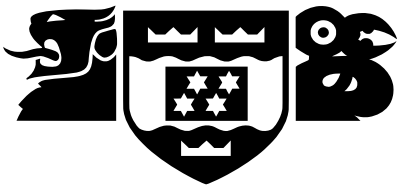
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui



School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Container-based Server Consolidation in Cloud with Evolutionary Computation Approaches

Boxiong Tan

Supervisors: Dr.Hui Ma, Dr.Yi Mei

Submitted in partial fulfilment of the requirements for
PhD.

Abstract

A short description of the project goes here.

Contents

Figures

Chapter 1

Introduction

1.1 Problem Statement

Cloud providers spend huge amount of money on data centers because a typical data center consumes as much energy as 25,000 households [28]. Thus, reducing the energy consumption becomes the major concern of Cloud providers. In addition, data centers and computation powers are the pillars of modern Cloud computing industry, software industry and etc. Reducing the cost of data centers will lead to a reduction of cost of softwares which consequently be beneficial to most people who access the Internet on a daily basis. Among several components that consume energy such as cooling system, physical machines (PMs) (e.g servers), and network devices, PMs accounts for 40% and have a huge improvement space, since they always run in a low utilization as observed by Barroso and Shen [7, 93] - from 10% to 50% of required resources on average.

In order to improve the resource utilization as well as reduce energy consumption, one way is cloud resource management [71]. Cloud resource management allocates resources to cloud users' applications, handles the workload fluctuations, and attempts to use as fewer number of PMs as possible to save energy. Before virtualization technology [107] was widely used, traditional data center assigns a PM for each application. Later on, PMs' utilization are largely improved by utilizing VMs to deploy applications. However, as a new trend of Service Oriented Architecture [101] appears in software industry; It separates a large centralized application into multiple distributed components (e.g web services). As most of web services only require a small amount of resources, using a VM for a web service would cause resource wastage inside the VM, consequently decreasing the utilization of PMs. Therefore, a new virtualization technology: containers [41, 98] and a new service model: Container as a Service (CaaS) [84] have been proposed to provide a finer granularity level of resource management. Container is an operating system (OS) level of virtualization which means multiple containers run on the same VM and share the OS. CaaS uses containers as the fundamental resource management units. That is, instead of deploying applications on VMs, applications are now deployed in containers which are running on top of VMs. This model lets us minimize the waste of resources as well as energy consumption [36]. Although this new technology gives an opportunity for better resource utilization, it also poses challenges to server consolidation.

Server consolidation [108] is the core strategy in improving the utilization as well as decreasing the overall energy consumption throughout the Cloud resource management processes as shown in Figure 1.1. Server consumption is essentially an optimization task where it adjusts applications' locations in PMs so that a minimum number of PMs is used. For a certain number of applications, the fewer number of PMs is used, the less energy is consumed.

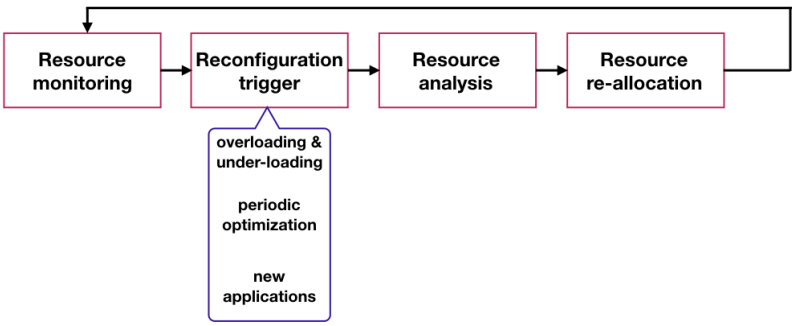


Figure 1.1: A workflow of resource management [75]

Three management processes: application initial placement [53], periodic optimization [75], overloading and under-loading adjustments [75] have distinct characteristics, hence, the server consolidation techniques applied on them can be roughly classified into two categories: static [119] and dynamic [9]. Static approaches use estimated resource utilization of applications (e.g average resource utilization of historical data) as the input to combine applications to a minimum number of PMs [75]. Static consolidation normally involves large number of applications and PMs, therefore, the optimization is quite time-consuming and often conducted in an off-line and a proactive (Cloud providers initiate) fashion. Periodic optimization belongs to this category. It takes a number of existing applications and re-allocate them into a number of PMs.

Static server consolidation are detailed explained in terms of VM-based and container-based Cloud. In a traditional VM-based Cloud, server consolidation can be described as, given a number of PMs which can be represented as resources (e.g CPU cores and RAM etc); a number of requests for fixed configurations of VMs (assume applications have been deployed in VMs), the configuration can also be represented as aforementioned resources; The objective is to allocate these requested VMs into a minimum number of PMs. The decision variable is the location of each requested VM. The basic constraint is that the aggregative resources of hosted VMs cannot exceed the PM's resource capacity.

In contrast, in a container-based Cloud, instead of allocating requested VMs in PMs, a set of containers (assume applications have been deployed into containers) represented as resources is first allocated to a number of fixed type VMs, we define it as the lower level of allocation. Then, these VMs are allocated to PMs, this is the upper level of allocation. The decision variables are the allocation of containers to VMs (lower level) and the allocation of VMs to PMs (upper level). For the lower level of allocation, the objective is to maximize the utilization of resources (e.g a balanced utilization among several resources), while the upper level objective is to minimize the number of PMs. The constraint is the demand of containers not to exceed the VM's capacity and the demand of hosted VMs not to exceed the PM's resource capacity. An additional constraint is that each container has its OS requirement which makes them cannot be simply packed into homogeneous VMs.

Dynamic approaches take one application each time, allocate it into one of the PMs [119]. As a dynamic problem often requires fast reaction such as overloading and underloading problems [10], the dynamic consolidation is conducted in an online fashion. In VM-based Cloud, the dynamic problem can be described as given a set of VMs and PMs, allocating these VMs to PMs iteratively so that the final solution reaches a near-optimal state. The key point is that the migration of VM can be conducted immediately after its position is decided. Hence, essentially, the problem can be simplified to allocating one VM to PMs. The difficulty is that the iterative process is hard to reach a global optimal because it normally

follows a greedy-based approach such as First Fit. In container-based Cloud, the problem is even complicated, if no VM can accommodate a container, a new VM must be created, which incurs a second level of deployment.

Application initial placement can be seen as either static: allocate a batch of new applications, or dynamic: allocate a new application each time. In this thesis, we will consider it in both ways.

Traditional VM-based server consolidation are modeled as bin-packing problems [69]. This is because VMs and PMs are naturally modeled as items and bins. Furthermore, server consolidation and bin-packing have the same optimization objective: minimize the number of bins/PMs. The complexity of bin-packing problem is NP-hard which means it is extreme time-consuming to find its optimal solution when the number of decision variables is large. Container-based server consolidation can be categorized as a bilevel optimization problem [25]. Bilevel optimization problems contain two level of optimization task: the outer optimization task is referred as the upper level task and the inner task is referred as the lower level task. The hierarchical optimization are typically non-convex and strongly NP-hard [110]. In our problem, two levels of optimization are both bin packing problems and they are cooperating.

Currently, most research focus on VM-based server consolidation and these methods cannot be directly applied on container-based consolidation because of the different structure. Only few research focus on container-based server consolidation problem. One of research is from Piraghaj and et al [84]. They first propose a VM-resizing technique that defines the types of VM based on analyzing the historical data from Google cluster data. Then, they propose a two-step allocation: first allocate containers to VMs and then allocate VMs to PMs. They propose simple heuristics on each level of allocation, thus, did not consider the interaction between two levels (see detailed discussion in Section 2.2.1). In addition, they propose a dynamic consolidation [83] using a series simple heuristics such as Random Host Selection Algorithm or First Fit Host Selection. Their resource allocation system completely relies on dynamic consolidation without using static methods. Although their system can execute allocation fast, the energy efficiency cannot be guaranteed. Another research [70] is the earliest study which realizes when deploying applications or containers, the VM size selection and VM placement should be considered together because they are interact with each other. They apply a fixed VM placement algorithm and considering a series of VM selection algorithms such as simple selection [45], Multiple selection, Maxsize, Consolidation-friendly. The results also proves that the interaction between two placement cannot be ignore.

The overall goal of this thesis is to develop new container-based server consolidation approaches to solve three problems: joint allocation of containers and VMs, periodic global optimization and dynamic consolidation.

1.2 Motivation

In this thesis, we aim at providing a series of approaches to continuously optimize the joint allocation of VMs and containers. A continuous optimization procedure mainly involves with three types of server consolidation: application initial placement, periodic consolidation, and dynamic consolidation. Different stages have distinct goals, therefore, they are considered as separated research questions. In addition, a scalability problem of static optimization is considered as an optional objective.

1. Joint allocation of containers and VMs (application initial placement),
   In this research, we take the joint allocation as a static problem which is fundamental

for server consolidation problem. At this stage, a set of containers is allocated to a set of VMs and these VMs are allocated to a set of PMs. This task is challenging because the problem is a bilevel optimization where each level is a bin packing problem. Exhaustive search of entire solution space is practically impossible, for the number of possible permutation of solution is huge. Current approaches [49, 83] use simple heuristics such as First Fit to solve the problem. These greedy-based heuristics do not consider the complex structure of the problem, therefore, often reach a local optimal solution.

2. Periodic consolidation,
   A periodic consolidation is conducted to improve the global energy efficiency in a periodical fashion. Data center constantly receives new allocations, releasing of old resources. These changing degrades the compact structure of a data center. Therefore, the data center needs a global optimization to improve the overall energy efficiency.

   The challenges are three folds, firstly, similar with initialization problem, the problem has two level of allocations and they interact with each other. Secondly, like VM-based consolidation, Container-based consolidation is considered as a multi-objective problem with minimization of migration cost as well as keeping a good energy efficiency. In bilevel optimization, multi-objective can be defined in either or both level, therefore, it further increases the complexity. Thirdly, consolidation is a time-dependent process which means the previous solution affects the current decision. Previous VM-based research only consider each consolidation as an independent process. As a consequence, although in one consolidation, the migration is minimized, It may lead to more migrations in the future consolidation. We will consider the robustness of consolidation and propose a novel time-aware server consolidation which takes the previous immediate consolidation and the future consolidation into consideration.

3. Dynamic consolidation,
   It takes one container and allocates it to VMs. Since the size of container can be dynamically adjusted, when the an application is under-provision or over-provision, the original container is halted, resized and re-allocated. Hence, there is a need to allocate this new container in real time.

   To solve a dynamic consolidation, heuristics and dispatching rules are often used [8, 44, 91, 94]. In this scenario, a dispatching rule is considered as a function that determines the priorities of VMs that a container can be placed. However, dynamic placement is much complex than bin-packing problem [69]. Because of its dynamic nature, human designed heuristics are ill-equipped in approximating solutions when the environment has changed [100].

   Hyper-heuristic methods, sepcifically, Genetic Programming (GP) technique [4] can learn from the best previous allocation and automatically evolves dispatching rules to solve this problem. GP has been applied in generating dispatching rules for bin-packing problem [18,100] and other scheduling problems [78]. The results have shown promising results.

   There are mainly two challenges, first, it is difficult to identify the related factors that construct the heuristic. Factors or features are the building blocks of heuristics. It is a difficult task because the relationship between a good heuristic and features are not obvious. Second, representations provide different patterns to construct dispatching rules. It is also unclear what representation is the most suitable for the consolidation problem.

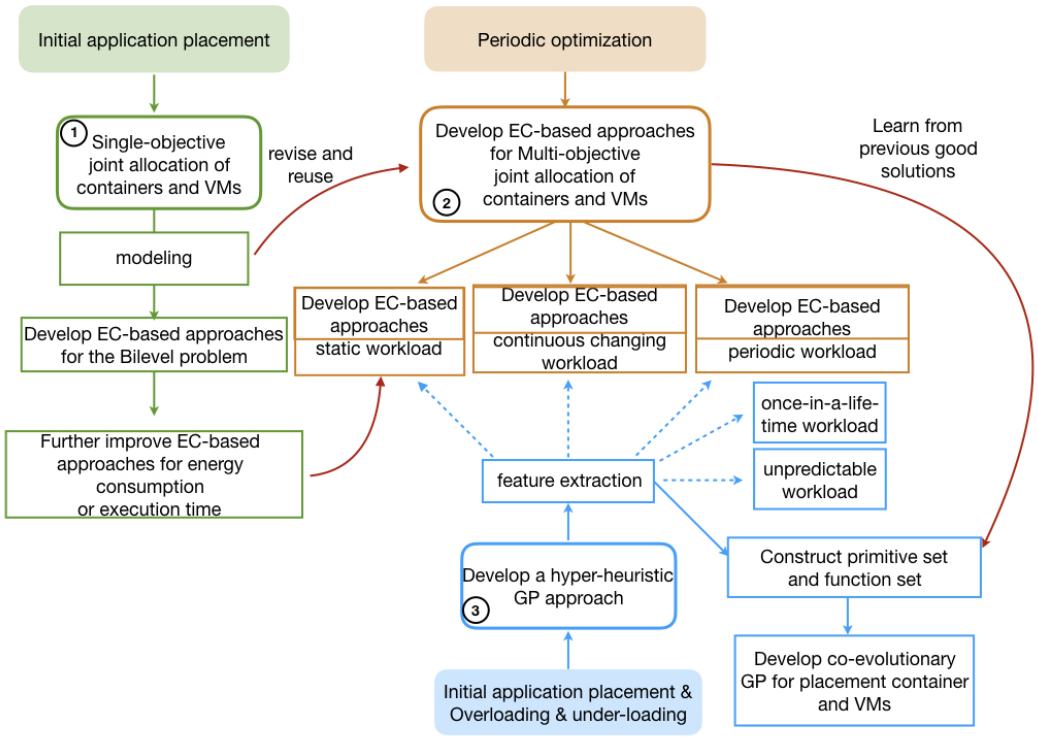


Figure 1.2: Relationship between objectives

1.3 Research Goals

As previous section introduced, we aims at providing a series of approaches to continuously optimize the a joint allocation of VMs and containers that considers three consolidation scenarios: application initial placement, periodic consolidation, and Dynamic consolidation.

1.3.1 Objective One: Develop EC-based approaches for the single objective joint allocation of containers and VMs

Currently, most research focus on VM-based server consolidation technique. They often modeled this problem as a vector bin-packing problem [124]. Container adds an extra layer of abstraction on top of VM. The placement problem has become a two-step procedure, in the first step, containers are packed into VMs and then VMs are consolidated into physical machines. These two steps are inter-related to each other. Previous research [84] solve this problem in separated steps where the first step allocate containers to VMs and the second step allocate VMs to PMs with simple bin-packing heuristics. According to Mann's [70] observation, we should consider these two allocation simultaneously to reach a near-optima solution.

In order to gradually solve the problem, we first ask three question, the answers will lead to the final solution.

1. How to model the bilevel server consolidation problem?

   Since there is no existing models for this problem, our first sub objective is to propose a descriptive single objective model for the bilevel optimization problem of joint allocation of container and VM. The reason to establish this model is because current server

consolidation models are mostly VM-based, they cannot be directly applied on bilevel problems. Therefore, variables, constraints and objective functions need to be clarified before applying any optimization algorithm. Each level of the problem will be formulated to a multi-dimensional vector bin packing problem. It is still unclear that which objective function is the best to capture the relationship between container and VM so that the overall energy is low. We will investigate several resource wastage models [42, 46, 121] and select a suitable one. In addition, several models have to be considered, including energy model [28], price model [1]. In addition, we will start from the simplest case - single dimension of resource - to more general multi-dimensional resources model. In this objective, we consider the applications' demand is provided by Cloud users and they are static which means the workloads of application remain constant over a period of time.

2. How to solve the problem with Evolutionary computation techniques?

   We will first develop a baseline approach that solve the problem using nested Evolutionary algorithms [97]. We will start from the simplest form: one dimensional bin-packing in each level to more complex multi-dimensional bin-packing.

   Nested methods have been used in solving bilevel problem for years, they are reported as effective approaches. We will investigate several approaches such as Nested Particle Swarm Optimization [66], Differential evolution (DE) based approach [2, 127] and Co-evolutioanry approach [64]. In order to adapt our problem to these existing approaches, we will develop suitable representations and genetic operators that are more suitable for this binary or discrete problem. In literature, there are various representations for packing problem, for example, Xiong et al [120] propose an indirect representation where the allocation of a VM to a PM is represented as a probability. While, Xu et al [121] propose a direct representation that groups VMs into several PMs.

3. How to improve the quality of solution in terms of energy consumption or algorithm execution time?

   Although nested approaches have been reported effective, they are very time consuming. Therefore, this sub objective intends to explore other directions to improve either better energy consumption or faster execution time. One possible direction is single-level reduction, which reduces the bilevel problem into a single dimensional problem. Clustering approaches such as K-means, DBScan can be useful in categorizing containers in predefined groups. Then, complementary containers can be grouped to reduce the variables of placement. Another possible solution is using a representation which embedded with simple heuristic (e.g First Fit); it allows to consider less number of PMs.

1.3.2 Objective Two: Develop EC-based approaches for the multi-objective joint allocation problem

As previously (see Section 1.2) mentioned, the task is multi-objective: minimizing the number of migration and minimizing the overall energy consumption. This two objectives are conflicting since intensive optimization may incur a large number migration. The first challenge is how to solve the multi-objective bi-level optimization problem. In addition, we consider propose a robust periodic optimization which means the placement of applications does not affect much from the variant workloads. Therefore, we divide workloads into five categories according to Fehling [**?**]: static, periodic, once-in-a-life-time, continuously changing, and unpredictable. Among five types of workloads, two of them: once-in-a-life-time

and unpredictable workloads are unsuitable for static placement, since their behavior are hard to foresee and plan, hence, they are normally solved by dynamic approaches which will be addressed in our third objective. For static, periodic, and continuously changing workload, we are going to design specific solutions. We also use three questions to guide our objective.

1. How to design an EC-based approach which solve the multi-objective joint allocation problem considering **static workload**?

   In this sub-objective, we will mainly focus on developing EC-based approaches to solve the multi-objective joint allocation problem. Since both container and VM can be migrated, that is, multi-objective exists on both levels. We will start from a simple case with considering multi-objective in lower level: Minimizing container migration and energy consumption. The single objective model developed in previous objective can be modified and reused. For multi-objective bilevel optimization, there are few studies using EC methods [32, 123]. However, most of them are designed for continuous problems. Therefore, new representations and operators need to be considered for discrete problem. We might use the experiences in previous objective.

   The assumptions for this objective, we will start from one dimensional of resource: CPU utilization. We will consider the static workload in this sub objective because they are common and easy to start with. We can utilize the representation and problem models from previous objective. However, we need to propose new genetic operators to adapt the multi-objective problem.

2. How to design an EC-based approach which considers **continuously changing workloads**? In comparison with static workload, continuously changing workloads (increasing or decreasing) is a more general case. They can be predicted with statistical regression approaches. The placement needs to consider their characteristics such as trend, changing ratio. Main difficulty is to develop a representation that allows the combined applications can be evaluated by previous designed fitness functions. One possible way is to combine complementary workloads into paired and treat the paired applications as a static application.

3. How to design an EC-based approach which considers **periodic workloads**?

   In this sub-objective, we will design a time-series aware EC-based server consolidation approach with considering periodic workload. Since periodic workload is one of the major type of workload, we will consider how to adapt it into the previously designed multi-objective algorithm. Previous research also consider periodic workload [74], however, they only consider combining paired complementary workloads, therefore, it is very limited. In contrast, we are going to consider combining multiple periodic workloads. We will start from the simplest case of stationary workload (the mean value of the series that remains constant over a time period). Depending on the measurement, direct and indirect representation will be explored.

1.3.3 Objective Three: Develop a hyper-heuristic Genetic Programming (GP) approach for automatically generating dispatching rules for dynamic consolidation

Previously, dynamic consolidation methods,including both VM-based and container-based, are mostly based on bin-packing algorithm such as First Fit Descending and human designed heuristics. As Mann's research [69] shown, server consolidation is more harder

than bin-packing problem because of multi-dimensional of resources and many constraints. Therefore, general bin-packing algorithms do not perform well with many constraints and specific designed heuristics only perform well in very narrow scope. Genetic programming has been used in automatically generating dispatching rules in many areas such as job shop scheduling [78]. GP also has been successfully applied in bin-packing problems [18]. Therefore, we will investigate GP approaches for solving the dynamic consolidation problem. We will start from considering one-level of problem: migrate one VM each time to a PM.

1. First, we will investigate which features and attributes are important when dealing with energy efficiency problem. As the basic component of a dispatching rule, primitive set contains the states of environment including: status of VMs (e.g. utilization, wastage), features of workloads (e.g. resource consumption). Furthermore, it has been shown that web workloads have such properties as correlation between workload attributes, nonstationarity, burstiness, and self-similarity [39]. Although there is no research has investigate how to use them to construct dispatching rules, there are extensive statistical analysis on workload [109]. The effectiveness of functional set and primitive set will be tested by applying the constructed dispatching rules on dynamic consolidation problem.
2. Develop GP-based methods for evolving dispatching rules. This sub-objective explores suitable representations for GP to construct useful dispatching rules. It also proposes new genetic operators as well as search mechanisms.

1.4 Published Papers

During the initial stage of this research, some investigation was carried out on the model of container-based server consolidation [105].

1. Tan, B., Ma, H., Mei, Y. and Zhang, M., "A NSGA-II-based Approach for Web Service Resource Allocation On Cloud". *Proceedings of 2017 IEEE Congress on Evolutioanry Computation (CEC2017).* Donostia, Spain. 5-8 June, 2017.pp.2574-2581

1.5 Organisation of Proposal

The remainder of the proposal is organised as follows: Chapter **??** provides a fundamental definition of the Container-based server consolidation problem and performs a literature review covering a range of works in this field; Chapter **??** discusses the preliminary work carried out to explore the techniques and EC-based techniques for the initialization problem; Chapter **??** presents a plan detailing this projects intended contributions, a project timeline, and a thesis outline.

Chapter 2

Literature Review

2.1 Background

2.1.1 Cloud computing

Cloud computing is a computing model offers a network of servers to their clients in a on-demand fashion. From NIST's definition [73], *"cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

To illustrate how it works, considering the case: a Cloud provider builds a data center which contains thousands of servers connected with network. These servers are virtualized which means they are partitioned into smaller units of resources called *Virtual Machines (VMs)* or *Containers* [41]. A web-based application provider can access and deploy their applications (e.g Endnote, Google Drive and etc.) in these resource units from anywhere in the world. Once the applications start serving, application users can use them without installing on their local computers.

Cloud computing involves three stakeholders (see Figure 2.1): Cloud providers, Cloud users (applications providers), and End (application) users [53]. Cloud providers build data centers, provide maintenance and resource management on hardware infrastructures such as servers. Cloud users develop and deploy applications on Cloud infrastructures. End users consumes applications developed by Cloud users and hosted by Cloud providers.

The detailed goal and objectives of stakeholders are described below.

- *Cloud providers*' goal is to increase the profit by boosting the income and reducing the expense. Their income comes from Cloud users' rental of servers or *Physical Machines (PMs)* in terms of resource quality (e.g 3.5GHz dual-core CPU), quantity (e.g 3 PMs), and time (e.g 1 year). Therefore, Cloud providers objective is to maximize utilization of computing resources. A high utilization brings two benefits, firstly, it increases income by accommodating more applications in limited resources. Secondly, it cuts the expense of energy consumption by packing applications in a minimum number of PMs so that idle PMs can be turned off.

- *Cloud users*' goal is also to increase the profit mainly through two objectives, attracting more End users and reduce the expense of resources. The first objective can be achieved by improving the quality of service as well as lower the fee for End users. Either way depends not only on the software entities but also the quality of service (QoS) offered by Cloud provider. The second objective can be achieved by a good es-

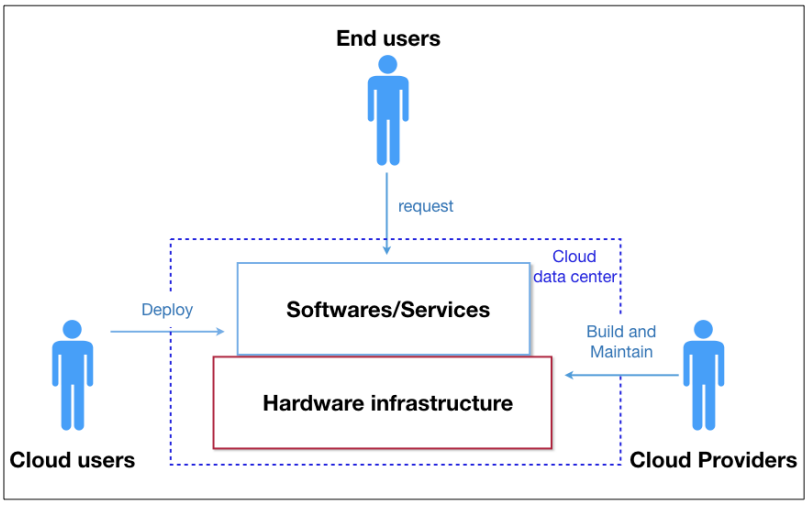


Figure 2.1: Stakeholders of Cloud computing

timation of the reserved resources, so that they do not rent insufficient or too much resources which cause performance degradation or wastage.

- *End Users'* goal is to obtain a satisfactory service. It is achieved by signing a Service Level Agreement (SLA) with Cloud users which constrains the performance of the services.

Cloud computing has three traditional service models [73]: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The relationship among three service models is showed in Figure 2.2. Service models of Cloud computing are critical in solving energy consumption problem because their distinct ways of managing resources have sever effect on the problem. These distinct ways of resource management mainly result from the responsibilities among stakeholders.

- IaaS, a Cloud provider hosts hardwares such as PMs and cooling infrastructure on behalf of Cloud users. Computational resources are often encapsulated in virtualized computing units called virtual machines (VMs). Cloud providers establish a number of types of VM for simplifying the management. The 'type' means a VM contains a certain quantity of resources such as 2-cores and 1 GB RAM. *Traditional* IaaS and PaaS use VM as the fundamental unit of resources.

  A typical procedure of an application deployment includes several steps. Initially, the Cloud user estimates the resources that their applications might consume and selects a type of VM which can satisfy the requirement. After the Cloud user has made the decision, he/she sends requests to Cloud providers for a number of VMs. Finally, Cloud providers received the request, provisioned and allocated these VMs to PMs.

- PaaS, a Cloud provider offers a platform which allows Cloud users to develop, test and deploy their applications on it.

  From resource management perspective, PaaS is sitting above the IaaS which means the underlying resource is still based on IaaS VM types. Different from IaaS, PaaS takes the responsibility of selecting VMs and allows Cloud users to focus on software development.

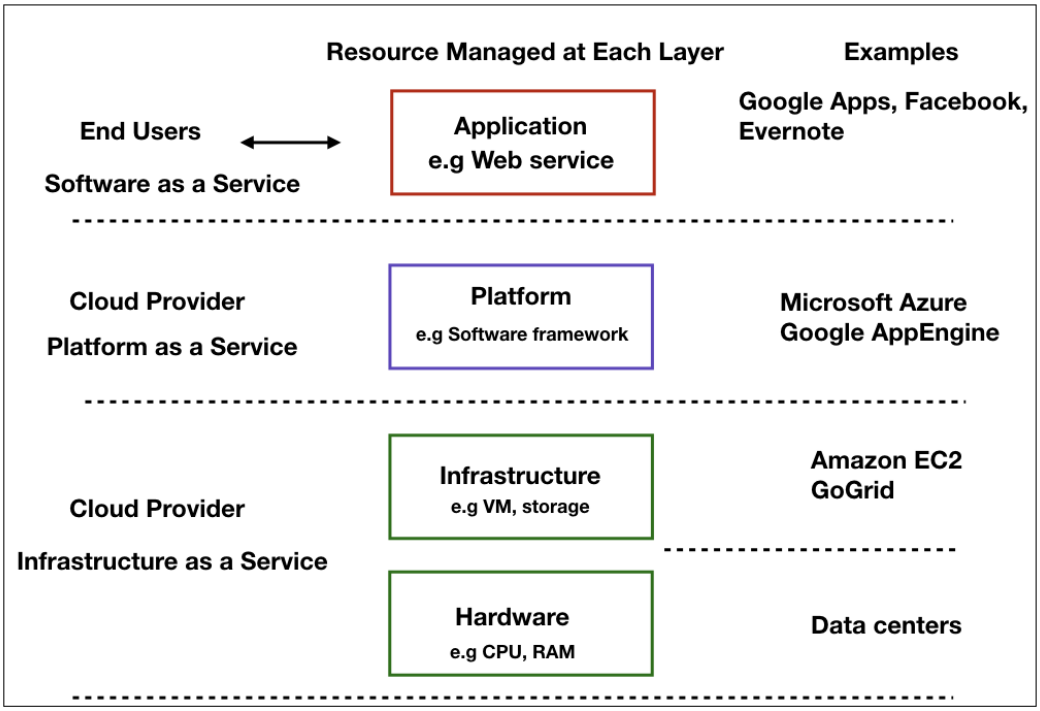


Figure 2.2: Cloud computing architecture [125]. IaaS provides the fundamental resources such as CPU cores, RAM. The resources are usually wrapped with various types of virtual machine. PaaS provides a software platform sitting on top of IaaS for Cloud users to develop applications. SaaS describes the relationship between applications and End Users.

- SaaS, Cloud users develop applications and deploy them on Cloud so that End users can access them via the Internet. Although this service model does not directly related to the resource management, it provides the fundamental reasons for resource management and optimization: applications receive fluctuated requests from End users. Because of the dynamic nature of workloads, the underlying resources must also be dynamic adjusted to meet the requirement.

Overall, Cloud computing has five chacteristics:

1. On-demand self-service: A Cloud user can require computing resources (e.g CPU time, storage, software use) without the interaction with Cloud provider.
2. Broad network access: Computing resources are connected and delivered over the network.
3. Resource pool: a Cloud provider has a "pool" of resources which are normally virtualized servers. In IaaS, it provides predefined sizes of VMs. In PaaS, the resources are 'invisible' to Cloud users who have no knowledge or ability to control.
4. Rapid elasticity: From the perspective of Cloud users, computing resources are assigned and released in real time. In addition, the resources assign to their software is "infinite". Therefore, Cloud users do not need to worried about the scalability of their applications.
5. Measured Service: Cloud provides an accurate measure of the usage of computing resources. It is fundamental to the pay-as-you-go policy.

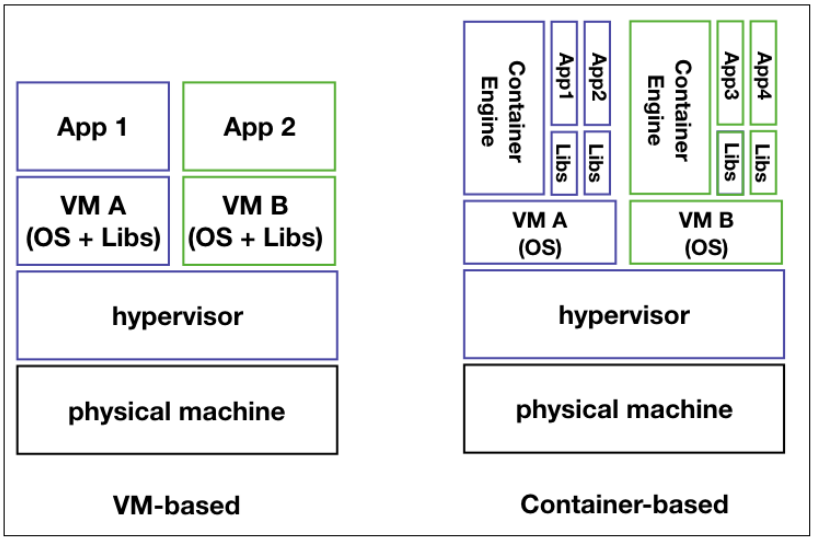


Figure 2.3: A comparison between VM-based and Container-based virtualization [83]

2.1.2 Virtualization

Virtualization [107] is the fundamental technology that enables Cloud computing. It partitions a physical machine's resources (e.g. CPU, memory and disk) into several independent units called virtual machines (VMs) or containers. This technology rooted back in the 1960s' and was originally invented to enable isolated software testing, because each virtualized unit can provide good isolation which means multiple applications can run in separated VMs within the same PM without interfering each other [99]. Soon, people realized that it can be a way to improve the utilization of hardware resources: With each application deployed in a VM, a PM can run multiple applications.

There are two classes of virtualization (see Figure 2.3): Hypervisor-based or VM-based and container-based virtualization.

Virtual machine

A virtualized system includes a new layer of software - the hypervisor or the vrirtual machine monitor (VMM). The VMM arbitrates accesses to the PM's resources so that guests' OS can share them [**?**]. In the previous decades VM-based hypervisors such as Xen [6], KVM [59], and VMware ESX [112] dominate this field.

Container

Container-based virtualization is also often addressed as operating-system-level virtualization. It includes two types of container: OS container and application container [**?**]. OS container (as shown in Figure 2.4) can run in both PM and VM. Each container provides an isolated environment. There are mainly three implementations of OS-level of containers: OpenVZ, Google's control groups, and namespace [90]. Google and Facebook have been using OS container for years and being beneficial for its lightweight and fast communication among applications.

In contrast of OS containers, an application container, such as Docker and Rocket, runs a single process. It allows to separate an applications into many components. With application container, it is easy to achieve auto-scaling on a single process. In comparison between OS and application container, the former can be seen as a VM runs multiple applications

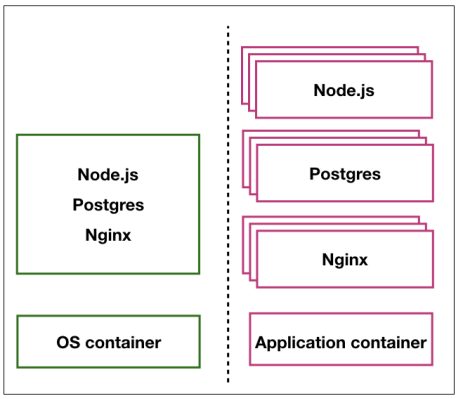


Figure 2.4: A comparison between OS container and Application container [**?**]

where each application can has its separated environment (e.g. libraries), while an application container act like a single unit in OS container and it can be allocated in both OS container and VM.

The recent development of application container such as Docker and Kubernetes [11] have attract the attention from both academia and industry. It provides a finer granularity of resource management by enabling an application level of operations including deployment, scaling, and migration.

Comparison between Container and VM

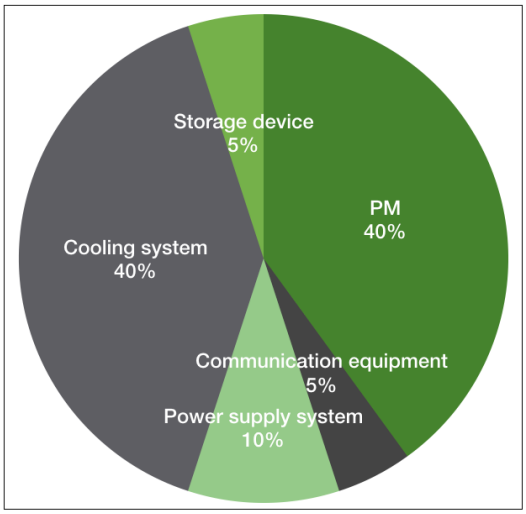
Several research [34, 41, 117] have compared VM-based and container-based virtualization. The advantages of container-based virtualization are mainly from three aspects:

- Low overhead, containers have a lightweight management. which generate much less overhead than a hypervisor. On the other hand, shared OS system also reduces the overhead on running multiple OSs.
- Containers have a near-native performance of CPU, memory, disk and network. While VM has a poor I/O performance [92]: up to 50% reduction of bandwidth (e.g hard disk and network). This defect also has a negative effect on the migration performance, since a VM-image is ranging from hundreds of MB to several GBs.
- Containers naturally support vertical scaling while VMs do not. Vertical scaling means a container can dynamically adjust its resources under the host's resource constraint.This feature offers a fine granularity management of resources.

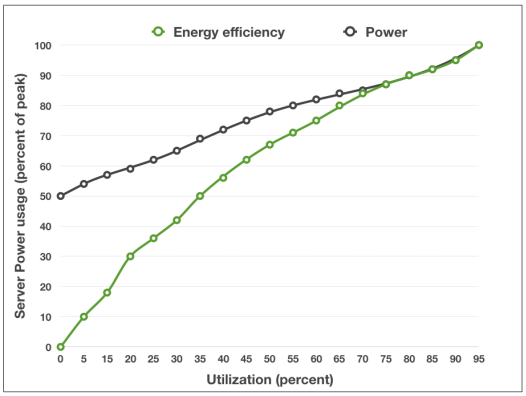
On the other hand, the disadvantages of containers are categorized in two aspects. Currently, containers have rather poor performance in isolation for memory, disk, and network. Security is immature in containers [11], therefore, high security required applications are not recommended to be deployed in such systems.

2.1.3 Energy Efficiency in Cloud Data centers

Apart from upfront investment, Energy consumption [55] is the major expense of data centers. Therefore, it is also the top concern of Cloud providers. Energy consumption is derived from several parts as illustrated in Figure 2.5a. Cooling system and servers or PMs account for a majority of the consumption. A recent survey [22] shows that the recent development



(a) Energy consumption distribution of data centers [89]



(b) Disproportionate between utilization and energy consumption [7]

of cooling techniques have reduced its energy consumption and now server consumption has become the dominate energy consumption component.

According to Hameed et al [47], servers are far from energy-efficient. The main reason for the wastage is that the energy consumption of servers remains high even when the utilization are low. Therefore, a concept of *energy proportional computing* [7] raised to address the disproportionate between utilization and energy consumption. This leads to using virtualization technology to achieve server consolidation.

2.1.4 Drawbacks of Traditional Service Models

There are three characteristics in IaaS which naturally lead to a low resource utilization.

- Resource over-provisioning
  As previous mentioned, VMs are either selected by Cloud users or automatically selected by software platforms. In either case, it requires an estimation of required resources. However, The accurate estimation is almost impossible because of unpredictable workloads; A simple way is to reserve more resources for ensuring the QoS at peak hours [20], rather than completely rely on auto-scaling, simply because auto-scaling is more expensive than reservation. However, the peak hours only account for a short period, therefore, in most of time, resources are wasted. In IaaS, the types of VM are a part of the contract, Cloud providers cannot simply change the type of VMs after provisioning.

- Unbalanced usage of resources
  Specific applications consume unbalanced resources which leads to vast amount of resource wastage [106]. For example, computation intensive tasks consume much more CPU than RAM; a fixed type of VM provides much more RAM than it needs. Because the tasks use too much CPU, they prevent other tasks from co-allocating. This also causes wastage.

- Heavy overhead of VM hypervisors and redundant operating systems (OSs)
  In VM-based resource allocation, heavy overhead is caused by the hypervisor of VMs and the separated operating systems running in the PM. A hypervisor manages and monitors the VMs running on a PM. The overhead of a hypervisor is heavier with the

increasing numbers of VM. Redundant operating system is another reason for overhead, as normal applications do not need specific operating systems; Commonly used OSs - such as Linux-based: RedHat, or Windows server versions - are well enough for their needs. Therefore, running applications in separated operating system simultaneously is unnecessary.

For traditional PaaS, Cloud providers can adjust the VMs' location and the type of VMs. Therefore, it overcomes the first drawback of IaaS. Because of PaaS is built upon IaaS, the one-on-one relationship between application and VM still exist. PaaS can only select the most suitable type instead of changing their sizes. Therefore, the unbalanced resource problem cannot be solved. In addition, PaaS brings a restriction for the applications deployed on it. PaaS build a software middle-ware to allow Cloud users' development. The middle-ware requires the deployed applications to be compatible with the environment, for example, Google App engine only allows certain programming languages and libraries. Therefore, the generality of PaaS is limited. It is urgent to provide a environment which supports automatic resource management as well as an editable programming environment.

2.1.5 Container as a Service

Container as a Service (CaaS) [**?**] is a new service model which is usually considered as a combination of IaaS and PaaS. CaaS uses containers and VM as its fundamental resource allocation unit as shown in Figure 2.3 on the right hand side.

CaaS has advantages of both IaaS and PaaS but without their disadvantages. On one hand - similar to PaaS - CaaS allows Cloud providers to manage resource in a fine granularity with containers, therefore, it may lead to high utilization of resources. On the other hand - similar to IaaS - CaaS allows customers to customize their software environment without being constrained by platforms. Therefore, it has more flexibility than PaaS.

2.1.6 Bilevel Optimization

A bilevel optimization [25] is a kind of optimization where one problem is embedded within another. The outer optimization is referred to as the *upper-level* problem, while the inner optimization is referred to as the *lower-level* problem .

The general formulation of a bilevel optimization problem can be defined as:

$$min_{x \in X, y} \quad F(x, y) \tag{2.1a}$$
$$s.t \quad G(x, y) \leq 0, \tag{2.1b}$$
$$min_{y} \quad f(x, y) \tag{2.1c}$$
$$s.t \quad g(x, y) \tag{2.1d}$$

The variables of problem, $x \in \mathbb{R}^{n_1}$ is the *upper-level variables* and $y \in \mathbb{R}^{n_2}$ is the *lower-level variables*. The function $F : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}$ and $f : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}$ are the *upper-level* and *lower-level objective functions* respectively. The function $G : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}^{m_1}$ and $g : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}^{m_2}$ are called the *upper-level* and *lower-level constraints* respectively.

Bilevel optimization problem has a hierarchical structure which may introduce difficulties such as non-convexity and disconnectedness even for simple cases such as bilevel linear programming problems is strongly NP-hard [**?**].

In practice, there are a number of problems that are bilevel in nature. For example, transportation related: work design, optimal pricing [17, 26], management: network facility location [103], and engineering related: optimal design [58].

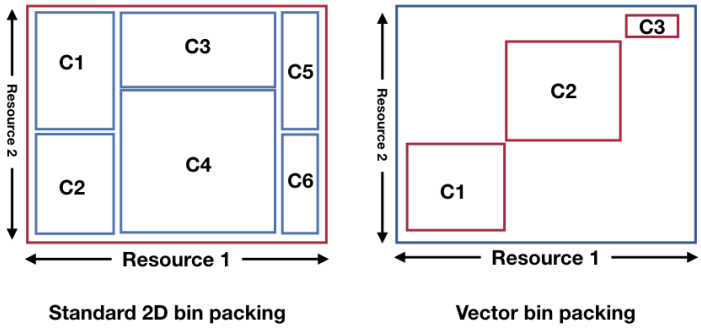


Figure 2.6: A comparison between standard bin packing and vector bin packing

2.1.7 Vector Bin Packing Problem

Vector bin packing problem is variant of standard bin packing problem (see Figure 2.6). It is also referred as multi-capacity [65] or multi-dimensional bin packing problem [120]. It is particularly suitable for modeling resource allocation problems where there is a set of bins with known capacities and a set of items with known demands [80].

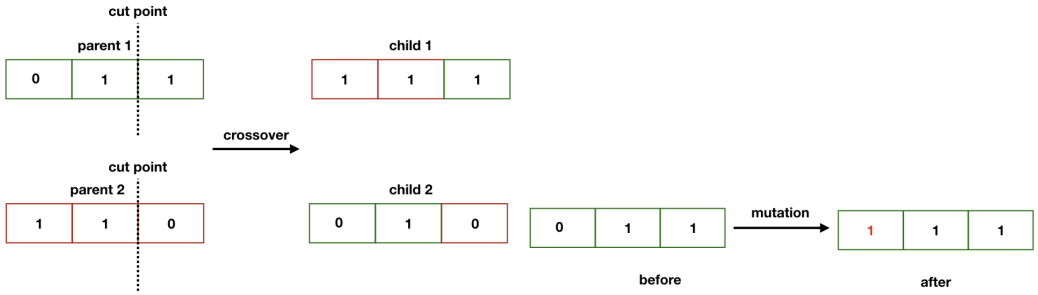
A d-dimensional Vector Bin Packing Problem (VBP_d), give a set of items $I^1, I^2, \ldots, I^n$ where each item has d dimension of resources represented in real or discrete number $I^i \in R^d$. A valid solution is packing I into bins $B^1, B^2, \ldots, B^k$. For each bin j and each dimension i, the sum of resources can not exceed the capacity of bin. The goal of Vector Bin Packing problem is to find a valid solution with minimum number of bins. Notice that, the items assigned to bins do not consider the positions in the bins, that is, there is no geometric interpretation of the items or bins [54]. Notice that, when $d = 1$, vector bin packing reduces to the classic *bin-packing* problem. Vector bin packing is an NP-hard problem in strong sense, as it is a generalized bin packing problem.

Several one dimensional bin packing algorithms have been generalized to vector bin packing such as generalized First Fit and Best Fit. In studies [], they suggest that an algorithm should keep the bins as "balanced" as possible, so that bins have better potential to accommodate more items.

2.1.8 Evolutionary Computation

Evolutionary Computation is a subfield of Artificial Intelligence. It is very well-known for its effective global search ability. These algorithms are inspired by biological mechanisms of evolution, social interactions and swarm intelligence. There are several distinguished characteristics of EC algorithms such as the use of a population-based search and the ability of avoiding local optima. A common process of EC algorithms is as follows. Initially, the population are randomly generated in the search space. During the evolution process, the population explores according to the evaluation of a or multiple predefined fitness functions. At the end of the evolution, the individual with the best fitness value is selected as the output.

The origins of evolutionary computation can be traced back to 1950s [3]. Since 1970s, the output of EC research has grown exponentially. The majority of current implementations of EC algorithms descend from three major approaches: *genetic algorithms*, *particle swarm optimization* and *genetic programming*.



(a) Crossover (b) Mutation

Genetic Algorithm

GA [51] was introduced by Holland. Since then, a large number of applications [29, 30] have applied GA as their search mechanism. In a GA, a population of chromosomes which represents the solutions of the problem are often represented as a fixed length string. Each entry or bit can be continuous, discrete or even alphabets depending on the problem. The evolution often starts from randomly generated population followed by an iterative process called generation. In each generation, the fitness value of chromosomes is evaluated according to a defined fitness function. Then, genetic operators such as mutation, crossover are applied on the solutions so that they are modified. As a search mechanism, these operators move solutions to explore the search space. New generation of solutions are then evaluated by the fitness function. This evolutionary procedure ends with a predefined a generation number or a satisfactory level of fitness has been reached.

Particle Swarm Optimization

Kennedy and Eberhart proposed PSO in 1995 [35]. It is a meta-heuristic algorithm inspired by the social behavior of birds. In PSO, each individual - a particle - searches the solution space. The underlying phenomenon of PSO is to use the collective knowledge to find the optimal solution.

At the initial state, each particle has a random initial position in the search space which is represented by a vector $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$, where D is the dimension of the search space. A particle has a velocity as $\vec{v}_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. The velocity is limited by a threshold $v_{\max}$ so that for any i and d, $v_{id} \in [-v_{\max}, v_{\max}]$. During the search process, each particle maintains a record of its best position so far, called the *personal best* (*pbest*). The best positions among all the personal best positions of its neighbors is the *global best* (*gbest*). The position and velocity of each particle are updated according to the following equations:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}, \tag{2.2}$$

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 \cdot r_{1i} \cdot (p_{id} - x_{id}^{t}) + c_2 \cdot r_{2i} \cdot (p_{pg} - x_{id}^{i}). \tag{2.3}$$

Here, t is the index of iteration. d is the index of dimension. The inertia weight w is used to balance the local search and global search abilities. The parameters c_1 and c_2 are the acceleration constants. r_{1i} and r_{2i} are random constants following the uniform distribution in the interval $[0, 1]$. p_{id} and p_{gd} denote the values of *pbest* and *gbest* in the d^{th} dimension of the i^{th} particle.

PSO was initiated to solve continuous optimization problems - position update function is developed for real numbers. To address discrete problems, Kennedy and Eberhart developed a binary PSO [56].

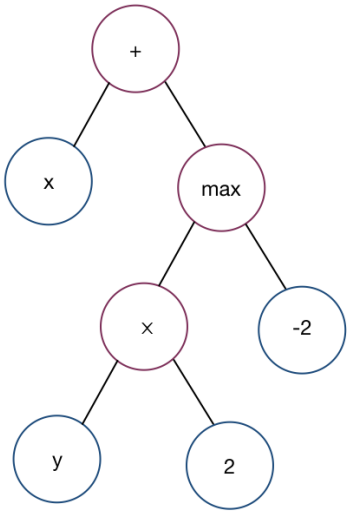


Figure 2.8: GP program that represents x + max(y $\times$ 2, -2)

Genetic Programming

Genetic programming [60] is an evolutionary computation technique, inspired by biological evolution, to automatically find computer programs for solving a specific task. In a GP population, each individual represents a computer program. In each generation, these programs are evaluated by a predefined fitness function, which accesses the performance of each program. Then, individuals will go through several genetic operators such as selection, crossover, and mutation. A number of top individuals will survive to the next generation while others will be discarded. The major difference between GA and GP is that, each GP individual is represented as a tree with variant depth instead of a string. This representation is particular suitable for a program. For example, a GP individual is showed in Figure 2.8 which is a program x + max(y $\times$ 2, -2). The variables {x, y} and constraint {-2, 2} are called terminal of the program. The arithmetic operations {+, $\times$, max } are called functions in GP. A GP individual is a specific combination of elements in terminal set and functional set. In order to observe the relationship between a function and its subtrees, the GP programs are usually presented to human users by using the *prefix* notation similar to a Lisp expression, for example, x + max(y $\times$ 2, -2) can be expressed as (+ (x (min ($\times$ y 2) -2))).

GP approach based hyper-heuristics (GP-HH) has been applied in many applications such as Job shop scheduling to evolve dispatching rules [78]. The term hyper-heuristics [27] means "heuristics to choose heuristics". *Dispatching rule* is essentially a heuristics [81] used in a scheduling context. Resource allocation problems are also in the scheduling category and they are often modeled as bin packing problems. GP-HH has been applied in generating heuristics for bin-packing problems [19,85,95]. These research have shown that GP-HH can generate excellent heuristics which have equal or better performance than human designed heuristics.

In the Cloud computing context, Cloud resource allocation usually has extra constraints such as multi-dimensional resources, migration costs, heterogeneous PMs etc. These constraints make the Cloud resource allocation problem much harder than original bin packing [69]. Therefore, traditional bin packing approaches such as First Fit Decreasing, Best Fit etc cannot perform well in this context. GP-HH, therefore, is a promising technique can be used to automatically generate heuristics under multiple constraints.

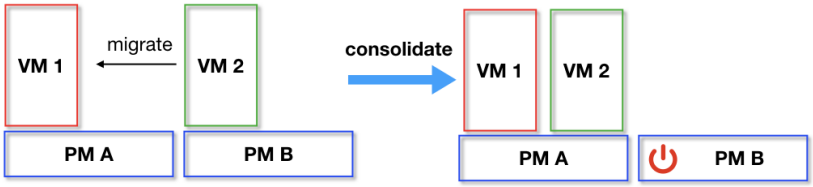


Figure 2.9: A Server Consolidation example: Initially, each PM runs an application wrapped with a VM in a low resource utilization state. After the consolidation, both VMs are running on PM A, so that PM B is turned off to save energy [7].

2.1.9 Server consolidation

Server consolidation packs a number of VMs on fewer number of PMs to improve the resource utilization and decrease the energy consumed by PMs. It is often applied to solve the problem of physical server sprawl [57]: a situation that more PMs are used in a low-utilized way.

From a broad technologies perspective, there are generally two technologies can be used to achieve server consolidation: clustering and virtualization. Clustering is used in a situation that the applications running in PMs are I/O intensive. This is because current virtualization technologies such as KVM [59] or Xen [6] have a 20% to 55% of reduction of I/O bandwidth (e.g disk reads and writes, network bandwidth) in comparison with non-virtualized PM [92]. Virtualization is more suitable for applications which require little CPU utilization (e.g 15%) and low I/O needs. Web services are mostly categorized into this group. Three major benefits of virtualization make it be the first choice for web-based application consolidation: No reliance on hardware, easy to provision and live migration.

Virtualization-based Server consolidation [125] utilized a dynamic migration technique (e.g pre-copy [23] and post-copy [50]) to resolve the low utilization problem by gathering applications into a fewer number of PMs (see Figure 2.9), so that the resource utilization of PMs are maintained at a high level. In the meanwhile, idle PMs can be turned off to save energy. Consolidation dramatically improves hardware utilization and lowers PM and cooling energy consumption.

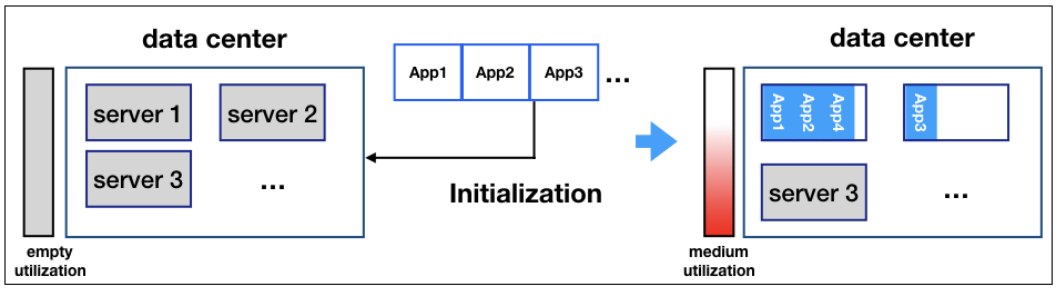
Server consolidation can be done in two ways: Static and Dynamic [109, 119] which are applied in different resource management scenarios (discuss in next section). In some scenarios, for example: new application initialization and global consolidation, involve large number of variables, therefore, it is very time-consuming job and often conducted in an off-line fashion. In other scenarios, when PMs are overloading or underloading, it requires fast a decision-making to migrate one or more VMs to reduce the burden on overloaded PM or improve the utilization. It migrates one VM at a time with a dynamic method.

General Server Consolidation Scenarios

The server consolidation in data center can be applied to three [75, 104] scenarios: Application initialization, Prediction and Global consolidation, and Dynamic resource management (see Figure 2.10).

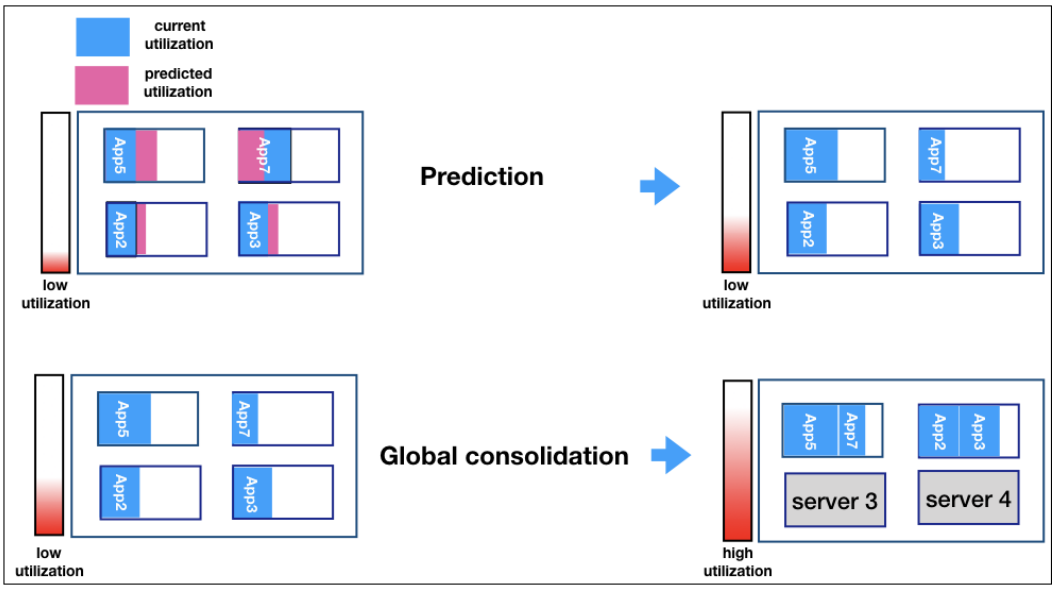
1. *Application initial placement* is applied when new applications or new VMs arrive and the problem is to allocate them into a minimum number of PMs [75].

   In this problem, a set of applications or VMs are waiting in a queue. The resource capacity of the PM and usage by applications are characterized by a vector of resource utilizations including CPU, memory and etc. Then, the allocation system must select



(a) Initialization

(b) Dynamic resource management



(c) Prediction and Consolidation

Figure 2.10: Three scenarios of resource management

a minimum number of PMs to accommodate them so that after the allocation, the resource utilizations remain high. The problem is to consider the different combinations of applications so that the overall resource utilization is high. This problem is naturally modeled as a static bin-packing problem [24] which is a NP-hard problem meaning it is unlikely to find an optimal solution of a large problem.

2. *Prediction and Global consolidation* is conducted periodically to adjust the current allocation of applications so that the overall utilization is improved.

   In this problem, time is discrete and it can be split into basic time frames, for example: ten seconds. A periodical operation is conducted in every N time frames. A cloud data center has a highly dynamic environment with continuous arriving and releasing of applications. Releasing applications cause hollow in PMs; new arrivals cannot change the structure of current allocation. Therefore, after the initial allocation, the overall energy efficiency is likely to drop along with time elapsing.

   In prediction, an optimization system takes the current applications' utilization records as the input. Make a prediction of their utilization in the next period of time. In Global consolidation, based on the predict utilization and the current allocation - including a list of applications/VMs and a list of PMs, the system adjusts the allocation so that the global resource utilization is improved.

   In comparison with initialization, instead of new arrivals, the global consolidation considers the previous allocation. Another major difference is that global consolidation needs to minimize the differences of allocation before and after the optimization. This is because the adjustment of allocation relies on a technique called live migration [23], and it is a very expensive operation because it occupies the resources in both the host and the target. Therefore, global optimization must be considered as a time-dependent activity which makes the optimization even difficult.

   In comparison with dynamic consolidation, global consolidation takes a set of VMs as input instead of one. Therefore, it is time consuming and often treated as a static problem.

3. *Dynamic resource management* is applied in three scenarios. **First**, it is applied when a PM is overloading. In order to prevent the QoS from dropping, an application is migrated to another PM. This is called hot-spot mitigation [75]. **Second**, it is applied when a PM is under-loading. Under-loading is when a PM is in a low utilization state normally defined by a threshold. At this moment, all the applications in the under-loading PM are migrated to other active PMs, so the PM becomes empty and can be turned off. This is called dynamic consolidation. **Third**, it is applied when a PM having very high level of utilization while others having low. An adjustment is to migrate one or more application from high utilized PMs to low ones. This is called load balancing.

   No matter which scenario it is, a dynamic resource management always involves three steps .

   - *When to migrate?* refers to determine the time point that a PM is overloaded or underloaded. It is often decide by a threshold of utilization.
   - *Which application to migrate?* refers to determine which application need to be migrated so that it optimize the global energy consumption.
   - *Where to migrate?* refers to determine which host that an application is migrated to. This step is called dynamic placement which is directly related to the consolidation, therefore, it is decisive in improving energy-efficiency.

Among three operations, dynamic placement is a dynamic and on-line problem. The term "dynamic" means the request comes at an arbitrary time point. An on-line problem is a problem which has on-line input and requires on-line output [16]. It is applied when a control system does not have the complete knowledge of future events.

There are two difficulties in this operation, firstly, dynamic placement requires a fast decision while the search space is very large (e.g hundreds of thousands of PMs). Secondly, migrate one application at a time is hard to reach a global optimized state.

Finally, a consolidation plan includes four major items:

1. A list of existing PMs after consolidation
2. A list of new virtual machines created after consolidation
3. A list of old PMs to be turned off after consolidation
4. The exact placement of applications and services

2.2 Related Work

In this section, we summarize the related works in the terms of three resource management processes: application initial placement, periodic consolidation and dynamic consolidation, corresponding to our proposed objectives. In each sub section, we analyze the container-based and VM-based approaches, discuss their models and methodologies.

2.2.1 Application Initial Placement

Application initial placement is one of the major process in resource management as discussed in Section 2.1.9. The task of application initial placement for cloud provider is to deploy a number of VMs or containers to a minimum number of empty physical machines (PMs), because energy consumption is proportional to the PM number.

This section first discusses container-based approaches including modeling and methodology. The modeling includes power model and wastage model. Then, we will discuss and reason their disadvantages in design. In order to remedy their problems, we will review a number of VM-based placement including traditional bin packing and more advanced approaches.

Container-based Application Initial Placement

This section will briefly discuss the models of container-based approaches. Furthermore, our focus is to discuss the design of separate concern of container placement and VM placement as well as the collaborative placement. This will lead to our first objective: Develop an algorithm to solve the joint allocation of containers and VMs.

There are only few container-based research in the literature which mainly follow the VM-based modeling. Piraghaj et al [83] and Mann [70] both consider a linear energy model which is widely adopted in the literature [118]. They consider two resources: CPU and memory. In addition, the constraints are allocated resources cannot exceed the PMs' capacity. In terms of wastage model, so far, they did not use any measure on the residual resources. In contrast, in most VM-based approaches, wastage has been widely considered. It will be discussed in the next section.

The design of three-tier structure: container-VM-PM raises a placement concern, optimize each tier separately and collaboratively. Piraghaj et al [83] consider it separately. Their

placement strategy replies on fast and simple heuristic: in each tier of placement, they apply First Fit algorithm. While their main contribution is not the placement strategy but an architecture for container-based resource management. In this architecture, it allows the cloud providers to customize the size or type of VM when allocating applications instead of traditional fixed size VM. They determine the size of VM by two steps, in a pre-execution phase, they perform clustering technique on historical workload data from Google Cluster Data. In this way, they assert that the applications with similar workload pattern can be categorized into the same group. In the execution phase, when a number of deployment requests of container arrives, if there are enough available resources in the existing VMs, they perform First Fit on containers. If new VMs need to be initiated, they first determines the size of VMs by designed policies. Then, the containers are allocated to certain size of VMs.

Mann's research [70] is the earliest study which realizes this two level of placement are interact with each other, therefore, they must be considered collaboratively. Different from Piraghaj's assumption, they consider the container is based on an existing IaaS where fixed types of VM are provided. Therefore, the problem becomes three folds: 1. VM size selection for containers, 2. Place containers, 3. VM placement, and they should be considered together. However, another concern is "why not placement as many containers as possible in a single VM which minimizes the overhead of hypervisor". The paper gives an answer of "Too big VMs limit the consolidation possibilities". In order to prove their interaction, they apply a fixed VM placement algorithm and considering a series of VM selection algorithms such as simple selection [45], Multiple selection, Maxsize, Consolidation-friendly. They discover that the final energy consumption varies with the selection algorithms. They claim that the performance is better when VM selection has more knowledge of the PMs' capacity. However, their study only focuses on the partial placement with fixed VM placement algorithm. The answer of "How these two level of placement interact ?" is still undiscovered.

There are mainly three reasons for us to propose a distinct approach from Piraghaj [83] to solve the container-based placement problem. First, their proposed architecture assumes arbitrary size of VM can be created when requests arrive. While our assumption is that the container-based architecture is based on traditional IaaS, where fix-size VMs provide the fundamental resources. Second, from the perspective of energy efficiency, the allocation of container and VM interact with each other. That is, the minimum number of VMs does not necessary lead to the minimum number of PMs, because the type of VMs also affect the results. Therefore, their approach cannot guarantee the energy consumption is near optimal. This inspires us to simultaneously allocate containers and VMs. Third, both approaches did not consider the OS requirement of containers or applications. We argue that this is another critical reason for deploying containers into different VMs besides the "limit the consolidation possibilities" argument states by Mann [70].

In order to solve the two-level of optimization problem, we believe one of the promising way is to model the problem as a bilevel optimization [25] as described in detailed in Section 2.1.6. Bilevel optimization naturally models the interaction so that this problem can be tackled by an optimization algorithm. The main challenge is that Bilevel optimization is a strongly NP-hard problem [**?**], therefore, traditional approaches such as Branch-and-bound [**?**] can only be applied in very small problems, or convert the problem into a single-level optimization problem. Evolutionary algorithms, on the other hand, becoming popular approaches in this field [96, 114].

VM-based Application Initial Placement

This section first reviews a number of traditional approaches for the VM placement problem. Their drawbacks will be presented. Then, a number of advanced approaches will be

Table 2.1: A Comparison of different models and approaches

Research	Resources	Algorithm	Power model	Wastage model	Objective
Xu et al [121]	CPU and RAM	GGA and Fuzzy multi-objective	Linear	balance resources	three
Gao et al [46]	CPU and RAM	Ant Colony Optimization	Linear	balance resources	Two
Ferdaus et al [42]	CPU, RAM, and IO	Ant Colony Optimization	Linear	Sum of resources	Single
Wang and Xia [113]	CPU and RAM	MIP	Cubical	No	Single
Wilcox et al [116]	CPU and RAM	GGA	Linear	Sum of resources	Single
Xiong and Xu [120]	CPU,RAM,Bandwidth,Disk	PSO	Non-linear	Sum of resources	Single

examined. We mainly study the following five aspects: resources, power model, wastage model, objective, and algorithm.

Most of the works model VM placement problem as variants of bin packing problem and propose extensions of greedy-based heuristics such as First Fit Decreasing (FFD) [**?**], Best Fit, Best Fit Decreasing [**?**] etc. However, as VM placement is an NP-hard problem, greedy-based approaches can not guaranteed to generate near optimal solutions. Mishra and Sahoo's paper [76] further analyze and discuss the drawbacks of these approaches. They found that, instead of standard bin packing, only vector bin packing is suitable for modeling resource allocation (see Section 2.1.7). Another drawback of traditional bin packing heuristic is that they do not consider the balance among resources which is a critical issue for vector bin packing problem. Their main contribution is that they list five principles for a good design of objective function, specially, the core idea is to capture the balance among resources.

Based on this insight, Gao et al [46] and Ferdaus et al [42] both propose an Ant Colony Optimization based metaheuristic using a vector algebra complementary resource utilization model proposed by Mishra [76]. They considered three resources CPU, memory, and network I/O with two objectives: minimizing power consumption and resource wastage. They apply the *Resource Imbalance Vector* to capture the imbalance among three resources. Meanwhile, they use a linear energy consumption function to capture the relationship between CPU utilization and energy [38]. Their solution was compared with four algorithms: Max-Min Ant System, a greedy-based approach, and two First Fit Decreasing-based methods. The results show that their proposed algorithm has much less wastage than other algorithms.

Xu and Fortes [121] propose a multi-objective VM placement approach with three objectives: minimizing total resource wastage, power consumption and thermal dissipation costs. They applied an improved grouping genetic algorithm (GGA) with fuzzy multi-objective evaluation. Their wastage by calculating as differences between the smallest normalized residual resource and the others. They also applied a linear power model to estimate the power consumption [67].They conduct experiments on synthetic data and compare with six traditional approaches including First Fit Decreasing (FFD), Best Fit Decreasing (BFD) and single-objective grouping GA. The results showed the superior performance than other approaches.

Wilcox et al [116] also propose a reordering GGA approach because GGA can effectively avoid redundancy [37]. They use a indirect representation [87] which represents the packing as a sequence. In order to transform the sequence into a packing, they applied an ordering operator which, in essence, is a first fit algorithm. This design naturally avoids infeasible solution, therefore, there is no need for constraint handling.

Wang and Xia [113] develop a MIP algorithm for solving large-scale VM placement problem under a *non-linear* power consumption model. Instead of considering the power consumption as a linear model like most researchers, they consider the CPU frequency can be adjust by dynamic voltage and frequency scaling (DVFS), therefore, the power consumption is a cubical power function of frequency. In order to solve the non-linear problem, they first

use a linear function to approximate the cubical function. Then, they first use the Gruobi MIP solver to solve the relaxed linearized problem. Then, they apply an iterative rounding algorithm to obtain the near optimal solution.

$$\delta = \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{d} (u_j^i - ubest_i)^2} \tag{2.4}$$

Xiong and Xu [120] propose a PSO based approach to solve the problem. Their major contribution is using a total Euclidean distance δ to represent the distance between current resource utilization and the optimal resource utilization (see equation 2.4) where d is the dimension of resources, u_j^i is the current resource utilization of j in a PM i, $ubest_i$ is the predefined optimal resource utilization (e.g 70% CPU utilization). Another contribution is their representation used in PSO. They represent the allocation of each VM to a PM as a probability and let particles search through the indirect solution space.

In summary, most of VM-based placement approaches consider two or three resources (I/O has not been considered in many approaches because they assume that network attached storage (NAS) is used as a main storage along the cluster [77]). After Mishra unreal the principles of vector bin packing, most research apply a balance-measure among resources as their objectives. EC approaches are widely used because they are better performed than traditional heuristics and faster than ILP methods.

2.2.2 Periodic consolidation

Periodic consolidation (see Section 2.1.9) is an process that optimizes the current allocation of resources in a periodic fashion [75]. This is because the cloud datacenter is a dynamic environment with continuous deployment and releases that causes degradation of the resource utilization, thus, the allocation needs to be adjusted when the performance degrades to a certain level. In comparison with application initial placement (see Section 2.2.1), the similarity is that they are both static approaches which consider a batch of applications and PMs. The difference is that periodic consolidation needs to take the cost of application migration into account, therefore, it is often considered as a multi-objective optimization problem.

Although periodic consolidation has been applied in VM-based Cloud for years [43,77], it has not been studied in the new context of container-based Cloud. Therefore, this section will first discuss VM-based models, especially the migration models. Secondly, we will discuss the approaches in periodic consolidation in the VM-context, specifically, in terms of the prediction of workload, these gaps existed in both VM and container context.

Murtazaev and Oh [77], Beloglazov et al [8] and Ferreto et al [43] realize that the migration process generates a large overhead so that it should be used as few as possible. In their migration model, they use the number of migration as the optimization objective. Murtazaev's approach minimize this number by developing an algorithm which always chooses a VM in the least loaded PM and attempts to allocate them on the most loaded PMs. Based on this idea, they develop a heuristic based on First and Best Fit. They select a candidate VM based on a surrogate weight of the resources it used. Beloglazov, on the other hand, considers different criteria for selecting candidate VMs. They not only considers the utilization of VMs but also the utilization of the original PM and target PMs. They also propose a simple heuristic: a modified Best Fit Decreasing to solve the problem. However, these two approaches develop their selection criteria in a greedy fashion which may lead to a local optimal. Ferreto proposes a preprocessing step before the placement algorithm. It first orders the VMs according to their workload variation. Then, it only performs placement on those VMs with the highest variability. These three papers provide some insight that a good

placement algorithm should consider more than the utilization of host and target PMs, but also the variation of workload.

Most previous consolidation approaches [21,40,111] only consider static workload. That is, they use a peak or average workload as a represented value as the consolidation input. In most of cases, this will lead to either low utilization: peak time only account for small proportion of the total time, or more migrations: extra migration are performed on workload changes. Therefore, the consolidation is more than aggressively concentrate workload on as few PM as possible, but also considers the robustness. The robustness is referred to the capability of enduring the variation of workload without make too many changes.

In order to achieve robustness, the workload variation must be taken into account. Bobroff [13] analyzed a large number of traces from real world datacenter. They categorize workloads into three main groups:

- Weak variability.
- Strong variability with weak periodic behavior.
- Strong variability with strong periodic behavior.

Workload with weak variability can be directly packed. The only problem is that their long-term workload can also be changed. For the second type of workload, it is hard or even impossible to predict its behavior. The third type of workload can be predicted. However, it is hard to find the applications with compensated workload patterns.

Meng et al [74] proposed a standard time series technique to extract the deterministic patterns (e.g trends, cycles and seasonality) and irregular fluctuating patterns from workloads' CPU utilization; they assume the periodic behavior of workload will preserve in the future and predict the irregular parts with two approaches: with and without explicit forecast error models. Then, applications are paired according to their negative correlation. They evaluate the workload prediction and application selection with a server consolidation task. They use First Fit to allocate paired applications. During the consolidation, The consolidation results show that they use 45% less PMs for hosting the same number of VMs. Furthermore, their approach is more robust since the variation of workload is considered. However, they only consider two complementary applications at a time.

2.2.3 VM-based Dynamic Consolidation Techniques

Forsman et al [44] propose two distributed migration strategies to balance the load in a system. *The push* strategy is applied on overloaded PM; it attempts to migrate *One* VM at a time to less loaded PMs. *The pull* strategy is applied on underutilized PMs request workload from heavier loaded PMs. Each of the strategy is executed on each PM as an intelligent agent. They share their status with each other through a communication protocol. There are several interesting features of their approach. First, they apply an adaptive high-load threshold (e.g 0.7 of overall CPU utilization) so that it considers the environment changes. Second, they use an EWMA algorithm to reduce the unnecessary migration because EWMA [52] is useful in smoothing out variations in the average load. Third, they applied an entropy to model the load distribution which is also applied in some previous approaches [61, 86]. Their system is agent-based which means large amount of communication may occur between nodes, this would certainly cost extra network resources which are not discussed. Therefore, we expect to design a centralized system, where all nodes are controlled by a controller.

Xiao et al [119] make two contributions, first, they build a quadratic energy model for the energy consumption of PM and a linear model for the energy consumption of migration [68].

Second, they propose an algorithm based on Multiplayer random evolutionary game theory to solve the problem. In their approach, VMs are mapped into players that take part in the evolutionary game. In each iteration, all the players choose their best feasible action, i.e, Migrate to a PM which can minimize the energy consumption. Some players will randomly choose PM to avoid being stuck at a local optimal. Their approach is compared with First Fit, Best Fit Increasing, Best Fit Decreasing, Greedy and Load Balance rule. The solutions show their approach can improve energy consumption greatly, especially in the scenario that the distributions of VMs are very centralized.

2.2.4 Techniques for Bilevel optimization

A number of studies have been conducted on bilevel optimization [25, 33]. Because of its NP-hard nature, approximation algorithms such as Karush-kuhn-Tucker approach [12, 48], branch-and-bound [5] are often applied to solve the problem. Most of these approaches are not applicable when the problem size increases.

As the complexity of the problem, practical problems with bilevel nature are often simplified into a single-level optimization problem which can achieve a satisfactory level instead of optimal. Classic algorithms often fail because of the nature of bilevel problem such as non-linearity, discreteness, no-differentiability, non-convexity etc.

Evolutionary methods have been applied to bilevel optimization problem since 90s. Mathieu et al [72] proposed an genetic algorithm (GA) based approach. It uses a nested strategy - the lower level is optimized with a linear programming method and the upper level apply a GA.

Oduguwa and Roy [79] proposed a co-evolutionary approach for bilevel problems. Two population are co-operated to find the optimal solution, where each population handles a sub-problem.

Wang et al [114] proposed an evolutionary algorithm based approach with a constraint handling technique. Their approach is able to handle non-differentiability at the upper level objective function, but not in constraints and lower level objective function. Later on, Wang proposed an improved version [115] that shows better performance than the previous version.

Particle Swarm Optimization [66] was also used in solving bilevel problems. A recent work is from Sinha et al [96], they propose a bilevel evolutionary algorithm (BLEAQ) works by approximating the optimal solution mapping between the lower level optimal solutions and the upper level variables. BLEAQ was tested on two sets of test problems and the results were compared with WJL [114] and WLD [115]. The results show BLEAQ is much faster than previous approaches. One major drawback of evolutionary algorithms is its high computation cost which limits the problem size from growing bigger.

Chapter 3

Preliminary Work

3.1 Related models

3.1.1 Workload model

Xavier el al [122] develops a *Resource-Allocation-Throughput (RAT)* model for web service allocation. The *RAT model* mainly defines several important variables for an atomic service which represents a software component. Based on this model, firstly, an atomic service's throughput equals its coming rate if the resources of the allocated VM are not exhausted. Secondly, increasing the coming rate will also increase an atomic service's throughput until the allocated resource is exhausted. Thirdly, when the resource is exhausted, the throughput will not increase as request increasing. At this time, the virtual machine reaches its capacity.

3.1.2 Power Model

Shekhar's research [102] is one of the earliest in energy aware consolidation for cloud computing. They conduct experiments of independent applications running in physical machines. They explain that CPU utilization and disk utilization are the key factors affecting the energy consumption. They also find that only consolidating services into the minimum number of physical machines does not necessarily achieve energy saving, because the service performance degradation leads to a longer execution time, which increases the energy consumption.

Bohra [14] develops an energy model to profile the power of a VM. They monitor the sub-components of a VM which includes: CPU, cache, disk, and DRAM and propose a linear model (Eq 3.1). Total power consumption is a linear combination of the power consumption of CPU, cache, DRAM and disk. The parameters α and β are determined based on the observations of machine running CPU and IO intensive jobs.

$$P_{(total)} = \alpha P_{\{CPU,cache\}} + \beta P_{\{DRAM,disk\}} \tag{3.1}$$

Although this model can achieve an average of 93% of accuracy, it is hard to be employed in solving SRAC problem, for the lack of data.

Beloglazov et al. [**?**] propose a comprehensive energy model for energy-aware resource allocation problem (Eq 3.2). P_{max} is the maximum power consumption when a virtual machine is fully utilized; k is the fraction of power consumed by the idle server (i.e. 70%); and u is the CPU utilization. This linear relationship between power consumption and CPU utilization is also observed by [62, 88].

$$P(u) = k \cdot P_{max} + (1 - k) \cdot P_{max} \cdot u \tag{3.2}$$

3.2 Problem Description

We consider the problem as a multi-objective problem with two potentially conflicting objectives, minimizing the overall cost of web services and minimizing the overall energy consumption of the used physical machines.

To solve the SRAC problem, we model an atomic service as its request and requests' coming rate, also known as frequency.

The request of an atomic service is modeled as two critical resources: CPU time $A = \{A_1, A_i, \ldots, A_t\}$ and memory consumption $M = \{M_1, M_i, \ldots, M_t\}$, for each request consumes a A_i amount of CPU time and M_i amount of memory. The coming rate is denoted as $R = \{R_1, R_i, \ldots, R_t\}$. In real world scenario, the size and the number of a request are both variant which are unpredictable, therefore, this is one of the major challenges in Cloud resource allocation. In this paper, we use fixed coming rate extracted from a real world dataset to represent real world service requests.

The cloud data center has a number of available physical machines which are modeled as CPU time $PA = \{PA_1, PA_j, \ldots, PA_p\}$ and memory $PM = \{PM_1, PM_j, \ldots, PM_p\}$. PA_j denotes the CPU capacity of a physical machine and PM_j denotes the size of memory. A physical machine can be partitioned or virtualized into a set of virtual machines; each virtual machine has its CPU time $VA = \{VA_1, VA_n, \ldots, VA_v\}$ and memory $VM = \{VM_1, VM_n, \ldots, VM_v\}$.

The decision variable of service allocation is defined as X_n^i. X_n^i is a binary value (e.g. 0 and 1) denoting whether a service i is allocated on a virtual machine n. The decision variable of virtual machine allocation is defined as Y_j^n. Y_j^n is also binary denoting whether a VM n is allocated on a physical machine j.

In this work, we consider homogeneous physical machine which means physical machines have the same size of CPU time and memory. The utilization of a CPU of a virtual machine is denoted as $U = \{U_1, U_n, \ldots, U_v\}$. The utilization can be calculated by Eq.3.3.

$$U_n = \begin{cases} \frac{\sum_{i=1}^{t} R_i \cdot A_i \cdot X_n^i}{VA_n}, \text{If } \frac{\sum_{i=1}^{t} R_i \cdot A_i \cdot X_n^i}{VA_n} < 1 \\ 1 \qquad\qquad\qquad\qquad\quad , \text{otherwise} \end{cases} \tag{3.3}$$

The cost of a type of virtual machine is denoted as $C = \{C_1, C_n \ldots, C_v\}$.

In order to satisfy the performance requirement, Service providers often define Service Level Agreements (SLAs) to ensure the service quality. In this work, we define throughput as a SLA measurement [82]. Throughput denotes the number of requests that a service could successfully process in a period of time. According to *RAT* model, the throughput is equal to the number of requests when the allocated resource is sufficient. Therefore, if a VM reaches its utilization limitation, it means that the services have been allocated exceedingly. Therefore, all services in that VM suffer from performance degradation.

Then we define two objective functions as the total energy consumption and the total cost of virtual machines:

minimize

$$Energy = \sum_{j=1}^{p} (k \cdot V_{max} + (1-k) \cdot V_{max} \cdot \sum_{n=1}^{v} U_n \cdot Y_j^n) \tag{3.4}$$

$$Cost = \sum_{j=1}^{p} \sum_{n=1}^{v} C_n \cdot Y_j^n \tag{3.5}$$

Hard constraint

A virtual machine can be allocated on a physical machine if and only if the physical machine has enough available capacity on every resource.

$$\begin{aligned} \sum_{n=1}^{v} VM_n \cdot Y_j^n \leq PM_j \\ \sum_{n=1}^{v} VA_n \cdot Y_j^n \leq PA_j \end{aligned} \tag{3.6}$$

Soft constraint

A service can be allocated on a virtual machine even if the virtual machine does not have enough available capacity on every resource, but the allocated services will suffer from a quality degradation.

$$\sum_{i=1}^{t} M_i \cdot R_i \cdot X_i^n \leq VM_n \tag{3.7}$$

3.3 Methods

As we have discussed, Multi-objective Evolutionary Algorithms are good at solving multi-objective problems and NSGA-II [31] has shown his effective and efficiency. NSGA-II is a well-known MOEA that has been widely used in many real-world optimization problems. In this paper we also adopt NSGA-II to solve the SRAC problem. We first propose a representation and then present a NSGA-II based algorithm with novel genetic operators.

3.3.1 Chromosome Representation

SRAC is a two-level bin-packing problem, in the first level, bins represent physical machines and items represent virtual machines. Whereas, in the second level, a virtual machine acts like a bin and web services are items. Therefore, we design the representation in two hierarchies, virtual machine level and physical machine level.

Figure 3.1 shows an example individual which contains seven service allocations. Each allocation of a service is represented as a pair where the index of each pair represents the number of web service. The first number indicates the type of virtual machine that the service is allocated in. The second number denotes the number of virtual machine. For example, in Figure 3.1, service #1 and service #2 are both allocated in the virtual machine #1 while service #1 and service #5 are allocated to different virtual machines sharing the same type. The first hierarchy shows the virtual machine in which a service is allocated by defining VM type and number. Note that, the VM type and number are correlated once they are initialized. With this feature, the search procedure is narrowed down in the range of existing VMs which largely shrinks the search space. The second hierarchy shows the relationship between a physical machine and its virtual machines, which are implicit. The physical machine is dynamically determined according to the virtual machines allocated on it. For example, in Figure 3.1, the virtual machines are sequentially packed into physical machines. The boundaries of PMs are calculated by adding up the resources of VMs until one of the resources researches the capacity of a PM. At the moment, no more VMs can be packed into the PM, then the boundary is determined. The reason we designed this heuristic is because a physical machine is always fully used before launching another. Therefore, VM consolidation is inherently achieved.

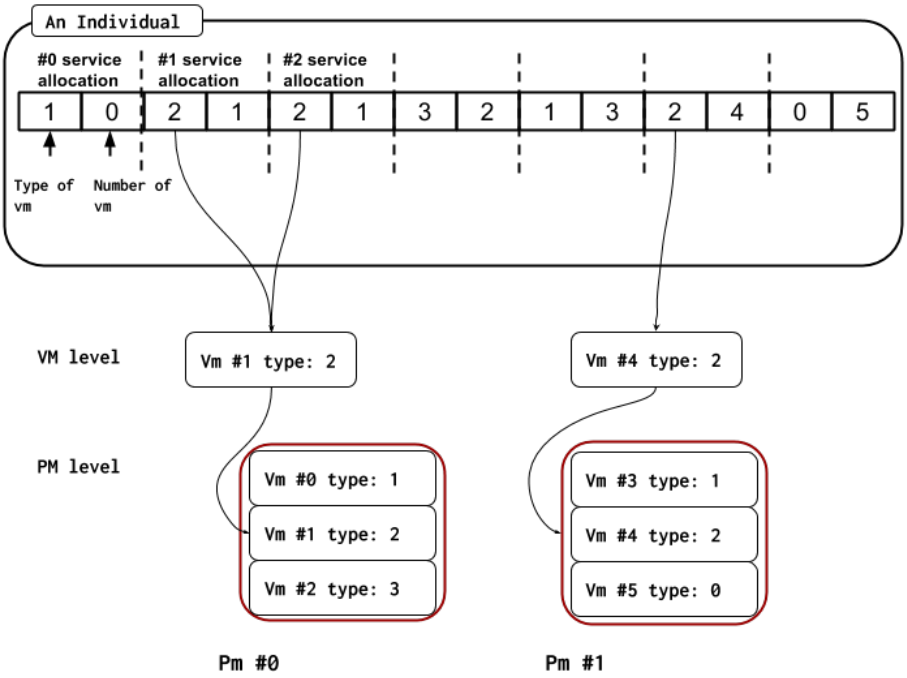


Figure 3.1: An example chromosome representation

Clearly, specifically designed operators are needed to manipulate chromosomes. Therefore, based on this representation, we further developed initialization, mutation, constraint handling and selection method.

3.3.2 Initialization

Algorithm 1 Initialization

Inputs:
VM CPU Time VA and memory VM,
Service CPU Time A and memory M
consolidation factor c
Outputs: A population of allocation of services

for Each service t **do**
  Find its most suitable VM Type
  Randomly generate a VM type $vmType$ which is equal or better than its most suitable type
  if There are existing VMs with $vmType$ **then**
    randomly generate a number u
    if $u <$ consolidation factor **then**
      randomly choose one existing VM with $vmType$ to allocate
    else
      launch a new VM with $vmType$
    end if
  else
    Create a new VM with its most suitable VM type
  end if
end for

The initialization (see Alg 1) is designed to generate a diverse population. In the first step, for each service, it is able to find the most suitable VM type which is just capable of running the service based on its resource requirements. In the second step, based on the

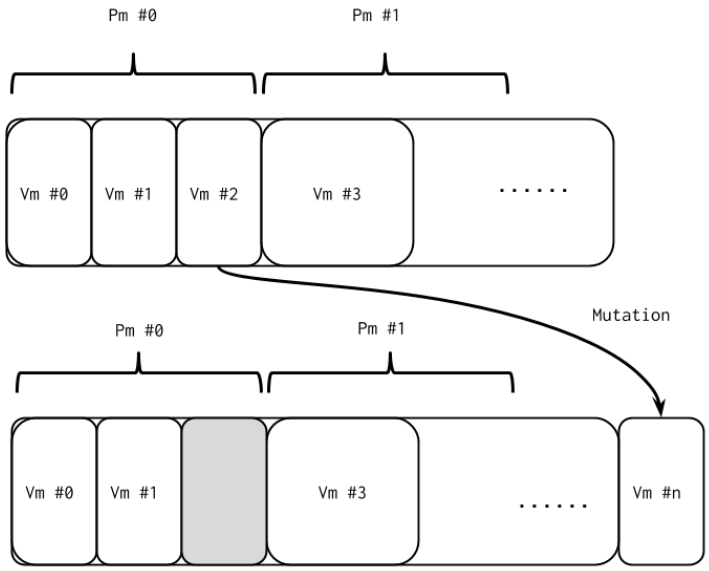


Figure 3.2: An example mutation without insertion that causes a lower resource utilization

suitable VM type, a stronger type is randomly generated. If there exists a VM with that type, the service is either deployed in the existing VM or launch a new VM. We design a consolidation factor c which is a real number manually selected from 0 to 1 to control this selection. If a random number u is smaller than c, the service is consolidated in an existing VM.

This design could adjust the consolidation, therefore, controls the utilization of VM.

3.3.3 Mutation

The design principle for mutation operator is to enable individuals exploring the entire feasible search space. Therefore, a good mutation operator has two significant features, the exploration ability and the its ability to keep an individual within the feasible regions. In order to achieve these two goals, firstly, we generate a random virtual machine type which has a greater capacity than the service needs. It ensures the feasible of solutions as well as exploration capability. Then, we consider whether a service is consolidated with the consolidation factor c.

The consolidation is conducted with a roulette wheel method which assigns fitness value to each VM according to the reciprocal of its current utilization. The higher the utilization, the lower the fitness value it is assigned. Therefore, a lower utilization VM has a greater probability to be chosen. At last, if a new VM is launched, it will not be placed at the end of VM lists. Instead, it will be placed at a random position among the VMs. The reason is illustrated in Figure 3.2. In the example, VM #2 is mutated into a new type and be placed at the end of the VM list. However, because of the size of VM #3 is too large for PM #0, the hollow in PM #0 will never be filled. This problem can be solved with the random insertion method.

3.3.4 Violation control method

A modified violation ranking is proposed to deal with the soft constraint, for the hard constraint is automatically eliminated by the chromosome representation. We define a violation

Algorithm 2 Mutation

Inputs:
An individual VM CPU Time VA and memory VM,
Service CPU Time A and memory M
consolidation factor c
Outputs: A mutated individual

```
for Each service do
    Randomly generate a number u
    if u < mutation rate then
        find the most suitable VM Type for this service
        Randomly generate a number k
        if k < consolidation factor then
            calculate the utilization of used VMs
            assign each VM with a fitness value of 1 / utilization and generate a roulette wheel according to
            their fitness values
            Randomly generate a number p, select the VM according to p
            Allocate the service
        else
            launch a new VM with the most suitable VM Type
            insert the new VM in a randomly choose position
        end if
    end if
end for
```

number as the number of services which are allocated in the degraded VMs. That is, if there are excessive services allocated in a VM, then all the services are suffered from a degraded in performance. The violation number is used in the selection procedure, where the individuals with less violations are always preferred.

3.3.5 Selection

Our design uses the binary tournament selection with a constrained-domination principle. A constrained-domination principle is defined as following. A solution I is considered constraint-dominate a solution J, if any of the following condition is true:

1. Solution I is feasible, solution is not,
2. Both solutions are infeasible, I has smaller overall violations,
3. Both solutions are feasible, solution I dominates solution J.

An individual with no or less violation is always selected. This method has been proved effective in the original NSGA-II paper [31].

3.3.6 Fitness Function

The cost fitness (Eq.3.5) is determined by the type of VMs at which web service are allocated. The energy fitness is shown in Eq.3.4, the utilizations (Eq.3.3) of VM are firstly converted into the utilizations of PM according to the proportion of VMs and PMs CPU capacity.

3.3.7 Algorithm

The main difference between our approach and the original NSGA-II is that our approach has no crossover operator.

That is, a random switch of chromosome would completely destroy the order of VMs, hence, no useful information will be preserved. Therefore, we only apply mutation as the

Table 3.1: Problem Settings

Problem	1	2	3	4	5	6
Number of services	20	40	60	80	100	200

exploration method. Then, the algorithm becomes a parallel optimization without much interaction between its offspring, which is often addressed as Evolutionary Strategy [63].

Algorithm 3 NSGA-II for SRAC

Inputs:
VM CPU Time VA and memory VM,
PM CPU Time PA and memory PM,
Service CPU Time A and memory M
consolidation factor c
Outputs: A Non-dominated Set of solutions

```
Initialize a population P
while Termination Condition is not meet do
    for Each individual do
        Evaluate the fitness values
        Calculate the violation
    end for
    non-Dominated Sorting of P
    calculate crowding distance
    while child number is less than population size do
        Selection
        Mutation
        add the child in a new population U
    end while
    Combine P and U { for elitism}
    Evaluate the combined P and U
    Non-dominated sorting and crowding distance for combined population
    Include the top popSize ranking individuals to the next generation
end while
```

3.4 Experiment

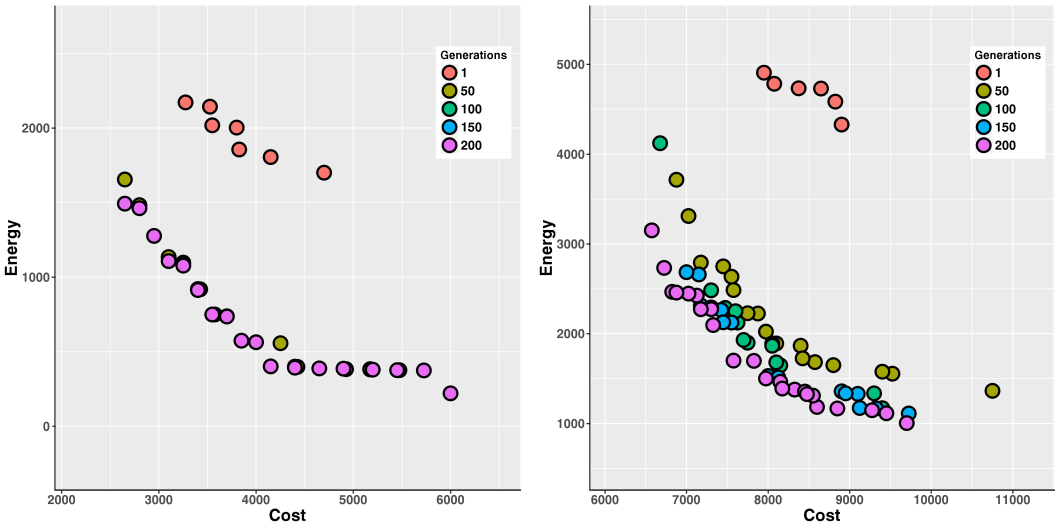
3.4.1 Dataset and Problem Design

This project is based on both real-world datasets *WS-Dream* [126] and simulated datasets [15]. The *WS-Dream* contains web service related datasets including network latency and service frequency (request coming rate). In this project, we mainly use the service frequency matrix. For the cost model, we only consider the rental of virtual machines with fixed fees (monthly rent). The configurations of VMs are shown in Table 3.2, the CPU time and memory were selected manually and cost were selected proportional to their CPU capacity. The maximum PM's CPU and memory are set to 3000 and 8000 respectively. The energy consumption is set to 220W according to [15].

We designed six problems shown in Table 3.1, listed with increasing size and difficulty, which are used as representative samples of SRAC problem.

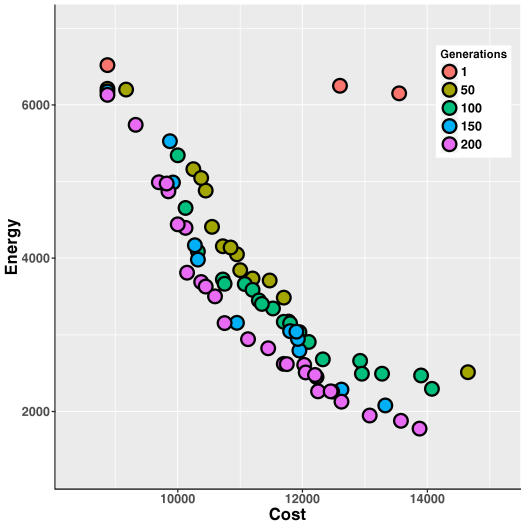
Selection Method with violation Control vs. without violation control

We conducted two comparison experiments. For the first experiment, we make a comparison between NSGA-II with violation control and NSGA-II without violation control. In second experiment, two mutation operators are compared. The first is the roulette wheel

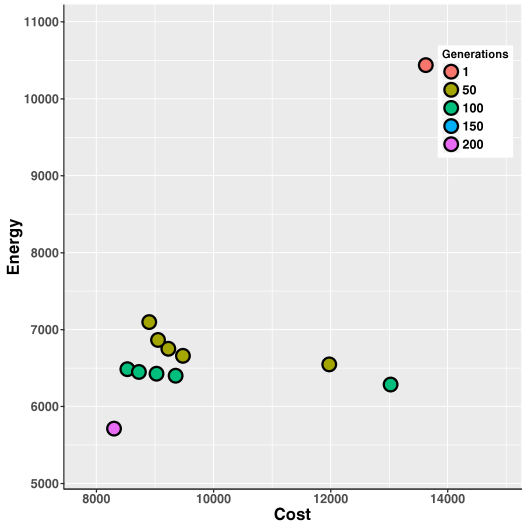


(a) Problem 1

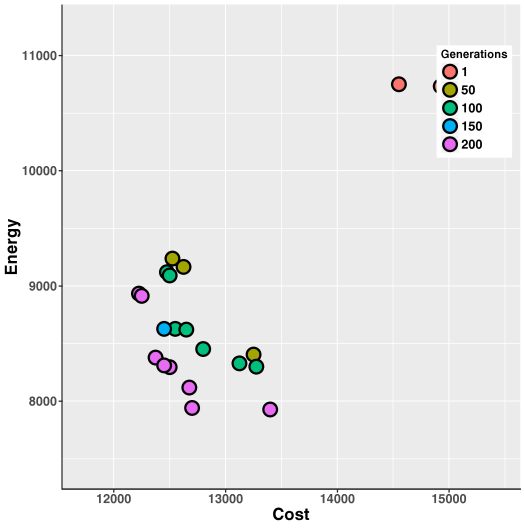
(b) Problem 2



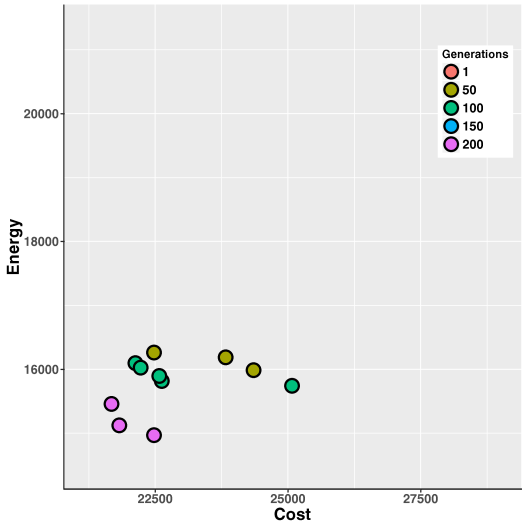
(c) Problem 3



(d) Problem 4



(e) Problem 5



(f) Problem 6

Figure 3.3: Non-dominated solutions evolve along with the generation

Table 3.2: VM configurations

VM Type	CPU Time	Memory	Cost
1	250	500	25
2	500	1000	50
3	1500	2500	150
4	3000	4000	300

mutation, the second is the mutation with greedy algorithm. The mutation with greedy algorithm is a variant of roulette wheel mutation. The only difference is that instead of selecting a VM to consolidate with fitness values, it always selects the VM with the lowest CPU utilization. Therefore, it is a greedy method embedded in the mutation.

The experiments were conducted on a personal laptop with 2.3GHz CPU and 8.0 GB RAM. For each approach, 30 independent runs are performed for each problem with constant population size 100. The maximum number of iteration is 200. k equals 0.7. We set mutation rate and consolidation factor to 0.9 and 0.01.

3.4.2 Results

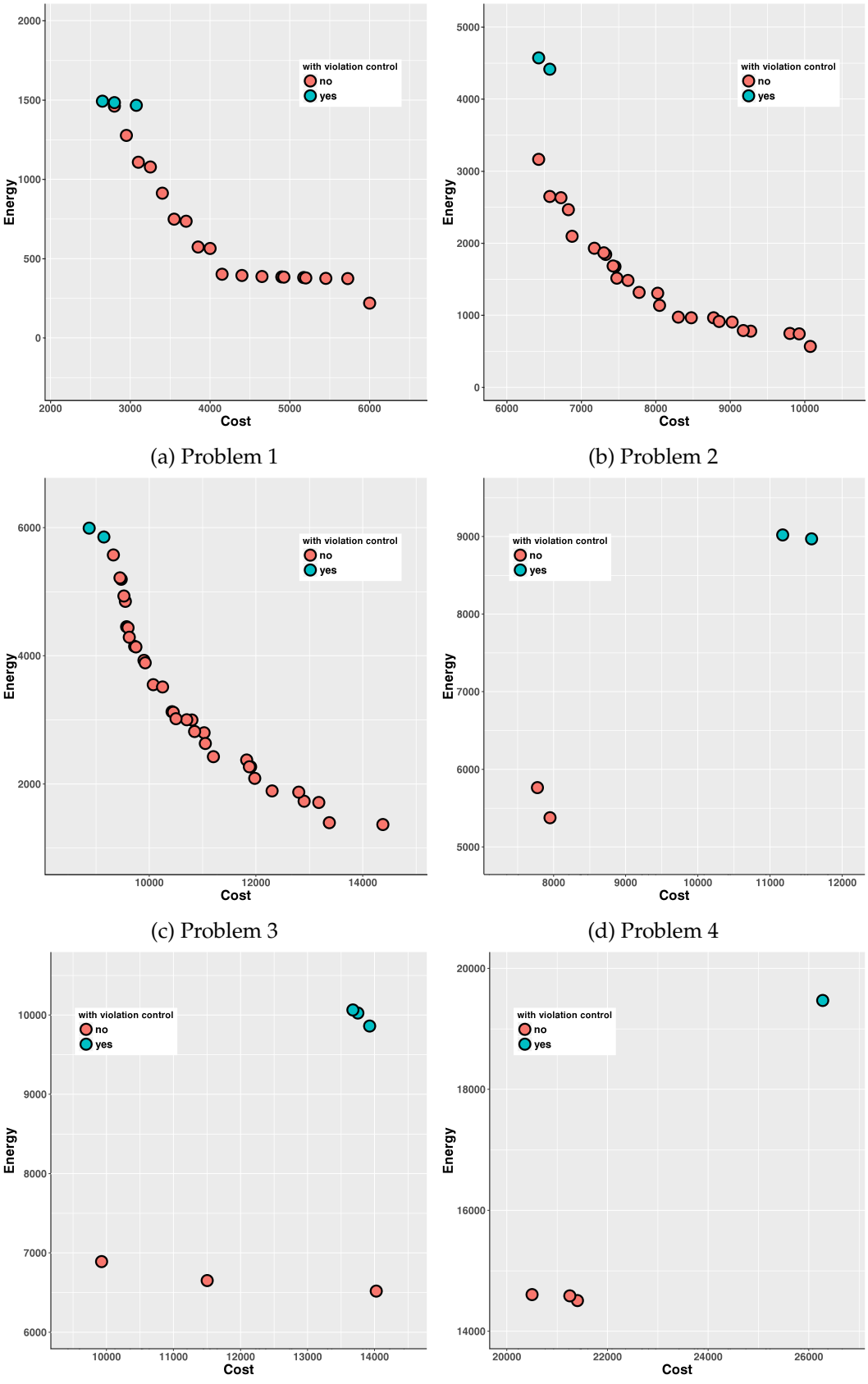
Table 3.3: Comparison between two Mutation methods

Problem	roulette wheel mutation		Greedy mutation	
	cost fitness	energy fitness	cost fitness	energy fitness
1	2664.6 ± 66.4	1652.42 ± 18.2	2661.7 ± 56.9	1653.2 ± 18.2
2	6501.1 ± 130.2	4614.0 ± 110.7	6495.37 ± 110.7	4132.5 ± 80.4
3	8939.2 ± 118.5	6140.7 ± 204.0	9020.5 ± 204.0	5739.6 ± 148.6
4	11633.7 ± 301.1	9301.9 ± 254.0	12900.6 ± 243.0	9376.3 ± 120.9
5	14102.0 ± 231.7	10164.8 ± 238.9	14789.2 ± 238.8	9876.3 ± 120.9
6	27194.3 ± 243.0	19914.4 ± 307.5	27654.2 ± 307.5	19187.1 ± 176.6

As we conducted the experiment for 30 runs, we first obtain an average non-dominated set over 30 runs by collecting the results from a specific generation from all 30 runs, and then apply a non-dominated sorting over them.

Firstly, we show the non-dominated solutions evolve along with the evolution process in Figure 3.3. These results come from selection method without violation control. As it illustrated, different colors represent different generations from 0th to 200th. For problem 1, because the problem size is small, the algorithm converged before 100 generations. Therefore, the non-dominated set from the 100th and 150th generations are overlapping with results from the 200th generation. For problem 2 and problem 3, it clearly shows the improvement of fitness values. For problem 4 onwards, the algorithm can only obtain a few solutions as the problem size is large, it is difficult to find solutions.

Then, the non-dominated sets of the last generation from two selection methods are compared in Figure 3.4. There are much fewer results are obtained from the violation control method throughout all cases. For the first three problems, the non-dominated set from the violation control method has similar quality as the no violation control method. From problem 4 onwards, the results from selection with violation control are much worse in terms of fitness values. However, most of the results from non-violation control selection have a high



(a) Problem 1

(b) Problem 2

(c) Problem 3

(d) Problem 4

(e) Problem 5

(f) Problem 6

Figure 3.4: non-dominated solutions comparison between selection with violation control and without violation control

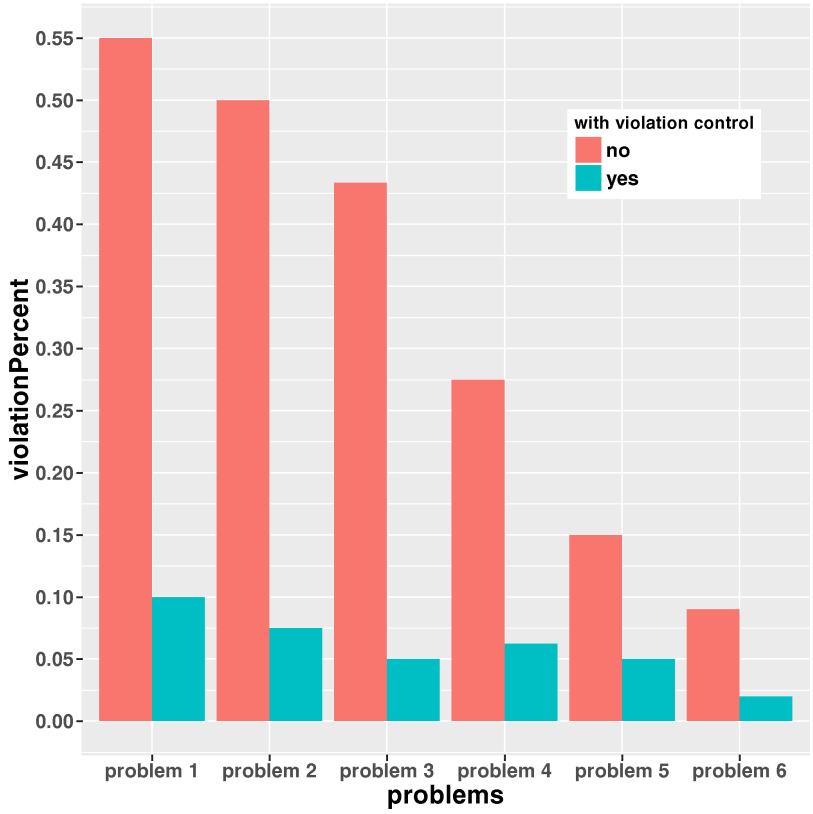


Figure 3.5: Violation Percentage comparison between selection with violation control and without violation control

violation rate. That is, the method without violation control is stuck in the infeasible regions and provide high-violation rate solutions.

From figure 3.5, we can observe the violation rate between two methods. It proves violation control has a great ability to prevent the individual from searching the infeasible region. On the other hand, without violation control, although, the algorithm can provide more solutions with better fitness values, most of them have a high violation rate over 10% which are not very useful in reality.

As we mentioned in previous section, the mutation rate and consolidation factor are set differently for the two methods. For the method with violation control, the mutation rate is set to 0.9 and the consolidation factor c is set to 0.01, this is because the feasible region is narrow and scattered. In order to avoid stucking in the local optima, a large mutation rate can help escape local optima. For the factor c, a larger percentage would easily lead the algorithm to infeasible regions. Therefore, it is set to a small number.

Mutation with roulette wheel vs. Mutation with greedy algorithm

Table 3.3 shows the fitness value comparison between mutation methods. According to statistics significant test, there is little difference between methods. The possible reason is the consolidation factor is set to 0.01. In each mutation iteration, there is only 1% probability that a service will be consolidated in an existed VM, therefore, the influence between different consolidation strategies is trivial.

3.5 Conclusion

In this paper, we first propose a multi-objective formulation of a two levels of bin packing problem, web service resource allocation in Cloud. It solves the resource allocation in IaaS and SaaS at the same time. Two objectives, minimizing the cost from service providers' perspective and minimizing the energy consumption from cloud provider's objective are achieved. Secondly, we propose a NSGA-II based algorithm with specific designed genetic operators to solve the problem. The results are compared with different variances of the algorithm. The results show our approach can solve the very complicate optimization problem.

With current work as a baseline, in future work, we could improve the quality of solutions as well as provide better violation control mechanisms.

Chapter 4

Proposed Contributions and Project Plan

This thesis will contribute to the field of Cloud Computing by proposing novel solutions for the joint allocation of container and VM, and to the field of Evolutionary Computation by proposing new representations and genetic operators in evolutionary algorithms. The proposed contributions of this project are listed below:

1. Propose a new model for the joint allocation of container and VM problem. New representations will be proposed to problem. The first EC-approach to solve the problem. Reduce the energy consumption of data centers.
2. Propose a new robustness measure for time-dependent server consolidation problem. New optimization algorithms not only consider the two-level of packing problem but also take the previous and next consolidation into considered.
3. Develp a GP-based hyper-heuristic approach to automatically generate dispatching rules for dynamic server consolidation problem. New functional and primitive set for constructing useful dispatching rules will be studied. New genetic operations and representations will be proposed.
4. Develop a preprocessing algorithm to reduce the number of variables in a large scale static consolidation problem without scacrificing too much performance. Useful features of server consolidation problem will be studied.

4.1 Overview of Project Plan

Six overall phases have been defined in the initial research plan for this PhD project, as shown in Figure **??**. The first phase, which comprises reviewing the relevant literature, investigating both VM-based and container-based server consolidation algorithms, and producing the proposal, has been completed. The second phase, which corresponds to the first objective of the thesis, is currently in progress and is expected to be finished on time, thus allowing the remaining phases to also be carried out as planned.

4.2 Project Timeline

The phases included in the plan above are estimated to be completed following the timeline shown in Figure **??**, which will serve as a guide throughout this project. Note that part

of the first phase has already been done, therefore the timeline only shows the estimated remaining time for its full completion.

4.3 Thesis Outline

The completed thesis will be organised into the following chapters:

- *Chapter 1: Introduction*
  This chapter will introduce the thesis, providing a problem statement and motivations, defining research goals and contributions, and outlining the structure of this dissertation.

- *Chapter 2: Literature Review*
  The literature review will examine in the existing work on VM-based and container-based server consolidation, discussing the fundamental concepts in this field in order to provide the reader with the necessary background. Multiple sections will then follow, considering issues such as static consolidation, dynamic consolidation, and large-scale of consolidation problem. The focus of this review is on investigating server consolidation techniques that are based on Evolutionary Computation.

- *Chapter 3: An EC Approach to the joint Allocation of Container and Virtual Machine*
  This chapter will establish a new model for the joint allocation of container and virtual machine problem. Furthermore, this chapter will introduce a new bi-level approach to solve this problem. One of the critical aspects of this approach is the representation of solution. Therefore, multiple representations will be proposed, analysed and compared.

- *Chapter 4: An EC Approach to the Time-dependent Global Server Consolidation*
  In this chapter, a robustness measure of time-dependent server consolidation will be proposed. Furthermore, a time-dependent optimization techniques will be employed to solve this problem. In the first, it will only consider the previous allocation results to minimize both cost of migration as well as the overall energy consumption of data center. Then, this approach will be generalized to consider next consolidation.

- *Chpater 5: A Genetic Programming-based Hybrid-heuristic Approach to Dynamic Consolidation*
  This chapter focuses on providing a Genetic Programming-based hybrid heuristic approach to automatic generate dispatching rules to dynamic consolidation problem.

- *Chapter 6: A Preprocessing Algorithm for Large-scale Static Consolidation*
  This chapter proposes a preprocessing algorithm for large scale static consolidation to reduce the number of variables so that the static consolidation can be more scalable.

- *Chapter 7: Conclusions and Future Work* In this chapter, conclusions will be drawn from the analysis and experiments conducted in the different phases of this research, and the main findings for each one of them will be summarised. Additionally, future research directions will be discussed.

4.4 Resources Required

4.4.1 Computing Resources

An experimental approach will be adopted in this research, entailing the execution of experiments that are likely to be computationally expensive. The ECS Grid computing facilities can be used to complete these experiments within reasonable time frames, thus meeting this requirement.

4.4.2 Library Resources

The majority of the material relevant to this research can be found online, using the universitys electronic resources. Other works may either be acquired at the universitys library, or by soliciting assistance from the Subject Librarian for the fields of engineering and computer science.

4.4.3 Conference Travel Grants

Publications to relevant venues in this field are expected throughout this project, therefore travel grants from the university are required for key conferences.

Bibliography

[1] AL-ROOMI, M., AND AL-EBRAHIM, S. Cloud computing pricing models: a survey. *Computing* (2013).

[2] ANGELO, J. S., KREMPSER, E., AND BARBOSA, H. J. C. Differential evolution for bilevel programming. In *2013 IEEE Congress on Evolutionary Computation (CEC)* (2013), IEEE, pp. 470–477.

[3] BÄCK, T., HAMMEL, U., AND SCHWEFEL, H.-P. Evolutionary computation - comments on the history and current state. *IEEE Trans. Evolutionary Computation 1*, 1 (1997), 3–17.

[4] BANZHAF, W., NORDIN, P., KELLER, R. E., AND FRANCONE, F. D. Genetic programming: an introduction, 1998.

[5] BARD, J. F., AND FALK, J. E. An explicit solution to the multi-level programming problem. *Computers & Operations Research 9*, 1 (Jan. 1982), 77–100.

[6] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T. L., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. *SOSP* (2003), 164.

[7] BARROSO, L. A., AND HÖLZLE, U. The Case for Energy-Proportional Computing. *IEEE Computer 40*, 12 (2007), 33–37.

[8] BELOGLAZOV, A., ABAWAJY, J. H., AND BUYYA, R. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Comp. Syst. 28*, 5 (2012), 755–768.

[9] BELOGLAZOV, A., AND BUYYA, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurrency and Computation - Practice and Experience 24*, 13 (2012), 1397–1420.

[10] BELOGLAZOV, A., AND BUYYA, R. Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under Quality of Service Constraints. *IEEE Transactions on Parallel and Distributed Systems 24*, 7 (2013), 1366–1379.

[11] BERNSTEIN, D. Containers and Cloud - From LXC to Docker to Kubernetes. *IEEE Cloud Computing* (2014).

[12] BIANCO, L., CARAMIA, M., AND GIORDANI, S. A bilevel flow model for hazmat transportation network design. *Transportation Research Part C: ... 17*, 2 (Apr. 2009), 175–196.

[13] BOBROFF, N., KOCHUT, A., AND BEATY, K. A. Dynamic Placement of Virtual Machines for Managing SLA Violations. *Integrated Network Management* (2007), 119–128.

[14] BOHRA, A. E. H., AND CHAUDHARY, V. VMeter: Power modelling for virtualized clouds. In *Parallel {& Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on (2010), Ieee, pp. 1–8.*

[15] BORGETTO, D., CASANOVA, H., DA COSTA, G., AND PIERSON, J.-M. Energy-aware service allocation. *Future Generation Computer Systems 28*, 5 (2012), 769–779.

[16] BORODIN, A., AND EL, R. Yaniv (1998): Online Computation and Competitive Analysis.

[17] BROTCORNE, L., LABBÉ, M., MARCOTTE, P., AND SAVARD, G. A Bilevel Model for Toll Optimization on a Multicommodity Transportation Network. *Transportation Science 35*, 4 (Nov. 2001), 345–358.

[18] BURKE, E. K., HYDE, M. R., AND KENDALL, G. Evolving Bin Packing Heuristics with Genetic Programming. *PPSN 4193*, Chapter 87 (2006), 860–869.

[19] BURKE, E. K., HYDE, M. R., KENDALL, G., AND WOODWARD, J. Automating the Packing Heuristic Design Process with Genetic Programming. *Evolutionary Computation 20*, 1 (Mar. 2012), 63–89.

[20] CHAISIRI, S., LEE, B.-S., AND NIYATO, D. Optimization of Resource Provisioning Cost in Cloud Computing. *IEEE Trans. Services Computing 5*, 2 (2012), 164–177.

[21] CHEN, M., ZHANG, H., SU, Y.-Y., WANG, X., JIANG, G., AND YOSHIHIRA, K. Effective VM sizing in virtualized data centers. In *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)* (2011), IEEE, pp. 594–601.

[22] CHO, J., AND KIM, Y. Improving energy efficiency of dedicated cooling system and its contribution towards meeting an energy-optimized data center. *Applied Energy 165* (Mar. 2016), 967–982.

[23] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. 273–286.

[24] COFFMAN JR, E. G., GAREY, M. R., AND JOHNSON, D. S. *Approximation algorithms for bin packing: a survey*. PWS Publishing Co., Aug. 1996.

[25] COLSON, B., MARCOTTE, P., AND SAVARD, G. An overview of bilevel optimization. *Annals OR 153*, 1 (2007), 235–256.

[26] CONSTANTIN, I. Optimizing frequencies in a transit network: a nonlinear bi-level programming approach. *International Transactions in Operational Research 2*, 2 (Apr. 1995), 149–164.

[27] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A Hyperheuristic Approach to Scheduling a Sales Summit. In *Practice and Theory of Automated Timetabling III*. Springer, Berlin, Heidelberg, Berlin, Heidelberg, Aug. 2000, pp. 176–190.

[28] DAYARATHNA, M., WEN, Y., AND FAN, R. Data Center Energy Consumption Modeling - A Survey. ... *Surveys & Tutorials* (2016).

[29] DE JONG, K. A. Are Genetic Algorithms Function Optimizers? *PPSN* (1992).

[30] DE JONG, K. A. Genetic Algorithms are NOT Function Optimizers. *FOGA* (1992).

[31] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6*, 2 (Apr. 2002), 182–197.

[32] DEB, K., AND SINHA, A. An Efficient and Accurate Solution Methodology for Bilevel Multi-Objective Programming Problems Using a Hybrid Evolutionary-Local-Search Algorithm. *dx.doi.org 18*, 3 (Aug. 2010), 403–449.

[33] DEMPE, S., DUTTA, J., AND LOHSE, S. Optimality conditions for bilevel programming problems. *... with Multivalued Mappings 55*, 5-6 (Oct. 2006), 505–524.

[34] DUA, R., RAJA, A. R., AND KAKADIA, D. Virtualization vs Containerization to Support PaaS. In *2014 IEEE International Conference on Cloud Engineering (IC2E)* (2014), IEEE, pp. 610–614.

[35] EBERHART, R. C., AND KENNEDY, J. *Particle swarm optimization, proceeding of IEEE International Conference on Neural Network*. Perth, 1995.

[36] ESPOSITO, C., CASTIGLIONE, A., AND CHOO, K.-K. R. Challenges in Delivering Software in the Cloud as Microservices. *IEEE Cloud Computing 3*, 5 (2016), 10–14.

[37] FALKENAUER, E. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics 2*, 1 (1996), 5–30.

[38] FAN, X., WEBER, W.-D., AND BARROSO, L. A. Power provisioning for a warehouse-sized computer. *ISCA* (2007), 13.

[39] FEITELSON, D. G. Workload Modeling for Performance Evaluation. *Performance 2459*, Chapter 6 (2002), 114–141.

[40] FELLER, E., RILLING, L., AND MORIN, C. Energy-Aware Ant Colony Based Workload Placement in Clouds. *GRID* (2011).

[41] FELTER, W., FERREIRA, A., RAJAMONY, R., AND RUBIO, J. An updated performance comparison of virtual machines and Linux containers. *ISPASS* (2015), 171–172.

[42] FERDAUS, M. H., MURSHED, M. M., CALHEIROS, R. N., AND BUYYA, R. Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic. *Euro-Par 8632*, Chapter 26 (2014), 306–317.

[43] FERRETO, T. C., NETTO, M. A. S., CALHEIROS, R. N., AND DE ROSE, C. A. F. Server consolidation with migration control for virtualized data centers. *Future Generation Computer Systems 27*, 8 (Oct. 2011), 1027–1034.

[44] FORSMAN, M., GLAD, A., LUNDBERG, L., AND ILIE, D. Algorithms for automated live migration of virtual machines. *Journal of Systems and Software 101* (2015), 110–126.

[45] GANESAN, R., SARKAR, S., AND NARAYAN, A. Analysis of SaaS Business Platform Workloads for Sizing and Collocation. *IEEE CLOUD* (2012), 868–875.

[46] GAO, Y., GUAN, H., QI, Z., HOU, Y., AND LIU, L. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *J. Comput. Syst. Sci. 79*, 8 (2013), 1230–1242.

[47] HAMEED, A., KHOSHKBARFOROUSHHA, A., RANJAN, R., JAYARAMAN, P. P., KOLODZIEJ, J., BALAJI, P., ZEADALLY, S., MALLUHI, Q. M., TZIRITAS, N., VISHNU, A., KHAN, S. U., AND ZOMAYA, A. Y. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing 98*, 7 (2016), 751–774.

[48] HERSKOVITS, J., LEONTIEV, A., DIAS, G., AND SANTOS, G. Contact shape optimization: a bilevel programming approach. *Structural and Multidisciplinary Optimization 20*, 3 (2000), 214–221.

[49] HINDMAN, B., KONWINSKI, A., ZAHARIA, M., GHODSI, A., JOSEPH, A. D., KATZ, R. H., SHENKER, S., AND STOICA, I. Mesos - A Platform for Fine-Grained Resource Sharing in the Data Center. *NSDI* (2011).

[50] HINES, M. R., DESHPANDE, U., AND GOPALAN, K. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev. 43*, 3 (July 2009), 14–26.

[51] HOLLAND, J. H. Outline for a Logical Theory of Adaptive Systems. *J. ACM 9*, 3 (1962), 297–314.

[52] HOLT, C. C. Author's retrospective on 'Forecasting seasonals and trends by exponentially weighted moving averages'. *International journal of forecasting 20*, 1 (Jan. 2004), 11–13.

[53] JENNINGS, B., AND STADLER, R. Resource Management in Clouds: Survey and Research Challenges. *Journal of Network and Systems Management 23*, 3 (2015), 567–619.

[54] JOHNSON, D. S. Vector Bin Packing.

[55] KAPLAN, J. M., FORREST, W., AND KINDLER, N. Revolutionizing data center energy efficiency, 2008.

[56] KENNEDY, J., AND EBERHART, R. C. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation* (1997), IEEE, pp. 4104–4108.

[57] KHANNA, G., BEATY, K. A., KAR, G., AND KOCHUT, A. Application Performance Management in Virtualized Server Environments. *NOMS* (2006).

[58] KIRJNER-NETO, C., POLAK, E., AND DER KIUREGHIAN, A. An Outer Approximations Approach to Reliability-Based Optimal Design of Structures. *Journal of Optimization Theory and Applications 98*, 1 (1998), 1–16.

[59] KIVITY, A., KAMAY, Y., LAOR, D., AND LUBLIN, U. kvm: the Linux virtual machine monitor. *... of the Linux ...* (2007).

[60] KOZA, J. R. Genetic programming - on the programming of computers by means of natural selection. *Complex adaptive systems* (1993).

[61] KUNKLE, D., AND SCHINDLER, J. A Load Balancing Framework for Clustered Storage Systems. *HiPC 5374*, 1 (2008), 57–72.

[62] KUSIC, D., KEPHART, J. O., HANSON, J. E., KANDASAMY, N., AND JIANG, G. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing 12*, 1 (2009), 1–15.

[63] LEE, K. Y., AND YANG, F. F. Optimal reactive power planning using evolutionary algorithms: A comparative study for evolutionary programming, evolutionary strategy, genetic algorithm, and linear programming. *IEEE Transactions on power systems 13*, 1 (1998), 101–108.

[64] LEGILLON, F., LIEFOOGHE, A., AND TALBI, E.-G. CoBRA - A cooperative coevolutionary algorithm for bi-level optimization. *IEEE Congress on Evolutionary Computation* (2012), 1–8.

[65] LEINBERGER, W., KARYPIS, G., AND KUMAR, V. Multi-Capacity Bin Packing Algorithms with Applications to Job Scheduling under Multiple Constraints. *ICPP* (1999), 404–412.

[66] LI, X., TIAN, P., AND MIN, X. A Hierarchical Particle Swarm Optimization for Solving Bilevel Programming Problems. In *Service-Oriented and Cloud Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 1169–1178.

[67] LIEN, C.-H., BAI, Y.-W., AND LIN, M.-B. Estimation by Software for the Power Consumption of Streaming-Media Servers. *IEEE Trans. Instrumentation and Measurement 56*, 5 (2007), 1859–1870.

[68] LIU, H., JIN, H., XU, C.-Z., AND LIAO, X. Performance and energy modeling for live migration of virtual machines. *Cluster computing 16*, 2 (2013), 249–264.

[69] MANN, Z. Á. Approximability of virtual machine allocation: much harder than bin packing.

[70] MANN, Z. Á. Interplay of Virtual Machine Selection and Virtual Machine Placement. In *Service-Oriented and Cloud Computing*. Springer International Publishing, Cham, Aug. 2016, pp. 137–151.

[71] MANVI, S. S., AND SHYAM, G. K. Resource management for Infrastructure as a Service (IaaS) in cloud computing - A survey. *J. Network and Computer Applications 41* (2014), 424–440.

[72] MATHIEU, R., PITTARD, L., AND ANANDALINGAM, G. Genetic algorithm based approach to bi-level linear programming. *RAIRO-Operations Research* (1994).

[73] MELL, P. M., AND GRANCE, T. The NIST definition of cloud computing. Tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, Gaithersburg, MD, 2011.

[74] MENG, X., ISCI, C., KEPHART, J. O., 0002, L. Z., BOUILLET, E., AND PENDARAKIS, D. E. Efficient resource provisioning in compute clouds via VM multiplexing. *ICAC* (2010), 11.

[75] MISHRA, M., DAS, A., KULKARNI, P., AND SAHOO, A. Dynamic resource management using virtual machine migrations. *IEEE Communications ... 50*, 9 (2012), 34–40.

[76] MISHRA, M., AND SAHOO, A. On Theory of VM Placement - Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach. *IEEE CLOUD* (2011), 275–282.

[77] MURTAZAEV, A., AND OH, S. Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing. *IETE Technical Review 28*, 3 (Sept. 2014), 212.

[78] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic Design of Scheduling Policies for Dynamic Multi-objective Job Shop Scheduling via Cooperative Coevolution Genetic Programming. *IEEE Transactions on Evolutionary Computation 18*, 2 (2014), 193–208.

[79] ODUGUWA, V., AND ROY, R. Bi-level optimisation using genetic algorithm. In *2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS 2002)* (2002), IEEE Comput. Soc, pp. 322–327.

[80] PANIGRAHY, R., TALWAR, K., UYEDA, L., AND WIEDER, U. Heuristics for vector bin packing. *research microsoft com* (2011).

[81] PANWALKAR, S. S., AND ISKANDER, W. A Survey of Scheduling Rules. *Operations Research 25*, 1 (1977), 45–61.

[82] PASCHKE, A., AND SCHNAPPINGER-GERULL, E. A Categorization Scheme for SLA Metrics. *Service Oriented Electronic Commerce 80*, 25-40 (2006), 14.

[83] PIRAGHAJ, S. F., CALHEIROS, R. N., CHAN, J., DASTJERDI, A. V., AND BUYYA, R. Virtual Machine Customization and Task Mapping Architecture for Efficient Allocation of Cloud Data Center Resources. *Comput. J. 59*, 2 (2016), 208–224.

[84] PIRAGHAJ, S. F., DASTJERDI, A. V., CALHEIROS, R. N., AND BUYYA, R. Efficient Virtual Machine Sizing for Hosting Containers as a Service (SERVICES 2015). *SERVICES* (2015).

[85] POLI, R., WOODWARD, J., AND BURKE, E. K. A histogram-matching approach to the evolution of bin-packing strategies. In *2007 IEEE Congress on Evolutionary Computation* (2007), IEEE, pp. 3500–3507.

[86] QIN, X., ZHANG, W., 0049, W. W., WEI, J., ZHAO, X., AND HUANG, T. Towards a Cost-Aware Data Migration Approach for Key-Value Stores. *CLUSTER* (2012).

[87] RADCLIFFE, N. J. Forma Analysis and Random Respectful Recombination. *ICGA* (1991).

[88] RAGHAVENDRA, R., RANGANATHAN, P., TALWAR, V., WANG, Z., AND ZHU, X. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News* (2008), ACM, pp. 48–59.

[89] RONG, H., ZHANG, H., XIAO, S., LI, C., AND HU, C. Optimizing energy consumption for data centers. *Renewable and Sustainable Energy Reviews 58* (May 2016), 674–691.

[90] ROSEN, R. Resource management: Linux kernel namespaces and cgroups. *Haifux* (2013).

[91] SARIN, S. C., VARADARAJAN, A., AND WANG, L. A survey of dispatching rules for operational control in wafer fabrication. *Production Planning and ... 22*, 1 (Jan. 2011), 4–24.

[92] SHAFER, J. I/O virtualization bottlenecks in cloud computing today. In *Proceedings of the 2nd conference on I/O virtualization* (2010).

[93] SHEN, S., VAN BEEK, V., AND IOSUP, A. Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. *CCGRID* (2015), 465–474.

[94] SHI, W., AND HONG, B. Towards Profitable Virtual Machine Placement in the Data Center. *UCC* (2011), 138–145.

[95] SIM, K., AND HART, E. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. *GECCO* (2013), 1549.

[96] SINHA, A., MALO, P., AND DEB, K. Efficient Evolutionary Algorithm for Single-Objective Bilevel Optimization.

[97] SINHA, A., MALO, P., AND DEB, K. A Review on Bilevel Optimization: From Classical to Evolutionary Approaches and Applications. *IEEE Transactions on Evolutionary Computation* (2017), 1–1.

[98] SOLTESZ, S., PÖTZL, H., FIUCZYNSKI, M. E., BAVIER, A. C., AND PETERSON, L. L. Container-based operating system virtualization - a scalable, high-performance alternative to hypervisors. *EuroSys 41*, 3 (2007), 275–287.

[99] SOMANI, G., AND CHAUDHARY, S. Application Performance Isolation in Virtualization. *IEEE CLOUD* (2009), 41–48.

[100] SOTELO-FIGUEROA, M. A., SOBERANES, H. J. P., CARPIO, J. M., HUACUJA, H. J. F., REYES, L. C., AND SORIA-ALCARAZ, J. A. Evolving Bin Packing Heuristic Using Micro-Differential Evolution with Indirect Representation. *Recent Advances on Hybrid Intelligent Systems 451*, Chapter 28 (2013), 349–359.

[101] SPROTT, D., AND WILKES, L. Understanding service-oriented architecture. *The Architecture Journal* (2004).

[102] SRIKANTAIAH, S., KANSAL, A., AND ZHAO, F. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems* (2008), San Diego, California, pp. 1–5.

[103] SUN, H., GAO, Z., AND WU, J. A bi-level programming model and solution algorithm for the location of logistics distribution centers. *Applied mathematical modelling 32*, 4 (Apr. 2008), 610–616.

[104] SVARD, P., LI, W., WADBRO, E., TORDSSON, J., AND ELMROTH, E. Continuous Datacenter Consolidation. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)* (2015), IEEE, pp. 387–396.

[105] TAN, B., MA, H., AND MEI, Y. A NSGA-II-based approach for service resource allocation in Cloud. *CEC* (2017), 2574–2581.

[106] TOMÁS, L., AND TORDSSON, J. Improving cloud infrastructure utilization through overbooking. *CAC* (2013), 1.

[107] UHLIG, R., NEIGER, G., RODGERS, D., SANTONI, A. L., MARTINS, F. C. M., ANDERSON, A. V., BENNETT, S. M., KÄGI, A., LEUNG, F. H., AND SMITH, L. Intel Virtualization Technology. *IEEE Computer 38*, 5 (2005), 48–56.

[108] VARASTEH, A., AND GOUDARZI, M. Server Consolidation Techniques in Virtualized Data Centers: A Survey. *IEEE Systems Journal* (2015), 1–12.

[109] VERMA, A., AND DASGUPTA, G. Server Workload Analysis for Power Minimization using Consolidation. *USENIX Annual Technical Conference* (2009), 28–28.

[110] VICENTE, L., SAVARD, G., AND JÚDICE, J. Descent approaches for quadratic bilevel programming. *Journal of Optimization Theory and Applications 81*, 2 (May 1994), 379–399.

[111] VISWANATHAN, B., VERMA, A., AND DUTTA, S. CloudMap - Workload-aware placement in private heterogeneous clouds. *NOMS* (2012), 9–16.

[112] WALDSPURGER, C. A. Memory resource management in VMware ESX server. *ACM SIGOPS Operating Systems Review 36*, SI (Dec. 2002), 181–194.

[113] WANG, Y., AND 0002, Y. X. Energy Optimal VM Placement in the Cloud. *CLOUD* (2016), 84–91.

[114] WANG, Y., JIAO, Y.-C., AND LI, H. An evolutionary algorithm for solving nonlinear bilevel programming based on a new constraint-handling scheme. *IEEE Trans. Systems, Man, and Cybernetics, Part C 35*, 2 (2005), 221–232.

[115] WANG, Y., LI, H., AND DANG, C. A New Evolutionary Algorithm for a Class of Nonlinear Bilevel Programming Problems and Its Global Convergence. *INFORMS Journal on Computing 23*, 4 (2011), 618–629.

[116] WILCOX, D., MCNABB, A. W., AND SEPPI, K. D. Solving virtual machine packing with a Reordering Grouping Genetic Algorithm. *IEEE Congress on Evolutionary Computation* (2011), 362–369.

[117] XAVIER, M. G., NEVES, M. V., ROSSI, F. D., FERRETO, T. C., LANGE, T., AND DE ROSE, C. A. F. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In *2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2013)* (2013), IEEE, pp. 233–240.

[118] XAVIER, M. G., ROSSI, F. D., DE ROSE, C. A. F., CALHEIROS, R. N., AND GOMES, D. G. Modeling and simulation of global and sleep states in ACPI-compliant energy-efficient cloud environments. *Concurrency and Computation - Practice and Experience 29*, 4 (Feb. 2017), e3839.

[119] XIAO, Z., JIANG, J., ZHU, Y., MING, Z., ZHONG, S.-H., AND CAI, S.-B. A solution of dynamic VMs placement problem for energy consumption optimization based on evolutionary game theory. *Journal of Systems and Software 101* (2015), 260–272.

[120] XIONG, A.-P., AND XU, C.-X. Energy Efficient Multiresource Allocation of Virtual Machine Based on PSO in Cloud Data Center. *Mathematical Problems in Engineering 2014*, 6 (2014), 1–8.

[121] XU, J., AND FORTES, J. A. B. Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments. *GreenCom/CPSCom* (2010).

[122] YAU, S. S., AND AN, H. G. Adaptive resource allocation for service-based systems. In *Proceedings of the First Asia-Pacific Symposium on Internetware* (2009), ACM, p. 3.

[123] YIN, Y. Genetic-algorithms-based approach for bilevel programming models. *Journal of Transportation Engineering* (2000).

[124] ZHANG, J., HUANG, H., AND WANG, X. Resource provision algorithms in cloud computing - A survey. *J. Network and Computer Applications 64* (2016), 23–42.

[125] ZHANG, Q., CHENG, L., AND BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications 1*, 1 (2010), 7–18.

[126] ZHANG, Y., ZHENG, Z., AND LYU, M. R. Exploring Latent Features for Memory-Based QoS Prediction in Cloud Computing. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on* (2011), pp. 1–10.

[127] ZHU, X., YU, Q., AND WANG, X. A Hybrid Differential Evolution Algorithm for Solving Nonlinear Bilevel Programming with Linear Constraints. In *2006 5th IEEE International Conference on Cognitive Informatics* (2006), IEEE, pp. 126–131.