(a)}_{i_1,\,i_2,\,j_1,\,j_2} \quad \leq \quad \sum_{a \in \mathcal{A}} \left(m^{(a)} \right)^2 n_{j_1,\,j_2} \tag{7}$$

Remarks: In this section, we presented a formalisation of the problem of optimal resource management in a cloud infrastructure, as a standard Integer Linear Programming (ILP) optimisation program. This can be solved in various ways, including recurring to standard solvers, as done for the experiments shown later in this paper, or writing custom and heuristic solvers, to conveniently speed-up computations.

The output of the above problem is constituted by:

1) the variables $x^{(a)}$ which, for each $a \in \mathcal{A}$, tell us whether to accept the application or not;
2) the variables $x^{(a)}_h$ which, for each $a \in \mathcal{A}$ and $h \in \mathcal{H}$, tell us whether component $\xi^{(a)}_i$ of $\mathcal{A}^{(a)}$ is to be deployed on host h or not;
3) the variables $y_{a,\,i_1,\,i_2,\,j_1,\,j_2}$ which, for each $a \in \mathcal{A}$, $i_1,\, i_2 \in \mathcal{A}^{(a)}$ and $(j_1,\, j_2) \in \mathcal{L}$, tell us whether traffic among $\xi^{(a)}_{i_1}$ and $\xi^{(a)}_{i_2}$ traverses link $(j_1,\, j_2)$, allowing us to properly configure the forwarding tables in the network, when SDN technologies are used.

We introduced two possible optimisation functions in said formulation, one to minimise costs incurred by the provider and the other one to minimise temporal interference among hosted applications. We will see, through a set of simulations, how these different policies impact on the overall performance as measured by applications.

C. SDN controller

A SDN controller enables network operators to programmatically modify network flows independently from physical network devices. In order to design our controller, we have considered a number of mandatory requirements for data centres hosting current and future cloud services: 1) End-hosts should be able to communicate efficiently with each other in the data centre using some of the -potentially-multiple communication paths. 2) Logic forwarding loops between pairs of hosts must be avoided 3) The controller must provide network support for any VM to migrate to any host without changing their IP addresses (this avoids breaking pre-existing TCP connections and session state). 4) The controller receives connectivity information from the optimisation logic (section III-B), translates this information into consistent forwarding rules and distribute them among the data centre switches). 5) A coherent *logical view* of the topology is mandatory for the proper management of the network. Therefore, the controller must periodically monitor the network and update the optimisation logic in case of any topological change (e.g., link failure).

As mentioned in section II, there are many technologies that can provide SDN functionality at some extent. However, we have designed our controller based on OpenFlow because there is a limitation in the other technologies using existing layer 2/layer 3 standards to provide *"as-is"* SDN solutions. That is, existing layer 2 and layer 3 network protocols face some combination of limitations in current data centres designs: Current IP networks require massive effort to configure and manage. Ethernet is vastly simpler to manage, but it lacks scalability as it relies on network-wide flooding to locate end hosts resulting in large state requirements and control message overhead that grows with the size of the network. Additionally, Ethernet forces paths to comprise a spanning tree thus avoiding any path redundancy and leading to poor utilisation of links in current multi-path data centre networks.

The OpenFlow protocol can combine the scalability of IP and the simplicity of Ethernet making it possible to build a system that maintains the same configuration-free properties as Ethernet, yet scales to large networks. This is possible because OpenFlow breaks the inflexible TCP/IP protocol stack allowing network switches to forward packets according to the information (headers) of different layers. This *vertical movement* in the protocol stack enables a richer set of forwarding rules that make it possible to identify any flow and allocate it anywhere in the network.

Specifically, our controller has been designed using a modular approach that allows a composition of functions that interact with the optimisation logic. We enhanced existing modules in the POX controller framework.[1] to implement the following functionality:
1) The *Host tracker* function monitors the location, MAC and IP addresses of hosts in the network. 2) Switches connectivity is monitored by the *Topology discovery* module using LLDP packets. 3) The *ARP responder* module replies to ARP queries using information received from the optimisation logic to avoid ARP broadcasts that can harm communication performance.
The information gathered by the Host tracker and Topology discovery modules is sent to the optimisation logic optimising virtual machines and network flows allocation. The SDN controller receives the configuration of flows in the network and creates the matching rules via the *Rule Creation* module. Finally, the resulting forwarding rules are sent to the data plane implementing the optimal configuration.

IV. Evaluation

We prototyped our design within CloudNetSim [1] and OMNeT++ [22], a mature, open-source simulation framework widely used—especially in networking research. OMNeT++ is a versatile framework that allows for a modular model design, while it also provides a simulation kernel and a handy API for modelling parallelisation. We leveraged pre-existing models, specifically by adapting the baseline models provided by the INET framework[2] to simulate servers and OpenFlow-compliant switches. However, as OMNeT++ is mainly geared at network simulations, we developed our own models for simulating CPU scheduling [1] and specifically hierarchical scheduling of VMs and applications within VMs, as needed in investigations on multi-tenant cloud environments under aggressive server consolidation policies.

Application Models: We developed a simple application model simulating a linear work-flow of VMs. User requests are triggered at a pre-fixed period, and they traverse in sequence the whole chain, needing a prefixed amount of processing time on each VM, then they are handed over to the next VM by sending a UDP message of a pre-fixed size. In the simulations performed for this paper, we used two-stages work-flows only, where we measured the round-trip times as measured by the client between sending each request and receiving the corresponding reply.

Network Topology: We ran a few exemplifying experiments simulating a small data centre with 8 available hosts distributed within 2 pods (see Figure 2), each including two ToR switches and two aggregation switches connected in a fat-tree-like structure, and a final core switch connecting the aggregation switches of both pods.

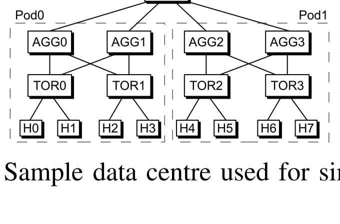


Figure 2: Sample data centre used for simulations.

This arrangement of is typical of data centre architectures for HPC, and is a classical, cheap and scalable way of deal-

[1]More information at: http://www.noxrepo.org/pox/documentation/

[2]More information at: http://inet.omnetpp.org/.

ing with data-intensive applications whose (aggregate) traffic would not fit within a traditional tree-only architecture.

A. *System Controller*

In our experimentation, the optimisation logic described above has been prototyped by using an exact formulation of the placement problem as a BLP optimisation program. This has been automatically generated and submitted to the GNU Linear Programming Toolkit (GLPK) solver. The obtained solution was automatically transformed into VM deployment decisions and switches forwarding table configurations.

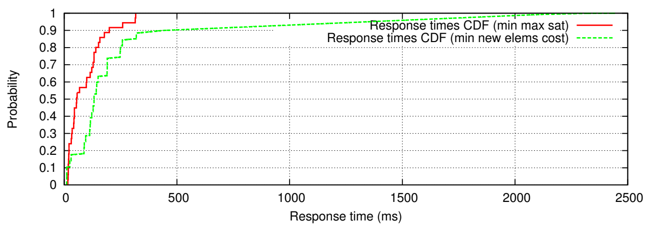
The latter configurations, within our CloudNetSim OMNeT++ based simulation environment, amounted to generating automatically two configuration files (an `.xml` and an `.ini` file) that are used to automatically populate the OMNeT++ topology representing the system, along with all the needed parameters. Among these files, we generate automatically models representing multiple VMs deployed on the same server and CPU, when needed. We also generate automatically the routing tables needed by the switches for dealing properly with the traffic, as needed by our optimisation framework. Actually, we could only use a partial implementation of the per-flow routing as enabled principally by SDN, however, a full implementation of the overall approach, including the simulation of an OpenFlow compliant SDN controller, is under way.

B. *Simulation Results*

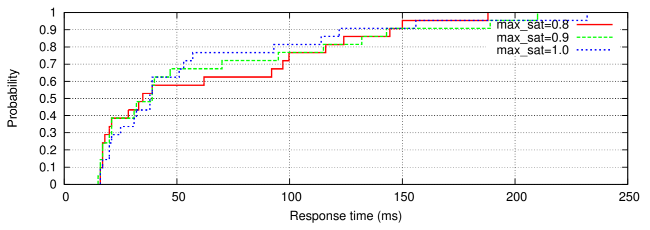
Through our simulation framework, we analysed the impact of the two optimisation objectives introduced above on the deployed applications performance. To this purpose, we deployed a variable number of dual-VM applications between 21 and 34, with randomly generated computing times and needed forward and backward message sizes. In order to present a compact representation of the obtained response-times, we use the obtained cumulative distribution function (CDF) of the overall set of response-times as gathered during 4 seconds of simulated time.

Figure 3.(a) compares the obtained CDFs when configuring the optimiser with the 2 different optimisation objectives described above: minimise cost of new occupied elements (red curve) vs minimise maximum elements saturation level (green curve). As it is expectable, the former policy pushes towards overly aggressive consolidation levels (both on networking and on computing resources), resulting in higher response-times. As the figure highlights, it is noteworthy that the worst-case response-times (nearly (2.5s) are greatly worse than the ones achieved by the latter policy (nearly 300ms). Indeed, this last policy tends to spread the workload across the available resources, minimising the level of sharing, resulting in higher responsiveness of the deployed applications.

Figure 3.(b) reports the CDFs obtained in a scenario with a lighter overall load, when the objective function minimising



(a)



(b)

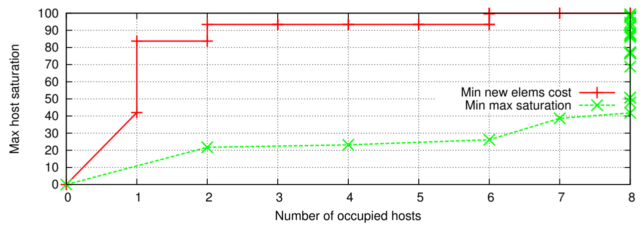
Figure 3: CDFs of the obtained response times, for the 2 different optimisation objectives (a) and with varying saturation thresholds (b).

cost of occupied resources was used, with different maximum saturation thresholds. In this case the load imposed on the data centre was not as high as in the case of Figure 3.(a). Still, it can be noticed that, as the saturation threshold allowed for the resources approaches 1.0, the worst-case response-times of the deployed applications tend to increase.

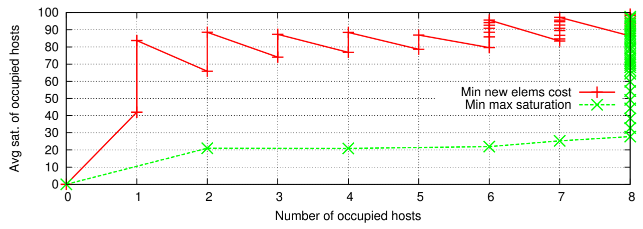
The difference in behaviour of the two optimisation policies may also be highlighted by observing some statistics on the data centre overall saturation levels. Figure 4.(a) reports the maximum saturation level of computing resources versus the number of occupied hosts as new applications are progressively accepted and deployed. As it can be seen, when minimising cost of new resources (red curve), the workload is initially kept very packed on a few hosts, then only when strictly needed, new hosts are progressively occupied. When minimising the maximum saturation (green curve), instead, applications are progressively deployed spreading across all available resources, and only when all of them are already loaded, does the policy start to re-deploy on the same resources, increasing progressively the maximum saturation level.

Figure 4.(b) is similar, but it reports the average saturation of only occupied hosts on the Y axis. In this case, the red curve goes up and down because the workload first fills a few hosts, then a new completely unloaded host is occupied, causing the average saturation of occupied hosts to drop, then fill again with more applications, until a new host is needed again, and so on.

Finally, focusing on the policy minimising the cost of new resources, we use a similar set of plots to compare what happens with the three maximum saturation levels of 0.8, 0.9 and 1.0. Figure 5 reports the obtained statistics.

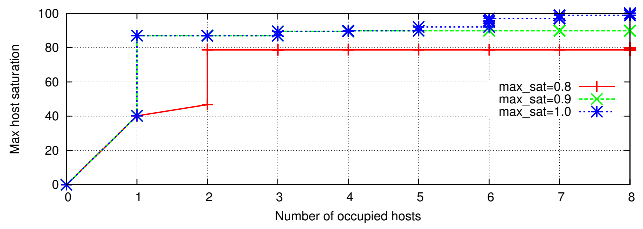


(a)

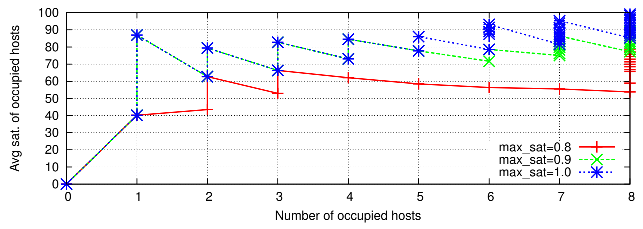


(b)

Figure 4: Obtained max saturation (a) and average saturation of occupied hosts (b) vs the number of occupied hosts as new applications are progressively accepted – comparison between the two considered objective functions.



(a)



(b)

Figure 5: Obtained max (a) and average (b) saturation of occupied hosts vs number of occupied hosts.

C. Solving Time

We applied the optimum GLPK-based solver to a number of scenarios obtained varying the number of pods and hosts per pod composing the topology. We measured the solving times obtained for these configurations using GLPK v4.45 running on an Intel(R) Core(TM)2 Duo CPU P9600 @ 2.66GHz (the solver does not take advantage of multiple CPUs, so only 1 CPU was used by GLPK).

Results are reported in Figure 6, where the X axis reports the overall number of hosts for the configuration, and the Y axis reports the solving time. The latter increases more than exponentially with the number of overall hosts in the physical topology (note the logarithmic scale on the Y axis). Therefore, the optimum solver can only be used with a small number of elements composing the physical topology.

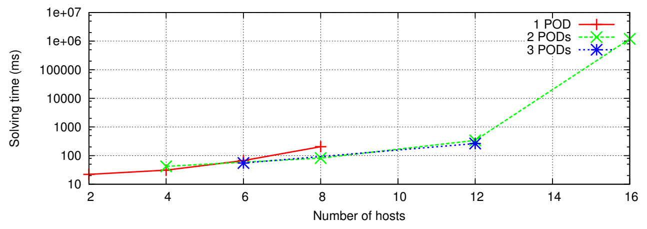


Figure 6: Solving time needed in a few sample scenarios

However, it is possible to apply a hierarchical decomposition of the optimisation problem, invoking a first instance of the solver figuring out which Pods to place the applications onto, then a second instance taking care of detailed placement within each Pod. For example, placement on 16 hosts spread across two pods can be dealt with:

- by applying the optimum solver for 2 pods and 16 overall hosts, taking up to 1 million ms (nearly 15 minutes) of solving time, as shown in Figure 6;
- by applying the optimum solver to a reduced problem placing onto two logical computing elements (the pods), taking nearly 20ms, then applying the solver again to the problem of placing within each pod the allocated components, across 8 hosts only, taking nearly 200ms; thus we would get a solution in nearly 220ms.

We defer an extensive evaluation of the effectiveness of hierarchical decomposition for large systems as future work.

V. Conclusions and Future Work

In this paper, we presented our design for optimum management of resources within cloud data centres, focusing on applications requiring massive data transfers and tight timing constraints. Our approach is based on an Optimisation Logic component, relying on knowledge of resource requirements of applications and status of the underlying physical resources. Then, an automatic SDN-based configuration of the network is leveraged to deal with redundant network architectures, allowing for per-VM and per-flow placement and routing of application workloads. A preliminary evaluation by simulation of our optimisation logic has been presented, highlighting the impact of its objective function on the achieved response-time of placed applications. Also, we reported a set of measurements on the solving time needed by the GLPK solver to provide an optimum solution, and we provided hints as to how to speed-up the solving process by hierarchical decomposition.

We plan to extend the described work in several directions. First, a prototype of the presented overall design is planned to be developed by modifying the OpenStack open-source framework. Second, we plan to tackle the optimisation problem as formalised in this paper by recurring to heuristics (e.g., along the lines of our prior work in [23]) that are faster and more scalable than an optimum general-purpose ILP solver, which thus may be more appropriate for dealing with dynamic scenarios in which placement decisions have to be taken dynamically. Third, the presented optimisation framework is being extended along various probabilistic paths, including considering applications with horizontal scalability requirements, along the lines of reasoning behind our prior research as appeared in [12], [13].

References

[1] T. Cucinotta and A. Santogidis, "Cloudnetsim - simulation of real-time cloud computing applications," in *Proc. of the 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2013)*, Paris, France, July 2013.

[2] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010.

[3] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Trans. on Services Computing*, vol. 5, no. 4, 2011.

[4] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010.

[5] M. Mazzucco, D. Dyachuk, and R. Deters, "Maximizing cloud providers' revenues via energy aware allocation policies," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010.

[6] F. Chang, J. Ren, and R. Viswanathan, "Optimal resource allocation in clouds," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010.

[7] F. Chang, R. Viswanathan, and T. L. Wood, "Placement in clouds for application-level latency requirements," in *IEEE CLOUD*, 2012, pp. 327–335.

[8] X. An, F. Pianese, I. Widjaja, and U. G. Acer, "Dmme: A distributed lte mobility management entity," *Bell Labs Technical Journal*, vol. 17, no. 2, pp. 97–120, 2012.

[9] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. INFOCOM*, 2012.

[10] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: automated data placement for geo-distributed cloud services," in *Proc. of the 7th USENIX conference on Networked systems design and implementation*, ser. NSDI'10, Berkeley, CA, USA, 2010, pp. 2–2.

[11] Z. Zhang, H. Wang, L. Xiao, and L. Ruan, "A statistical based resource allocation scheme in cloud," in *Cloud and Service Computing (CSC), International Conference on*, Dec. 2011.

[12] K. Konstanteli, T. Cucinotta, and T. Varvarigou, "Optimum allocation of distributed service workflows with probabilistic real-time guarantees," *Springer Service Oriented Computing and Applications*, vol. 4, no. 4, pp. 229–243, 10 2010.

[13] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou, "Admission control for elastic cloud services," in *Proc. of the IEEE Fifth International Conference on Cloud Computing (CLOUD 2012)*, Honolulu, Hawaii, USA, June 2012.

[14] H. Zheng, J. Yang, and W. Zhao, "QoS probability distribution estimation for web services and service compositions," in *IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, December 2010, pp. 1 –8.

[15] F. Zhang, J. Cao, H. Cai, J. Mulcahy, and C. Wu, "Adaptive virtual machine provisioning in elastic multi-tier cloud platforms," in *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, 2011, pp. 11–19.

[16] J. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *INFOCOM, 2012 Proc. IEEE*, 2012, pp. 2876–2880.

[17] D. Dutt *et al.*, "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," 2013.

[18] R. Sherwood *et al.*, "FlowVisor: A Network Virtualization Layer," Tech. Rep., October 2009.

[19] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow," in *Proc. of the 2010 internet network management conference on Research on enterprise networking*, ser. INM/WREN'10, San Jose, CA, USA, 2010.

[20] T. Cucinotta, F. Checconi, G. Kousiouris, D. Kyriazis, T. Varvarigou, A. Mazzetti, Z. Zlatev, J. Papay, M. Boniface, S. Berger, D. Lamp, T. Voith, and M. Stein, "Virtualised e-learning with real-time guarantees on the irmos platform," in *Proc. of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2010)*, Perth, Australia, 12 2010, pp. 1–8.

[21] K. Oberle, D. Cherubini, and T. Cucinotta, "End-to-end service quality for cloud applications," in *Proc. of the 10th International Conference on Economics of Grids, Clouds, Systems and Services (GECON)*, Zaragoza, Spain, Sept. 2013.

[22] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proc. of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, ser. Simutools '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 60:1–60:10.

[23] T. Cucinotta and G. Anastasi, "A heuristic for optimum allocation of real-time service workflows," in *Proc. of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2011)*, Irvine, CA, December 2011.