On The Classification of Resource Consolidation Management Problems

by

Steven Lonergan
B.Sc., University of Victoria, 2010

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

On The Classification of Resource Consolidation Management Problems

by

Steven Lonergan
B.Sc., University of Victoria, 2010

Supervisory Committee

Dr. U. Stege, Supervisor
(Department of Computer Science, University of Victoria)

Dr. Y. Coady, Departmental Member
(Department of Computer Science, University of Victoria)

Supervisory Committee

Dr. U. Stege, Supervisor
(Department of Computer Science, University of Victoria)

Dr. Y. Coady, Departmental Member
(Department of Computer Science, University of Victoria)

ABSTRACT

This thesis focuses on computatal problems regarding the allocation of resources within a data center that services a cloud. This problem is formally know as Resource Consolidation Managmenet (RCM). In this thesis we analyse current RCM methods from the literature with respect to computational problem defintions and propose a framework to allow the classification and comparison of RCM solutions. With a decade of research in the field, this framework should be intuitive such that any researcher can easily use it to define computational problems in the field of RCM and reverse engineer problem definitions from exisiting solutions for RCM Problems. Finally our framework should be extendable: as the field continues to grow, the framework should be able to adapt to meet future needs.

Besides presenting the framework, we analyse computational problems obtained by the framework in terms of its classical complexity. We show several of those problems to be **NP**-complete and discuss variants that are solvable in polynomial time. A further contribution is the exploration of different comparison tools for solutions of RCM problems.

Contents

Supervisory Committee **ii**

Abstract **iii**

Table of Contents **iv**

List of Tables **vi**

List of Figures **vii**

Acknowledgements **viii**

Dedication **ix**

1 Introduction **1**

1.1 Motivation 2

1.2 Contributions 3

1.3 Thesis Overview 4

2 Tools and Terminology **5**

2.1 Basic Terminology for RCM Problems 6

2.2 Basics From Computational Complexity 8

2.2.1 **NP**-completeness Proofs via Reduction and Certificate 11

2.3 Computational Problems Related to RCM 12

2.3.1 Bin Packing Problems 12

2.3.2 Knapsack Problems 13

3 Related Work **15**

4 A Framework for RCM Problems **19**

4.1 Framework 19

4.1.1 Framework Formulation 19
4.1.2 Framework Construction 21
4.1.3 Criteria Selection 22
4.2 Framework Applications 25
4.2.1 Practical Applications 26
4.3 Case Studies 28
4.3.1 Remarks 33
4.4 Computational Complexity of RCM Variants 34
4.4.1 Framework Member Analysis: **NP**-completeness 34
4.5 Framework Member Analysis: Polynomial Time Results 39

5 Summary and Future Work **43**
5.1 Summary 43
5.2 Future Work 45
5.2.1 Framework Extensions 45
5.2.2 Analysis Extensions 45

Bibliography **47**

List of Tables

Table 4.1 Application of the framework to RCM solutions. Note that in the last row priorities are given where applicable. 29

List of Figures

Figure 2.1 Relationship between **P**, **NP**, **NP**-Complete and **NP**-hard: adapted from [13] 11

Figure 4.1 A visual representation of breaking tasks into non-redundant sets. The figure has been simplified to ensure legibility. 25

Figure 4.2 Transformation of RCM analysis 27

ACKNOWLEDGEMENTS

I would like to thank:

My Family, For supporting me in through all the ups and downs and for always just being a phone call away.

Dr. Ulrike Stege, for providing more support than I could ever imagine.

Dr. Yvonne Coady, for getting me excited about computer science in my first year at UVic.

Andrea Pugh, for the long nights and phone calls that helped make the last two years easier.

My wonderful friends, for helping me get through the tough times.

Naomi and Megan, for letting me pretty much live at their house during the last month of grad school and for teaching me that peanut butter is delicious on EVERYTHING!

Our deepest fear is not that we are inadequate. Our deepest fear is that we are powerful beyond measure. It is our light, not our darkness, that most frightens us. Your playing small does not serve the world. There is nothing enlightened about shrinking so that other people wont feel insecure around you. We are all meant to shine as children do. Its not just in some of us; it is in everyone. And as we let our own lights shine, we unconsciously give other people permission to do the same. As we are liberated from our own fear, our presence automatically liberates others.

Adapted from Marianne Williamson

DEDICATION

To A.P. for all the love and support along the way.

Chapter 1

Introduction

Integrating computer science seamlessly into our everyday lives through the use of computers and software has been a major theme over the past decade. Cloud computing, through services such as Apple's iCloud [6], the Google Cloud Platform [5] and Microsoft's cloud services [7] is starting to explode into the mainstream. Joining projects such as Google's development of driver-less cars, and California's approval to allow them on highways, crowd sourcing projects such as Wikipedia [9], Folding@Home [3], SETI@Home [8] and reCAPTCHA, computer science is beginning to be recognized in daily life.

Behind all of the marketing buzz and catch phrases of cloud computing is the central idea of transforming computation into a utility, similar to that of electricity or water [25, 12]. Cloud computing achieves this by selling computational power to users. Note that the users are not required to know where or how the physical machines are being managed. The concept of utility computing is similar to that of electricity: users want to use power whenever they need it, but rarely think about where it actually comes from. A cloud provider acting behind the scenes manages the distribution of computational power to its users.

This thesis focuses on the computational challenges that service providers face when managing resources. Studying problems on the management of resources within the cloud has gained considerable momentum in the last decade, although it was originally proposed in the 1960s. Basic concepts from that time can be found in a presentation by John McCarthy [27] in 1961 and a book by Parkhill [38], published in 1966. Limitations and the cost of hardware contributed to the slow transition from idea to practice. One can view cloud computing in two different ways: the *user's view* and the *provider's view*. Each has a unique perspective of the cloud, and each

has goals that need to be met. For the *cloud provider*, each user must be satisfied while maintaining their own constraints. What the provider is required to do to satisfy a user is formally detailed in a contract, typically referred to as a *Service Level Agreement* or *SLA*. SLAs are legal documents that ensure the user a certain level of service for a certain price. The *cloud user* on the other hand defines the level of service they expect using the SLA and then demands to be able to access the cloud without any interruptions.

Typically, the service provider must continuously move resources to ensure constant user satisfaction. This task of continuously feeding resources to users is commonly referred to as *Resource Consolidation Management (RCM)*[1].

1.1 Motivation

A number of different approaches to RCM problems are presented in the literature. It is no surprise that these, and the general description of RCM, allow a variety of different RCM problem definitions. Many have common elements and may differ only in very small details. For instance most RCM solutions are concerned with optimizing resource usage [48, 28]. Additionally certain RCM solutions optimize the number of changes that need to occur to optimize resource usage [28].

Due to the large scale of data centers, providers need to rely on solutions that do not require human intervention. Realistically, no human could react fast enough to make useful changes[2]. The changes are left to *autonomous agents*. How these agents make their decisions is something that varies considerably between solutions. Utility functions play a large part in some [48], as do alternative ranking measures [54]. Other approaches are based on combinatorics [28].

We observed in the literature that empirical methods are frequently used. Comparisons are based on time [28, 54], number of physical machines used [54], number of movements between configurations (known as migrations) [54] and almost any other metric imaginable. These metrics, often presented in the context of some custom data center or simulator available to the researchers, are frequently stated without reference to any common benchmark [28, 54, 48]. When benchmarks are used it is typically done ad hoc by implementing versions of the problem to use as a control.

[1] The author recognizes that there are many different names proposed in the literature instead of RCM. Examples are found in [28, 55, 48].

[2] No human could react with the speed necessary before the solution becomes outdated

Yasir *et al.* implement different RCM variants that serve as benchmarks for the empirical testing of their novel methods [54].

This leads to the question: how can we compare RCM problems and their solutions? A difficulty is the lack of a common terminology or framework to express computational RCM problems. In this thesis we present a framework that will bring consistency into the world of RCM problems, as it can serve as a platform to allow meaningful comparisons of different variants of computational RCM problems. The framework enables a formal analysis of the computational complexity of the RCM problem as a whole.

1.2 Contributions

Our thesis' contributions are as follows. First we propose a general framework encompassing computational RCM problems. The framework asks a series of binary questions that can be applied to RCM solutions offered in the literature. With this tool a researcher can analyse and identify solutions in the literature that are comparable to each other or to their own work. By using our framework it is possible to reverse engineer the computational problem definitions that the solutions were set out to solve. This approach gives a researcher a way to generate a problem definition from previous work, where a problem definition is lacking.

Second, the framework establishes common platform that allows for a discussion on which methods of analysis are best suited for RCM variants. We highlight the benefits of having a theoretical analysis along side an empirical approach. These theoretical tools include asymptotic analysis of algorithms, and classical complexity analysis of problem definitions. Asymptotic analysis perfectly complements empirical methods that are heavily used in the field, while classical complexity allows the researcher to get an understanding of the difficulties of the computational problem itself. We show **NP**-completeness for different RCM variants captured by our framework. The methods used in the proofs are simple and require only basic reductions. We also explore restrictions of computational RCM variants that can be solved in polynomial time.

Third, an intangible contribution is found in the simplicity in which the theoretical methods work in tandem with empirical methods in the RCM setting. Therefore, fruitful collaborations between the systems and theory communities will advance the area.

1.3 Thesis Overview

The remainder of the thesis is organized as follows. Chapter 2 gives a background into the terminology used in RCM as it pertains to the cloud computing community and introduces basic terminology for computational complexity. Chapter 3 is related work. Chapter 4 contains the main results of this thesis and is separated into two main sections: in the first, we propose our framework for RCM problems. In the second, we focus on a computational complexity analysis of problems that we obtain from our framework. Chapter 5 contains a summary along with a discussion on future work.

Chapter 2

Tools and Terminology

With hardware becoming less expensive, cloud computing is starting to expand at a fast rate, placing more importance on being able to efficiently manage resources. In this chapter general terminology for computational RCM problems is given. Two different perspectives exist in cloud computing. First, from a *users perspective*, cloud computing provides computational services (processing power, storage, etc.) for a fee that is based on resources consumed by the user. Much like other utilities, users are removed from the lower level details of implementation and are simply given the resources they require. Second, *providers* of cloud services, such as Google [4] and Amazon [1], manage a set of *physical machines* linked together to serve clients. The level of service is agreed upon individually with each user. Naturally each provider has a finite set of physical machines that make up a pool of resources. Throughout this thesis a set of physical machines and resources are terms that are used interchangeably. A challenge the provider faces is allocating enough resources to each user to satisfy their needs. These needs are formalized in terms of legal contracts called *Service Level Agreements* or simply SLAs.

To give a better understanding of SLAs, consider some user A. An SLA for A could demand that every request to A's web service has to be served in x seconds. The objective of the provider is to ensure that A's web service has the resources it needs at any given time, guaranteeing no more than a delay of x seconds. If the provider is successful then A is considered satisfied. The problem of reconfiguring resources in a way to satisfy each SLA is at the heart of resource consolidation management.

2.1 Basic Terminology for RCM Problems

Definition 2.1.1. A *data center* is a set, $P = \{p_1, p_2, \ldots, p_n\}$, of *physical machines*.

Unless otherwise stated all physical machines in the data center are identical. The set of all tasks that need to be satisfied by the service provider is given as set T. In this thesis we assume all tasks to be pairwise independent.

Definition 2.1.2. The set of all client requests, refered to as *tasks* is a set $T = \{t_1, t_2, \ldots, t_l\}$.

Each machine is represented as a vector of capacities and each task is represented by a vector of requirements. These vectors are special resource vectors.

Definition 2.1.3. A *resource vector* is a vector, $R = [r_1, r_2, \ldots, r_m]$, where each $r_i \in R$ represents a resource such as CPU or memory. We require any two resource vectors to be pairwise comparable. That is, for resource vectors R_1 and R_2, $R_1[i]$ is the same type of resource as $R_2[i]$.

Two resource vectors, $B_U(.)$ and $B_L(.)$, represent upper and lower bounds for physical machines. These bounds, set by the provider, do not represent the physical limits of the machines, but instead can act as warning signs that a physical machine needs attention.

Definition 2.1.4. Let $p \in P$. An *upper bound* resource vector is a vector $B_U(p) = [u_1, u_2, \ldots, u_m]$.

Definition 2.1.5. Let $p \in P$. A *lower bound* resource vector is a vector $B_L(p) = [l_1, l_2, \ldots, l_m]$.

Definition 2.1.6. Each machine $p \in P$ is assigned a resource vector of *capacities* $\phi_p = [c_1, c_2, \ldots, c_m]$, a resource vector of upper bounds $B_U(p)$ and a resource vector lower bounds $B_L(p)$.

The resource vector of capacities serves as the physical bounds of the machines. In this thesis, if not stated otherwise, we will make use of the physical capacities as bounds only.

The *tasks* to be executed on the cloud must be assigned by the provider to physical machines in the data center.

Definition 2.1.7. Each *task*, t, is assigned a resource vector of *requirements*, $\rho_t = [r_1, r_2, \ldots, r_m]$, to be satisfied.

We observe that ρ_t and ϕ_p are pairwise comparable for any t and p since both are resource vectors.

It is common practice to have tasks contained within *virtual machines* that run on physical machines [33, 29, 51]. Since multiple virtual machines can co-exist on a single physical machine, multiple user tasks can be co-located on the same physical machine.

Definition 2.1.8. Given a physical machine, $p \in P$, the set of tasks located on p is defined as $TASKS(p) = \{t_1, t_2, \ldots, t_k\}$. We call $TASKS(p)$ the assignment of p.

Definition 2.1.9. For $p \in P$, an assignment is *valid* if for each $i \leq m$:
$\sum_{t_j \in TASKS(p)} \rho_{t_j}[i] < U(p)[i]$. Otherwise we consider the assignment to be *invalid*.

Whenever a machine has greater demand than capacity, service can quickly deteriorate leaving SLAs unsatisfied. The bounds established in Definition 2.1.6 act as buffer zones. When an upper bound is reached it means the physical machine is close to having no more resources to allocate. When a lower bound has not been met it indicates that this machine could possibly be turned off to save energy.

Definition 2.1.10. A physical machine $p \in P$ is *overloaded* if for some, $i \leq m$, $\sum_{t \in TASKS(p)} \rho_t[i] > B_U(p)[i]$.

Definition 2.1.11. A physical machine $p \in P$ is *underloaded* if for some, $i \leq m$, $\sum_{t_j \in TASKS(p)} \rho_t[i] < B_U(p)[i]$.

In practice, over- and underloading of physical machines occurs because a task's resource requirements can fluctuate at any time, requiring more or less resources to satisfy the associated SLA. These fluctuations cause tasks to be migrated within the data center. The topology of the network of physical machines can be considered when deciding how and where to move a task through the data center [18, 15]. Considering the topology of the network allows the data center to be thought of as a graph [18].

Definition 2.1.12. A data center with physical machines P, can be represented as a graph $G = (P, E)$, where P represents the graph's *nodes* and E is the set of *edges*, which represent connections between physical machines $p_i,\ p_j \in P$[1]. Each $e \in E$ takes the form $e = (p_i, p_j)$.

[1]These connections can be wired connections, connections via wifi or any other type of connection. For the purpose of this thesis we distinguish only whether or not two physical machines can communicate, not how they accomplish it.

A path in the graph is a connected sequence of physical machines that starts at some node p_s and when followed, ends at some terminal node p_t.

Definition 2.1.13. A *path* in a graph $G = (P, E)$ between p_s and $p_t \in P$ is an ordered set, $PATH(p_s, p_t) = (p_s = p_{i_1}, p_{i_2}, \ldots, p_{i_l} = p_t)$ with $(p_{i_j}, p_{i_{j+1}}) \in E$ for $1 \leq j < l$. The length of the path, $|PATH(p_s, p_t)| = l - 1$.

For any two nodes p_i and p_j many different paths may exist. A noteworthy type of path is one of the shortest length.

Definition 2.1.14. A *shortest path* between two nodes p_s and p_t is a set $PATH(p_s, p_t)$ where $|PATH(p_s, p_t)|$ is smallest.

Because each link in the data center can be restricted to a finite amount of resources at any given moment, it is important to ensure that tasks move to a new physical machine in an efficient manner. These moves within the data center are refered to as *migrations* of data within the data center.

Definition 2.1.15. Let $G = (P, E)$ and p_i, $p_j \in P$. Further, given a configuration of task assignments moving the assignment of a task from a machine p_i to a machine p_j is called a *migration*. We permit migrations only if $(p_i, p_j) \in E$[2].

A migration typically occurs when a machine is either overloaded and must move one or more tasks to a new machine, or when a machine is being underloaded and must be cleared off to be powered down. The number of migrations that occur is exactly the length of the path the task took between its starting machine and the machine it ended on.

2.2 Basics From Computational Complexity

Long before modern computers were used to solve computational problems in the blink of an eye, the ancient Greeks were developing algorithms to solve computational problems. Efficiency must have been a huge concern since the algorithms were applied by hand. Even with the birth of modern computers complexity issues still arise, despite computation no longer being done by hand. The question of what is feasible

[2]Another way to define the weight of migrations is to count one for a migration of the tasks from one machine to another, independent of the path length. However, in a system where the network topology is not a complete graph it is meaningful to weigh a migration in terms of its length since a longer migration may be more expensive and therefore should be avoided.

when using a computer is a central question in computer science. The idea of feasible computation was a driving force behind the seminal work by Cook in 1971 [20][3], which explored and defined a set of computational decision problems that are likely not capable of being computed efficiently.

Throughout this thesis, only the decision versions of computational optimization problems are considered. A *decision problem* is a computational problem for which the answer is only yes or no. Whereas an *optimization problem* minimizes or maximizes some aspect of the problem and outputs the best possible answer. The reader interested in further discussion of decision and optimization problems is directed to [13].

The set of problems unlikely to be computed efficiently are know as **NP**-complete problems. These problems are a subset of the class *Nondeterministic Polynomial Time* or **NP** for short. The class **NP** is the set of all computational decision problems for which given candidate solutions can be verified in a polynomial amount of time. Computational decision problems that are **NP**-complete are thought of as the hardest problems to solve in the class **NP**.

Since the results of Cook and Levin, the most famous problem in Computer Science was born. Does $\mathbf{P} = \mathbf{NP}$? is the question posed by Cook [20], the answer to which will net its author one million dollars [2]. A reader looking for an in-depth discussion on this is directed to either [42] or [13] both of which do an excellent job detailing this question. Unless otherwise stated, in this thesis we assume that $\mathbf{P} \neq \mathbf{NP}$.

Resource Consolidation Management concerns itself with finding *efficient* algorithms. But what is an efficient algorithm? From a classical computational complexity point of view an efficient algorithm is one that requires no more than a polynomial amount of time to solve relative to its input size. The set of all problems that are computable in a polynomial amount of time is known as the complexity class **P**. We say that problems in **P** are *efficiently solvable*.

Definition 2.2.1. *Complexity class P [21]:* as the set of concrete decision problems that are polynomial-time solvable.

Unfortunately polynomial time solutions are not known for all decision problems. However, there exists a set of decision problems that can be verified efficiently. This set, for which **P** is a subset, is known as the class of *Nondeterministic Polynomial Time* problems or **NP**.

[3]This result was independently shown in Russia by L.A. Levin and published in 1973 [34].

Definition 2.2.2. *Complexity Class* **NP**[21]: The *complexity class* **NP** is the class of languages that can be verified by a polynomial-time algorithm.

Directly from Definition 2.2.2, Lemma 2.2.1 follows.

Lemma 2.2.1. *[13]* $\mathbf{P} \subseteq \mathbf{NP}$.

An important aspect of complexity theory and a fundamental element to the work of Cook [20] is the polynomial time reducibility. Given decision problems A and B, we are interested to know if we can transform A into B in polynomial time. Of interest to us are transformations that require only a polynomial amount of time.

Definition 2.2.3. [21]

Suppose that we have a procedure that transforms any instance α of A into some instance β of problem B with the following characteristics:

1. The transformation takes polynomial time.
2. The transformation is answer-preserving. That is, the answer for α is "yes" if and only if the answer for β is also "yes".

We call such a procedure a *polynomial-time reduction*, denoted $A \leq_P B$.

Definition 2.2.3 implies that if problem B is solvable in polynomial time, then so is Problem A.

Definition 2.2.4. A decision problem B is **NP**-*hard* if $A \leq_p B$ for every $A \in \mathbf{NP}$ [21]

We note that to show a reduction from every problem in **NP** to problem B, it is sufficient to only select one known **NP**-hard problem, A, and show $A \leq_p B$. This is sufficient because by Definition 2.2.4 we know that every problem in **NP** is polynomial time reducible to A. Therefore showing that A is polynomial time reducible to B shows that for any problem in **NP** there exists a polynomial time reduction to A and a second independent polynomial time reduction to B. Overall this two part reduction requires only polynomial time, showing that it suffices to select only one known **NP**-hard problem.

An important part of the work by Cook [20] is a definition of a set of problems that are members of **NP** but likely are not in **P**. This set of languages, known as **NP**-complete, is characterized as having verifiable solutions in polynomial time, but at the same time having no known way of solving the problems in polynomial time. Further each problem that is **NP**-complete is polynomial time reducible to any other problem that is **NP**-complete. This fact is central in the definition of **NP**-completeness.

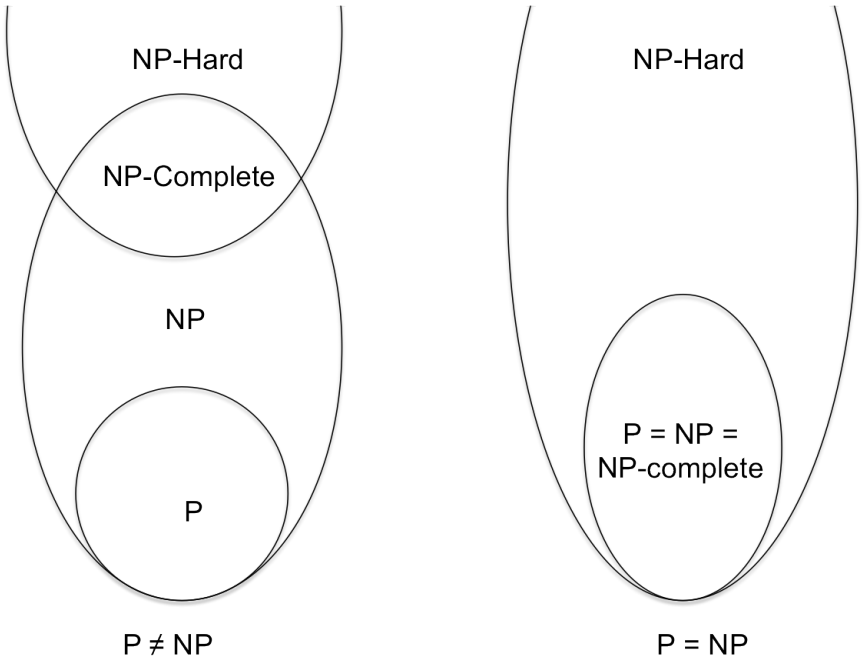

Figure 2.1: Relationship between **P**, **NP**, **NP**-Complete and **NP**-hard: adapted from [13]

.

Definition 2.2.5. A decision problem L is **NP**-*Complete* [21] if:

- L is a member of **NP**.
- L is **NP**-hard.

The relation between **P**, **NP**, **NP**-complete and **NP**-hard is summarized pictorially in Figure 4.2.1.

2.2.1 NP-completeness Proofs via Reduction and Certificate

Let us consider two problems A and B and let us assume that A is **NP**-complete. Further, the complexity of B is unknown. To show that B is **NP**-complete B must,

according to Definition 2.2.5, be both a member of **NP** and be **NP**-hard.

To show that B is a member of **NP** an algorithm needs to be produced that, given a candidate solution as input, verifies its correctness in polynomial time. A candidate solution is often also refered to as a *certificate*.

An alternate way of showing membership is via a polynomial time reduction. By showing $B \leq_p A$ we can also establish membership because any instance of B can be solved by transforming it into an instance of A, then solving A in **NP** time (since A is a member of **NP**). Therefore, since we did a polynomial time reduction, we answer B in polynomial time plus the time required to solve A, which overall is no worse than **NP** time showing that B itself is a member of **NP**. Note that this reduction does not show hardness!

As discussed above, to show that a problem is **NP**-complete we need to show that the problem is **NP**-hard. Here we give an alternate approach to the one outlined above and look at the relation of two problems when one is a special case of the other.

Given two computational problems P_1 and P_2 where P_1 is known to be **NP**-complete, assume that P_1 is a special case of P_2. Without loss of generality let us assume that P_2 has a polynomial time algorithm. Since P_1 is a special case of P_2 every instance of P_1 is also an instance of P_2. This means that we could use P_2 on an input of P_1 to solve P_1. However this would yield a polynomial time solution for an **NP**-complete problem and this cannot happen. Therefore we see that P_2 must be at least as hard as P_1.

2.3 Computational Problems Related to RCM

This section addresses a set of core computational problems used in this thesis. These problems all relate to RCM and are used in proofs throughout this thesis.

2.3.1 Bin Packing Problems

In this thesis several variants of packing problems are mentioned as examples, and to show **NP**-completeness of RCM problems.

Problem 2.3.1. BIN PACKING:
Input: A finite set U of items, a size $s(u) \in Z^+$ for each $u \in U$, a positive integer bin capacity B, and a positive integer K.

Question: Is there a partition of U into K disjoint sets $U_1, U_2, \ldots, U_K$ such that for each set U_i, i with $1 \leq i \leq K$, $\sum_{u \in U_i} s(u) < B$? In other words, is the sum of the sizes of the items in each U_i B or less?

BIN PACKING is known to be **NP**-complete [26]. Problem 2.3.1 considers packing of only items that have one constraint. In the context of RCM, considering only one constraint is problematic since service providers rarely consider only CPU, memory, etc. This variant replaces the single size with a vector of constraints.

Problem 2.3.2. VECTOR BIN PACKING:
Input: A finite set U of items, each $u \in U$ a vector $S(u) = [s_1(u), s_2(u), \ldots, s_d(u)]$ where each $s_j(u) \in Z^+$ for $1 \leq j \leq d$, and a vector if positive integers $B = [b_1, b_2, \ldots, b_d]$, and a positive integer K.
Question: Is there a partition of U into K disjoint sets $U_1, U_2, \ldots, U_K$ such that for each set U_i and for each dimension j, $1 \leq j \leq d$, $\sum_{u \in U_i} s_j(u) \leq b_j$?

Also, VECTOR BIN PACKING is **NP**-complete [50].

2.3.2 KNAPSACK Problems

Related to the BIN PACKING problem is the KNAPSACK problem. Given a single knapsack and a set of items each with a weight and value, the KNAPSACK problem asks what set of items (that fit into the knapsack) yields the best profit.

Problem 2.3.3. KNAPSACK
Input: Finite set U, for each $u \in U$ a size $s(u) \in Z^+$ and a value $v(u) \in Z^+$, and positive integers B and K.
Question: Is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \leq B$ and such that $\sum_{u \in U'} v(u) \geq K$?

KNAPSACK is **NP**-complete and can be found as one of Karp's original 21 problems [30]. Similar to BIN PACKING above, KNAPSACK is not directly applicable to RCM because we can neither model multiple constraints on the tasks, nor can we model more than one physical machine. We next focus on a variant that considers multiple knapsacks instead of just one.

Problem 2.3.4. MULTIPLE KNAPSACK
Input: A finite set U, for each $u \in U$ a size $s(u) \in Z^+$ and a value $v(u) \in Z^+$, and

positive integers B, K and N.
Question: Is there a subset $U' \subseteq U$ that partitions U' into N disjoint sets, $U'_1, U'_2, \ldots,$ U'_N such that for each $U'_i, \sum_{u \in U'} s(u) \leq B$ and $\sum_{U'_i \in U'} \sum_{u \in U'_i} v(u) \geq K$?

MULTIPLE KNAPSACK is know to be **NP**-complete since KNAPSACK is a special case of MULTIPLE KNAPSACK.

Chapter 3

Related Work

Although proposed in the 1960s by John McCarthy [27] and Douglas Parkhill [38], research on resource consolidation management started to gain traction again in the early 2000s. The work of Walsh *et al.* [48] served as a starting point for work in this thesis. Their study focused on utility functions as an approach to solve the allocation problem. The idea behind using utility functions is assigning numeric scores to indicate the level of preference a certain user has for a proposed bundle of resources. They then assign one machine to act as a global arbiter that decides the best global utility for the system and hands out resources based on the numeric preferences. The two-level approach of local resource managers that govern a subset of users and a global arbiter that deals with managers at the node level allows the arbiter to react at a higher level, only having to meet the requirements of the managers and not individual user tasks.

Multiple further approaches were based on the use of utility functions based on this approach [16, 19, 47, 45, 46, 22].

The work in [19] took and added a user interface to create a system that could be deployed with more user friendly features. The focus was to explore the use of utility functions in the cloud, but with the ability to test the environment in a more realistic setting than the initial work by Walsh *et al.* [48]. Several years later Das *et al.* extended this model in an attempt to make it even more user-friendly [22].

The work by Tesauro *et al.* found in [47, 46] focuses on methods to assist the prediction of future resource demands to preemptively configure the data center. The work in [47] focuses on Reinforced Learning (RL) strategies, while [46] uses a hybrid between RL and queuing theories.

Methods other than utility functions have been employed over the past decade.

In 2010, Yazir *et al.* proposed a solution that considered preference functions using a method other then utility functions [54]. This approach is based on the PROMETHEE method proposed by Mareschal in 1987 [36]. PROMETHEE is a rank generator that generates rankings within the data center that are acted on autonomously.

Hermier *et al.* propose a method using strictly packing problems called Entropy [28]. This work did not use preference generation methods, but instead uses packing problems to model the data center and allocation. Entropy uses a BIN PACKING approach to find an optimal resource allocation and then runs a KNAPSACK approach to determine a way to reconfigure the task assignments using the least number of migrations. Entropy also modeled the data center as a graph to check if the proposed set of migrations is feasible. The graph is used to model which groups of migrations can happen concurrently. Recent work has continued to use the idea of the data center as a graph, and focused on further graph techniques to solve computational RCM variants. [18, 15].

With the advancements of work in *Virtual Machines* [40], research quickly focused on virtual environments that allowed several user tasks to be co-located on the same physical machine. In theory, no two virtual machines interfere with each other, although in practice that is not always the case [49]. The use of virtual machines was first studied in 2006 in work by Almeida *et al.* [11]. Currently, models using virtual machines are the standard [11, 54, 28] due to the benefits of being able to assign more than one virtual machine to a single physical machine.

Validation is typically through an empirical study that sets up a testing environment that is either a simulation (such as in [54]) or a physical test bed (such as in [48]). Methods rarely can be directly compared due to variabilities in testing methods, such as hardware, number of machines, and networking capabilities.

The majority of the work in the field tends to only use an empirical approach for the analysis of solutions [54, 28, 32, 33, 37, 44, 48]; analysis methods from computational complexity can be found in [44, 43, 10, 52, 29]. While many more papers mention computational complexity results, typically they come in the form of a brief observation as apposed to using rigorous arguments. An example is the work of Walsh *et al.* [48].

Research that is notable was done by Speitkamp and Bichler [43]. They break the problem into several parts such as optimizing purchasing of servers and optimizing the server power that is currently available. Rigor is used to first define a base problem

of simple allocation of virtual machines to physical machines. Next they introduce several extra constraints to further the definition to consider migrations and constrains on the virtual machines themselves. Computational hardness is shown using BIN PACKING in their reduction. The work is then extended to study approximation algorithms and empirical tests are run to support their conclusions. They also note that although a problem is intractable, it should not be assumed that it is impossible to solve. Managers should know which instance sizes cause undesirable running times in their data center [43].

This suggests that it is important to have an understanding of the complexity, and to continue the theoretical analysis of approaches, even after a complexity class of the problem is addressed. Confusion around the definitions of **NP** and **NP**-hard appear to exist in the literature. For example, authors state that a problem is impossible for non-trivial input sizes [10, 39]. Although often infeasible, these problems are decidable and therefore computable.

The study in [10] claims that:

> "**NP**-hard problems are problems that are as hard as the hardest problems in class **NP**. **NP** is the set of problems such that, when given a solution, whether it is a true(ly optimal) solution or not can be verified in polynomial time, i.e., $\mathcal{O}(n^c)$ time, where n is the problem size (the number of items in the packing problem) and c is a constant. **Naturally, finding an optimal solution needs more time, for example, exponential time $\mathcal{O}(c^n)$, and it is impossible in practice for not a small n**"

.

Where Pisinger in [39] states:

> "The KCLP is **NP**-hard in the strong sense and also in practice very difficult to solve. Only small sized instances can be solved to optimality thus when dealing with real-life instances of large size heuristic approaches are applied."

The bold sentence in the first quote implies that finding optimal solutions to problems within **NP** is not possible in polynomial time. This is untrue since $\mathbf{P} \subset \mathbf{NP}$, and problems in **P** by definition are solvable efficiently. Both quotes imply that solving **NP**-hard problems optimality is only possible for the smallest input values. Although solving problems for larger input sizes may not be practical, further analysis is needed before one can assume that the use of heuristics is a must.

Further complexity work has been done using different variants of PARTITION [44]. Interestingly enough, in the conference version of the paper the overview of the hardness proof is stated incorrectly [44] and a technical report is cited for further details of the correct proof [17].

To the best of the author's knowledge no comprehensive, extendible framework that applies to methods both past a present has been proposed in the literature.

Chapter 4

A Framework for RCM Problems

In this chapter, we present the main results of the thesis, seperated into two main sections. The first introduces our framework for computational RCM problems. The framework is then applied to several existing RCM solutions found in the literature. The second section of this chapter focuses on the computational complexity analysis of the RCM problems obtained by our framework.

4.1 Framework

So far we have not defined RCM as a computational problem, as no single such definition could be identified in the literature. Instead, multiple problem definitions co-exist.

We define a framework that categorizes different variants of computational RCM problems and provides corresponding problem definitions. Each problem definition generated by the framework can be analysed and compared to other such problems. Further each proposed solution to RCM can be bucketed into its corresponding slot in the framework.

4.1.1 Framework Formulation

When studying the literature we see that:

1. No study has been done on the relationship of different variants of RCM.
2. There is no agreed upon physical or simulator testbed for testing RCM solutions.

In this thesis we only address the first item and leave the second for future work. Common goals and themes on RCM emerge from the literature, such as the overall goal of RCM is to satisfy the users of the data center. This leads to similar problem definitions that typically are close variants to each other.

To highlight similar, but different, problem definitions, two problems from the literature are selected. A basic version of RCM is to consider a set of items that need to be placed on a set of physical machines. This version does not take into account any aspect of the current data center configuration and in its reconfiguration does not consider migrations. Here it is assumed that all of the tasks fit on the physical machines of the data center, and the goal is to figure out an optimal placement to minimize the number of physical machines used. This problem is called Simplified Resource Consolidation Management and can be found in an article by Walsh *et al.* [48].

Problem 4.1.1. Simplified Resource Consolidation Management
Input: A set of tasks T, a set of physical machines P, for each $p \in P$, a resource vector $\phi(p)$, and positive integer N, $N \leq |P|$.
Question: Can each $t \in T$ be placed onto a physical machine $p \in P$ such that no p exceeds $\phi(p)$ and no more than N physical machines are used?

This version of RCM is equivalent to Vector Bin Packing. Details of this transformation can be found in Theorem 4.4.1

The second variant is taken from the work of Hermier *et al.* [28]. It contains a problem that appears very similar to Simplified Resource Consolidation Management, but it also considers migrations.

Problem 4.1.2. Simplified Resource Consolidation Management with Migrations:
Input: A set of tasks T, a set of physical machines P, for each $p \in P$, $\phi(p)$, and positive integers M and N with $N \leq |P|$.
Question: Can each $t \in T$ be placed onto a physical machine $p \in P$ such that no p exceeds its capacity $\phi(p)$, no more than N physical machines are used, and no more than M migrations occur?

We will show that this problem is equivalent to Cost Bin Packing (Theorem 4.4.3). Note that Problem 4.1.1 and Problem 4.1.2, although similar, do not share a common problem definition as Cost Bin Packing and Vector Bin Packing are different computational problems.

Our answer for classifying and providing concrete problem definitions is the below described framework that represents a family of problem definitions. The framework is built on a set of binary questions that, when answered, generate a binary string. This string identifies a problem definition for the solution and moreover allows two problems to be identified as identical or not.

4.1.2 Framework Construction

One of the fundamental components of the framework is the set of binary questions that is used to generate a problem definition. To facilitate the construction of the set of binary questions, an approached proposed for the construction of utility functions by Keeney and Raiffa [31] is used.

We consider work by Keeny and Raiffa from the book *Decisions With Multiple Objectives: Preferences and Value Tradeoffs* [31] that suggests dissecting the problem at hand, using the following four principles to generate a set of constraints:

- Completeness: The selection of attribute(s) covers all the important aspects of the initial problem.
- Operational: Attributes must be usable and meaningful.
- Non-redundancy and decomposability: Redundancy can be avoided via decomposing attributes into smaller, independent ones.
- Minimum Size: Keep the constraint set small without sacrificing the three conditions given above.

Applying these four principles we derived 7 constraints. The framework is presented below as Definition 4.1.1.

Following the conditions above ensures the framework is useful and reduces the unnecessary expansion of the list to include redundant or useless constraints. It also ensures that the constant set is practical and sufficiently cover the problem. From these principles we derived the framework presented below in Definition 4.1.1. A preliminary version was presented in the work in progress track at Cloud 2012 [35].

Definition 4.1.1. *Resource Consolidation Management:*
Input: To define the input the researcher must construct a set of conditions C by answering yes or no to each of the following questions. A more elaborate explanation of each question can be found below.

1. Are reconfigurations performed at discrete time steps?
2. Does the computation use uncertainty?
3. Is maximization of task completion considered?
4. Does it minimize resource usage in terms of physical machines?
5. Are migrations minimized?
6. Can tasks be added or removed?
7. Do any of the tasks have strict deadlines?

An answer to each of these questions with an ordered list of constraints defining the priority order, along with a set of physical machines P, a set of tasks T and a possibly empty initial configuration P_I defines our input.
Question: Can we satisfy the given tasks under condition set C while complying with the priority order?

When satisfying the set of constraints it is possible to arrive at a situation where two constraints need to be optimized that are not independent of each other. As an example consider the two constraints of minimizing resource usage and minimizing the number of migrations as is the case of Entropy [28]. Entropy finds the optimal placement of tasks within the system and then achieves this by moving the least number of machines. This gives preference to ensuring the system is always optimally packed, and afterwards minimizes the number of moves required to achieve this. The corresponding problem definition prioritizes the minimization of resources over the minimization of migrations. If, instead, we would prioritize minimization of migrations over the minimization of resources, then in general optimal solutions to the two problems may differ.

Finally we note that maximization of task completion is set to no in the trivial cases where we make the assumption that all tasks can be placed on the physical machines.

4.1.3 Criteria Selection

The constraints that form the set C are not complete. Our selection of 7 criteria serves as a starting point for which C continues to grow as the area of RCM develops.

The only requirement to add a new constraint is to follow the same selection criteria outlined. This extends the lifetime of the framework past current and past solutions.

Time Steps

Cloud systems are dynamic by nature and as a result providers must be prepared to react to changes within the data center. For instance, when used for web services, spikes in traffic can cause a task to require more than its current assignment of resources to be satisfied. Conversely, if there is a drop in traffic, then a task could be assigned more resources than required for it to be satisfied, leaving unused resources tied up. These changes can be monitored in the data center at certain intervals. These intervals, or *discrete time steps*, are when reconfigurations occur.

Uncertainty

Due to the dynamic nature of tasks, uncertainty can play a role in resource management when an operator wants to predict the status of the system. As discussed, previous research has focused on different techniques for forecasting demand [16, 45, 46, 47].

Task Satisfaction

Maximization of task satisfaction is something that arises when a data center does not have sufficient resources to satisfy all tasks. Some subset of tasks now has to remain unsatisfied for some period of time. Again we note that this constraint is set to no in the trivial cases where the assumption is made that all tasks can be placed on the physical machines

Resource Usage

Resource usage is at the core of the resource consolidation problem. Minimizing the amount of resources used allows providers to turn off unused resources. When physical machines are switched off a provider saves money because powering that physical machine is no longer required.

Migrations

When a task must be moved within the data center it is important to ensure it is able to get from its current physical machine to a new physical machine in the

shortest amount of time possible. A migration occurs when a task changes from one physical machine to an other(Definition 2.1.15. Minimization of migrations refers to limiting the number of such changes that occur within the data center during any given reconfiguration.

Removal or Addition of Tasks

The ability to add or remove tasks stems from the dynamic nature of clouds. Typically as a cloud progresses through time it will have new clients and clients that no longer require the cloud's services. It is important to be able to reflect this in the model for RCM.

Task Deadlines

Although task deadlines are not common in today's cloud work, they are common in the cluster and grid cousins of cloud computing. If a client requires that a task must be completed by a certain deadline, it has to be factored into RCM since these deadlines are parts of SLAs. We are aware that currently deadlines are not commonly found in SLAs, however, Yazir *et al.* [53] argue that SLAs will continue to evolve towards higher-level definitions allowing for more complicated requirements.

Summary

The process of criteria selection has been simplified in Figure 4.1. The figure shows how a general, high level problem was broken into multiple levels. Top down, each level brings us closer to the final set by breaking general criteria into smaller, more specific ones. We stopped at the point when each constraint on a level satisfied our four principles for constraint selection.

When using the framework to generate a problem definition from a solution in the literature it is important to answer the questions using only information available in the solution. No additional assumptions should be made.

In the case where the framework is used by a researcher to generate a problem definition before a solution is proposed we recognize certain criteria could be controversial. The constraint of resource minimization could to some researchers appears redundant, which would break one of the main principles for generation of the constraint list. But this constraint is not considered in every solution in the literature.

The work of Walsh *et al.* instead always allocates all of the resources available, deciding which sets of tasks would benefit with more resources[48]

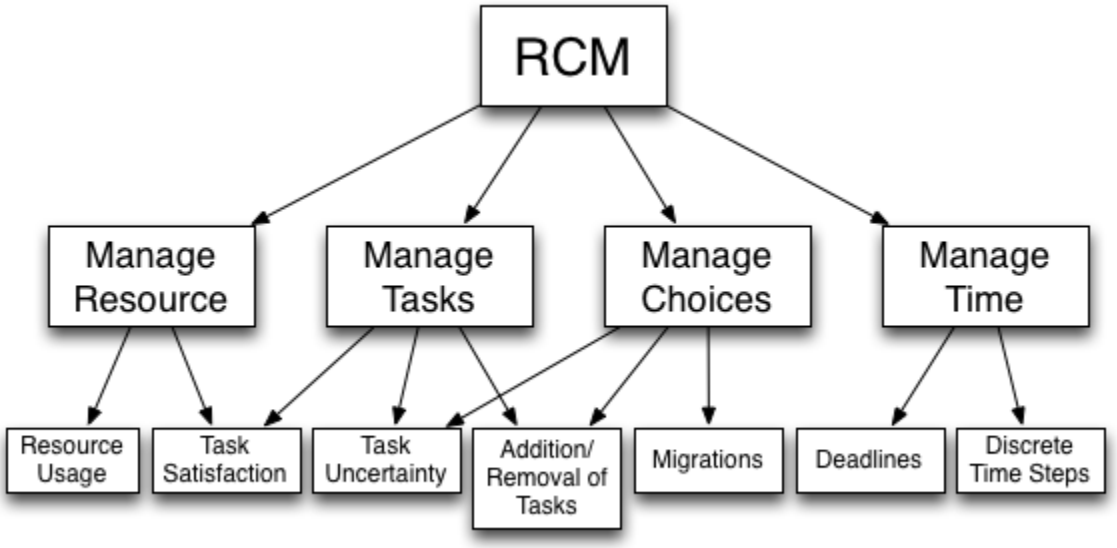

Figure 4.1: A visual representation of breaking tasks into non-redundant sets. The figure has been simplified to ensure legibility.

4.2 Framework Applications

With the framework established, we focus our attention on how to use it. The framework itself is flexible and can generate problem descriptions in two different ways. First, a researcher can use the framework to design the problem description using the set of binary questions. Second, the framework can be applied to a previously proposed solution by analyzing the solution, and then applying the framework's questions. This approach will produce a problem definition for the solution at hand.

A benefit of our framework is that it gives researchers a common terminology to compare any RCM solution, regardless of its origins. Currently a common technique used in analysis is to provide the results of an experiment carried out on a simulator or physical cloud [28, 19, 28, 54]. Empirical studies are often considered practical since they conduct testing on a real life system, but they have their weaknesses. For instance, two empirical test results are not guaranteed to be directly comparable. Further, scalability issues are also often hard to detect with only an empirical trial.

4.2.1 Practical Applications

The practical applications of our framework are twofold. First it establishes a common platform to compare different variants of RCM. Second, given a common platform it allows for easier analysis and comparison of RCM problems.

The ability to analyse computational problems and algorithms is a main competency found in a computer scientist's tool belt. Without concrete analysis the validity of a solution can be questioned. Further, how can we guarantee two variants are comparable?

We discuss and select three different analysis tools. A combination of computational complexity and empirical methods are suggested to provide a complete analysis of any problem definition generated by the framework. Methods such as running time analysis which analyse a solution by abstracting operations from specific hardware is applied to capture issues surrounding scalability. Empirical methods, which are directly tied to hardware, capture how a certain solution will behave on a real world system. Finally, while the two previous tools focus on specific solutions to a problem definition, formal complexity analysis of computational problems is suggested. This allows a researcher to get an understanding of the difficulty of the problem.

From the related work it is clear that empirical methods are heavily used and valued by the community, but used alone empirical methods might not catch issues of scalability. The simulations used for the trials might only generate certain instances of the problem, and as highlighted in [24] neglect instances where a solution might not preform well. Also typically simulators are not made public, so other researchers wanting to duplicate results might not be able to do so.

We suggest augmenting comparison methods through the use of theoretical tools. The addition of asymptotic analysis, such as Big-Oh, does not take away from other methods already applied. Instead the two different methods work together to give a provider a deep understanding of the problem solution.

Big-Oh analysis is the possible hidden, large constants in the running time. This weakness is seen in the seminal work by Robertson and Seymour on graph minor theorem [41] 'towers' of 2s are hidden when a Big-Oh analysis is done. Erik Demaine, in a keynote at IWPEC 2008, asked 'What's 2 to the 2 to the 2 to the 2 ... between friends?' also highlighting the concern.

In Addition to asymptotic analysis, it can also be meaningful to look at the average case running time, but it is only meaningful if it can be shown that the average running

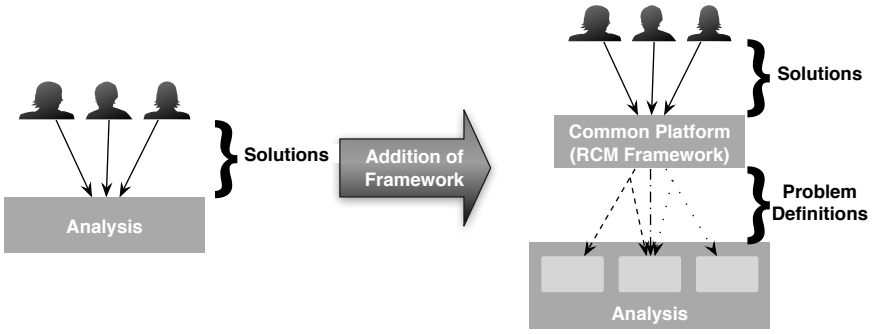

Figure 4.2: Transformation of RCM analysis

time happens with high probability. A famous example of this would be Quick Sort. With Quick Sort typically the average case is considered when Big-Oh analysis is done, which is $O(n \log n)$. The worst case of Quick Sort is n^2, but happens with very low probability [21].

Even though the two analysis methods above complement each other, we still have only analysed the problem from the perspective of a solution and neglected to look at the problem definition as a whole. Further analysis should be done on the computational problem as a whole, and for this we turn to classical computation complexity.

Using classical computational complexity tools we can show a computational problem to be a member of a certain complexity class or to be hard for a certain complexity class, such as outlined in Chapter 2. Looking at the classical complexity of a computational problem is important before a solution is proposed. It gives a general idea of the approach a researcher should take when developing a solution. Knowing which complexity class a problem is a member of should act as a guide to different techniques for a solution. To give an example, if the problem is known to be **NP**-complete, then trying to develop a polynomial time exact solution would be a waste. We emphasis here that these results should only serve as a guide, and that each problem should be treated differently.

4.3 Case Studies

We now present case studies on four RCM variants found in the literature to show how our framework is applied to existing solutions. The following solutions are considered for the case studies in this section: Entropy by Hermenier *et al.* [28] and the three models by Yazir *et al.* [53].

First we highlight how the framework is applied to each of the models by producing a problem definition for each solution. From there we show how comparisons can be done in a meaningful way. The results are summarized in Table 4.1.

The first variant is the solution proposed in [53] and uses a multi criteria decision method based on PROMETHEE [36]. The framework is demonstrated by going through each binary question and answering it based on the information available in the published papers.

1. *Discrete time steps:* Yes. Reconfigurations are computed and carried out in a reactive manner that is defined by the conditions in the data center.

2. *Does the computation consider uncertainty:* Yes. Three of the six criteria they use are defined as variability indicators. Some level of uncertainty is considered at each stage of the computation since these values are derived from past values and they determine a potential future level. Further, due to the use of fuzzy numbers by the family of PROMETHEE methods, uncertainties are inherent.

3. *Maximization of tasks completion:* No. The authors assume that they are able to satisfy all given tasks. When a task is added and insufficient resources exist to place it in the data center, the task is ignored and not added.

4. *Minimization of resource usage:* Yes. The authors aim to place the tasks such that the minimum amount of resources is used overall.

5. *Minimization of migrations:* No. In this paper a migration takes place when a particular machine is overloaded, but no attempt is made to minimize them.

6. *Task addition and removal:* Yes. The authors indicate at what point and how a virtual machine can be added or removed from the system.

7. *Tasks deadlines:* No. The authors do not consider a case where a given task has a deadline that must be met.

Table 4.1: Application of the framework to RCM solutions. Note that in the last row priorities are given where applicable.

Condition	PROMETHEE [53]	SDM [53]	FFD [53]	Entropy [28]
1) Discrete time steps for Reconfigurations	Yes	Yes	Yes	Yes
2) Uncertainty in computation	Yes	No	No	Yes
3) Maximization of Task Completion	No	No	No	No
4) Minimization of Resources Used	Yes	Yes	Yes	Yes
5) Migrations Minimized	No	No	No	Yes
6) Addition/Removal of Tasks	Yes	Yes	Yes	Yes
7) Strict Deadlines	No	No	No	No
Priorities	None	None	[4]	[4, 5]

Given the answers to the binary questions, this variant emits the problem definition as follows:

Problem 4.3.1. PROMETHEE RCM VARIANT
Input: A set R of resources, a set T of tasks with resource constraints, an initial configuration I, and a condition set $C = \{$Discrete time, Uncertainty, Minimization of resource usage, Addition/removal of tasks$\}$ representing "yes" answers to the questions as obtained above and no priority order.
Question: Can we satisfy the given tasks under conditions set C while complying with the priority order?

The process is now repeated to show how to derive a problem definition from the solution proposed by Hermenier *et al.* [28]. Entropy works in two stages by first packing the physical machines tightly, and then finding the least migrations required to realize the proposed configuration from the first step.

1. *Discrete time steps:* Yes. Entropy reconfigures the system at discrete time steps throughout the execution of the program.

2. *Does the computation consider uncertainty:* Yes. Statistics are used in the monitoring of the data center, which is fed in as input to each instance of bin packing.

3. *Maximization of tasks completion:* No. The authors assume that at any point they will be able to completely satisfy the tasks at hand. For instance, they use the assumption that the bin packing algorithm is always able to produce some packing that is then used in further steps of their algorithm.

4. *Minimization of resource usage:* Yes. The authors attempt to pack all of the virtual machines into as few physical machines as possible.

5. *Minimization of migrations:* Yes. The algorithm is broken into two parts. The second part focuses on minimizing the number of migrations to achieve the packing generated in the first part. This gives minimization preference to resources over migrations. In other words migrations are minimized given an already minimal virtual machine packing.

6. *Task addition and removal:* Yes. Since the authors repeat the packing at each configuration, a task can be added or removed each time the packing procedure

is started. It is worth noting that a procedure cannot be added during the second stage of the solution, since it relies on the packing from the first stage.

7. *Tasks deadlines:* No. There is no notion of task deadlines.

Since two constraints are required to be optimized here it is noted that minimization of resources is given the priority over migrations. The answers to these questions yields the following problem definition:

Problem 4.3.2. Entropy RCM Variant
Input: A set R of resources, a set T of tasks with resource constraints, initial state I and a condition set $C = \{$Discrete time, Uncertainty, Minimization of resource usage, Minimization of migrations, and addition/removal of tasks$\}$ with priority given to minimization of resources over migrations.
Question: Can we satisfy the given tasks under condition set C while complying with the priority order?

Once again we repeat the process for Simple Distributed Method (SDM). This method makes all decisions by randomly selecting machines and sending them to random destinations.

1. *Discrete time steps:* Yes. SDM selects the machines and moves them within the system at discrete time steps.

2. *Does the computation consider uncertainty:* No. No prediction models are used and although randomness is used, it is not used in an attempt to gain an advantage or a better solution.

3. *Maximization of tasks completion:* No. The authors assume that the system is capable of handling all the tasks in set T, and no attempt is made to select certain tasks to be left out and unsatisfied.

4. *Minimization of resource usage:* Yes. The end goal is to minimize the total number of machines used in the system.

5. *Minimization of migrations:* No. The system chooses where to send tasks randomly. Thus no attempt is made to minimize the number of migrations that occur in the system.

6. *Task addition and removal:* Yes. At any discrete time step there is the possibility to add or remove items in the system.

7. *Tasks deadlines:* No. There is no notion of task deadlines.

No priorities are needed for this instance. The answers to these questions yields the following problem definition:

Problem 4.3.3. SDM RCM Variant
Input: A set R of resources, a set T of tasks with resource constraints, initial state I and a condition set C = {Discrete time, Minimization of resource usage, and addition/removal of tasks} with no priorities.
Question: Can we satisfy the given tasks under conditions set C while complying with the priority order?

Finally we analyse the last method which is the First Fit Decreasing (FFD) method. This method takes a task that needs to be placed in the data center and iterates through each physical machine, placing it on the first machines that has room to accommodate it.

1. *Discrete time steps:* Yes. FFD selects the machines and moves them within the data center at discrete time steps.

2. *Does the computation consider uncertainty:* No. The algorithm simply takes a task to be placed and iterates through the list of physical machines until space is found.

3. *Maximization of tasks completion:* No. It is assumed that there will always be room for the task to be placed. If that is not the case the algorithm just rejects a solution.

4. *Minimization of resource usage:* Yes. The end goal is to minimize the total number of machines used in the data center.

5. *Minimization of migrations:* No. The algorithm places a task in the first opening, this does not take into account the number of migrations that might be required to accomplish this.

6. *Task addition and removal:* Yes. At any of the discrete time steps an item may be either added or removed to the task set T.

7. *Tasks deadlines:* No. There is no notion of task deadlines.

No priorities are needed here and the answers to these questions yields the following problem definition:

Problem 4.3.4. FFD RCM Variant
Input: A set R of resources, a set T of tasks with resource constraints, initial state I and a condition set C = {Discrete time, Minimization of resource usage, and addition/removal of tasks} with no priorities.
Question: Can we satisfy the given tasks under conditions set C while complying with the priority order?

4.3.1 Remarks

Inspecting Table 4.1, we observe that two of the solutions, FFD and SMD, have the same problem definition. Therefore, the two solutions can be directly compared. We notice that the other two problem definitions, PROMETHEE and Entropy, not only have different problem definitions, but neither match the definitions of SDM or FFD. Overall the four solutions generated three different problem definitions.

Focusing on PROMETHEE and Entropy, we discuss what to do when two problem definitions differ. First note that the set of yes answers for PROMETHEE is a subset of the questions answered for Entropy (the only difference is migrations are considered for Entropy). Can any meaningful comparison be done here?

We argue that meaningful comparison can be done when the two solutions are compared, with respect to the definition given by set of constraints for PROMETHEE. Generally, if two solutions exist that produce condition set C_1 and C_2 from the framework where $C_1 \subset C_2$ and where either 1) the priority sets match, or 2) if they are different and C_1 has fewer constraints as well as the constraints it shares with C_2 have matching rank. In the first case no extra care has to be taken when doing the comparison. In case 2 care must be taken since the priorities do not match and a bi-directional comparison might not be possible. More precisely, we can compare the solutions only with respect to C_1. We argue this comparison can be performed because both solutions sufficiently cover the constraints in C_1, so regardless of the approach the provider selects, the desired constraints will be covered by both.

We cannot have a comparison with respect to C_2 because both solutions do not cover the constraint set. Therefore after doing the analysis if the provider selects

the solution that generated C_1 they will not have a solution that covers all of the constraints.

The comparison rule can be thought of as comparing down, we can only compare solutions with respect to the largest subset of the constraint sets. What happens if we have two problem definitions where neither is a proper subset of the other? Unfortunately not much can be done in these cases.

4.4 Computational Complexity of RCM Variants

With a common platform established in Section 4.1 we turn our attention to the computational complexity of established computational problems. We show that members of our proposed framework are **NP**-complete and further explore restrictions that we can prove to be in the complexity class **P**. At the same time, we explore the relationships between BIN PACKING, KNAPSACK and computational RCM problems. In the literature BIN PACKING and KNAPSACK are frequently referenced as closely related problems to RCM. We also propose one new **NP**-complete problem, COST BIN PACKING (Problem 4.4.3), that to the best of the authors‘ knowledge does not appear in the literature.

4.4.1 Framework Member Analysis: NP-completeness

This section focuses on showing **NP**-hardness for selected problems obtained by our framework. All of the discussed problems are also **NP**-complete, since they are all members of **NP**. Verifying membership of **NP** works similar for each problem and is straight forward and therefore omitted. We refer the reader to Chapter 2 where we explain how membership proofs for **NP** can be obtained.

Minimization of Resources

At the heart of RCM is the minimization of resources used by the data center. When only considering resource minimization, the problem description that we obtain from our framework is:

Problem 4.4.1. RCM Minimizing Resource Usage:
Input: As per Definition 4.1.1 with $C = \{$Minimization of Resources$\}$, and a positive integer r.

Question: Can we satisfy the given tasks under conditions set C while complying with the priority order, using at most r resources?

Theorem 4.4.1. *Problem 4.4.1 is* **NP***-complete.*

Proof. We show **NP**-hardness via a reduction from Vector Bin Packing to Problem 4.4.1.

Given for Vector Bin Packing is a set of items, $U = \{u_1, u_1, \ldots, u_n\}$, for each $u \in U$ a vector $S(u) = [s_1(u), s_2(u), \ldots, s_e]$, a vector of positive integers $B = [b_1, b_2, \ldots, b_e]$, and an integer K. We ask if we can partition U into U_1, U_2, ..., U_K such that for each i and j: $\sum_{u \in U_i} s_j(u) \leq b_j$?

Given for the RCM variant defined as in Problem 4.4.1 is a set of tasks $T = \{t_1, t_2, \ldots, t_m\}$, for each $t \in T$ a resource vector of requirements $\rho(t) = [r_1(t), r_2(t), \ldots, r_d(t)]$, a finite set of physical machines P, for each $p \in P$ a resource vector of capacities $\phi(p) = [c_1, c_2, \ldots, c_d]$, and a positive integer R.

We ask if there exists a $P' \subseteq P$, $|P' \leq R$, such that we can place each $t \in T$ on exactly one $p \in P'$ with: for each $p \in P'$ and i, $\sum_{t \in TASKS(p)} r_i(t) \leq c_i$?

The transformation is as follows:

1. $m := n$
2. $d := e$
3. $T = U$
4. $R := K$
5. For each p, $\phi(p) := B$
6. For each $t \in T$ with $t = u$, $\rho(t) := S(u_i)$

Given this transformation we see that we have a yes-instance for Vector Bin Packing iff we have a yes instance for Problem 4.4.1. This follows from observing that each U_i corresponds to $TASKS(p_i)$, that is, this corresponds to partitioning the task set onto the physical machines. □

Resources and Migrations

The next addition considered is migrations in the data center. Migrations occur when a physical machine is overloaded and tasks must be moved between physical machines. Migration is formally defined in Definition 2.1.15.

Since we now minimize two constraints, we must give priority to one. Here we consider minimization of resources having priority over minimization of migrations.

Problem 4.4.2. RCM Minimizing Resource Usage and Migrations:
Input: As per Definition 4.1.1 with C ={Minimization of Resources, Minimization of Migrations} and positive integers R and M with priority order: 1) minimization of resources, 2) minimization of migrations.
Question: Can we satisfy the given tasks under conditions set C while complying with the priority order, using at most R physical machines and using at most M migrations?

Before analyzing the computational complexity of Problem 4.4.2, we introduce, and show **NP**-completeness of, an auxiliary problem: Cost Bin Packing. We use Cost Bin Packing in our reduction to show that Problem 4.4.2 is **NP**-hard. Cost Bin Packing is similar to Vector Bin Packing but starts with an initial packing of the bins. Its goal is to repackage the bins such that the weight of each bin is each dimension satisfies the constraints while not allowing more then M movements of items.

Problem 4.4.3. Cost Bin Packing:
Input: A finite set U of items, for each $u \in U$ a vector $S(u) = [s_1(u), s_2(u), \ldots, s_d(u)]$ where $s_j(u) \in Z^+$ for $1 \leq j \leq d$, a vector of positive integers $B = [b_1, b_2, \ldots, b_d]$, positive integers K and M and an initial partition of U into L disjoint sets $I_1, I_2, \ldots, I_L$.
Question: Does there exist a partition of U into K disjoint sets $U_1, U_2, \ldots, U_K$ such that for each i and j: $\sum_{u \in U_i} s_j(u) \leq b_j$ and $\sum_{k=0}^{L-K} |U_k - I_k| \leq M$. Here $U_k - I_k$ denotes the set difference.

Lemma 4.4.2. Cost Bin Packing *is* **NP**-*complete.*

Proof. We show that Vector Bin Packing reduces to Cost Bin Packing.

Given for Vector Bin Packing is a set of items, $U = \{u_1, u_2, \ldots, u_n\}$, for each $u \in U$ a vector $S(u) = [s_1(u), s_2(u), \ldots, s_e(u)]$, a vector of positive integers $B = [b_1, b_2, \ldots, b_e]$, and an integer K. We ask if we can partition U into U_1, U_2, ..., U_K such that for each i and j: $\sum_{u \in U_i} s_j(u) \leq b_j$.

Given for Cost Bin Packing is a set of items, $U' = \{u'_1, u'_2, \ldots, u'_{n'}\}$, for each $u' \in U'$ a vector $S'(u') = [s'_1(u'), s'_2(u'), \ldots, s'_d(u')]$ a vector of positive integers $B' = [b'_1, b'_2, \ldots, b'_d]$, positive integers K', M' and a partition of U' into L disjoint sets I_1,

$I_2, \ldots, I_L$. We ask if there exist a partition of U' into K' disjoint sets $U'_1, U'_2, \ldots, U'_{K'}$ such that for each i and j: $\sum_{u' \in U'_i} s'_j(u') \leq b'_j$ and $\sum_{k=0}^{L-K'} |U'_k - I_k| \leq M$

The transformation is as follows:

1. $U' := U$
2. $S'(u') := S(u)$ for each item $u' \in U'$, $u' = u$
3. $L = n$
4. $B' := B$
5. $K' := K$
6. $M := n$
7. for all j, $1 \leq j \leq n$, $I_j := \{u_j\}$

Given this transformation we see that we have a yes-instance for VECTOR BIN PACKING iff we have a yes instance for COST BIN PACKING, since with $M = n$, every item in $U' = U$ can be moved. □

Theorem 4.4.3. *Problem 4.4.2 is* **NP***-complete.*

Proof. We reduce from COST BIN PACKING. Given for COST BIN PACKING is a set of items, $U = \{u_1, u_1, \ldots, u_n\}$, for each $u \in U$ a vector $S(u) = [s_1(u), s_2(u), \ldots, s_e(u)]$, a vector of positive integers $B = [b_1, b_2, \ldots, b_e]$, positive integers K, M and a partition of U into L disjoint sets $I_1, I_2, \ldots, I_L$. We ask if there exists a partition of U into K disjoint sets $U_1, U_2, \ldots, U_K$ such that for each i and j: $\sum_{u \in U_i} s_j(u) \leq b_j$ and $\sum_{k=0}^{L-K} |U_k - I_k| \leq M$

Given for the RCM variant defined as in Problem 4.4.2 is a set of tasks $T = \{t_1, t_2, \ldots, t_m\}$, for each $t \in T$ a resource vector of requirements $\rho(t) = [r_1(t), r_2(t), \ldots, r_d(t)]$, a set of physical machines P, for each $p \in P$ a resource vector of capacities $\phi(p) = [c_1, c_2, \ldots, c_d]$ and an assignment of tasks $TASKS(p)$, and positive positive integers R,M'. We ask if we can place each $t \in T$ on exactly one $p \in P$ such that for each $p \in P$ and i: $\sum_{t \in TASKS(p)} r_i(t) \leq c_i(p)$, no more then R physical machines are used, no more then M' tasks have migrated? Note that the task assignment for the physical machines is now reconfigured.

The transformation is as follows:

1. $T = U$
2. For each p, $\phi(p) := B$
3. For each $t \in T$ with $t = u$, $\rho(t) := S(u_i)$
4. $R = K$
5. $|P| = L$
6. For each i, $1 \leq i \leq L$, $TASKS(p_i) := I_i$

Given this transformation we see that we have a yes-instance for Cost Bin Packing iff we have a yes instance for Problem 4.4.2.

□

Task Completion

In the previous problem definitions we assumed that we can always satisfy all tasks. In practice this is not always the case. But which tasks do not get satisfied? We revisit the problems above to now consider these problems for the case that we want to maximize the amount of tasks that can be satisfied. Since we consider several optimization constraints, we are required to set the priority.

Problems 4.4.1 and 4.4.2 are redefined to include the additional constraint.

Problem 4.4.4. RCM Maximizing Task Completion While Minimizing Resource Usage:
Input: As per Definition 4.1.1 with C ={Minimization of Resources, Maximize Task Completion}, and a priority ordering: 1) Maximization of task completion, 2) Minimization of Resources.
Question: Can we satisfy the given tasks under conditions set C while complying with the priority order, using at most r resources using only m migrations?

Problem 4.4.5. RCM Maximizing Task Completion While Minimizing Resource Usage and Migrations:
Input: As per Definition 4.1.1 with C ={Minimization of Resources, Minimization of Migrations, Maximize Task Completion} with a priority ordering: 1) task completion 2) minimization of resources, 3) minimization of migrations.
Question: What is the most resource efficient way to satisfy the given tasks under

conditions set C, using at most r resources, at most m migrations and satisfying at least t tasks?

We show that the two problems presented above have known **NP**-complete special cases.

Lemma 4.4.4. *Problem 4.4.1 is a special case of Problem 4.4.4.*

Proof. Consider an instance I_1 of Problem 4.4.1. We argue that it has a corresponding instance I_2 of Problem 4.4.4 where $t = |T|$. Observe that if I_1 is a yes-instance of Problem 4.4.1 then zero tasks are left unsatisfied. Therefore, I_1 is a yes instance iff I_2 is a yes instance. □

And by similar logic:

Lemma 4.4.5. *Problem 4.4.1 is a special case of Problem 4.4.5.*

Proof. Consider an instance I_1 of Problem 4.4.2. We argue that it has a corresponding instance I_2 of Problem 4.4.5 where $t = |T|$. Observe that if I_1 is a yes-instance of Problem 4.4.2 then zero tasks are left unsatisfied. Therefore, I_1 is a yes instance iff I_2 is a yes instance. □

The next Theorem follows directly from Lemma 4.4.4 and Lemma 4.4.5.

Theorem 4.4.6. *Problem 4.4.4 and Problem 4.4.5 are* **NP***-complete.*

4.5 Framework Member Analysis: Polynomial Time Results

As seen above all variants so far defined and analysed by our framework are **NP**-complete. This causes challenges because the ultimate goal is to find optimal solutions at speeds that allow an optimal reconfiguration of the system to occur. Restrictions, outside of the proposed constraints, can be applied to the RCM variants which allow for faster, optimal computation. The tradeoff comes at the cost of a general solutions, meaning their application might be limited. We also address restrictions that have no effect on the complexity of the solution in this section.

Bin Sizes

By the definition of BIN PACKING (Definition 2.3.1), VECTOR BIN PACKING (Definition 2.3.2) and COST BIN PACKING (Definition 4.4.3), the sizes of the bins are fixed. Do we gain any benefit from relaxing this restriction and allowing bin sizes of variable sizes?

We show that relaxing the bin sizes does not change the complexity of the problem. Observe that if all of the bins are of some constant size that is a special case of a different version of bin packing where bin size are allowed to vary. And since we know that BIN PACKING with constant sized bins is **NP**-complete, then it follows that a version with variable bin sizes must at least as hard.

Restricting Item Size

Imposing restrictions on size of items is the next variant we inspect. Unlike the restriction placed on bin size, restricting the size of item in a certain way allows solutions via a polynomial time algorithm. If we restrict each item to require some constant amount of resources, then a straight forward greedy approach can be used to pack the bins optimally, and in polynomial time.

Theorem 4.5.1. BIN PACKING *with constant sized items is solvable in a polynomial amount of time.*

Proof. The following greedy algorithm produces an optimal solution when each item to be packed is of constant size. First we formally state the BIN PACKING variant with restricted item sizes.

Problem 4.5.1. BIN PACKING WITH CONSTANT SIZE ITEMS
Input: A finite set U of items, a positive integer item size S, where $s(u) = S$ for each $u \in U$, a positive integer bin capacity B, and a positive integer K. Note this input machines that of Definition 2.3.1 except we set each item size to be some constant.
Question: Is there a partition of U into K disjoint sets $U_1, U_2, \ldots, U_K$ such that for each set U_i, i with $1 \leq i \leq K$, $\sum_{u \in U_i} s(u) < B$?

Since each disjoint set, or bin, is of the same size, and each item is also of some constant size we need only compute the number of items that can be placed into some U_i that respects $\sum_{u \in U_i} s(u) < B$. We call this c.
Algorithm:

1. Compute c.
2. Reject if $|U| > K \times c$.
3. If $|U| \leq K \times c$ construct the sets U_i as follows: U_1 is assigned the first c items of U $(u_1, u_2, \ldots, u_c)$. The second bin, U_2 is assigned the next c items, and so on. This continues until we run out of items to assign.

The algorithm is correct: each item is the same size it can be swapped with any other item in any other bin producing an equivalent solution. Further if we have more then $c \times K$ items we reject since it is not possible to partition U in K disjoint sets. □

An example where this happens is as follows: if a monitoring system is in place that can notice when the size of the user tasks are all identical, it can switch algorithms to exploit the above algorithm. This result extends to VECTOR BIN PACKING by noting that restricting each item to have a constant amount of resources over all dimensions allows for a similar greedy algorithm to be used.

Further we are able to derive similar results when we restrict item, or task, size in the case of KNAPSACK. A brief overview of the algorithm is as follows: 1) Sort items by value, 2) Continue to select the item with the highest remaining value until knapsack is full.

Multiple Dimension Restrictions

Finally we ask what if we divide each dimension into n units of some constant size, and one or more units are assigned to satisfy a task. Two interesting cases present themselves from this restriction. Consider a task that requires three units of resources to be computed. Are those three chunks required to live on the same physical machine to be satisfied? If yes, then by definition this is BIN PACKING. If no, then we are able to derive a polynomial time algorithm for this problem. The work of Balasubramanian *et al.* in [14] discusses this problem and shows that a greedy approach is optimal. Note this problem, taken directly from [14] is an optimization problem.

Problem 4.5.2. RESOURCE ALLOCATION IN DISTRIBUTED SYSTEMS [14]
Input: A set $V = \{1, 2, \ldots, M\}$ of M severs, and a set $R = \{1, 2, \ldots, L\}$ of L resources (e.g. CPU, memory, or bandwidth) that are to be assigned to these servers. The throughput of a server m, $1 \leq m \leq M$, denoted by T_m, is a function $T_m : 2^R \rightarrow$

$\mathcal{R}^+$.

Goal: Maximize the sum of the throughputs of the servers, $\sum_{m=1}^{M} T_m$, subject to the constraint that every such resource is assigned at most one server

It is not immediately clear how practical it would be to split tasks over multiple physical machines in a cloud setting, and exploring this is left as future work.

Chapter 5

Summary and Future Work

The field of cloud computing has flourished over the last decade. During this time many solutions for RCM have been proposed and implemented with the goal of satisfying clients better. In the last few years, cloud computing has been a topic in the spotlight resulting in an even larger research focus on the area. This thesis focused on computational problems regarding the allocation of resources within a data center that serves as (or part of) a cloud. This problem is formally know as Resource Consolidation Management. As clouds continue to grow, the data centers required to serve them do so as well. In this thesis we analysed current RCM methods from the literature with respect to their computational problem definitions, and proposed a framework to allow classification and comparison of RCM solutions.

5.1 Summary

An overview of the contributions of this thesis is highlighted below.

- A comprehensive, and open ended framework that captures different variants of computational RCM problems present in the current literature.
- A study of the classical computational complexity of computational RCM problem that we defined from our framework.
- A discussion on the types of analysis tools for RCM problems and suggestions on best practices.
- A look at restrictions of RCM problems that allow polynomial time algorithms.

- Examples of **NP**-completeness proofs for RCM problems that serve as a guide for the successful complexity analysis of many variants of known **NP**-complete problems.

Framework

In Chapter 4 a unified framework for RCM problems is presented. This novel framework defines a family of problem definitions. The framework is easily extendable. We anticipate that this will stimulate discussion within the community and allow ownership of the framework by the community itself.

Our framework allows the reverse engineering of RCM solutions from the literature. This is easily accomplished because the framework is based on a set of binary questions that are to be answered to establish the formal problem definition.

Computational Complexity

After obtaining concrete problem definitions using the framework we studied the computational complexity of selected problems. Showing the classical complexity of core RCM problem variants, we found many computational RCM problems analysed are **NP**-complete. Although not a surprising result, we did also explore, and propose, restrictions that allowed certain variants to be members of **P**. We accomplished all of the hardness results without the need of any non-standard proof techniques. One goal was to show the ease at which one can perform classical complexity analysis.

Analysis Tools

Classical complexity analysis is one of the tools we suggest for the analysis of RCM problems. Empirical methods are already heavily used in the field, but when used alone, presented some weakness. In order to give a well rounded analysis at the algorithm level we suggested the addition of asymptotic analysis, such as Big-Oh. When used in combination with empirical methods a researcher is able to get a more complete picture, including the scalability of their approaches.

Classical complexity analysis of the computational problems can serve as a guide that recommends algorithmic approaches the researcher should aim for.

5.2 Future Work

We start the discussion of future work with an overview of the next set of constraints that will be added to the framework.

5.2.1 Framework Extensions

Network Topology

Network topology is important to consider because in practice, a data center will likely not have each physical machine connected directly to every other machine. Therefore we must build some idea of paths through the network if we hope to perform migrations of tasks within the data center. In terms of the complexity of these variants, and the ability to model networks with graphs, solutions and **NP**-completeness proofs will start to borrow more from the area of graph theory.

Task and Data Location

A related question to network topology is: is all the data a user needs to run their task located on the same physical machines as the task itself. Currently we consider every task to be contained within a virtual machine that is self sufficient, but this might not always be the case. New challenges include increased communication between physical machines both host part of a users task. Further reconfigurations become more challenging as tasks could be housed on multiple physical machines within the data center. New metrics, such as some idea of closeness between physical machines, will need to be proposed to address these new concerns.

5.2.2 Analysis Extensions

Framework Analysis: Task Deadlines

Task deadlines are addressed in the framework section of this paper, but no computational complexity results for these problems are shown in this thesis. Consider RCM variants where tasks can have deadlines that indicate the latest possible time to be completed. Here we formally define, in terms of our framework, a variant with task deadlines and minimization of resource usage.

Problem 5.2.1. RCM Variant Considering Task Deadlines and Minimization of Resources

Input: As outlined in Definition 4.1.1 with C ={Minimization of Resources, Maximize Task Completion, Time Steps}, a set of tasks $T = \{t_1, t_2, \ldots, t_m\}$, for each $t \in T$ a resource vector of requirements $\rho(t) = [r_1(t), r_2(t), \ldots, r_d(t)]$ and a positive integer deadline $D(t)$, a set of physical machines P, for each $p \in P$ a resource vector of capacities $\phi = [c_1, c_2, \ldots, c_d]$, and positive integer R. Further, we assume that each task can be completed in a unit time step.
Question: Can we satisfy each task t before deadline $d(t)$ using at most R physical machines and such that at any time step: $\sum_{t \in TASKS(p)} r_i(t) \leq c_i$?

We postulate that this variant is **NP**-complete.

Approximation Algorithms

This thesis focused only on classical complexity models. This led to results that show the intractability of problems leading to the conclusion that RCM, unless heavily restricted, is not a classically easy problem to solve. What was not addressed was, for example, a formal analysis of approximation algorithms and parameterized complexity analysis. Although approximation algorithms do not yield optimal results, it is possible to have polynomial time algorithms that get within a very small factor of optimal exact results. These results could be very meaningful since a solution that is 'good enough' is often sufficient if it can be computed very fast. We wish to extend the framework and give a formal analysis of how well approximation algorithms could preform for members of the framework.

Classical complexity is not the only computational complexity analysis that is useful. Fixed-parameter tractable algorithms, at a high level, look at a computational problem through the lens of a parameter for that problem. The added benefit is that if a problem is *fixed-parameter tractable* we get algorithms that are in practice very fast, albeit not polynomial time [23].

Benchmarking

Benchmarking solutions with a known, standard benchmark is an important tool when using empirical trials. We would like to look into a benchmark that can be part of the analysis component of our framework. With a common problem definition given by our framework, finding appropriate benchmarks is now possible.

Bibliography

[1] Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2. Accessed: August 21, 2012.

[2] Clay mathematics institute: P vs np problem. http://www.claymath.org/millennium/P_vs_NP/. Accessed: September 4, 2012.

[3] Folding@home. http://folding.stanford.edu/English/HomePage. Accessed: September 3, 2012.

[4] Google app engine. http://code.google.com/appengine. Accessed: September 4, 2012.

[5] Google cloud platform. http://cloud.google.com/. Accessed: September 3, 2012.

[6] icloud. https://www.icloud.com/. Accessed: September 3, 2012.

[7] Microsoft cloud services. http://www.microsoft.com/oem/en/products/other/Pages/cloud_services.aspx#fbid=qWydYODP7b6. Accessed: September 4, 2012.

[8] Seti@home. http://setiathome.ssl.berkeley.edu/. Accessed: September 3, 2012.

[9] Wikipedia. http://www.wikipedia.org/. Accessed: September 3, 2012.

[10] Y. Ajiro and A. Tanaka. Improving packing algorithms for server consolidation. In *Proceedings of the International Conference for the Computer Measurement Group (CMG)*, pages 399–406, 2007.

[11] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian. Resource Management in the Autonomic Service-Oriented Architecture. In *ICAC '06: IEEE International Conference on Autonomic Computing*, pages 84–92, 2006.

[12] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[13] S. Arora and B. Barak. *Computational complexity: a modern approach*, volume 1. Cambridge University Press, 2009.

[14] Sowmya Balasubramanian, Ron Desmarais, Hausi A. Müller, Ulrike Stege, and S. Venkatesh. Characterizing problems for realizing policies in self-adaptive and self-managing systems. In *Proceeding of the 6th international symposium on Software engineering for adaptive and self-managing systems*, SEAMS '11, pages 70–79, New York, NY, USA, 2011. ACM.

[15] N. Bansal, K.W. Lee, V. Nagarajan, and M. Zafer. Minimum congestion mapping in a cloud. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 267–276. ACM, 2011.

[16] Mohamed N. Bennani and Daniel A. Menasce. Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 229–240, 2005.

[17] Henri Casanova, David Schanzenbach, Mark Stillwell, and Frédéric Vivien. Resource Allocation using Virtual Clusters. Rapport de recherche RR-6692, INRIA, 2008.

[18] WK Chan, L. Mei, and Z. Zhang. Modeling and testing of cloud applications. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 111–118. IEEE, 2009.

[19] D.M. Chess, A. Segal, I. Whalley, and S.R. White. Unity: Experiences with a Prototype Autonomic Computing System. In *2004. Proceedings. International Conference on Autonomic Computing*, pages 140–147, 2004.

[20] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[21] T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[22] R. Das, J.O. Kephart, I.N. Whalley, and P. Vytas. Towards Commercialization of Utility-based Resource Allocation. In *ICAC '06: IEEE International Conference on Autonomic Computing*, pages 287–290, 2006.

[23] Rod G. Downey and R. Fellows. *Parameterized complexity*. Monographs in computer science. Springer, 1999.

[24] Sally Floyd and Vern Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001.

[25] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *GCE: Grid Computing Environments Workshop*, pages 1–10, 2008.

[26] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[27] S.L. Garfinkel and H. Abelson. *Architects of the Information Society: 35 Years of the Laboratory for Computer Science at MIT*. Mit Press, 1999.

[28] Fabien Hermenier, Xavier Lorca, Jean M. Menaud, Gilles Muller, and Julia Lawall. Entropy: A Consolidation Manager for Clusters. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 41–50, 2009.

[29] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible. Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 72–79. IEEE, 2011.

[30] R.M. Karp. Reducibility among combinatorial problems. *50 Years of Integer Programming 1958-2008*, pages 219–241, 2010.

[31] R.L. Keeney and H. Raïffa. *Decisions with multiple objectives: preferences and value tradeoffs*. Wiley series in probability and mathematical statistics. Applied probability and statistics. Cambridge University Press, 1993.

[32] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application Performance Management in Virtualized Server Environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 373–381, 2006.

[33] A. Kochut. On Impact of Dynamic Virtual Machine Reallocation on Data Center Efficiency. In *Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on*, pages 1 –8, 2008.

[34] L.A. Levin. Universal sequential search problems, 1973.

[35] S. Lonergan, Y.O. Yazir, and U. Stege. A framework for classification of resource consolidation management problems. In *IEEE 5th International Conference on Cloud Computing. Work in progress track*, pages 972–973. IEEE, 2012.

[36] B. Mareschal. Aide a la Decision Multicritere: Developpements Recents des Methodes PROMETHEE. *Cahiers du Centre d'Etudes en Recherche Operationelle*, pages 175–241, 1987.

[37] Daniel A. Menasce and Mohamed N. Bennani. Autonomic Virtualized Environments. In *ICAS: Proceedings of the International Conference on Autonomic and Autonomous Systems*, pages 28–37. IEEE Computer Society, 2006.

[38] D.F. Parkhill. *The challenge of the computer utility.* Number p. 246 in The Challenge of the Computer Utility. Addison-Wesley Pub. Co., 1966.

[39] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382–392, 2002.

[40] Gerald J. Popek and Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *ACM Communications*, 17(7):412–421, 1974.

[41] N. Robertson and P.D. Seymour. Graph minors—a survey. *Surveys in combinatorics*, 103:153–171, 1985.

[42] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2006.

[43] B. Speitkamp and M. Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *Services Computing, IEEE Transactions on*, 3(4):266 –278, oct.-dec. 2010.

[44] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova. Resource allocation using virtual clusters. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 260–267. IEEE, 2009.

[45] G. Tesauro, R. Das, W.E. Walsh, and J.O. Kephart. Utility-Function-Driven Resource Allocation in Autonomic Systems. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 342–343, 2005.

[46] G. Tesauro, N.K. Jong, R. Das, and M.N. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In *ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, pages 65–73. IEEE Computer Society, 2006.

[47] Gerald Tesauro. Online Resource Allocation Using Decompositional Reinforcement Learning. In *AAAI'05: Proceedings of the 20th National Conference on Artificial Intelligence*, pages 886–891. AAAI Press, 2005.

[48] William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi Das. Utility Functions in Autonomic Systems. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, pages 70–77. IEEE Computer Society, 2004.

[49] Jinpeng Wei, Xiaolan Zhang, Glenn Ammons, Vasanth Bala, and Peng Ning. Managing security of virtual machine images in a cloud environment. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 91–96, New York, NY, USA, 2009. ACM.

[50] Gerhard J. Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Information Processing Letters*, 64(6):293 – 297, 1997.

[51] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-Box and Gray-Box Strategies for Virtual Machine Migration. In *4th USENIX Symposium on Networked Systems Design and Implementation*, pages 229–242, 2007.

[52] R. Yanggratoke, F. Wuhib, and R. Stadler. Gossip-based resource allocation for green computing in large clouds (long version). *KTH Royal Institute of Technology, https://eeweb01. ee. kth. se/upload/publications/reports/2011/TRITA-EE*, 36, 2011.

[53] Yağız Onat Yazır, Chris Matthews, Roozbeh Farahbod, Adel Guitouni, Stephen Neville, Sudhakar Ganti, and Yvonne Coady. Dynamic and Autonomous Resource Management in Computing Clouds through Distributed Multi Criteria Decision Making. Technical Report DCS-334-IR, University of Victoria, Department of Computer Science.

[54] Yağız Onat Yazır, Chris Matthews, Roozbeh Farahbod, Stephen Neville, Adel Guitouni, Sudhakar Ganti, and Yvonne Coady. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In *IEEE CLOUD: 2010 IEEE 3rd International Conference on Cloud Computing*, 2010.

[55] Y.O. Yazır. *Multiple Criteria Decision Analysis in Autonomous Computing: A Study on Independent and Coordinated Self-Management*. PhD thesis, University of Victoria, 2011.