Network-Aware Impact Determination Algorithms for Service Workflow Deployment in Hybrid Clouds

Hendrik Moens*, Eddy Truyen†, Stefan Walraven†, Wouter Joosen†, Bart Dhoedt* and Filip De Turck*
* Ghent University, IBBT Department of Information Technology
Gaston Crommenlaan 8/201, B-9050 Gent, Belgium
† Katholieke Universiteit Leuven, DistriNet Research Group, Dept. Computer Science
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
e-mail: hendrik.moens@intec.ugent.be

***Abstract*—In recent years, many service providers have started migrating their service offerings to cloud infrastructure. Sometimes, parts of the service workflow can however not be moved to cloud environments. This can occur due to client policies, or because some services are linked to physical client-site devices. The result of the migration is then a hybrid cloud environment, where part of the services are executed within the client network, while most of the processing is moved to the cloud.**

Migration to the cloud enables a more flexible deployment of services, but also increases the strain on underlying networks as most tasks are partially handled in a remote cloud, and no longer just in the local network. An important question that providers must answer before new service workflows are deployed is whether they can provide the workflow with sufficient quality of service, and whether the deployment will impact existing service workflows. In this paper we discuss strategies based on multi-commodity flow problems, a subset of graph flow problems that can be used to determine whether new service workflows can be sufficiently provisioned, and whether the addition of new workflows can negatively impact the performance of existing flows. We evaluate the proposed solution by comparing the performance of three approaches with respect to the number of successful workflows and with respect to their execution speed.

***Index Terms*—Hybrid clouds, Distributed Computing, Workflow Deployment**

I. Introduction

Traditionally, many service providers install and maintain servers and devices on a client's site. Upgrading a service, or adding new features to an existing service, often increases the load on management servers. If these servers are located client-site, it may be required to add new servers, or upgrade existing ones. The requirement of upgrading servers significantly increases costs, and delays the roll-out of services. For some customers, this additional cost may be prohibitive, preventing them from using some of the offered services.

Migrating these management servers to the cloud resolves these issues, as resources can be added on-demand, and nearly instantaneously. Sometimes, specific tasks can however not be executed in the cloud, as they must be executed at the client site, for policy reasons, or because they make use of physical devices present on-site. An example of this can be found in medical communication systems, where physical, on-premise devices are needed in every room in a hospital. This leads to a hybrid cloud where a part of the services are executed in a remote cloud environment, with near infinite capacity, and part of the services are executed in a client-site network, with network and server capacity constraints.

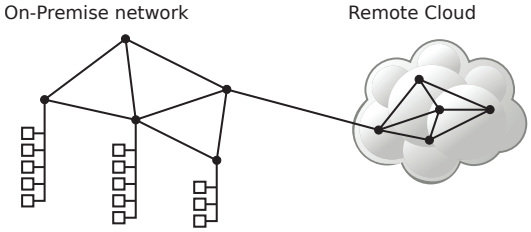


Fig. 1: An example configuration. An on-premise network containing physical terminals that interact with users is connected to a cloud that handles relocatable tasks.

While migrating servers to the cloud enables a more dynamic selection of offered services, moving the services further away from the on-premise devices impacts the underlying network, as the services must still communicate with each other. The available services are still limited depending on the capabilities of both the required devices, and the capacity of the client-site underlying network. While service selection can become more dynamic in the cloud, it is important that the service provider, who offers the services, can determine whether he is capable of providing services that require on-premise resources with sufficient quality guarantees before they are deployed, and that the provider can ensure the service will not interfere with other, already deployed services.

The setup is illustrated in Figure 1, where an on-premise, client-site network, and a cloud are connected. Several terminals, illustrated as boxes, exist within the on-premise network, and their functionality cannot be moved to the cloud. This paper describes a strategy for determining whether service workflows can be provisioned on the network created by the client-site network of which the topology is known, in combination with a public cloud with (near) infinite capacity, and what the impact of these workflows on existing flows is. In our analysis we focus on a medical communication system

978-3-901882-48-7

use case. The bottleneck in the system is assumed to be in the private environment and the uplink to the public cloud environment, so the topology of the public cloud does not have to be known. We will refer to this as the Network-Aware Impact Determination (NAID) problem. The developed algorithms determine whether services can be provisioned, and the quality with which they can be provisioned.

To achieve this, we make use of multi-commodity flows [1], a specific category of network flow problems. Specifically, we describe an extension of the maximum concurrent flow problem [2], that is workflow-aware and maximizes the realized demand of individual workflows.

In the context of this paper, a service workflow is a sequence of services that either communicate continuously, or for which a certain bandwidth must be reserved during a specific timeframe. Possible timeframes can, for example, be the day or night shift in a hospital. It is impossible to know beforehand when in the timeframe a workflow will need to be executed, and considering the medical use case, it is of critical importance that the flow can always be executed. To prevent unnecessary restrictions on the algorithm results, small enough timeframes should be chosen, and if a workflow executes during multiple timeframes, the NAID algorithm must be executed for each of the used timeframes. The process of choosing such timeframes is however out of scope for this paper. We also assume services are CPU constrained, as this applies in our use case, but we also explain how additional constraints such as disk I/O could be handled.

The remainder of this paper is structured as follows. The next section describes related work. In Section III, we will detail the NAID problem parameters. Subsequently, we will discuss the multi-commodity flow problem in Section IV. In Section V we will formally describe three NAID algorithms, based on a conversion of the problem to a multi-commodity flow graph. This is followed by Section VI, where we evaluate the approach, after which we will state our conclusions in Section VII.

II. Related work

Multi-commodity flow problems [1] are a specific class of network problems, and can be used to model various network-problems such as several network routing problems [3], [4], [5], virtual network allocation [6], and design of fault-tolerant networks [7]. These approaches however work on the network level, and focus on routing flows from one network node to another. We on the other hand add service information to the input network, and focus on service-to-service routing: only the service that is executed matters, not where this service is executed, as long as server constraints are respected.

The approach described in this paper has similarities with the application placement problem [8]. Application placement is used to determine the location of applications within networks [8], [9], [10] or clouds [11], [12], [13], taking into account the demand for each application. Application placement is used to coordinate applications. This work however focuses on the coordination of service workflows, rather than the management of individual services. In [14] network-aware placement of services is discussed, but the focus is the management of datacenters with specific layouts, so the techniques discussed cannot be directly applied to client-site networks. Furthermore, the system assumes bandwidth is the only limitation, ignoring CPU limitations. Our approach however incorporates CPU limitations and can be applied to varying network layouts. In [9] and [15], an application placement algorithm based on a conversion to a network problem is discussed, but the physical network is not taken into account. Our work by contrast specifically focuses on the underlying network.

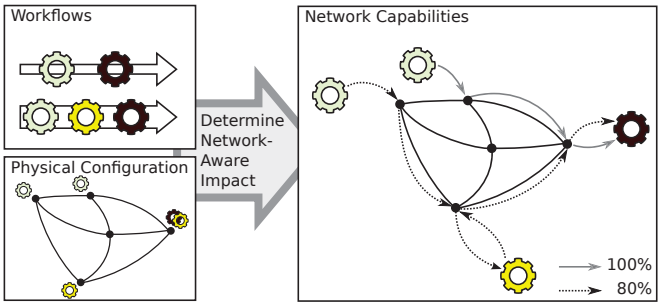


Fig. 2: The Network-Aware Impact Determination (NAID) problem takes as input a collection of service workflows and a network containing servers on which services run. As an output, the problem determines the share of the requested capacity that can be placed for each workflow.

Our approach further differs from application placement approaches as we assume that the services are already placed. We rather focus on determining which service workflows can successfully execute, given a specific configuration. Thus the approach discussed in this work can be used in conjunction with existing application placement techniques, the application placement techniques being used to determine the service locations, and the NAID algorithms to determine the achievable workflows taking into account these service locations.

The NAID problem is similar to the service matching problem [16]. As in [17], we assume the service specification is known, but while the authors relax the capacity limit to achieve a polynomial time algorithm, we on the other hand focus specifically on these capacity constraints. By focusing on whether the required capacity for offering the services is present in the network, rather than on which specific service instances are used within the compositions, we similarly achieve polynomial time algorithms.

III. Network-Aware Impact Determination (NAID)

An overview of the NAID problem is shown in Figure 2. As an input, the problem takes a collection of service workflows, and a physical configuration. A service workflow is a sequence of services that communicate with each other. We focus on workflows that are repeated frequently. If many workflows exist that are only executed once, generic workflows can be determined based on this information, and used as input for the NAID algorithms. Different workflows can require different

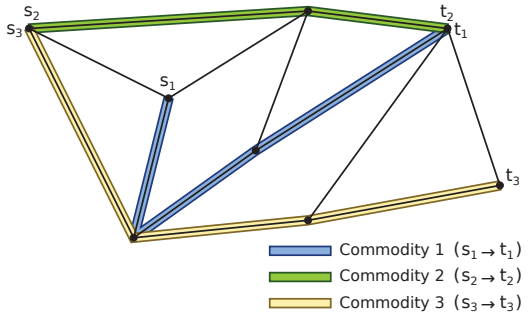


Fig. 3: An example multi-commodity flow with three flows. Sources are marked with s, sinks with t.

amounts of network capacity, and the capacity that must be provisioned can differ between subsequent services, as one service could e.g. implement a filter, drastically reducing bandwidth needed for subsequent steps. The physical configuration is determined by the servers, the network topology, and by the services that are active on these servers.

The NAID algorithm determines a possible flow on the network, respecting network and server capacities. The result of the algorithm is, for every workflow, the amount of its requested capacity that can be provisioned. The approach focuses on the feasibility of realizing the flows, and not of other quality metrics of the flows such as end-to-end delay, latency or cost. To incorporate such metrics, cost and latency models, such as those used in [18], could be used to add additional constraints, preventing some links from being used by some of the workflows.

In Figure 2, an example is shown with two workflows making use of three services. The physical network consists of five servers, and instances of the different services are running on specific servers. The algorithm then determines a possible network flow for both workflows, resulting in the example in 100% demand satisfaction for the first workflow, and 80% satisfaction for the second workflow.

In our approach, a single service can be used in multiple service workflows, services can be instantiated on multiple servers, and multiple services can exist on one server.

IV. Multi-commodity flow

Flow network problems are a class of problems where a flow is moved from a source to a sink within a directed network where edges have a limited capacity. Well-known examples are the maximum flow problem, where the maximum possible flow between source and sink is determined, and the minimum-cost flow problem, where a given amount of flow must be moved between a source and sink node at the minimum cost.

A multi-commodity flow problem [1] is an extension of these flow network problems, where more than one source and sink can exist, and not one, but multiple flows must be routed through the network. An example of a multi-commodity flow problem is shown in Figure 3. In this example, three commodities exist, resulting in three separate flows. The difference between multi-commodity flows and regular flows with multiple source and sink nodes, is that, in the latter case, the flow entering the sink can come from any of the source nodes, while in the multi-commodity flow system, only flow from a specific source may enter the sink.

A specific multi-commodity flow problem is the maximum concurrent flow problem [2]. The maximum concurrent flow problem is an optimization problem that strives to maximize the share of demand of each commodity that is satisfied. This problem treats all commodities equally, and ensures that an equal share z of the demand of each commodity is met.

Multi-commodity flows are a natural representation of many networking problems, as in these scenarios multiple servers communicate with each other, and the communication flows must move from a specific server to a specific target server. The maximum concurrent flow problem is an interesting starting point when considering the NAID, as unlike most multi-commodity flow problems, which are NP hard, it can be represented using linear programming (as opposed to integer linear programming), making it solvable in polynomial time [19]. We will, in Section V, extend the basic maximum concurrent flow problem to handle workflows, where apart from a source and a sink, multiple intermittent nodes exist without losing this polynomial character.

We will now formally define the maximum concurrent flow problem for a capacitated directed graph $G(N, E)$, with a collection of nodes N, and a collection of edges E. A capacity value $\mathrm{cap}(e)$ is associated with every edge $e \in E$.

Within this network, multiple commodities C exist. Each commodity has a source and a sink, and a demand $\mathrm{d}(c)$ between both. The flow passing over network edges $e \in E$ for commodity $c \in C$ is represented by the variable $\mathrm{f}(e, c)$.

The objective of the optimization is to find a network flow that moves a maximum percentage of demand from the source to the sink of the commodities. This percentage is represented by the variable z, thus making the optimization objective $\max(z)$.

The optimization is subject to two constraints. First, there is a *flow conservation constraint*, shown in Equation (1). For this, we first define $\mathrm{f}(n, c)$, the net flow for a commodity $c \in C$ on a node $n \in N$, in Equation (2). This value is determined by subtracting the sum of outgoing flows from the sum of incoming flows. For nodes n that are neither source nor sink of a commodity c, this sum must be zero, as no flow may be lost. For source nodes, only outgoing flows exist, ensuring the total flow in the node is negative. For sink nodes, that only have incoming flows, the total flow is positive.

$$\mathrm{f}(n,c) = \begin{cases} -z \times \mathrm{d}(n_1) & \text{If } n \text{ source of } c \\ z \times \mathrm{d}(n_1) & \text{If } n \text{ sink of } c \\ 0 & \text{Otherwise} \end{cases} \quad (1)$$

$$\mathrm{f}(n,c) = \sum_{(m,n)\in E} \mathrm{f}((m,n),c) - \sum_{(n,m)\in E} \mathrm{f}((n,m),c) \quad (2)$$

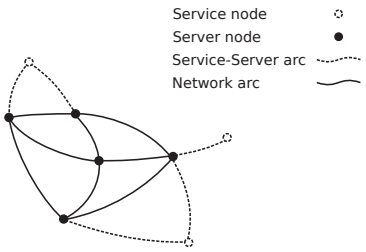


Fig. 4: The NAID problem as a multi-commodity flow network. Each arc in the Figure consists of two directed edges.

The second constraint is the *capacity constraint*, shown in Equation (3), which is added for every edge $e \in E$, and ensures that the sum of all flows passing over an edge does not exceed the edge's capacity.

$$\sum_{c \in C} \mathrm{f}(e, c) \leq \mathrm{cap}(e) \tag{3}$$

V. Algorithms

We will first describe how NAID can be converted to a graph problem. Subsequently, we will define extensions to the maximum concurrent flow problem, ensuring the resulting linear problem formulation can be applied to the graph to solve the NAID problem.

A. Graph model

The general concept of multi-commodity flows can be used to model the capacity used by service workflows in networks. To achieve this, the problem described in Section III must first be adapted to a graph. An example of such a network is shown in Figure 4. The graph contains nodes for all servers and services. The servers are connected with edges according to the physical network, and capacity constraints are added for these edges based on the capacity of the links. Services are connected to the servers on which they execute by adding links in two directions. The capacity of these edges is unlimited, as limitations to server bandwidth are handled by the edges between server nodes. These edges will still be subject to other constraints, that will be described in the next section, as using this edge implies a service on a server is used, which in turn consumes CPU server resources. Within this approach, routers can be included and modeled as a server on which no services are running.

Workflows can be constructed by creating a commodity for every pair of services, and chaining these workflow commodities together. For a workflow

$$w_i : a \xrightarrow{d_{a,b}} b \xrightarrow{d_{b,c}} c$$

this implies creating two separate commodities: (a, b) with demand $d_{a,b}$ and (b, c) with demand $d_{b,c}$. When the multi-commodity flow algorithm is executed, two flows will be created for the commodities. Together, these flows form the entire workflow. This is illustrated in Figure 5, where a relevant fragment of a larger network is shown. In the figure, the workflow is executed as follows:

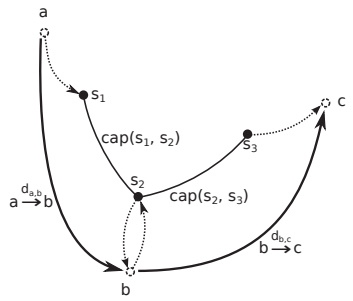


Fig. 5: The workflow between services a, b and c consists of two separate commodities, (a, b) with demand $d_{a,b}$ and (b, c) with demand $d_{b,c}$.

1) First the service a is executed on server s_1.
2) The result of the execution is moved over the network to s_2.
3) The service b is executed on server s_2.
4) The result of the second execution is moved over the network to s_3.
5) The service c is executed on server s_3.

This approach can easily be extended to non-linear workflows, used in e.g. broadcasting scenarios, by linking multiple commodities together in different ways. In this paper, we will however focus specifically on linear workflows applicable to medical communications systems.

It is possible that a single service runs on multiple servers, as illustrated in Figure 4. In this case the optimization process will route flow over the service instance that results in the best global z value. Special care must be taken to ensure that, when an incoming arc to a service is used, its corresponding outgoing arc, going back to the same server is used. This is achieved by adding specific constraints that are discussed in the next subsection.

B. Formal model

In the previous section we have described how the NAID problem can be converted to a graph problem that can be solved using a variant of the maximum concurrent flow problem. With regard to the formal problem as described in Section IV, a few constraints must be added to take into account server CPU limitations and to ensure the different commodities can be correctly chained together to create service workflows.

The problem makes use of a collection of servers $s \in S$, that are connected using edges $(u, v) \in S^2$ with capacity $\mathrm{cap}(u, v) \in \mathbb{R}$. Each server s has available CPU_s, and does not have bandwidth limitations as these are handled by the network capacities.

Additionally, a set of services $a \in A$ exits. We assume that the input flow c_a^{in} of a service a is proportional to its output flow c_a^{out}. Using a proportionality constant for the service this is expressed as $c_a^{in} = \mathrm{r}_{in}^{out}(a) \times c_a^{out}$. We use this rate as separate steps in a workflow can have different bandwidth requirements as discussed in Section III. Similarly, we assume that the amount of CPU resources utilized by

the service are related to the throughput of the service, so $CPU_a = \mathrm{r}_{in}^{CPU}(a) \times c_a^{in}$.

The system also contains a collection of workflows, $w \in W$, represented as a chain of services and demands:

$$w_i : a_{i_1} \xrightarrow{d_{1,2}} a_{i_2} \xrightarrow{d_{2,3}} \cdots \xrightarrow{d_{n-1,n}} a_{i_n} \tag{4}$$

An entire workflow can be characterized using a sequence of services and an input demand d_{w_i}. The demand for subsequent links can be determined using the $\mathrm{r}_{in}^{out}(a_i)$ variables, as it relates input and output flows, and as the output flow of a service in the sequence is the input flow of the next service. The resulting recursive formula for determining $d_{j,j+1}$ is shown in Equations (5) and (6).

$$d_{0,1} = d_{w_i} \tag{5}$$

$$d_{j,j+1} = d_{j-1,j} \times \mathrm{r}_{in}^{out}(a_{i_j}) \tag{6}$$

As discussed previously, each pair of workflow components $(a_{i_j}, a_{i_{j+1}})$ corresponds to a commodity, with demand $d_{j,j+1}$. The different workflow commodities must however be connected to each other: flow going into a service for a commodity must also leave the same service, on the same server, for the next workflow commodity. This *workflow chain relation constraint*, expressed in Equation (7), ensures that, if a flow enters a service from a server, it must go back to the same server for the next workflow commodity.

$$\mathrm{r}_{in}^{out}(a_{i_n}) \times \mathrm{f}((s, a_{i_n}), c_i) = \mathrm{f}((a_{i_n}, s), c_{i+1}) \tag{7}$$

This constraint is added for every sequence $(a_{i_{n-1}}, a_{i_n})$, $(a_{i_n}, a_{i_{n+1}})$, with associated commodities c_i, c_{i+1}, that is part of a workflow, and for every server s on which service a_{i_n} can be executed.

A second flow-based constraint is added to ensure that a flow between two services does not pass over other services, but only moves over the server network. This *server flow constraint* is shown in Equation (8) for outgoing flows and in Equation (9) for incoming flows. These constraints are added for every service $a \in A$, server $s \in S$, and commodity $c \in C$ going from service a_1 to a_2.

$$\mathrm{f}((a, s), c) = 0 \text{ (unless } a = a_1) \tag{8}$$

$$\mathrm{f}((s, a), c) = 0 \text{ (unless } a = a_2) \tag{9}$$

In the model as discussed up until now, the only limitation is bandwidth. While for some cases this may be sufficient, in practice, the available CPU usage can be a bottleneck as well. This adds an additional constraint, the *CPU capacity constraint*, which is shown in Equation (3), and which is added for every server $s \in S$.

$$\sum_{(s,a)\in E} \mathrm{r}_{in}^{CPU}(a) \times \sum_{c \in C} \mathrm{f}((s, a), c) \leq CPU_s \tag{10}$$

In this equation we make use of the proportionality of CPU usage to input flow, which we discussed earlier, to determine the CPU usage of individual services. While we focus on CPU constraints in this paper, similar constraints can analogously be added to model other resources, such as e.g. disk I/O.

This formulation in itself is not sufficient: the initial service of a workflow does not have any input flow, and is thus ignored by this constraint. We resolve this by defining an additional service a_0, which does not consume any CPU, which is active on all servers, and which is prepended to all workflows. The only purpose of this service is to ensure the first workflow service a_{w_1} of a workflow w has an input flow, which can then be used to correctly enforce the capacity constraint. This service is artificial, so the flow for any workflow commodity (a_0, a_{w_1}) which starts in this service may not pass over server-server links. To enforce this, the constraint in Equation (11) is added for all $e \in E$ of the type $(s_1, s_2) \in S^2$, and all commodities $c \in C$ for which the flow starts in a_0.

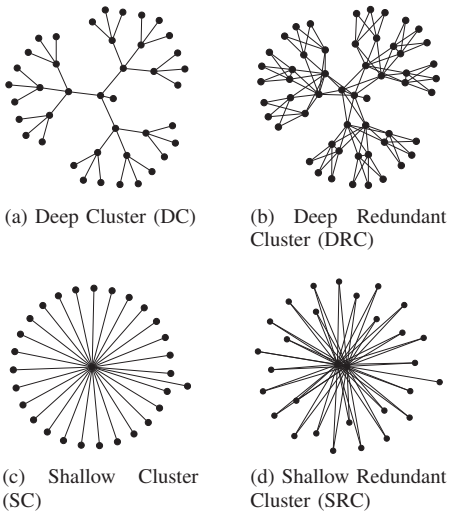
$$f((s_1, s_2), c) = 0 \tag{11}$$

It is of note that within the presented model, flows are assumed to be splittable. This assumption is acceptable for the network nodes, as network packets can be split, but may not always hold for some services. Within the model, no direct support is offered for such services, as adding constraints to achieve this would make it impossible to achieve polynomial execution speeds. We suggest two approaches to handle such a situation: (1) requiring more resources than strictly necessary, to ensure a feasible flow can be mediated after execution, or (2), ensuring only a single instance of this service is present so the flow is forced to make use of this unique service.

C. Linear Programming Algorithms

The formal model as defined previously can be used as input for a Linear Programming (LP) solver. We have implemented three variants of the NAID algorithm using the CPLEX[20] LP solver:

- NAIDz solves the NAID problem as described previously, that is: every workflow realizes an equal share z of its demand. A disadvantage of this approach is that a single bottleneck can limit the maximum z value, lowering the quality of other workflows that do not suffer from the bottleneck.
- A variation on the model, NAIDz_w can be achieved by assigning an individual value z_w for every workflow w, rather than assigning a global z value for all workflows, and maximizing $\sum_{w \in W} z_w$. This ensures individual bottlenecks cannot limit the performance of other workflows. If contention for capacity occurs, this approach could however cause starvation for some flows, causing some flows to achieve exceptionally high z_w values at the cost of the z_w values of other flows.
- A final variation, NAIDzz_w, combines properties of both algorithms: it first uses NAIDz to determine a maximal global z value. Subsequently, it assigns individual shares z_w to every workflow, maximizing their sum, as in NAIDz_w. In this approach, the value of z is however used as a minimal value of the z_w values, ensuring no starvation of workflows occurs. The obvious disadvantage of this approach is that, as opposed to other variants, two optimizations must be executed.



(a) Deep Cluster (DC)

(b) Deep Redundant Cluster (DRC)

(c) Shallow Cluster (SC)

(d) Shallow Redundant Cluster (SRC)

Fig. 6: The networks used in the evaluations.

These algorithms can be used to determine demand satisfaction of a specific workflow configuration. Using the algorithms, the satisfaction of a current service workflows configuration can be determined. When new workflows are to be added, the impact on the fulfillment of existing workflows can be determined by re-evaluating the configuration when the workflow is added. The difference between the original and new z (or z_w) values then characterizes the impact of adding the workflows.

VI. Evaluation results

In this section, we will first describe the evaluation setup. Then, the performance of the three algorithms is compared in terms of two quality metrics and four different network setups, after which we evaluate the algorithms' execution speed.

A. Evaluation Setup

The two metrics to determine solution quality are: (1) the amount of successfully provisioned service workflows, and (2) minimum, maximum and average achieved demand of workflows. Both metrics are evaluated for different simulated network topologies. The used networks are shown in Figure 6, and are all variations on a star network. The leave nodes are servers with a 2GHz CPU, while inner nodes are routers, on which no services can be executed. The networks shown in Figures 6a and 6c have varying depths, and use an edge capacity of 1Gbps. The networks in Figures 6b and 6d are similar, but add redundancy by connecting each server and router to two nodes on a higher level. In this case we assign an edge capacity of 500Mbps to ensure the global capacity is equal to the capacity in the other two cases. In all four cases, the root of the network is connected directly to a central node without intermediate branches. The root of each of these networks is connected to a cloud, which is represented as a server with infinite capacity, using a link with infinite capacity, as we assume the constraints in the network are caused by the local network and servers.

In the evaluations, we use 40 services, where for every service a the value $\mathrm{r}_{in}^{CPU}(a)$ is randomly chosen in the interval $[\frac{1}{2}, 2]$, as we assume services with a high flow will in general require a large amount of CPU resources, and $\mathrm{r}_{in}^{out}(a)$ is chosen in the interval $[\frac{1}{5}, 5]$, as relatively large differences between input and output flows can occur. In both cases, the randomization is executed in such a way that i and $\frac{1}{i}$ have equal probabilities of occurring. The services on which the services can be executed are chosen as follows: (1) there is a 10% chance that the service will only run on all servers, and a 10% chance it will only run on a specific server, and (2) otherwise, the amount of servers on which the service can be executed will be chosen uniformly from the range $[1, |S_x|]$, with S_x the set of non-router server nodes on which services can be executed.

We then generate 30 workflows, by, for each workflow, randomly choosing 5 services from the previously described set. As mentioned in Section V-A, these workflows are linear. It is possible for a service to occur more than once in a workflow, but not directly adjacent to one another. For each flow, we randomly determine a total load $l \in [800, 1200]$. We choose this range as the cumulative client-site capacity in the test networks is $\pm 27 Gbps$, which is used up if every workflow uses $900 Mbps$. As not every workflow commodity needs to pass over the local network, we use a higher average to ensure a high system load exists. The choice of using workflows of 5 services is inspired by the medical scenario, where a flow of the type *terminal, server, cloud, server, terminal* is common, and where the terminal and server are connected using proprietary technologies, ensuring this server cannot itself be migrated to the cloud. In the evaluation, we will also discuss the algorithm's performance with varying workflow lengths.

In these problem models, most of the workflows can be partially executed in the cloud, but on average, a significant load on local network will exist as well. The varying rates ensure that strongly different bandwidth requirements between pairs of services, caused by nodes such as filters, occur.

First we will discuss the results for the two qualitative metrics; afterwards we will evaluate the execution speed of the algorithms.

B. Ratio of successfully provisioned service workflows

The data in this section has been generated by using the randomly generated problems discussed previously, repeating the evaluation for the different networks and NAID algorithms. The presented results have been averaged over 500 executions.

We will first evaluate the quality of the different algorithms by comparing the amount of service workflows that can be provisioned on the network. In Figure 7 we show the amount of workflows that succeed for the different algorithms and network configurations. In the chart, we show how many workflows, on average, achieve 100%, $> 80\%$ and $> 50\%$ of their demand. The latter two are interesting to consider when the requirement of achieving 100% demand satisfaction is not strictly required.

We observe that the NAIDz_w and NAIDzz_w approaches consistently outperform the NAIDz approach. Using the

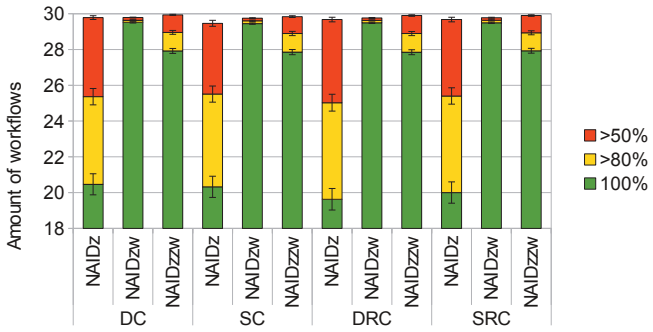


Fig. 7: The average amount of workflows that achieve 100%, 80% and 50% of their requested demand.

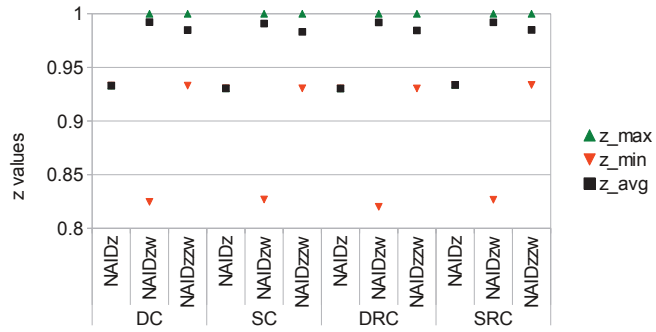


Fig. 9: The average, minimum and maximum achieved z values of the different algorithms.

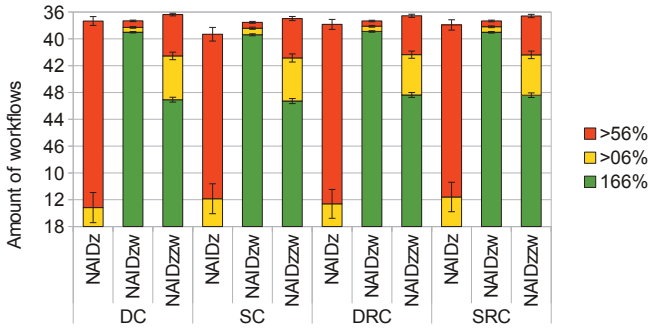


Fig. 8: The average amount of workflows that achieve 100%, 80% and 50% of their requested demand, using only problem models for which non-successful flows occur.

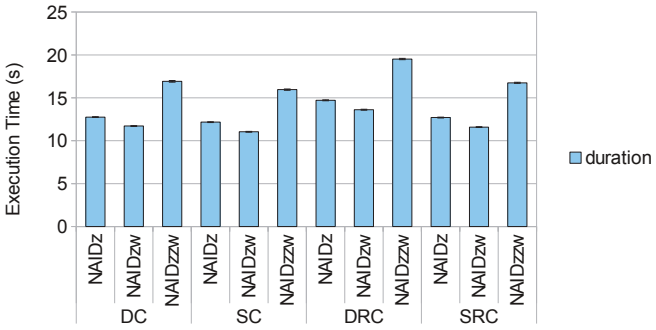


Fig. 10: The execution speed of the different evaluations.

NAIDz_w approach, more workflows achieve 100% success, but the NAIDzz_w approach has more applications achieving at least 50% of their demand. These differences become even more pronounced when we filter out execution results that fully achieve the demand, focusing entirely on the more difficult problems in the evaluation sets. This is shown in Figure 8, where, due to the premise, no workflows achieve 100% demand satisfaction using the NAIDz algorithm.

C. Achieved workflow demand

The average, minimum and maximum z_w values are shown in Figure 9. For the NAIDz algorithm, these three values are always the same, as only a single z value is determined, while for the other algorithms differing values are achieved. The minimal z_w value in the NAIDzz_w algorithm is the same as the z value of NAIDz, while a higher average and maximum are achieved. Forcing the NAIDzz_w algorithm to achieve at a minimum the z value of the NAIDz algorithm constrains the problem, a limit that does not exist in the NAIDz algorithm. This enables the latter to achieve a higher average z_w, but at the cost of a significantly lower minimal workflow satisfaction.

It is interesting to note that, while the evaluations are used on four different networks with varying internal complexity, the quality metrics are nearly identical for the different cases. This is interesting, as increased problem complexity can, as we will show in the next section, increase the execution time. The results show that part of the network complexity can be removed, while similar quality results can still be achieved.

These experiments were repeated for workflow lengths ranging from 2 to 9, leading to similar results, indicating that the problem complexity is mainly influenced by the total workflow load rather than by the length of the workflow. These results are however not shown due to space constraints.

D. Execution speed

Execution speeds were evaluated using an Ubuntu server with Intel Core i3 2.93GHz processor and 4GiB memory.

In Figure 10, the execution speed of the evaluation discussed in the previous section are shown. The NAIDz and NAIDz_w both require a similar amount of execution time, with NAIDz_w executing marginally faster, while the NAIDzz_w approach executes slower. The differences between NAIDz and NAIDz_w are, for this input, negligible. It is of note that, while NAIDzz_w executes two LP optimizations, its duration is less than the sum of the durations of the other algorithms. This could be caused by the additional constraints added in the NAIDzz_w algorithm that limit the search space, ensuring the CPLEX solver can find a solution faster during its second execution.

Subsequently, we evaluated the execution speed of the algorithms using varying system parameters. Increasing the amount

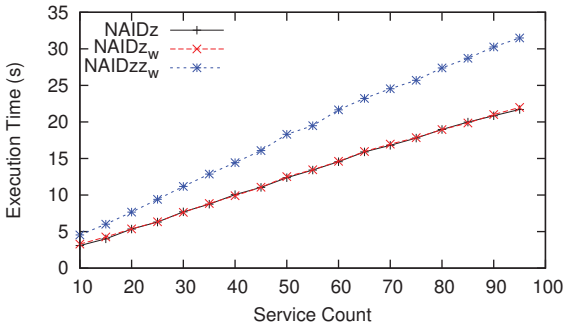


Fig. 11: Execution speeds with varying service counts, $|W| = 30$, $|S_x| = 30$.

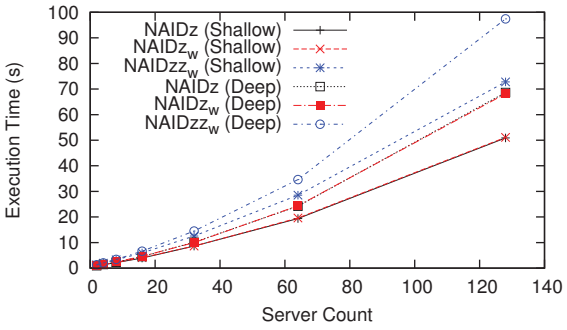


Fig. 13: Execution speeds with varying server counts, $|W| = 30$, $|A| = 30$.

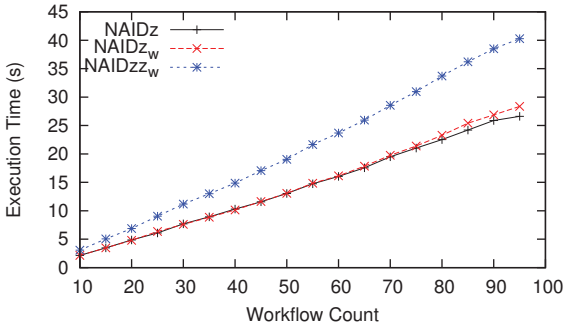


Fig. 12: Execution speeds with varying workflow counts, $|A| = 30$, $|S_x| = 30$.

of services, as shown in Figure 13 increases the execution time of each of the algorithms linearly. This increase is to be expected as the amount of nodes in the input graph discussed in Section V-A increases, which increases its complexity. The execution time curve of NAIDzz_w is steeper than those of NAIDz and NAIDz_w, as it combines both algorithms. Increasing the amount of workflows, as shown in Figure 12 causes a steeper execution time increase, again linear, as in this case, more commodities are included in the model. For higher workflow counts, the execution speed of NAIDz_w increases compared to that of NAIDz, as increasing the amount of workflows increases the amount of z_w decision variables in the LP formulation of the NAIDz_w algorithm.

In Figure 13, we evaluate the execution speeds of the different algorithms for two types of star configuration, once using a shallow network where all servers are connected to a central router as in Figure 6c, and once using a deep network using two levels of depth, similar to the network in Figure 6a. As the amount of servers increases, so does the execution time of the algorithm. The times for the shallow and deep models diverge as the amount of servers increases, due to an increasing impact of the network complexity.

E. Algorithm comparison

Both NAIDz_w and NAIDzz_w perform significantly better than NAIDz if less than 100% demand satisfaction is permissible. The choice for algorithms then depends on the context: if the average demand satisfaction must be maximized, the NAIDz_w algorithm performs best, while if the goal is to maximize the amount of workflows that achieve acceptable performance, the NAIDzz_w algorithm performs best. If less than 100% demand satisfaction is not tolerated, the three algorithms have an equal qualitative performance, in which case NAIDz is preferable as it executes faster. The choice of whether to use NAIDz, NAIDz_w or NAIDzz_w thus depends upon the context. We also note that the results show that, to improve the execution speed of the algorithms, part of the internal network structure can be abstracted.

VII. Conclusions

In this paper, we have described and evaluated three approaches, based on extensions to the linear programming formulation of the maximum concurrent flow problem, to determine the impact of adding service workflows in a hybrid cloud scenarios. The three NAID algorithms have differing quality properties, and can thus be used in different contexts. Specifically, one algorithm executes the fastest, another leads to higher average workflow satisfaction, whereas the last maximizes the workflow achievement first, ensuring a minimum quality level for all workflows is achieved first preventing workflow starvation. We found that, for the considered cases, at most 100s execution time is needed, and that abstracting the underlying network decreased execution times up to 30%, without significantly reducing placement quality.

In future work, we will incorporate the presented algorithms in a management framework, where we will determine how the information generated by these algorithms can be used in conjunction with intelligent application placement techniques.

Acknowledgement

Hendrik Moens is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). This research is partly funded by the IBBT CUSTOMSS[21] project.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows*. Prentice Hall, New Jersey, 1993.

[2] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," *J. ACM*, vol. 37, no. 2, pp. 318–334, 1990.

[3] B. Awerbuch and T. Leighton, "Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks," in *Proceedings of the 26th Symposium on Theory of Computing STOC*. ACM, 1994, pp. 487–496.

[4] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[5] V. Kolar and N. B. Abu-Ghazaleh, "A multi-commodity flow approach for globally aware routing in multi-hop wireless networks," *Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications PERCOM06*, pp. 308–317, 2006.

[6] W. Szeto, Y. Iraqi, and R. Boutaba, "A multi-commodity flow based approach tovirtual network resource allocation," *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM 03*, vol. 6, pp. 3004–3008, 2003.

[7] Y. L. Y. Liu, D. Tipper, and P. Siripongwutikorn, "Approximating optimal spare capacity allocation by successive survivable routing," pp. 198–211, 2005.

[8] J. Rolia, A. Andrzejak, and M. Arlitt, "Automating enterprise application placement in resource utilities," in *Self-Managing Distributed Systems: 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2003*. Springer, 2004, pp. 118–129.

[9] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proceedings of the 6th international conference on World Wide Web*, 2007, pp. 331–340.

[10] C. Adam and R. Stadler, "Service middleware for self-managing large-scale systems," *IEEE Transactions on Network and Service Management*, vol. 4, no. 3, pp. 50–64, Dec. 2007.

[11] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based resource management for cloud environments," in *Proceedings of the 6th International Conference on Network and Service Management (CNSM 2010)*, 2010, pp. 1–8.

[12] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. D. Turck, "Feature placement algorithms for high-variability applications in cloud environments," in *Proceedings of the 13th Network Operations and Management Symposium (NOMS 2012)*, 2012, pp. 17–24.

[13] H. Moens, J. Famaey, S. Latré, B. Dhoedt, and F. De Turck, "Design and evaluation of a hierarchical application placement algorithm in large scale clouds," in *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, 2011, pp. 137–144.

[14] C. Low, "Decentralised application placement," *Future Generation Computer Systems*, vol. 21, no. 2, pp. 281–290, 2005.

[15] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguadé, "Utility-based placement of dynamic web applications with fairness goals," in *Proceedings of the 11th Network Operations and Management Symposium (NOMS 2008)*. IEEE, 2008, pp. 9–16.

[16] J. Brønsted, K. M. Hansen, and M. Ingstrup, "Service composition issues in pervasive computing," *IEEE Pervasive Computing*, vol. 9, no. 1, pp. 62–70, Jan. 2010.

[17] K.-T. Tran, N. Agoulmine, and Y. Iraqi, "Cost-effective complex service mapping in cloud infrastructures," in *Proceedings of the 13th Network Operations and Management Symposium (NOMS 2012)*, 2012, pp. 1–8.

[18] M. Hajjat, X. Sun, Y.-w. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: Planning for beneficial migration of enterprise applications tothe cloud," in *Proceedings of the ACM SIGCOMM 2010 conference*, 2010, pp. 243–254.

[19] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.

[20] (2011) IBM ILOG CPLEX 12.3. [Online]. Available: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer

[21] (2011) CUSTOMSS: CUSTOMization of Software Services in the cloud. [Online]. Available: http://www.ibbt.be/en/projects/overview-projects/p/detail/customss