Novel Genetic Algorithm with Dual Chromosome Representation for Resource Allocation in Container-based Clouds

Boxiong Tan, Hui Ma, Yi Mei
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email:{Boxiong.Tan, Hui.Ma, Yi.Mei}@ecs.vuw.ac.nz

***Abstract*—Containerization does not only support fast development and deployment of web applications but also provides the potential to improve the energy efficiency in cloud data centers. In container-based clouds, containers are allocated to virtual machines (VMs) and VMs are allocated to physical machines (PMs). This new architecture requires consolidation algorithms to select heterogeneous VMs to host containers and consolidate VMs to PMs simultaneously. Existing server consolidation techniques in VM-based clouds can hardly be applied because of the two-level architecture of the container-based clouds. This paper proposes a novel genetic algorithm (GA) with dual chromosome representation to solve the problem. The experiments show that the proposed GA achieves significantly higher energy efficiency than the compared state-of-the-art algorithms on a wide range of test problems.**

***Index Terms*—container, server consolidation, cloud resource allocation, genetic algorithm, evolutionary computation**

I. Introduction

Container-based clouds [1] have gradually become the pillar of the modern software industry with the rise of micro-services and server-less applications. With containers, it is easier for application providers to pack, migrate, and deploy web applications than using virtual machines (VMs). Internet companies such as Google and Microsoft run their applications with millions of containers in their data centers.

Server consolidation is a technique to reduce energy consumption in data centers. In container-based clouds, server consolidation is difficult because of the resource allocation in container-based clouds is two-level: container allocation on VMs and VMs allocation on physical machines (PMs). The extensive server consolidation algorithms [2] in VM-based clouds can hardly be reused.

To solve the two-level resource allocation in container-based clouds, two major difficulties have puzzled researchers. The first difficulty is the interaction between container allocation and VM allocation. In other words, the best container allocation may not lead to the best VM allocation. Hence, the allocation at the two levels must be conducted simultaneously. Secondly, both levels of allocation are vector bin packing problems [3] which are NP-hard.

This research aims to develop a novel resource allocation technique for two-level container-based clouds to minimize the energy consumption of data centers. To address the two-level resource allocation problem, this work proposes a Genetic Algorithm (GA)-based approach.

GA [4] has been successfully applied to various combinatorial optimization problem over its fifty-year history [5]. GA has a strong ability to avoid local optimums. In [6], GA has been used to solve the VM allocation problem. The key difficulty of applying GA to solve the problem is the design of representations of solutions and genetic operators that can evolve solutions. This is because a good representation narrows the search space and good operators are able to accelerate the searching for near-optimal solutions.

The overall goal is to propose a GA-based approach for the resource allocation problem in container-based clouds. More specifically, we propose a new dual-chromosome representation, new genetic operators, and evaluate the proposed GA by comparing with the state-of-the-art algorithms: BestFit Descending [7] and the single-chromosome GA [8].

II. Related works

In this section, we first review the related works of the server consolidation problem in container-based clouds. We then provide a brief background of genetic algorithms.

A. Existing Approaches

Currently, most server consolidation approaches in container-based clouds treat the problem as a dynamic problem [2], which means they allocate one container at a time when the request arrives. Piraghaj [9] and Kaur [10] approach the problem by considering a set of predefined VM types in a cloud and applying First Fit heuristic to allocate VMs to PMs. At the migration stage, containers are allocated according to rules such as the Least Full Host Selection Algorithm (LFHS) [9] and the First Fit Host Selection (FFHS). These rules are mostly greedy-based heuristics which are designed for fast resource allocations.

The major drawbacks of the above methods are on two aspects. First, as Wolke et al. [2] concluded, dynamic approaches are worse than static approaches because dynamic approaches migrate containers frequently which lead to extra

network overhead. Secondly, greedy-based heuristics often provide local optimal solutions. On the other hand, Wolke et al. [2] suggest that the static approaches are more suitable for the initial allocation of containers because the initial allocation normally has a longer time tolerance. Therefore, we aim at allocating a batch of containers to a number of new VMs and PMs. Although the static approach might have longer computational time, it generally provides better performance than greedy approaches.

As container-based clouds emerged recently, most research lacks the functionality of selecting VM types. Guan et al. [11] consider one type of VM and each PM are filled with ten VMs. Then, they propose an Integer Linear Programming (ILP) to allocate containers. However, only considering one predefined types of VMs not only is inflexible to meet different resource requirements but also leads to the waste of resources. Furthermore, allocating more VMs leads to more overheads. In the perspective of their approach, it is known that the computational time of ILP-based approaches grows exponentially with the increase of the problem size.

A multi-objective genetic algorithm approach [8] has been proposed to solve the two-level container allocation problem. This approach proposes a single-chromosome representation to optimize two-level resource allocation. However, their approach is limited with the design of genetic operators because it does not use a crossover operator and relies completely on mutation for local search. Intuitively, the crossover can enhance the searchability of GA. Therefore, it motivates us to design an effective GA-based method with crossover operators. For this purpose, we design a new representation.

III. Problem Model

Given a set of containers $1, \ldots, N$, the *resource allocation in a container-based cloud* allocates containers to VMs $1, \ldots, V$, then allocate the VMs to a set of PMs $1, \ldots, P$, so that the aggregated energy consumption AE of all the PMs are minimized. The following equations calculate the overall energy consumption AE and the energy consumption E_p of an individual PM.

$$\min\ AE = \sum_{p=1}^{P} E_p \cdot \lceil u_{cpu}(p) \rceil \tag{1}$$

$$s.t.\ \sum_{n=1}^{N} C_n \cdot x_{nv} \leq VC_v,\ \sum_{n=1}^{N} M_n \cdot x_{nv} \leq VM_v \tag{2}$$

$$\sum_{v=1}^{V} VC_v \cdot y_{vp} \leq PC_p,\ \sum_{v=1}^{V} VM_v \cdot y_{vp} \leq PM_p \tag{3}$$

$$\sum_{n=1}^{N} x_{nv} = 1 \tag{4}$$

Where Eq.(2) defines a constrain that the total resource requirement of containers cannot exceed the capacity of the target VM v. Eq.(3) defines a constrain that the total resource requirement of VMs cannot exceed the capacity of the target PM p. Eq.(4) is that a container can only be deployed once.

E (Eq. (1)) [12] aggregates all PMs' energy consumption E_p if a PM p is activated. The term $\lceil u_{cpu}(p) \rceil$ returns 1 when the CPU utilization of a PM p is greater than 0, and 0 otherwise. E^{idle} and E^{max} are the energy consumption when a PM is idle and fully used, respectively.

$$E_p = E_p^{idle} + (E_p^{max} - E_p^{idle}) \cdot u_{cpu}(p) \tag{5}$$

In our model, containers, VMs, and PMs are associated with two types of resources, CPU and memory. Containers' CPU and memory requirements are denoted as C_n and M_n; A VM has CPU and memory capacities VC_v and VM_v; A PM has CPU and memory capacities PC_p and PM_p. The amount of resources is defined by a domain (i.e. CPU capacity of an entity) of $[1, \ldots, R]$. In addition, each VM has overheads of CPU and memory denoted as $OC(v)$ and $OM(v)$. The overheads are also represented as resources.

This work considers a data center with homogeneous PMs. We consider a number of types of VMs which constrain the combination of VC and VM into $Type_v$. For containers, we consider the one-to-one mapping between applications and containers. That is, we define the domain of containers' resource requirement between 1 and the capacity of the largest VM type.

The two-level allocation model mainly reflects in modeling the resource utilizations. At the containers–VMs level, the CPU and memory utilization of a VM are computed with Eq. (6) and Eq. (7). The binary variable x_{nv} (i.e. 0 or 1) indicates whether a container n is allocated on a VM v.

$$\mu_{cpu}(v) = \frac{\sum_{n=1}^{N} C_n \cdot x_{nv}}{VC_v} \tag{6}$$

$$\mu_{mem}(v) = \frac{\sum_{n=1}^{N} M_n \cdot x_{nv}}{VM_v} \tag{7}$$

At the VMs–PMs level, the CPU and memory utilization of a PM is computed with Eq. (8) and Eq. (9). The utilization of a PM aggregates the used resources by the VMs which are deployed on the PM. The binary variable y_{vp} denotes if a VM v is allocated on a PM p.

$$u_{cpu}(p) = \frac{\sum_{v=1}^{V} (OC(v) + \mu_{cpu}(v) \cdot VC_v) \cdot y_{vp}}{PC_p} \tag{8}$$

$$u_{mem}(p) = \frac{\sum_{v=1}^{V} (OM(v) + \mu_{mem}(v) \cdot VM_v) \cdot y_{vp}}{PM_p} \tag{9}$$

IV. The Proposed Dual-Chromosome GA

This section introduces the design of our GA-based approach, which includes the representation, genetic operators, and the fitness function.

A. Algorithm

Our proposed algorithm is a standard GA [4] with our problem-specific representation and operators.

B. Representation

The representation of an individual consists of two separate chromosomes: one for container allocation and the other for VM allocation. Fig. 1 shows the design of the chromosomes, the input, and the output of the algorithm. A complete allocation solution can be obtained by a decoding procedure.

Both chromosomes are vectors of integer values. In the chromosome of container allocation, each value represents the indexes of the containers in the original input. The length of the chromosome is the total number of containers.

In the chromosome of VM allocation, each entry represents a VM type with the value taken from types of VMs. The length of the VM allocation vector equals the length of the container allocation vector. This is because we may use at most N VMs for allocating N containers (one-to-one mapping).

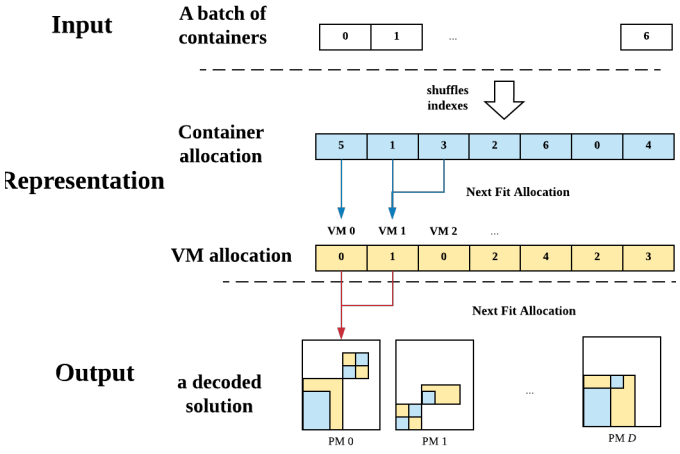


Fig. 1: Representation

To decode an individual to an allocation solution, we apply a bin packing heuristic – Next Fit algorithm in both levels. At the container–VM level, containers are packed sequentially into VMs. For example, In Fig. 1, after *container 5* is packed to *VM 0*, the following *container 1* cannot fit into *VM 0*. Therefore, we close *VM 0* and open *VM 1* to accommodate *container 1*. The closed VMs are never checked again later on. This decoding ends when all containers are allocated. Similarly, VMs are packed into PMs using the same rule. A decoded solution includes both levels of allocation as well as the types of the used VMs.

On one hand, this representation guarantees the feasibility of solutions by applying Next Fit heuristic to decode the solution. Therefore, we do not need extra constraint handling methods. On the other hand, this representation is able to cover the entire solution space so that we may find the optimal solution.

C. Initialization

For each individual, we randomly shuffle the indexes of containers to generate container allocation chromosomes. For initializing VM allocation chromosomes, we uniformly generate the types of VMs.

D. Crossover

A good crossover leads to high utilization of VMs' resources. We apply the order 1 crossover [13] to pass the useful permutation to the next generation. For the chromosome of VM allocation, we apply the single-point crossover [14].

The order 1 crossover randomly selects a sequence of consecutive entries from one parent. The remaining values are placed in the child with the same order in the other parent. For example, in Fig 2, container 3 and 4 are selected and copied from parent to child 1. Then, the same containers (3 and 4) are crossed out from parent 2. The rest values are copied from parent 2 starting from the second cut point and rolling back to the head, e.g. container 1, 5, and 2. The same rules are applied to the second child.

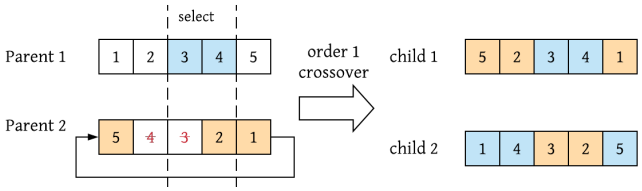


Fig. 2: Order 1 crossover

The single-point crossover first randomly cuts a chromosome into two parts. A child inherits one part from parent 1 and the other part from parent 2.

E. Mutation

The mutation operator provides a local search mechanism that aims at exploring the neighbors of the current individual. For each level of chromosomes, we design a mutation operator to achieve this goal.

Switch mutation randomly selects two entries on container allocation chromosome and switch their values. This mutation changes the allocation of two containers.

Change VM type mutation loops through the VM allocation chromosome and changes the value uniformly from the VM type list by a probability. This mutation modifies the types of VMs.

F. Fitness Function

The fitness of a solution is determined by the energy consumption of all PMs resulted from the decoding process. The energy consumption AE is calculated according to Eq. (1), which aggregates the energy consumption of all the PMs allocated with containers.

V. Experiment

A. Design of Experiments

The overall goal of the experiment is to test the performance of the dual-chromosome GA in terms of energy consumption. To this end, we conduct experiments on real-world datasets [15] by comparing with two benchmark algorithms (single-chromosome GA and BestFit Descending) in terms of energy consumption.

B. Dataset and Test instance

We use a real-world dataset where applications' resource requirement is recorded (AuverGrid trace [15]). We assume homogeneous PMs with the resource capacity of [3300 MHz, 4000 MB]. The energy consumption for the fully utilized PM is set to 135W.

To test the algorithms' performance on a variety of VM configurations, we design three sets of VM type with increasing numbers of VM types (see Table II). The first set (yellow

area) contains five VM types. The second set includes seven VM types (yellow and green areas). The third set includes all ten VM types. We designed four test instances (see Table I). We test the algorithms on these instances for each VM type settings, which includes a total of 12 experiments.

TABLE I: Instance Settings

Instances	1	2	3	4
Number of containers	100	200	500	1000

TABLE II: Configuration of VMs

VM types	CPU (MHz)	Memory (MB)	VM types	CPU (MHz)	Memory (MB)
1	660	800	6	660	2400
2	1320	1600	7	1320	2800
3	1320	2800	8	660	2800
4	1980	2400	9	1980	1200
5	2310	2800	10	2310	1600

C. Benchmark Algorithms

The single-chromosome GA is proposed in [8] to solve two-level container allocation problem.

Two differences prevent us from directly reusing their algorithm. First, their algorithm is a multi-objective version of GA called NSGA-II. Second, one of the major assumptions in their approach is that they allow *VM overbooking* [2] which tolerates a certain degree of utilization overhead. In our assumption, VM overbooking is not allowed. Therefore, in order to make a comparison, we only adopt their representation and re-design the initialization and mutation operator. For the initialization, since we disallow *VM overbooking*, we disallow more containers to be allocated to a VM if the VM does not have enough capacity. Therefore, for a chromosome, the types of VMs are randomly generated and the containers are shuffled and allocated to VMs with First Fit. The mutation operator randomly switches two containers.

The BestFit Descending algorithm has been used for selecting existing VMs in container-based clouds.

In this work, the BestFit Descending with sum rule [7] is implemented to make a comparison. Since BestFit Descending does not have the functionality to select VM types, in order to create new VMs, we always select the largest VM type (Type five in Table II).

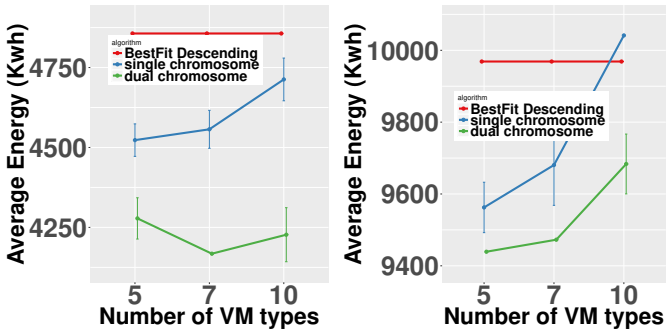
D. Parameter Settings

The parameter settings for both single- and dual-chromosome GAs are listed in Table III. In addition, we employ the elitism with size 10 and tournament selection with size 7. These parameter settings are standard and widely used. For single-chromosome GA, we set its mutation rate 0.8 because it completely relies on mutation to search.

All the algorithms were implemented in Java version 8 and the experiments were conducted on i7-4790 3.6 GHz with 8 GB of RAM memory running Linux Arch 4.14.15. We applied Wilcoxon signed rank test to test the statistic significance.

TABLE III: Parameter Settings

Parameter	Description
crossover rate x	80%
mutation rate for dual-chromosome GA	10%
mutation rate for single-chromosome GA	80%
elitism	top 10 individuals
Number of generations	1000
Population	100
Selection	tournament selection (size = 7)



(a) Test instance 3 (b) Test instance 4

Fig. 3: Comparison of the average energy consumption among the three algorithms for the three VM type configurations

E. Experiment Results

This section illustrates the performance comparison among the three algorithms in terms of energy consumption.

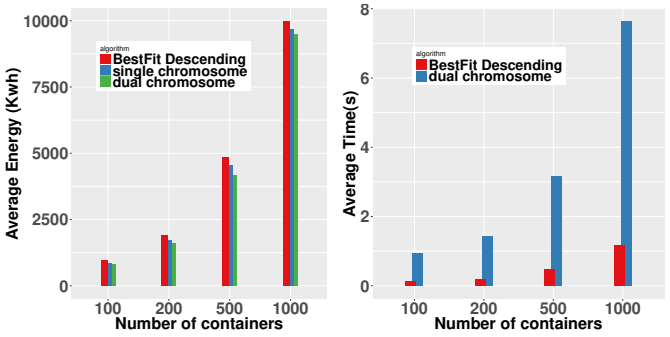
Fig. 3 shows the comparison of the average energy consumption among the three algorithms for test cases 3 and 4. We only show two instances because the pattern of instance 1 and 2 are similar to instance 3. BestFit Descending algorithm (red line) uses the most energy consumption. Single-chromosome GA (blue line) shows better performance than BestFit Descending in most of the cases except one. Dual-chromosome GA has shown the best performance in all test instances. The Wilcoxon signed rank test shows that it is significantly different between our GA and others with a confidence interval of 95%.

The energy consumption of the solution generated by the BestFit Descending is not affected by the number of VM types because it always uses the largest VM type (see Section V-C). Additionally, Best Fit Descending is a deterministic algorithm, which means with the same input set of containers, the output energy will always be the same.

Fig.4 shows the accumulated energy and time compared with the increasing number of containers. Our dual-chromosome GA has advantages on all instances in terms of energy consumption meanwhile it takes 7 times longer computation time than the BestFit Descending approach.

F. Insights

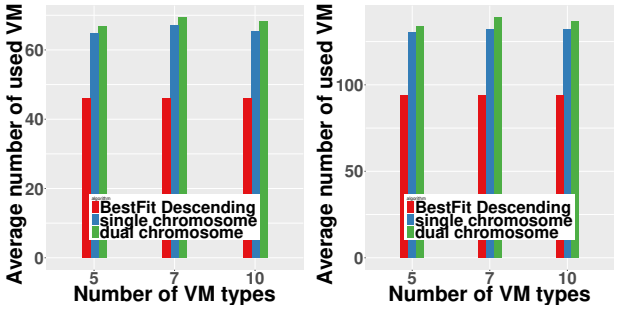
These experiments provide two insights for the container allocation problem. The first insight is that the least number of VMs does not necessarily lead to the least energy consumption. Fig. 5 shows the average number of VMs used in



(a) Accumulated energy (b) Computation time

Fig. 4: Energy and time comparison with the increasing number of containers

test instances 3 and 4. As we see, BestFit Descending (red bar) always uses the least number of VMs because it chooses the largest VM type in default. However, the largest VM type (Type five in Table II) is account for 70% of total resources in a PM. That means the maximum resources that can be used by BestFit is only 70% of the total resources in a PM because a PM can only accommodate one largest VM.



(a) Test instance 3 (b) Test instance 4

Fig. 5: Average number of used VMs

In contrast, the solutions generated by both single- and dual-chromosome GAs use more VMs than BestFit. When we examine the VM types that GAs used, we found that in most cases, GAs found the complementary VMs. For example, in Table II, the combination of VM types (2, 4), (3, 9), (2, 2, 1) is able to use 100% of PM's resources. We observed that GAs can always find these complementary VM types to maximize utilization. The best combination of VM types may not use the least number of VMs. As in our test instances, dual-chromosome GA always generates better performance with more VMs than the single-chromosome GA.

The second insight is that providing more VM types may have a negative influence on algorithms because the redundant VM types extend the search space. In Fig.3, test instance 3 represents a pattern where the dual-chromosome GA achieves the best performance applying on seven VM types and the performance drops on ten VM types. As mentioned above, GA searches for complementary VM types to fully utilized PMs. The performance increases when providing seven VM types because the additional VM types are useful to construct complementary VM types. However, when providing ten types, the additional types (compared to 7 VM types) are redundant, therefore, it only leads to longer searching time and worse performance (worse fitness). For example, in instance 4, since the size of containers is much larger than the previous instances, the search space is also much larger. The performance of both GAs decreases.

VI. Conclusion

This work proposes a dual-chromosome GA to solve the resource allocation problem in container-based clouds. The experiments ran on real-world datasets with the comparison of the single-chromosome GA and a BestFit Descending algorithm. The results show that our proposed GA performs much better than the compared algorithms in all test instances.

References

[1] B. Familiar, *Microservices, IoT and Azure: leveraging DevOps and Microservice architecture to deliver SaaS solutions*. Springer, 2015.

[2] A. Wolke, M. Bichler, and T. Setzer, "Planning vs. dynamic control: Resource allocation in corporate clouds," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 322–335, 2016.

[3] L. U. R.Panigrahy, K.Talwar, "Heuristics for vector bin packing," January 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/heuristics-for-vector-bin-packing/

[4] M. D. Vose and A. Hall, "Modeling Simple Genetic Algorithms," in *Evolutionary Computation*. Elsevier, 1996, vol. 3, no. 4, pp. 453–472.

[5] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, and J.-J. Li, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Applied Soft Computing*, vol. 34, pp. 286–300, 2015.

[6] Z. Ding, Y.-C. Tian, and M. Tang, "Efficient fitness function computation of genetic algorithm in virtual machine placement for greener data centers," in *IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2018, pp. 181–186.

[7] K. Maruyama, S. K. Chang, and D. T. Tang, "A general packing algorithm for multidimensional resource requirements," *International Journal of Computer & Information Sciences*, vol. 6, no. 2, pp. 131–149, 1977.

[8] B. Tan, H. Ma, and Y. Mei, "A NSGA-II-based approach for service resource allocation in cloud," in *IEEE Congress on Evolutionary Computation (CEC)*, 2017, pp. 2574–2581.

[9] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers," in *IEEE International Conference on Data Science and Data Intensive Systems*, 2015, pp. 368–375.

[10] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 48–56, 2017.

[11] X. Guan, X. Wan, B. Choi, S. Song, and J. Zhu, "Application oriented dynamic resource allocation for data centers using docker containers," *IEEE Communications Letters*, vol. 21, no. 3, pp. 504–507, 2017.

[12] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07, New York, NY, USA, 2007, pp. 13–23.

[13] P. Poon and J. Carter, "Genetic algorithm crossover operators for ordering applications," *Computers and Operations Research*, vol. 22, no. 1, pp. 135 – 147, 1995, Genetic Algorithms.

[14] R. B. Agrawal, K. Deb, and R. Agrawal, "Simulated binary crossover for continuous search space," *Complex systems*, vol. 9, no. 2, pp. 115–148, 1995.

[15] S. Shen, V. v. Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 465–474.