Optimization of resource usage in virtualized environments

Jakub Górski

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2016-30
Optimization of resource usage in virtualized environments
(Packing of virtual machines with respect to resource, bandwidth and placement constraints.)

Jakub Górski
jakub.b.gorski@gmail.com

June 13, 2016

Master’s thesis work carried out at Ericsson AB.

Supervisors: Jörn W. Janneck, jorn.janneck@cs.lth.se
Peter Kanderholm, peter.kanderholm@ericsson.com
Examiner: Krzysztof Kuchcinski, krzysztof.kuchcinski@cs.lth.se
Abstract

Today’s markets are heavily concentrated on Cloud Computing, where trends project an increasing amount of services living therein. Because these data centers, named Clouds, provide ability to dynamically scale the resources assigned to a service. It is a technology which allows products to remain competitive in a constantly changing market. Consumers profit from this solution, as it is much easier to maintain the quality of a given service (QoS) by allocating more resources within a Cloud, on the contrary it lies within the Cloud provider’s interest to fit as many services possible within their Cloud. This poses the question of how resources integral to facilitation of a service are meant to be allocated to enable efficient utilization of resources within the Cloud.

Services consisting of several Virtual machines are placed on as few physical machines as possible using a 3-dimensional Multiple Subset-sum problem model approach. The virtual machine placement case considered is Infrastructure as a Service (IaaS), and its implementation is facilitated by Java Constraint Programming Library (JaCoP) developed by Prof. Krzysztof Kuchcinski and PhD Radosław Szymanek. Results from this work deem constraint programming a suitable approach for enhancing an existing virtual machine deployment process in an IaaS Cloud.

Keywords: JaCoP, constraint programming, multiple knapsack, variable bin-packing, virtual machine placement, bandwidth analysis, subset sum
Acknowledgements

Firstly, thank you Prof. Jonas Skeppstedt for recommending me as a master thesis worker. Thank you Peter Kanderholm, together with Filip Lange for guidance, and faith in my work. Thank you Prof. Ümit V. Çatalyürek at the Ohio State University, and Martin Quinson at French Research Institute IRISA for being open to correspondence. Big thank you to my friends Tobias Johannesson, Fredrik Gullstrand, and Billy Johansson for numerous discussions and brainstorming. A big thank you to Mentor Jörn Janneck at Lund University helping me maintain focus throughout the thesis. Lastly, a huge thank you goes to Professor Kuchcinski, as this master thesis would not have been possible without his support, and vast knowledge of constraint programming.
# Contents

1 Problem Description and Applications .................................................. 7
  1.1 Data Center Architecture ............................................................... 8
      1.1.1 Virtual Network Functions (VNFs) ........................................ 9
      1.1.2 Virtual Machines (VMs) ....................................................... 9
  1.2 Placement Algorithm Goals .............................................................. 10
  1.3 Related Work ..................................................................................... 12

2 Problem Model as Combinatorial Optimization ......................................... 15
  2.1 0/1 Knapsack Problem (0-1 KP) ....................................................... 15
  2.2 0/1 Multiple Knapsack Problem (0-1 MKP) ......................................... 16
  2.3 Multidimensional Multiple Subset-Sum Problem (MDMSSP) .................. 16
  2.4 Proposed Problem Model ................................................................ 17
  2.5 Discarded Problem Models ................................................................. 20
  2.6 JaCoP Constraint Problem Solver ....................................................... 20
      2.6.1 Introduction by Sudoku example ............................................ 20
      2.6.2 Search Heuristics ................................................................ 21
      2.6.3 Implementation .................................................................... 24

3 Results ...................................................................................................... 27
  3.1 Placement Quality ............................................................................. 29
      3.1.1 Testing methodology ............................................................. 29
      3.1.2 Results .................................................................................. 29
      3.1.3 Discussion ............................................................................ 30
  3.2 Placement with Bandwidth Analysis .................................................... 31
      3.2.1 Testing Methodology ............................................................. 31
      3.2.2 Results .................................................................................. 32
      3.2.3 Discussion ............................................................................ 33

4 Conclusion ............................................................................................... 35

Bibliography ............................................................................................... 37
Appendix A JSON specifications
Chapter 1
Introduction

Because market trends fluctuate, and demand on a service may vary on daily basis, it is important to remain flexible with regards to how much computer resources are used to keep an on-line service running in an acceptable manner. Because of this, the market has evolved such that entire companies move their services into large data centers, filled with physical machines, where said service may dynamically scale to allocate more or less resources on-demand. These data centers are named Clouds, and are often provided by third-parties such as Google or Amazon, which allow companies to remain competitive in a growing market, such that a given company’s service may scale together with the market itself. However, usage of Clouds are not limited to running services, but also computationally heavy processes, which is why Clouds still have unexplored potential, and also why their usage is often referred to as Cloud Computing, as it is a more general notion encompassing the possibilities of a Cloud.

Cloud computing trends, the everything as a service (XaaS) model in particular, is a highly flexible market which has been rapidly growing since 2007 [9, p.623]. Over the years, during which companies like Amazon have invested in Cloud Computing, there have been efforts to optimize these clouds with respect to various aspects, such as energy consumption reduction [24], and physical machine usage minimization [16, p.5]. These objectives are realized by controlling the utilization of physical machine resources through Virtual Machine (VM) placement algorithms. Historically most VM placement algorithms have been on-line [16, p.7], due to the dynamic nature of Cloud data centers, on account that VMs may regularly migrate across machines. However, one thing which these algorithms don’t excel at is initial deployment, which becomes especially important in infrastructure as a service (IaaS) models.

Placement of VMs in data centers with respect to resource utilization, and placement constraints is NP-hard [12, p.471][13, p.9]. It is a difficult problem on whose solution the quality [1] of nigh all Cloud services depend [2]. Efficient VM placement becomes increasingly important, as current software as a service (SaaS) (eg. Google Apps and Facebook) Cloud Computing trends project exponential growth since 2013 [9, p.623].
This trend is further augmented by Spotify's decision in Q1 2016 to move into the Google Cloud Infrastructure [23]. Resource utilization of such configurations is of essence, which can be achieved by optimizing virtual machine placement.

It lies within a Cloud Service provider’s interest [24] to fit as many cloud services as possible on available hardware, a dilemma which depends on multiple rules and availability of resources. On the contrary, a Cloud Service customer is primarily interested in the responsiveness, thereby quality of the service (QoS) [9], which depends on many resources integral to facilitation of bandwidth within the Cloud. In Infrastructure as a Service (IaaS) configurations, where groups of VMs are assigned to a specific network topology, VM placement with bandwidth analysis becomes of essence.

This master thesis’ focus is an IaaS VM placement case, which considers several groups of VMs communicating in an Artificial Neural Network Topology (see Figure 1.4b) thereby contributing to the family of off-line multi-objective network-performance maximizing VM placement algorithms [16]. Solutions consist of VM placements which meet, or exceed a predetermined bandwidth objective that the system must handle. Virtual machines are placed such that minimum amount of machines are expended, and bandwidth into system meets said bandwidth objective. Machine resources considered are: memory, storage, units of CPUs and two categories of network bandwidth resources denoted by Link and vSwitch throughput.

Quality of proposed algorithm’s solutions will be determined by how evenly resources are consumed across machines. Implementation uses constraint programming, hierarchical heuristics facilitated by Java Constraint Programming Library (JaCoP), and a combinatorial problem model formulation which facilitates such a solution format. Algorithm’s viability of enhancing an existing VM deployment strategy will be discussed, which is the main goal of this thesis devised by Ericsson.

Introduction section focuses on a clear definition of what this thesis regards as a data center, machine, VM, and how VM network topology groups are specified. A more elaborate VM placement problem formulation will follow, which will make use of information about physical machine elements within the data center, and VM elements which are to be placed inside said data center.

Introduction is concluded with how the problem is modeled as a constraint-programming problem.

1.1 Data Center Architecture

The data center in Figure 1.1 is the top-level resource which cumulatively represents all machines’ resources which are described in Figure 1.2a. In real-life scenarios machines are assigned to nodes, which this thesis will regard as simply another abstraction.

Of the resources shown in Figure 1.2a, the bandwidth facilitating vSwitch (Virtual Switch), is denoted as a VM in itself. The vSwitch facilitates local communication between VMs that share a machine, as shown in Figure 1.3a. Outgoing machine traffic taxes both vSwitch and Link resources, where Link is another bandwidth facilitating resource, which is consumed by communication across machines. Communication traffic between machines needs to pass through the vSwitch prior to exiting via a machine Link, as shown
1.1 Data Center Architecture

![General data center model](image)

**Figure 1.1:** General data center model employed in this thesis.

Because the vSwitch is essentially a VM it will be regarded as a preexisting resource-consumer, whose RAM, vCPU, disk consumption has been taken into account prior to a VM placement.

### 1.1.1 Virtual Network Functions (VNFs)

In this thesis VNFs are entities which group VMs together, and describe how those VMs communicate within that group. Communication described by VNF groups obeys a particular network topology, such as *Partially Connected Mesh Network Topology* [Figure 1.4a] or *Artificial Neural Network Topology* [Figure 1.4b]. Thesis will focus on the *Artificial Neural Network Topology*, as it is a simple topology where all VMs evenly distribute traffic amongst themselves.

### 1.1.2 Virtual Machines (VMs)

VMs are elements which consume resources as described in [Figure 1.2b]. They may also contain placement rules in the form of *Affinities* and *Anti-Affinities*, which respectively denote whether certain VMs must be placed on the same physical machine, or must never be placed on the same physical machine. For a particular VM placement to be valid such placement rules must be obeyed.

As mentioned in section 1.1, the vSwitch is regarded as a VM itself. Each machine is inherently required to have one vSwitch to facilitate VM communication within a machine. Outgoing machine traffic taxes the vSwitch prior to its exit through Link.

vSwitch is the only VM element which may not have any of the aforementioned Anti-Affinity placement rules, as each machine is required to have a vSwitch. It also provides...
1. Problem Description and Applications

(a) Each machine’s resource provisioning model. The vSwitch is part of a VM which every machine is required to have. A constant enabling conversion between units PPS (Packets Per Second) and Bps is given.

(b) Virtual machine resource consumption model with placement rules. Affinities force VMs to share the same machine, whereas Anti-affinities prohibit VMs from sharing the same machine.

Figure 1.2: Description of VM and Machine models.

1.2 Placement Algorithm Goals

Given machines with a known amount of available resources (Figure 1.2a), VM elements with known resource consumption (Figure 1.2b), their Anti-/Affinity placement rules (see subsection 1.1.2), and place within the VNF group they belong to (see Figure 1.5b), a placement problem with bandwidth analysis will be solved. With the example placement in Figure 1.5 in mind, an algorithm with the following properties will be proposed:

i VM placement that obeys Anti-/affinity placement rules.

ii Performs bandwidth analysis once placement is complete. This analysis is done according to the each VNF group’s Artificial Neural Network Topology (see Figure 1.4b).

iii VM placement obeys machine resource constraints (RAM, Disk, vCPU).

iv Optionally permit usage of a bandwidth objective which placed system must handle (vSwitch, Link).
1.2 Placement Algorithm Goals

(a) Dotted arc denotes simplex communication from VM1 to VM2. Black arcs illustrate how this communication is actually facilitated. Traffic taxes the machine’s vSwitch.

(b) Dotted arc denotes simplex communication from VM1 to VM2 across two machines. Black arcs illustrate how the two machines’ vSwitch and Link facilitate this communication. Traffic taxes both machine’s respective Link and vSwitch.

Figure 1.3: An illustration of the discrepancy between how VMs want to communicate (dotted arcs), and which resources machines expend to facilitate simplex communication (black arcs).
1.3 Related Work

The foundation to this thesis’ VM placement solution is a packing problem formulation named Knapsack Problem (KP). The methodology described by KP is analogous to packing a real life rucksack, such that items placed inside said rucksack do not exceed its weight and volume limit. Also, the cumulative value of the items placed into the rucksack should have as much value as possible. Less formally, it can be said that a fridge won’t be on the rucksack’s packing list, as it weighs a lot, takes a lot of volume, is not practical, and therefore has almost zero value. However, a map and compass weigh very little, take up small volume, and are very useful, which means they have a great value.

Previous work on VM placement algorithms largely employ variants of a packing algorithm named the Multiple Knapsack Problem [1] [10], where aforementioned rucksack analogy is applied just the same, but for several rucksacks, such that cumulative value of items packed into all rucksacks is as large as possible. The most related VM placement work is [3], which employs a Multidimensional Multiple Knapsack Problem model. It similarly implies that several rucksacks are packed, and the cumulative value of all packed items is maximized just like before, however now the rucksacks and items have multiple weights and volumes respectively, thereby making the problem multidimensional.

Importantly, [10] showed that hierarchical cloud structures can be represented as placement constraints that are modeled as matrices. In this work a different approach is employed, such that only VM placements are modeled as matrices, whereas the hierarchical cloud structure is maintained by generating constraints which identify how VMs communicate given a placement, such that a bandwidth objective can be met. In order to compensate for certain performance penalties heuristics are employed (see 2.6.2) which avoid searching where there are no solutions.

According to [10, p. 5] 2013 VM Placement Literature Review, publication [14] proposes an on-line bandwidth-maximizing VM placement algorithm, which amongst its drawbacks lists significant compute-resource wastage for the purpose of maximizing network utilization. Similarly, the proposed algorithm in this thesis is off-line for the same reason, as bandwidth analysis is a very computationally intensive element.

Just as in publication [10] Modeling and Placement of Cloud Services with Internal Structure, this thesis proposes a VM placement with bandwidth analysis, but with a simpler network topology model. However, this an exact bandwidth analysis algorithm as opposed to the 2-approximation algorithm that minimizes traffic between VMs which [10, p.3] proposes. The penalty for using an exact algorithm is longer computation time.

Because publication [3] shows that Multiple Multidimensional Knapsack Problem can be used to model the VM placement problem, it follows that fundamental Knapsack Problem’s special case Subset-sum Problem (when item weights and item values are the same) is also suitable as a problem model. Analogously, the Subset-sum Problem’s objective is to fully utilize rucksack capacities by packing as many items as possible. This Subset-sum Problem special case lays the foundation to this thesis’ algorithm, and is discussed in chapter 2.
1.3 Related Work

(a) Partially Connected Mesh Network Topology. Network traffic from and to $s$ is unevenly distributed amongst VMs through bidirectional arcs.

(b) Artificial Neural Network Topology. Network traffic from $s$ is evenly distributed amongst all VMs according to a communication hierarchy using directed arcs. This thesis will use a bidirectional arc case, where $VM7$ and $VM8$ also would distribute their result downwards in the communication hierarchy, and back to traffic source $s$.

Figure 1.4: Two main network topologies used to describe VM group communication within a VNF. Figure 1.4a provides a flexible topology with point-to-point connections. Figure 1.4b employs interconnection of VMs similar to a $k$-partite graph. VNFs are unaware of how this communication will be facilitated by the data center (see Figure 1.3). This thesis focuses on the topology shown in Figure 1.4b.
1. Problem Description and Applications

(a) Information regarding placement rules. Each VM may have Affinities, or Anti-affinities towards other VMs (see subsection 1.1.2). In this example all VMs have Anti-/Affinity placement rules.

(b) Information regarding VNF groups which the VMs belong to. Each VNF’s internal communication employs bidirectional Artificial Neural Network Topology, as explained in Figure 1.4b.

(c) Resultant placement on two machines. VNF groups have now been scattered across machines due to VM placement. Note how Affinity and Anti-affinity rules from Figure 1.5a are obeyed.

(d) Communication commences after VM placement. VNF traffic obeys Figure 1.5b description. Dotted bidirectional arcs depict a duplex communication scenario of Figure 1.3.

Figure 1.5: VM placement problem example. Solved with respect to Affinity, Anti-affinity, and VNF groups.
Chapter 2
Problem Model

The problem model is built on a subset-sum version of Multidimensional 0/1 Multiple Knapsack Problem (0-1 MMKP), such that knapsacks accommodate three volumes, factor in placement constraints, and placement-dependent bandwidth analysis. Simpler Knapsack variants will be presented prior to the proposed algorithm’s 0-1 MMKP based model.

2.1 0/1 Knapsack Problem (0-1 KP)

The 0/1 Knapsack Problem (0-1 KP) is an NP-hard combinatorial optimization problem [6], which throughout the centuries has been found to have applications in cryptography [13], networking [20], economics [7], and scheduling [25]. The formal definition of 0-1 KP [17, p.11] follows:

**Definition 2.1.1. 0/1 Knapsack Problem**: Given \( n \in \mathbb{Z}^+ \) items, their item weights \( w_j \) item profits \( v_j \) and one knapsack of weight capacity \( c > 0 \). Choose a subset of the \( n \) items such that the cumulative item profit inside the knapsack is maximized without exceeding the knapsack’s weight capacity \( c \):

Maximize: \[ \sum_{j=1}^{n} v_j x_j \]  \hspace{1cm} (2.1)

Subject to: \[ \sum_{j=1}^{n} w_j x_j \leq c. \] \hspace{1cm} (2.2)

\[ x_j \in \{0, 1\}, \quad j = 1, \ldots, n, \] \hspace{1cm} (2.3)

where \( x_j \) is a binary variable governing whether item \( j \) should be packed into the knapsack. When \( x_j = 1 \) item is included in the knapsack, and excluded from the knapsack when \( x_j = 0 \).
2. Problem Model as Combinatorial Optimization

2.2 0/1 Multiple Knapsack Problem (0-1 MKP)

Due to the nature of 0-1 KP it can be scaled up to multiple knapsacks, which then becomes the 0/1 Multiple Knapsack Problem (0-1 MKP), which is an NP-hard problem. 0-1 MKP permits more complex problem formulations than 0-1 KP, and has applications in radar technology \[11\], Cloud systems \[1,5\], multiprocessor scheduling \[4\], and energy management \[18\]. The formal definition of 0-1 MKP \[17, p.13\] follows:

**Definition 2.2.1.** 0/1 Multiple Knapsack Problem: Given \(n \in \mathbb{Z}^+\) items, their item weights \(w_j\) item profits \(v_j\) and \(m \in \mathbb{Z}^+\) knapsacks of weight capacities \(c_i\). Choose a subset of the \(n\) items such that the cumulative item profit inside each knapsack is maximized without exceeding each knapsack’s weight capacity \(c_j\):

Maximize:
\[
\sum_{i=1}^{m} \sum_{j=1}^{n} v_j x_{ij} \tag{2.4}
\]

Subject to:
\[
\sum_{j=1}^{n} w_j x_{ij} \leq c_i, \quad i = 1, ..., m, \tag{2.5}
\]
\[
\sum_{i=1}^{m} x_{ij} \leq 1, \quad j = 1, ..., n, \tag{2.6}
\]
\[
x_{ij} \in \{0, 1\}, \quad i = 1, ..., m, \quad j = 1, ..., n. \tag{2.7}
\]

Here \(x_{ij} = 1\) indicates that item \(j\) should be packed into knapsack \(i\), and Equation 2.5 ensures that capacity constraint of each knapsack is satisfied. Constraint Equation 2.6 ensures that each item is chosen at most once.

**Remark (Subset-sum Problem).** A special case in Definitions 2.1.1, 2.2.1 when \(w_j = v_j\) \[21, p.2\] is termed as the Subset-Sum Problem, and Multiple Subset-Sum Problem respectively \[17, p.157\]. Their objective is to find a subset of items whose sum is closest to, but not exceeding, knapsack capacity.

Mentioned subset-sum case in Definition 2.2’s remark is particularly important to this thesis, as it will be used to compute RAM, vCPU, disk consumption in machines.

2.3 Multidimensional Multiple Subset-Sum Problem (MDMSSP)

By using 0-1 MKP special case where \(w_j = v_j\) (see Definition 2.2’s remark) a Multiple Subset-Sum Problem (MSSP) is formulated. In order to facilitate a problem model accounting for machine three resources (RAM, vCPU, disk), items will need several weights, and knapsacks several capacities. Therefore MSSP will be scaled up to a Multidimensional Multiple Subset-Sum Problem (MDMSSP):

**Definition 2.3.1.** Multidimensional Multiple Subset-Sum Problem: Given \(d \in \mathbb{Z}^+\) dimensions and \(n \in \mathbb{Z}^+\) items, their item weights \(w_{jr}\) and \(m \in \mathbb{Z}^+\) knapsacks of weight
2.4 Proposed Problem Model

Building upon the MDMSSP combinatorial problem model described in section 2.3, additional constraints will be supplemented to factor in VM Anti-/Affinity placement restriction rules (see subsection 1.1.2), and bandwidth resource constraints in Link and vSwitch.

Bandwidth constraints are modeled such that for each MDMSSP knapsack in section 2.3 two new knapsacks will be created, whose capacities are governed by Link and vSwitch bandwidth resources present in each machine. During bandwidth analysis, if a traffic edge spans across machines, then both machines’ vSwitch and Link are consumed (Figure 1.3b). Whereas, if an edge resides within one machine, only that machine’s vSwitch is consumed (Figure 1.3a).

The following will build on the MDMSSP Definition 2.3.1, with modifications to accommodate for Anti-/Affinity placement restrictions rules (see subsection 1.1.2), and VNF groups that specify how VMs distribute traffic amongst themselves (see Figure 1.5b).

Definition 2.4.1. Multidimensional Multiple Subset-Sum Problem with item placement restrictions and bandwidth analysis: Given \( d \in \mathbb{Z}^+ \) dimensions and \( n \in \mathbb{Z}^+ \) items, their item weights \( w_{jr} \) and \( m \in \mathbb{Z}^+ \) knapsacks of weight capacities \( c_{ir} \). Choose a subset of the \( n \) items such that the cumulative item weights inside each knapsack is maximized without exceeding each knapsack’s weight capacities \( c_{ir} \):
exceeding each knapsack’s weight capacities $c_{ir}$:

Maximize: $\sum_{r=1}^{d} \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ir} x_{ij}$ \hfill (2.12)

Subject to: $\sum_{r=1}^{d} \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ir} x_{ij} \leq c_{ir}, \quad i = 1, \ldots, m,$ \hfill (2.13)

$\sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, \ldots, n,$ \hfill (2.14)

$x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n,$ \hfill (2.15)

$x_{ij} = 1$ indicates that item $j$ should be packed into knapsack $i$, and Equation 2.13 ensures that the $d$ different capacities of each knapsack are satisfied. Constraint Equation 2.14 denotes that each item must be packed.

An item $j$ may optionally have placement restrictions, such that it must not share knapsack with another item $q$. Similarly, there’s an optional placement restriction, such that an item $j$ must share knapsack with another item $q$. For each such item $j$ with a placement rule towards an item $q$ following constraints are formulated:

Anti-affinities: $x_{ij} = 1 \implies x_{ij} \neq x_{iq} \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \quad q = 1, \ldots, n, \quad j \neq q,$ \hfill (2.16)

Affinities: $x_{ij} = 1 \implies x_{ij} = x_{iq} \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \quad q = 1, \ldots, n, \quad j \neq q.$ \hfill (2.17)

Let there be a known $g \in \mathbb{Z}^+$ amount of VNFs denoted as graphs $G_p = (E_p, V_p, \Omega, s)$, such that $p = 1, \ldots, g$. Each graph $G_p$ has a known $k \in \mathbb{Z}^+$ amount of layered groups denoted $L_{p,i}$, such that the amount of layered groups is $G_p$ graph specific. It is also known to which layered group $L_{p,i}$ each of the $j = 1, \ldots, n$ packed items belong to, such that $\exists j \in L_{p,i}$. Thereby making each item $j$ a vertex, such that $\forall j \in V_p$.

Let there be a traffic source $s$ generating an unknown amount of traffic $t \in \mathbb{Z}^+$. Edges’ $(j, q) \in E_p$ weight $\omega_{iq} \in \Omega$ is directly dependent on traffic $t$ originating from $s$, where $j = 1, \ldots, n$ and $q = 1, \ldots, n$. The edge weight relation depends on how many items there are on previous layered groups, such that distributive traffic is employed as shown in Equation 2.22.

Edges $(s, p) \in E_p$ between $s$ and all items $p \in L_{p,1}$, which is the layered group whose layer is $l = 1$ are formed. Traffic $t$ originating from $s$ is thereby evenly distributed between items in $p$ layered groups $L_{p,1}$. Within each individual $G_p$ even distribution of traffic between each layered group’s items occurs, all the way to the top layered group $L_{p\text{,max}(k)}$, where $\max(k) \in G_p$.

Let there be two known additional capacities to each knapsack denoted $c_{i,d+1}$ and $c_{i,d+2}$ which represent vSwitch and Link resources respectively. Let there also be a known bandwidth objective $b \in \mathbb{Z}^+$. 

18
### 2.4 Proposed Problem Model

#### Bandwidth analysis problem

Bandwidth analysis problem is defined in full by constraints in equations \(2.18\)-\(2.23\), which if satisfied, imply that placed system handles bandwidth objective \(b\).

\[
t \geq b, \text{ where } t \text{ is unknown,}
\]

\[
\sum_{p=1}^{g} \sum_{l=1}^{\max(k)-1} \sum_{j \in L_{p,l}} \sum_{q \in L_{p,l+1}} \omega_{jq} \cdot t_{ijq} + \sum_{p=1}^{g} \sum_{l=1}^{\max(k)-1} \sum_{j \in L_{p,l}} \sum_{q \in L_{p,l+1}} \omega_{iq} \cdot t_{ijzq}
\]

\[
+ \sum_{p=1}^{g} \sum_{j=1}^{n} \omega_{kj} \cdot x_{ij} \leq c_{i,d+1}, \text{ for all } L_{p,k} \in G_p, \quad i = 1, ..., m, \quad z = 1, ..., m, \tag{2.19}
\]

\[
\sum_{p=1}^{g} \sum_{l=1}^{\max(k)-1} \sum_{j \in L_{p,l}} \sum_{q \in L_{p,l+1}} \omega_{jq} \cdot t_{ijzq}
\]

\[
+ \sum_{p=1}^{g} \sum_{j=1}^{n} \omega_{kj} \cdot x_{ij} \leq c_{i,d+2}, \text{ for all } L_{p,k} \in G_p, \quad i = 1, ..., m, \quad z = 1, ..., m, \tag{2.20}
\]

\[
t_{ijzq} = x_{ij} \land x_{zq}, \text{ where } i = 1, ..., m, \quad j = 1, ..., n, \quad z = 1, ..., m, \quad q = 1, ..., n, \quad x \in \{0, 1\} \tag{2.21}
\]

\[
\omega_{jq} = \frac{t}{\prod_{f=1}^{l+1} |L_{p,f}| \cdot \sum_{p=1}^{g} |L_{p,1}|}, \text{ where } j = 1, ..., n, \quad q = 1, ..., n, \tag{2.22}
\]

\[
\omega_{kj} = \frac{t}{\sum_{p=1}^{g} |L_{p,1}|}, \text{ where } j = 1, ..., n. \tag{2.23}
\]

To test whether traffic travels inside or across machines, \textbf{Equation 2.21} performs boolean operation on the placement variables \(x \in \{0, 1\}\), whose result is 1 if such communication occurred. \textbf{Equation 2.21} tests for communication on same machine for \(t_{ijq}\) and across machines for \(t_{ijzq}\).

To cover the case of vSwitch consumption, \textbf{Equation 2.19} does a cumulative sum of knapsack placement possibilities where edges go across knapsacks, and within existing
knapsacks, such that vSwitch capacity $c_{i,d+1}$ is honored. Link consumption calculated by Equation 2.20 performs the same calculation, but only takes edges going across knapsacks into account, such that Link capacity $c_{i,d+2}$ is honored. Equation 2.22 defines that items within layered groups $L_{p,d}$ employ distributive traffic, whereas Equation 2.23 defines that traffic $t$ originating from vertex $s$ is evenly distributed amongst items $j$ in layered groups adjacent to $s$.

This concludes the definition of the proposed Multidimensional Multiple Subset-Sum Problem with item placement restrictions and bandwidth analysis.

2.5 Discarded Problem Models

This thesis' initial approach to the VM placement problem was a hypergraph partitioning problem model. Inspired by Prof. Ümit V. Çatalyürek’s work on hypergraph partitioning [8], the intent was to model VMs as vertices, and machines as hypervertices. Weights would be attributed to directed edges in the hypergraph to denote communication costs, and additional weights to both vertices and hypervertices to denote consumption various resources. Due to the difficulties of modeling how arc weights are denoted depending on the entities they pass (machines) on their way to their target, and whether the edge consumes a machine’s Link and vSwitch or only vSwitch, it was difficult to employ High Performance Computing (HPC) hypergraph partitioning libraries to attack this problem.

The model was discarded, as amount of different weight variables required by this particular VM placement could not be facilitated by HPC hypergraph partitioners.

2.6 JaCoP Constraint Problem Solver

This section serves as a short introduction to constraint programming, and the features used to implement this thesis’ proposed VM placement algorithm.

Combinatorial problem model defined in 2.4.1 is expressed in terms of the Java Constraint Problem Library (JaCoP). A library implemented by Prof. Krzysztof Kuchcinski and PhD. Radosław Szymanek, which received a bronze medal in the annual competition of constraint programming solvers MiniZinc Challenge 2015 [22].

JaCoP facilitates integer, floating-point, logical, and set constraints, [15] of which integer constraints were chosen for solving this particular VM placement problem. Packing of VMs on machines is implemented through weighted sums of Finite Domain Variables (FDV)[15, p.11], which denote unknown variables whose value is restricted by an integer domain. For example, a FDV $X$ can be restricted to values within domain $X::[1..5]$. This property is used to let VMs consume machine resources up to a specific machine resource’s upper integer domain limit. Similarly, traffic edge weight FDVs can be created with successively smaller domains, such that search-space is reduced in a safe way.

2.6.1 Introduction by Sudoku example

Permit the premise that constraint programming can solve a Sudoku problem. The objective is to fill a $9 \times 9$ grid with numbers ranging $1, \ldots, 9$, such that each column, row, and
2.6 JaCoP Constraint Problem Solver

<table>
<thead>
<tr>
<th>Indomain method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndomainMin</td>
<td>selects a minimal value from the current domain of FDV</td>
</tr>
<tr>
<td>IndomainMax</td>
<td>selects a maximal value from the current domain of FDV</td>
</tr>
<tr>
<td>IndomainMiddle</td>
<td>selects a middle value from the current domain of FDV and then left and right values</td>
</tr>
<tr>
<td>IndomainRandom</td>
<td>selects a random value from the current domain of FDV</td>
</tr>
<tr>
<td>IndomainSimpleRandom</td>
<td>faster than IndomainRandom but does not achieve uniform probability</td>
</tr>
<tr>
<td>IndomainList</td>
<td>uses values in an order provided by a programmer</td>
</tr>
<tr>
<td>IndomainHierarchical</td>
<td>uses indomain method based provided variable-indomain mapping</td>
</tr>
</tbody>
</table>

Table 2.1: Value Selection Heuristics. Cited from JaCoP documentation [15, p.83].

the nine $3 \times 3$ boxes are restricted to contain unique numbers.

Since each cell inside the grid may only contain the numbers $1, \ldots, 9$, the cells are suitable to be expressed as Finite Domain Variables with domains $[1, \ldots, 9]$, which corresponds to the integers permitted by the rules of Sudoku.

JaCoP constraint programming library provides a readily applicable constraint named `alldifferent([x1, x2, \ldots, xn])` which in this case would cause the FDVs to be different within their permitted domains $[1, \ldots, 9]$. What follows is the vectorization of each row, column, and $3 \times 3$ box such that the constraint may be imposed.

After imposing `alldifferent` constraint the Sudoku problem has been successfully modeled, and a solution space has been thereby defined. By searching within the defined solution space using a Depth First Search a set of correct solutions can be found. In this thesis however, the focus is obtaining just one solution. Behavior of the Depth First search can be altered using several heuristics provided by the JaCoP library.

### 2.6.2 Search Heuristics

JaCoP facilitates tools for defining own search methods, by modifying existing ones, through search-plugins [15, p.49]. In the presented solution, only a subset of the variables used in 2.4.1 are part of the search. Employed search method builds on a Depth First Search (DFS), which organizes the search-space into a search tree [15, p.49]. Every node within the search tree is subject to become a choice point, where FDVs chosen by a variable selection heuristic Table 2.2 are assigned a value by a value selection heuristic Table 2.1. Subsequent constraint propagation is triggered once a FDV has been assigned a value. Should a constraint not be satisfied during this constraint propagation, then search is cut at that point in the tree, thereby preventing searching sub-trees where there are no solutions.

Chosen variable selection heuristic is MostConstrainedDynamic, which is motivated by how constraints on FDVs pile up during bandwidth analysis, which figuratively becomes a red thread which the MostConstrainedDynamic variable selection heuristic in Table 2.2 can follow. However, this does not mean MostConstrainedDynamic is the best variable selection heuristic, as SmallestDomain variable selection heuristic outperforms MostConstrainedDynamic when there are few VM placement constraints.

In order to minimize computation time a hierarchical heuristic was chosen, such that each FDV can be subjected to a different value selection heuristic. The motivation behind
2. Problem Model as Combinatorial Optimization

Comparator Description

- **SmallestDomain**: selects FDV which has the smallest domain size
- **MostConstrainedStatic**: selects FDV which has most constraints assigned to it
- **MostConstrainedDynamic**: selects FDV which has the most pending constraints assign to it
- **SmallestMin**: selects FDV with the smallest value in its domain
- **SmallestMax**: selects FDV with the smallest maximal value in its domain
- **LargestDomain**: selects FDV with the largest domain size
- **LargestMin**: selects FDV with the largest value in its domain
- **MaxRegret**: selects FDV with the largest difference between the smallest
- **WeightedDegree**: selects FDV with the highest weight divided by its size

Table 2.2: Variable Selection Heuristics. Cited from JaCoP documentation [15, p.83].

this is because VM placement variables are an $m \times n$ logical matrix, whose element’s domains are $[0..1]$. Bandwidth variables on the other hand are positive integers with domains typically reaching $[0..10000]$. Because of this FDV’s related to bandwidth do not benefit from being treated with the same variable selection heuristic [15, p.83]. More specifically, important variables consist of the following heuristics which act on the solution-search tree:

- **0/1 Placement Matrix FDVs: IndomainRandom.**
  Perhaps a counterintuitive choice of value selection heuristic, as IndomainMax would cause a VM to be placed immediately. However, using IndomainMax skews the VM placement. Because the $[0..1]$ domains residing in the placement matrix will be immediately assumed to be 1, and because the matrix traversal is iterative, some machines become more heavily utilized than others, which is why IndomainRandom is preferred. Therefore it can be said that IndomainRandom produces a more balanced VM placement.

- **Incoming VNF traffic FDVs: IndomainMax.**
  Incident traffic is not maximized, however should always be as large as possible even when the optional bandwidth objective is not used.

- **Communicating VMs not sharing machine variable: IndomainMin.**
  This FDV is the boolean result of a Reified constraint [15, p.74] which is true when two VMs do not share a machine. Because VMs communicating across machines can be detrimental to IaaS Cloud performance, the value selection heuristic will always assume this value is false.

- **Communicating VMs sharing machine FDVs: IndomainMax.**
  This FDV is the boolean result of a Reified constraint [15, p.74] which is true when two VMs share a machine. VMs communication within machines is more preferred than VMs communicating across machines. The value selection heuristic will always assume this value is true.

- **Cumulative bandwidth into VNFs FDVs: IndomainMax.**
  Selects the maximum integer value permitted by FDV’s domain, such that incident traffic into Cloud can be as large as possible. This does not maximize bandwidth, only selects the largest value.
2.6 JaCoP Constraint Problem Solver

- All above mentioned FDVs: **IndomainHierarchical**.
  Individually attributed value selection heuristics for all previously mentioned FDVs is made possible by supplying the attributes with *IndomainHierarchical*. This value selection heuristic provides variable-indomain mapping by supplying indomain methods using a Java HashMap<T, Indomain<T>>.

- All other FDVs: **IndomainRandom**.
  This is the default value selection heuristic for all variables which the Indomain-Hierarchical value selection heuristic does not supply an indomain method for. Essentially unused because all variables already have an indomain method supplied using IndomainHierarchical.

Traffic across machines is more undesirable than traffic within machines, which is imposed by attributing IndomainMax value selection heuristic to traffic arcs between VMs within one machine, and IndomainMin to traffic arcs for VMs communicating across machines.

To summarize, MostConstrainedDynamic variable selection heuristic, and above listed value selection heuristics are employed by the Depth First Search (DFS) based restart-search seen in Listing 2.1, from which no-good elements are derived, such that every restart contributes with new no-goods. Search is run until one solution is found. If no solution is found, or solution-space is exhausted, then the entire search routine fails. Upon failure an assumption is made, that the failure is due to lack of machines to place VMs on, and search procedure is promptly restarted with an extra machine, which thereby expands the search-space.

It is important to mention that only a certain subset of FDV variables have been deemed suitable to become selection points within the DFS search tree. Theoretically however, every FDV within the problem model could be explicitly treated with a different value selection heuristic. Whether this is something desirable has not been investigated, but will be discussed later in **chapter 4**.


2. Problem Model as Combinatorial Optimization

This section serves as an overview of the JaCop implementation stated in 2.4.1. Input to this implementation consists of manually defined machine sizes, and variably specified VMs, Link, and vSwitch, which can be seen in Appendix A. Primary constraints employed are linear integer constraints \[ \text{[15], p.22} \], which ensure that knapsack capacities are never exceeded. Linear integer constraints are a weighted sums being constrained by a FDV value which relates to one of the operators: \(<, \leq, >, \geq, =\). For affinities and anti-affinities conditional constraints are employed, which have the nature IfThen(constraint1, constraint2), where constraint1 would typically express a placement-matrix equality statement, and thereby dynamically impose constraint2 if satisfied.

The bandwidth analysis part of procedure 2 is structured such that all possible edge weights are added to a Link and vSwitch ArrayList<IntVar> respectively, where IntVar stands for a FDV. The value of each edge depends on whether traffic actually occurred according to placement, which is decided by the result of \(t_{ijzq}\) (See Equation 2.21),

// It conducts the search with restarts from which the no-goods are derived.
*/
public boolean searchWithRestartsHierarchical() {
    /* immediate fail condition */
    if (fail == true)
        return false;
    /* initialize */
    boolean result = false;
    boolean timeout = true;
    /* Define DFS search */
    search = new DepthFirstSearch<IntVar>();
    /* employ MostConstrainedDynamic FDV variable selection heuristic, and */
    /* IndomainHierarchical value selection heuristic which supplies indomain */
    /* mapping for each FDV individually. Fall back to IndomainRandom value */
    /* selection heuristic for FDVs which IndomainHierarchical does not */
    /* specify indomain mapping for */
    SelectChoicePoint<IntVar> select = new SimpleSelect<IntVar>(
        vars.toArray(new IntVar[1]), new MostConstrainedDynamic<IntVar>(),
        new IndomainHierarchical<IntVar>(/* hierarchicalMap, new IndomainRandom<IntVar>(*/));
    /* prepare no-goods collection */
    NoGoodsCollector<IntVar> collector = new NoGoodsCollector<IntVar>(
        search.setExitListener(collector);
        search.setTimeOutListener(collector);
        search.setExitListener(collector);
    /* search with restarts */
    while (timeout) {
        search.setPrintInfo(false);
        /* restart search after 1000 nodes */
        search.setNodesOut(1000);
        result = search.labeling(store, select);
        timeout &= collector.timeOut;
        /* prepare for search restart */
        search = new DepthFirstSearch<IntVar>();
        collector = new NoGoodsCollector<IntVar>(
            search.setExitListener(collector);
            search.setTimeOutListener(collector);
            search.setExitListener(collector);
        }
        return result;
    }
}

Listing 2.1: Solution search routine employed. Builds on restart-search with no-goods collection.

2.6.3 Implementation

This section serves as an overview of the JaCop implementation stated in 2.4.1. Input to this implementation consists of manually defined machine sizes, and variably specified VMs, Link, and vSwitch, which can be seen in Appendix A. Primary constraints employed are linear integer constraints \[ \text{[15], p.22} \], which ensure that knapsack capacities are never exceeded. Linear integer constraints are a weighted sums being constrained by a FDV value which relates to one of the operators: \(<, \leq, >, \geq, =\). For affinities and anti-affinities conditional constraints are employed, which have the nature IfThen(constraint1, constraint2), where constraint1 would typically express a placement-matrix equality statement, and thereby dynamically impose constraint2 if satisfied.

The bandwidth analysis part of procedure 2 is structured such that all possible edge weights are added to a Link and vSwitch ArrayList<IntVar> respectively, where IntVar stands for a FDV. The value of each edge depends on whether traffic actually occurred according to placement, which is decided by the result of \(t_{ijzq}\) (See Equation 2.21),
whose behavior is implemented with an AndBool constraint. This means that the sum-
mation of each Link & vSwitch knapsack is mostly trailing zeros, among which a few are
edges carrying traffic. Therefore it can also be said that implementation of Procedure 2
exploits the fact that constraints can be applied to arrays of dynamic size.

Procedure 1 MDMSSP Placement

1: function MDMSSP($m, n, d, x_{m,n}, w_{n,d}, c_{n,d}, \text{Affinities}, \text{Anti-Affinities}$)
2: \hspace{1em} $m \leftarrow m$ amount of Machines
3: \hspace{1em} $n \leftarrow n$ amount of VMs
4: \hspace{1em} $d \leftarrow d$ amount of different knapsack volumes
5: \hspace{1em} $x_{m,n} \leftarrow 0/1$ placement matrix of size $m \times n$
6: \hspace{1em} $w_{n,d} \leftarrow n$ item weights with $d$ weight categories
7: \hspace{1em} $c_{n,d} \leftarrow n$ knapsacks with $d$ volumes
8: \hspace{2em} \textbf{for} $j = 1$ to $n$ \textbf{do}
9: \hspace{3em} \textbf{impose constraint} $\sum_{i=1}^{n} x_{i,j} = 1$ \hspace{1em} $\triangleright$ Each VM must be placed once.
10: \hspace{2em} \textbf{for} $r = 1$ to $d$ \textbf{do}
11: \hspace{3em} \textbf{for} $i = 1$ to $m$ \textbf{do}
12: \hspace{4em} \textbf{impose constraint} $\sum_{j=1}^{n} w_{j,r} \cdot x_{i,j} - c_{i,r} \leq c_{i,r}$ \hspace{1em} $\triangleright$ Honor knapsack capacities.
13: \hspace{2em} \textbf{for each item} $j$ which has Affinity to an item $q$ where $j, q \in n$ and $j \neq q$ \textbf{do}
14: \hspace{3em} \textbf{for} $i = 1$ to $m$ \textbf{do}
15: \hspace{4em} \textbf{impose conditional constraint} if $x_{i,j} = 1$ then $x_{i,j} = x_{i,q}$
16: \hspace{2em} \textbf{for each item} $j$ which has Anti-Affinity to an item $q$ where $j, q \in n$ and $j \neq q$ \textbf{do}
17: \hspace{3em} \textbf{for} $i = 1$ to $m$ \textbf{do}
18: \hspace{4em} \textbf{impose conditional constraint} if $x_{i,j} = 1$ then $x_{i,j} \neq x_{i,q}$
Problem Model as Combinatorial Optimization

Procedure 2 Bandwidth Analysis

1: function BWA($m$, $n$, $x_{m,n}$, $g$, $p$, $L_{g,p}$, $V_p$, $E_p$, $\Omega$, $s$, $L_m$, $S_m$, $t$, $b$)
2:   $m \leftarrow m$ amount of Machines
3:   $n \leftarrow n$ amount of VMs
4:   $x_{m,n} \leftarrow 0/1$ placement matrix of size $m \times n$
5:   $g \leftarrow g$ amount of VNFs
6:   $l \leftarrow l$ amount of layered groups in a VNF
7:   $L_{g,k} \leftarrow k$ layered groups in $g$ graphs, where $k$ is VNF-specific
8:   $\Omega \leftarrow$ set of edge weights (see 2.22, 2.23)
9:   $s \leftarrow$ traffic source vertex
10:  $t \leftarrow$ unknown traffic generated by vertex $s$
11:  $b \leftarrow$ bandwidth objective
12:  $G_p \leftarrow (V_p, E_p, \Omega, s)$ all VNFs
13:  $L_m \leftarrow m$ amount of Link knapsacks
14:  $S_m \leftarrow m$ amount of vSwitch knapsacks
15:  $L'_{m,x} \leftarrow x$ amount of usages of $m$ Links, where $x$ is of unknown size
16:  $S'_{m,y} \leftarrow y$ amount of usages of $m$ vSwitches, where $y$ is of unknown size
17:   for $p = 1$ to $g$ do
18:     for $i = 1$ to $k - 1$, where $\max(k) \in G_p$ do
19:       for $j = 1$ to $n$ do
20:         for $q = 1$ to $n$ do
21:           for $i_1 = 1$ to $m$ do
22:             $S'_{i_1,y+1} \leftarrow \omega_{s,j} \cdot x_{i_1,j}$  \hspace{1cm} \triangleright \text{Traffic from s.}$
23:             $L'_{i_1,x+1} \leftarrow \omega_{s,j} \cdot x_{i_1,j}$  \hspace{1cm} \triangleright \text{Traffic from s.}$
24:           for $i_2 = 1$ to $m$ do
25:             $S'_{i_2,y+1} \leftarrow \omega_{i_1,j} \cdot t_{i_1,i_2,q}$  \hspace{1cm} \triangleright \text{VM traffic within machines consuming vSwitch. See Equation 2.21}$
26:             $L'_{i_2,x+1} \leftarrow \omega_{i_1,j} \cdot t_{i_1,i_2,q}$  \hspace{1cm} \triangleright \text{VM traffic across machines consuming vSwitch. See Equation 2.21}$
27:           for $i = 1$ to $m$ do
28:             impose constraint $\sum_{x=1}^{\max(x)} L'_{i,x} \leq L_i$  \hspace{1cm} \triangleright \text{Honor Link knapsack capacities.}$
29:             impose constraint $\sum_{y=1}^{\max(y)} S'_{i,y} \leq S_i$  \hspace{1cm} \triangleright \text{Honor vSwitch knapsack capacities.}$
30:           impose constraint $t \geq b$  \hspace{1cm} \triangleright \text{Meet bandwidth objective b.}$
Chapter 3
Results

Because this work is based on a model whose quality, at this point, is unknown identification of placement quality has been a priority. It can be therefore said that work related to the validation of the presented VM placement model is beyond the scope of this thesis. Thus, placement quality is evaluated.

Results are presented in two parts. One dedicated presenting the VM placement algorithm, and the other presenting VM placement with bandwidth analysis. A predefined set of VMs are going to be placed in a predefined set of datacenters found in Table 3.1 and Table 3.2. The vSwitch is a preexisting resource consumer, whose resource consumption is specified in Table 3.1.

These placements will not be subjected to affinity and anti-affinity placement restriction rules, as such placements would essentially be predefined, which would hinder an evaluation case where the placement algorithm, and heuristic search algorithm, have free reign.
### Table 3.1: VM resource requirement metrics.

<table>
<thead>
<tr>
<th>VM size</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>XLarge</th>
<th>XXLarge</th>
<th>vSwitch</th>
</tr>
</thead>
<tbody>
<tr>
<td>vCPU</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>RAM [MB]</td>
<td>512</td>
<td>1024</td>
<td>2048</td>
<td>4096</td>
<td>8192</td>
<td>32</td>
</tr>
<tr>
<td>Disk [MB]</td>
<td>1024</td>
<td>2048</td>
<td>3072</td>
<td>8192</td>
<td>16384</td>
<td>512</td>
</tr>
</tbody>
</table>

### Table 3.2: Datacenter configurations. Where Datacenter #1 is populated with Small Machines, whereas Datacenters #2,#3 house Medium Machines. See Table 3.3.

<table>
<thead>
<tr>
<th>Datacenter</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Medium</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Large</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>XLarge</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>XXLarge</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Total [# VMs]</td>
<td>8</td>
<td>16</td>
<td>28</td>
</tr>
</tbody>
</table>

### Table 3.3: Machine resource configuration. 1 PPS = 650Bps.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Machine</td>
<td>32</td>
<td>8192</td>
<td>50</td>
<td>50000</td>
<td>100000</td>
</tr>
<tr>
<td>Medium Machine</td>
<td>32</td>
<td>16384</td>
<td>50</td>
<td>50000</td>
<td>100000</td>
</tr>
</tbody>
</table>
3.1 Placement Quality

3.1.1 Testing methodology

Proposed placement algorithm will be evaluated by introducing 3 different datacenter configurations, which will be comprised of VMs in Table 3.1. Datacenter configurations are shown in Table 3.2, inside which VMs are placed on machines seen in Table 3.3. The machines differ only in RAM size, which is motivated by the necessity of keeping machines small enough for the placement algorithm to make decisions, but big enough to house the largest VM specified in Table 3.1.

Because the proposed algorithm will always look for solutions where VMs fit on minimal amount of machines, it is important to look at the quality of those solutions. This will be evaluated on VNF groups, which house VMs that do not have any placement restriction rules. Quality of a solution is determined by how evenly resource consumption is distributed across machines, given that the placement algorithm has received free reign due to the lack of affinity, and anti-affinity placement restrictions.

3.1.2 Results

Table 3.4: Geometric mean machine of % resource consumption in Datacenter #1 placement when placed on Small Machines (see Table 3.3). Table represents average resource consumption for 30 consecutive placements, and their approximate deviations. Values are: geometric mean ± standard deviation.

<table>
<thead>
<tr>
<th>Datacenter #1</th>
<th>Small Machine 0</th>
<th>Small Machine 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM [%]</td>
<td>0.5745 ± 0.2012</td>
<td>0.6127 ± 0.2012</td>
</tr>
<tr>
<td>vCPU [%]</td>
<td>0.4739 ± 0.1268</td>
<td>0.4933 ± 0.1268</td>
</tr>
<tr>
<td>Disk [%]</td>
<td>0.1763 ± 0.0618</td>
<td>0.1867 ± 0.0618</td>
</tr>
</tbody>
</table>

Table 3.5: Geometric mean of % machine resource consumption in Datacenter #2 placement when placed on Medium Machines (see Table 3.3). Table represents average resource consumption for 30 consecutive placements, and their approximate deviations. Values are: geometric mean ± standard deviation.

<table>
<thead>
<tr>
<th>Datacenter #2</th>
<th>Medium Machine 0</th>
<th>Medium Machine 1</th>
<th>Medium Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM [%]</td>
<td>0.6026 ± 0.1472</td>
<td>0.7721 ± 0.1276</td>
<td>0.7809 ± 0.104</td>
</tr>
<tr>
<td>vCPU [%]</td>
<td>0.8958 ± 0.1061</td>
<td>0.8593 ± 0.1135</td>
<td>0.8927 ± 0.0955</td>
</tr>
<tr>
<td>Disk [%]</td>
<td>0.3678 ± 0.099</td>
<td>0.4881 ± 0.0835</td>
<td>0.4916 ± 0.0705</td>
</tr>
</tbody>
</table>
Table 3.6: Geometric machine of % resource consumption in Datacenter #3 placement when placed on Medium Machines (see Table 3.3). Table represents average resource consumption for 30 consecutive placements, and their approximate deviations. Values are: geometric mean ± standard deviation.

<table>
<thead>
<tr>
<th>Datacenter #3</th>
<th>Medium M. 0</th>
<th>Medium M. 1</th>
<th>Medium M. 2</th>
<th>Medium M. 3</th>
<th>Medium M. 4</th>
<th>Medium M. 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM [%]</td>
<td>0.5284 ± 0.0513</td>
<td>0.5428 ± 0.0909</td>
<td>0.8328 ± 0.097</td>
<td>0.8038 ± 0.0763</td>
<td>0.7648 ± 0.1015</td>
<td>0.7557 ± 0.0849</td>
</tr>
<tr>
<td>vCPU [%]</td>
<td>0.979 ± 0.0438</td>
<td>0.9229 ± 0.0806</td>
<td>0.9376 ± 0.0596</td>
<td>0.7934 ± 0.1048</td>
<td>0.6918 ± 0.1325</td>
<td>0.7076 ± 0.11</td>
</tr>
<tr>
<td>Disk [%]</td>
<td>0.304 ± 0.0269</td>
<td>0.3515 ± 0.074</td>
<td>0.5084 ± 0.0788</td>
<td>0.512 ± 0.0331</td>
<td>0.4805 ± 0.054</td>
<td>0.4891 ± 0.05</td>
</tr>
</tbody>
</table>

3.1.3 Discussion

The most vital resource vCPU, and RAM are relatively spread in the cases in Table 3.4 and Table 3.5. Congestion in vCPU availability in Table 3.6 made the placement algorithm skew the VM placement, such that Machines 0 through 2 have a greater vCPU load than Machines 4 and 5. Despite the placement matrix being treated with an IndomainRandom heuristic, there is still some bias. The order of the FDV variables in the placement matrix dictate the order of evaluation. Bias becomes apparent as vCPU usage is appears to be descending when seen from left, to right, where left is Machine 0 and right is Machine 5. Incidentally, the vCPU capacity constraint is also the first capacity checked for validity upon VM placement, as it’s regarded as the most vital one. Thereby the order of FDV evaluation plays a role in how balanced a placement is, which is a behaviour stemming from the chosen search heuristic.
3.2 Placement with Bandwidth Analysis

3.2.1 Testing Methodology

Datacenters’ VMs in Table 3.2 will be put in particular network configurations show in Figure 3.1. Placement commence without a bandwidth objective, then with a bandwidth objective. Network resources are taxed as previously stated in Figure 1.3. As mentioned in the beginning of this chapter, affinity and anti-affinity placement restriction rules will not apply for these placements, such that the quality of the placement can be evaluated.
### 3. Results

#### 3.2.2 Results

**Table 3.7:** Placement with bandwidth analysis of Datacenter #1 when bandwidth is not maximized. Values are: geometric mean of percentage utilization ± standard deviation.

<table>
<thead>
<tr>
<th>Datacenter #1</th>
<th>Small Machine 0</th>
<th>Small Machine 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM [%]</td>
<td>0.6617 ± 0.208</td>
<td>0.5213 ± 0.208</td>
</tr>
<tr>
<td>vCPU [%]</td>
<td>0.5133 ± 0.1166</td>
<td>0.4595 ± 0.1166</td>
</tr>
<tr>
<td>Disk [%]</td>
<td>0.2019 ± 0.0658</td>
<td>0.1587 ± 0.0658</td>
</tr>
<tr>
<td>Link [%]</td>
<td>0.3764 ± 0.0876</td>
<td>0.342 ± 0.2032</td>
</tr>
<tr>
<td>vSwitch [%]</td>
<td>0.954 ± 0.1215</td>
<td>0.8501 ± 0.0618</td>
</tr>
</tbody>
</table>

External traffic [PPS] | 31403 ± 13686.78

**Table 3.8:** Placement with bandwidth analysis of Datacenter #1 when bandwidth is maximized. Values are: percentage resource utilization.

<table>
<thead>
<tr>
<th>Datacenter #1</th>
<th>Small Machine 0</th>
<th>Small Machine 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM [%]</td>
<td>0.3137</td>
<td>0.9411</td>
</tr>
<tr>
<td>vCPU [%]</td>
<td>0.2667</td>
<td>0.7334</td>
</tr>
<tr>
<td>Disk [%]</td>
<td>0.0808</td>
<td>0.3030</td>
</tr>
<tr>
<td>Link [%]</td>
<td>0.5</td>
<td>0.2083</td>
</tr>
<tr>
<td>vSwitch [%]</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

External traffic [PPS] | 62502

**Table 3.9:** Placement with bandwidth analysis of Datacenter #2 when bandwidth is not maximized. Values are: geometric mean of percentage utilization ± standard deviation.

<table>
<thead>
<tr>
<th>Datacenter #2</th>
<th>Medium Machine 0</th>
<th>Medium Machine 1</th>
<th>Medium Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM [%]</td>
<td>0.7629 ± 0.1373</td>
<td>0.7508 ± 0.139</td>
<td>0.631 ± 0.1578</td>
</tr>
<tr>
<td>vCPU [%]</td>
<td>0.9646 ± 0.0625</td>
<td>0.9014 ± 0.1128</td>
<td>0.7834 ± 0.1144</td>
</tr>
<tr>
<td>Disk [%]</td>
<td>0.4717 ± 0.0894</td>
<td>0.4733 ± 0.0906</td>
<td>0.3963 ± 0.1044</td>
</tr>
<tr>
<td>Link [%]</td>
<td>0.3252 ± 0.0988</td>
<td>0.2425 ± 0.1227</td>
<td>0.236 ± 0.1381</td>
</tr>
<tr>
<td>vSwitch [%]</td>
<td>0.7959 ± 0.2333</td>
<td>0.5265 ± 0.2904</td>
<td>0.5232 ± 0.3182</td>
</tr>
</tbody>
</table>

External traffic [PPS] | 23748 ± 3651.48
### 3.2 Placement with Bandwidth Analysis

**Table 3.10:** Placement with bandwidth analysis of Datacenter #2 when bandwidth is maximized. Values are: percentage resource utilization.

<table>
<thead>
<tr>
<th>Datacenter #2</th>
<th>Medium Machine 0</th>
<th>Medium Machine 1</th>
<th>Medium Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM [%]</td>
<td>0.7514</td>
<td>0.7828</td>
<td>0.6575</td>
</tr>
<tr>
<td>vCPU [%]</td>
<td>0.9333</td>
<td>1</td>
<td>0.7333</td>
</tr>
<tr>
<td>Disk [%]</td>
<td>0.4848</td>
<td>0.4646</td>
<td>0.4242</td>
</tr>
<tr>
<td>Link [%]</td>
<td>0.3758</td>
<td>0.35</td>
<td>0.3759</td>
</tr>
<tr>
<td>vSwitch [%]</td>
<td>0.9523</td>
<td>1</td>
<td>0.8018</td>
</tr>
</tbody>
</table>

| External traffic [PPS] | 45092 |

**3.2.3 Discussion**

Bandwidth analysis in Table 3.7 without a bandwidth objective portrays a great deviation in found handleable duplex traffic. Once given a bandwidth objective as seen in Table 3.8 it finds very good solutions for small systems. There is a high chance that the solution in Table 3.8 is a global optimum, as vSwitch is completely exhausted on both machines. None of the machine’s resources are overloaded, however a great margin of resource balance has been sacrificed to meet the bandwidth objective, which is clearly observed in Machine 1’s 94% RAM consumption, which is a contrast to Machine 0’s 31% RAM consumption. Algorithm’s primary objective has become bandwidth, and resource consumption is now secondary.

Despite maximizing bandwidth in Table 3.10 the external traffic value is not as high as in Table 3.8, which speaks for room for improvement in the solution search heuristic. Tailoring heuristics for VM placement and VM placement with bandwidth analysis separately would likely lead to much better solutions. Such tailoring could be applied for communication edges with rules which govern probability of an edge leading to a functioning placement.

Because bandwidth analysis generates a lot more constraints, previously tested Datacenter #3 could not be analyzed for bandwidth, and will be regarded as a timeout. However could be evaluated in the future if a computer with enough RAM is facilitated for testing.
3. Results
Chapter 4

Conclusion

As stated in Chapter 3 this work builds on a model whose quality is unknown, and its evaluation will be listed as future work. All material is presented such that it obeys Ericsson NDA. Also, validation of the VM placement model is beyond the scope of this thesis.

In summary, the proposed VM placement algorithm has built on an NP-hard (0/1 Multiple Knapsack) problem model, and added several NP-complete sub-problems in order to factor in the various resources machines provide, which VMs consume. Its implementation in the Java Constraint Programming Library (JaCoP) consisted of 1780 LOC Java code, out of which 650 LOC were used to build an abstract model of a Cloud data center.

Out of the evaluated approaches, hypergraph partitioning, and combinatorics, the latter proved feasible. Utilizing the JaCoP constraint programming library’s versatility, combined with abstractions inherent to Java, a model of a data center was successfully implemented. In order verify applicability of the combinatorial problem model a 0/1 Knapsack implementation in JaCoP trialled with success. However, because knapsack items are denoted by both weight and cost, where cost is not required for this particular VM placement, it was scrapped by making knapsack item weights and costs equal. By making use of a 0/1 Knapsack’s special case where each item’s weight = cost the packing model became a Subset-sum.

The Subset-sum was scaled to Multiple Subset-sum to factor in multiple machines, and then scaled to 3-dimensional Multiple Subset-sum to factor three machine resources: RAM, vCPU, and disk. At this point VM placement was facilitated. In order to model VM communication, the items packed by the 3-dimensional Multiple Subset-sum communicate across the knapsacks they are packed into, thereby consuming capacities of two new knapsacks (Link and vSwitch) which are attributed to each machine.

The complexity imposed on calculation of the Link and vSwitch knapsacks proved to be an obstacle. An alternative implementation of the current algorithm, such that memory is dynamically allocated upon constraint evaluation, proved to be slower however, which leaves the bandwidth analysis very RAM intensive. Because of this, the bandwidth analyzing Procedure 2 has been made optional, such that Procedure 1 can be run on its own.
4. Conclusion

Bandwidth analysis time complexity makes it applicable to verification of placement optimality of small systems. However large placements with bandwidth analysis put the NP-hard complexity of this problem to trial, leaving it unfit for systems of sizes such as Datacenter #3 in 3.2. As future work however, it could be extended to place several smaller systems one after another, with a degree of optimality, such that they can be part of a bigger, alas less optimal system.

Expressing VMs as communicating items in a Knapsack problem formulation, making those items communicate, and identifying their means of communication is inefficient. In order to make the proposed solution applicable for large systems heuristics need to be tailored. Proposed algorithm uses a restart-search described in subsection 2.6.2 and has been adapted using facilities provided by JaCoP. However, some parts of it can be further tailored to bandwidth analysis in particular, which spells for future work in this area.
Bibliography


Appendices
Appendix A

JSON specifications


```
{
    "vmUUID": "100000",
    "vmName": "VM 0",
    "vmRamConsumption": 384,
    "vmDiskConsumption": 1024,
    "vmVcpuConsumption": 3,
    "vmPpsConsumption": 0,
    "vmBpsConsumption": 0,
    "groupId": 1,
    "vmID": 1,
    "vmAffinities": [],
    "vmAntiAffinities": [],
    "packetByteSize": 650
}
```

Listing A.2: JSON specification of the Link.

```
{
    "linkUUID": "100",
    "linkName": "Linecard 1x10GB",
    "maxLinkBps": 650000000,
    "packetByteSize": 650,
    "preferredUnit": "BPS"
}
```

Listing A.3: JSON specification of the vSwitch.

```
{
    "vSwitchUUID": "1000",
    "vSwitchName": "Openstack V2",
    "vSwitch": {
        "vmUUID": "1000",
        "vmName": "Openstack V2 vSwitch VM",
        "vmRamConsumption": 256,
        "vmDiskConsumption": 100,
        "vmVcpuConsumption": 2,
        "vmPpsConsumption": 0,
        "vmBpsConsumption": 0,
        "groupId": 0,
        "vmAffinities": [],
        "vmAntiAffinities": [],
        "packetByteSize": 650
    },
    "vSwitchMaxPps": 500000,
    "vSwitchMaxBps": 650000000,
    "packetByteSize": 650,
    "preferredUnit": "PPS"
}
```
Optimering av resurshantering i virtualiserade miljöer

POPULÄRVETENSKAPLIG SAMMANFATTNING Jakub Górski

Dagens värld omgivar oss av teknik, och nya tekniska ord och benämningar tillkommer jämt. Främst bland dessa förekommer ordet ”Cloud” som är något de flesta använt, men kanske inte riktigt känner till. Arbetet föreslår en kombinatorisk modell för optimering av resursanvändning i Infrastructure as a Service (IaaS) Cloud system.


Problemets plan i att packa flera ryggsäckar på bästa sätt. Man vill inte överskriva ryggsäckens volym, inte packa för tungt, och man vill gärna välja vad man packar i ryggsäcken så man får plats med så mycket som möjligt. På ryggsäckarnas packningslista så kommer t.ex. inte kylskåpet befinna sig, men möjligtvis kartan och kompassen som väger lite och tar liten plats.

Man kan på samma sätt säga att man packar Virtuella Maskiner ner i flertalet servrar. Med skillnaden att datorerna har tre volymer (arbetsminne, processorkraft och diskutrymme) jämfört med ryggsäcken, och Virtuella Maskiner som packas ner i servrarna förbrukar dessa tre volymer.
