Federal State Autonomous Educational Institution for Higher Education
National Research University Higher School of Economics

Faculty of Computer Science
Applied Mathematics and Information Science

BACHELOR'S THESIS

Research project

"Algorithms for Virtual Machine Consolidation with Migration Cost"

Prepared:
Student of group 175, 4th year of study, Ilya Pauzner

Supervisor:
Principal engineer, Andrey Tikhonov

Moscow 2021

Contents

Abstract

Virtualization technologies are the cornerstone of cloud computing. Analogous to the disk defragmentation problem, there is a virtual machine consolidation problem. The problem consists of creating a mapping between logical resources (e.g. virtual machines resource consumption) and physical resources (e.g. servers resource capacity). The optimization goal is to use as few hosts as possible, thus bringing virtual machines from many hosts together to a smaller number of hosts, incurring minimal possible migrations. The useful result of this procedure is the opportunity to shut down freed hosts. This paper presents a 2-staged algorithm for this problem, the first part minimizing active hosts count, second part minimizing migrations.

1 Problem statement

The definition and variable naming are not consistent in different papers, we will use the notation from Bartók and Mann 2015. V is the set of all virtual machines, and P are the set of all physical machines (hosts). Each host p has capacity $cap(p)$, which is a 2-dimensional vector $(CPU, memory)$. Each virtual machine v has a load (how much resources it uses), a 2-dimensional vector $(CPU, memory)$.

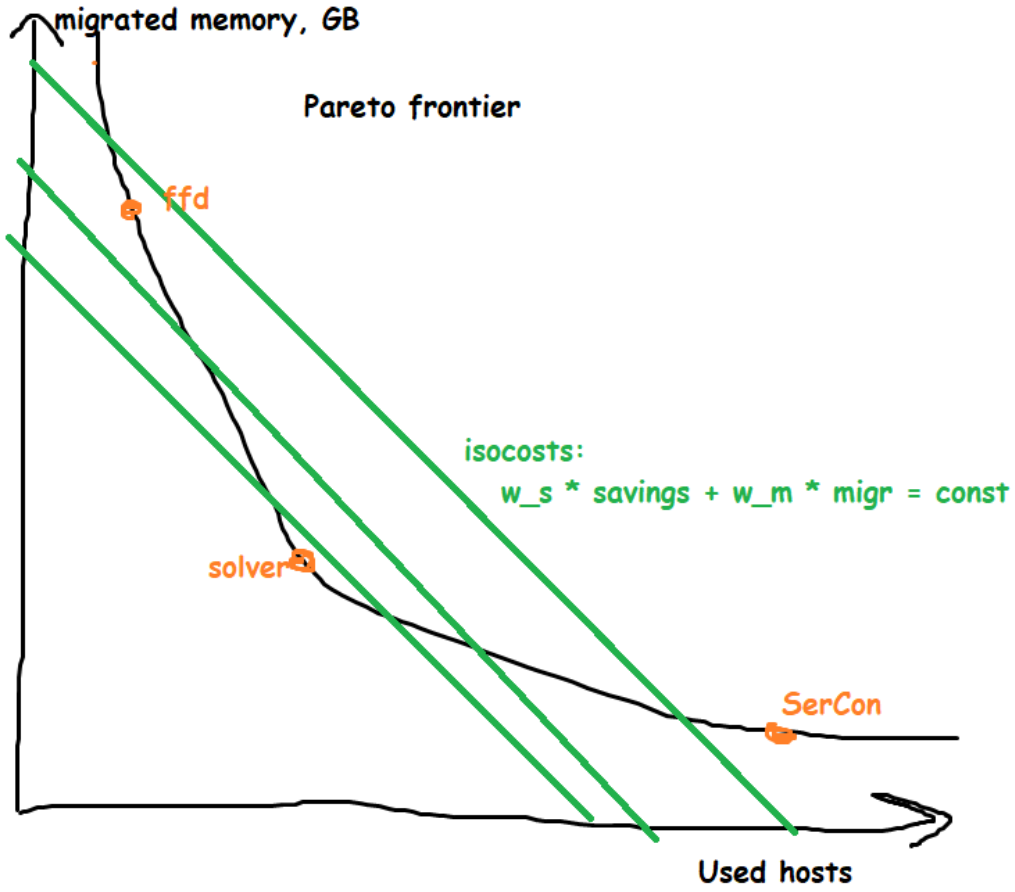
Goal of the problem is to create a mapping $map : V \to P$, where $map(v) = p$ means that virtual machine v is placed on host p. There is a necessary capacity constraint: every host should have enough capacity to bear the load of its virtual machines: $\sum_{map(v)=p} load(v)_{type} \leq cap(p)_{type} \forall type \in (cpu, memory)$. Optimisation objectives include:

1 Active hosts score: $active(old, new) = \#(\forall p \in P \ s.t. \sum_{new(v)=p} load(v) = 0)$, number of non-empty hosts, it should be minimized.

2 Migration score: $migr(old, new) = \sum_{old(v)\neq new(v)} load(v)_{memory}$, amount of migrated memory, it should be minimized.

Since the problem has two (actually, opposing each other) objectives, there exists some set of Pareto-optimal solutions. Figure 1.1 shows it, using some algorithms' performance as reference points.

Figure 1.1: Pareto curve



Live migration is the core virtualization technology allowing to move the virtual machines from one server to another incurring small (hundreds of milliseconds) downtime, thus nearly transparent to the user. It is implemented in many hypervisors, such as VirtualBox and QEMU. The way it works is described in Clark et al. 2005.

The most common implementation of live migration is pre-copy migration. Storage is usually managed via network drives, thus eliminating the need to copy it, only RAM needs a transfer. It goes as follows: virtual machine memory is copied over a network from the source host to the destination host in iterations, the next one copying pages which were changed since the previous one. When the changed pages set become sufficiently small, the virtual machine pauses, and the rest of its memory is copied to the destination host. Since all the state now was copied to the destination host, one can start a virtual machine there.

2 Problem significance

Using an algorithm from this paper one can move virtual machines from servers and shut down them when they are free. Since the industry of cloud computing now is worth billions of dollars (Amazon, Microsoft, and Google have it as a significant part of their business), the economy of electricity, both direct and indirect such as cooling costs, can save a valuable amount of money. The amount of energy consumed by cloud data centers is comparable to that of Argentina and continues to grow 11% annually, as it was stated in Mathew, Sitaraman, and Shenoy 2011. Cloud systems can be divided into two types, private and public cloud.

Public cloud provides its services to customers, ranging from IaaS such as Amazon EC2 and to PaaS such as Amazon S3. Therefore, there are tight SLAs managed by the contract with the user and they should be maintained. Many large companies such as cell phone operators, banks, and social networks (e.g. Instagram) are using this service to scale easily without maintaining their server fleet.

Private cloud is the service offered by the IT department within the private internal network of the company and used for company needs. Historically, most of the public cloud providers started by exporting their private cloud (Google Cloud Services, Yandex Cloud). Being an internal service gives more flexibility than in the public cloud. It allows doing unsafe things, such as simply shutting down the virtual machine on the source server and starting it on the destination server. It is possible if the IT department knows that virtual machine contains replicated stateless service, which can be arbitrarily restarted (to which most of the Docker ecosystem obey).

3 Literature review

The problem considered in this paper is a variation of the classic bin-packing problem which is widely studied and described in classic textbooks such as Papadimitriou and Steiglitz 1982 and Korte and Vygen 2012.

Virtual machine consolidation is a more special problem arising in cloud computing. Its main difference from bin-packing is having migration cost (see below), which corresponds to the cost of moving virtual machines from their existing places to places prescribed by new, more dense, arrangement. The following articles' review does not describe the whole picture of literature on virtual machine consolidation but rather attempts to mention few different significant approaches.

Bartók and Mann 2015

The authors consider the objective function being a weighted sum of the migration score *migr* (the total migrated bytes of RAM) and the active hosts score *active* (how many hosts have at least 1 virtual machine deployed).

$$migr(old, new) = \sum_{old(v) \neq new(v)} load(v)_{memory} \tag{1}$$

$$active(old, new) = \#(\forall p \in P \; s.t. \sum_{new(v)=p} load(v) = 0) \tag{2}$$

$$f(old, new) = \alpha \cdot migr(old, new) + \beta \cdot active(old, new)$$

Apart from this, they introduce a constraint on the migration score:

$$migr(old, new) < K$$

It is necessary, since migration is a long and costly operation and can possibly result in failure. They use a branch-and-bound approach similar to those from ILP-solvers but using problem-specific knowledge to build more efficient algorithms. The search tree consists of vertices. Vertices on level i mean "the first

i virtual machines are allocated on some hosts". Edges to the children are created by placing $i + 1$st virtual machine on some hosts. Infeasible solutions are eliminated. Some other optimizations they used include:

- Incremental computation of scores, using results in parent nodes
- Careful children ordering using heuristics
- Maintaining the lower bound on cost, used for search tree pruning

Hermenier et al. 2009

The authors consider the *active* score first; the migrations score is secondary. They also use a branch-and-bound-like approach, but using another set of optimizations:

- Using the similarity between VMs - they fall into several types, so if one of a kind fits a host, others do it too
- Maintaining not only the lower bound but also the upper bound by using the First-Fit-Decreasing heuristic

However, in this paper, the authors include migration planning in their scope: what is the sequence of virtual machine migrations, leading from an old mapping to a new mapping? After selecting the best *active* score placement, their algorithm chooses the migration sequence with the best migration score. Their migrations score is more complicated than *migr* - it is calculated as the sum of migration step scores (each step can include more than 1 virtual machine, and its score is the maximum of migrated VM's memory amounts).

Challita, Paraiso, and Merle 2017

This article is a review article; it just describes existing approaches without developing a new one. The approaches the authors cite include:

- Greedy heuristics, such as First-Fit and Least-Load.
- Ant Colony Optimisation - a generic probabilistic optimization tool, somehow similar to Simulated Annealing
- Subgraph isomorphism algorithms - by reducing the VM placement to an NP-hard (just as original) subgraph isomorphism problem, which has well-developed algorithms for solving it.
- Stochastic Integer Programming - a way to probabilistically solve Integer Linear Programming problem

Murtazaev and Oh 2011

In this paper, the authors offer a simple heuristic, similar to First-Fit-Decreasing in the bin-packing problem.

Their algorithm works the following way:

```
while True:
    least_loaded_host = min(hosts, key=calc_load)
    failed = False
    for vm in sorted(vms on least_loaded_host, reverse=True):
        if not (try to move vm to another active host):
            return
```

The advantages of this approach include: short running time (linear on the product of $\|V\|$ and $\|P\|$, while other algorithms have exponential complexity), the solution is generated step by step by moving VMs, so it is not necessary to produce a migrations timetable from the initial configuration to the desired one. However, these advantages are outweighed by the fact that the answer is not guaranteed to be optimal. Moreover, it is possible to construct a configuration. The algorithm's answer is blatantly incorrect (for example, where the least-loaded machines contain a large virtual machine, with the size of about half of the host, it is impossible to move it anywhere).

This algorithm can be used as an initial approximation and an achievable upper bound for more sophisticated algorithms.

Brandão 2016

Filipe Brandão from Universidade do Porto in his article from 2016 presents a generic way to solve bin-packing problems efficiently. In this paper, the two-step approach is used. In the first step, Mixed Integer Programming Model is created, which encodes the problem statement and goals. In the second, the model is fed to a general-purpose MIP solver (such as GLPK or CBC), which produces a solution.

The main culprit of the article is to develop model, which is computationally easier to solve than the straightforward one:

$$\forall type, host: \sum_{map(vm)=host} load(vm)_{type} \cdot taken(vm) \leqslant capacity(host)_{type} \quad (3)$$

$$\forall vm: \ taken(vm) \in \{0, 1\} \quad (4)$$

Bin-packing (NP-complete itself) is reduced to a specific calculation (minimum flow satisfying several constraints) on a graph derived from a problem. This graph can be vastly compressed without weakening the constraints, thus providing a much simpler model, similar to Gilmore and Gomory 1961.

This approach proves to be very effective and is implemented in the thesis' code.

4 Solution

The algorithm described below is the particular case of the general idea in optimization called multistage optimization. It consists of making multiple consecutive steps, optimizing different parts of the problem. A well-known example of this method is coordinate descent, which is coordinate-wise gradient descent,

optimizing one coordinate at a time.

The algorithm consists of two consecutive stages, the first one optimizing *active* (see equation 2) score and the second one optimizing *migr* (see equation 1) score for the fixed *active* score.

4.1 Optimizing *active* score

This problem is a particular case of the well-known problem of bin-packing, 2-dimensional vector bin-packing. Vector bin-packing problem goes as follows:

1 Vector space V, weights of items (defined below)

2 Finite set (with repetitions) $V \subset S$, items weights

3 Finite set (with repetitions) $V \subset S$, bins capacities

The goal is to assign items I to bins B, so that sum of all items in one bin does not exceed (by every coordinate) its capacity. An optimization objective is non-empty bins count, and it should be minimized.

Substituting R^2 for V, virtual machines for items, and hosts for bins, we get our original problem. Vector bin-packing is a well-developed optimization problem and has many useful heuristics with proven upper bounds. Panigrahy et al. 2011 describes some of them in great detail. First-Fit-Decreasing is one of the most popular approximate algorithms of this class.

First-Fit-Decreasing pseudocode:

```
Sort bins in arbitrary order.
Sort items in decreasing lexicographic order.
For item in items:
    For bin in bins:
        if item fits bin:
            bin.append(item)
            break
```

Algorithm works in $O(n(\log m + \log n))$ where n means items count and m means bins count. To achieve this complexity, one should store all bins in ordered multiset so that operation "Find first suitable bin" can be implemented through lower_bound. It's approximation rate for *active* score is just $\frac{11}{9}$ from optimum for 1-dimensional case. 2-dimensional is harder to analyze. However, it is possible to prove a reasonable upper bound.

The major drawback of First-Fit-Decreasing is that it does not distinguish whether a virtual machine previously was on this host or another. Taking this into account, one can deduce that *migr* score of First-Fit-Decreasing's solution can be absurdly high. The next part is dedicated to lowering *migr* score without any harm to *active* score.

4.2 Optimizing *migr* score

Our goal in this part is to rearrange virtual machines in such a way, that *active* score is the same, but *migr* score decreases. For the sake of simplicity, we will only consider permutations between hosts which have at least 1 virtual machines.

Key concept which will help to develop algorithm for this problem is *hole*. We consider partition of one host into non-overlapping parts (see figure 4.1).

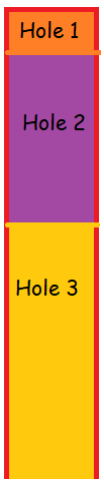


Figure 4.1: Hole system

A *hole* is one such part and is characterized by its dimensions (CPU and RAM). A hole can fit a virtual machine if hole dimensions are not smaller than virtual machine dimensions. We will partition all hosts into holes in the following way: each virtual machine occupies a hole of its size, free space is evenly cut in holes of the size of the largest virtual machine.

Now, when there is such a partition and virtual machines are assigned to holes, one can describe a large class of virtual machine arrangements, where virtual machines are somehow rearranged, obliging

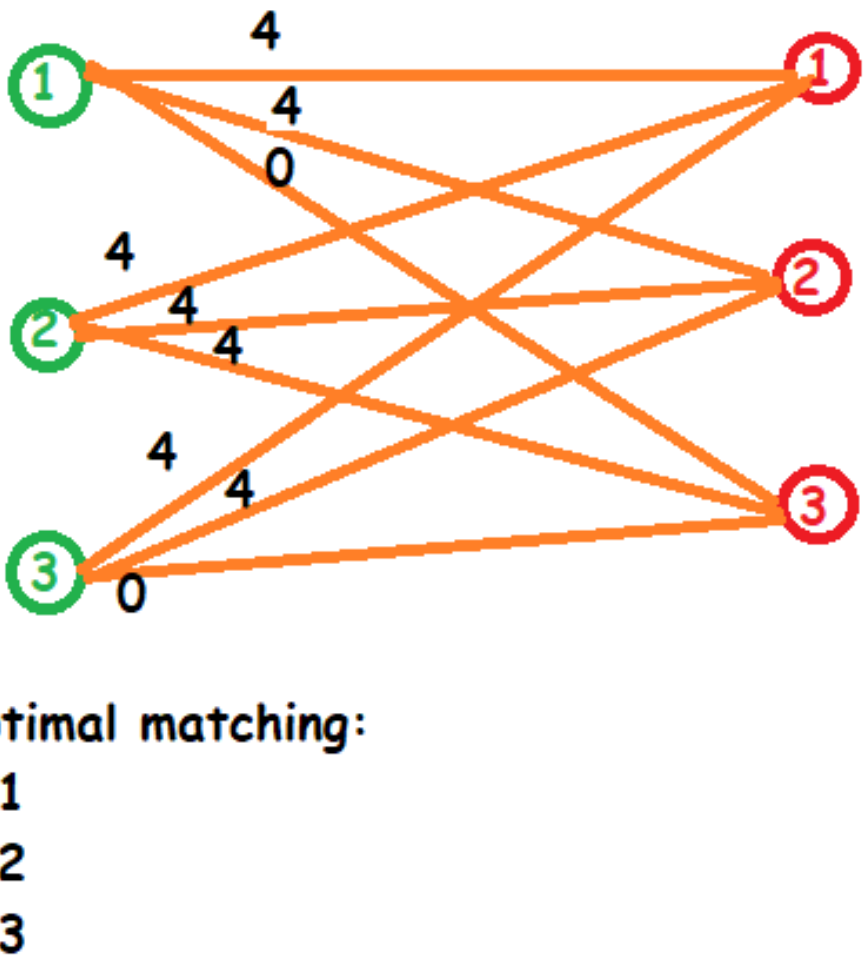
to the property that each virtual machine lays in a sufficiently large hole. Migration cost of putting the virtual machine into a hole can take 3 possible values (see figure 4.2):

1 0, if virtual machine was on this host before and hole is sufficiently large

2 RAM, if virtual machine was not on this host before and hole is sufficiently large

3 ∞, if hole is not sufficiently large

Figure 4.2: Optimal matching



It can be easily shown that the total cost of an arrangement is finite if the arrangement is feasible and is equal to *migr* cost. Therefore, we need to assign

virtual machines to holes in such a way that cost is minimal. Mathematically speaking, we need to find minimum weight matching in a bipartite graph.

This problem is widely known as the assignment problem: there are n workers and n jobs; one needs to create a timetable where all jobs are done with the minimum total cost. The Hungarian Algorithm, described in Kuhn 2005 can be used to solve it in polynomial time.

5 Results

https://github.com/ilya-pauzner/thesis

Code is written in Python 3 programming language. NumPy library is used for fast numeric computations and Pandas is used for experiments results processing and visualisation.

5.1 Code structure

File	Contents
main.py	Consolidation algorithms
generate_tests.py	Test generators
results.csv	Evaluation results

5.2 Algorithms and data structures

Each virtual machine and host is represented as NumPy array of shape (2) of numbers $(CPU, memory)$. This facilitates easy validation of capacity constraints.

Algorithm	Function in main.py
Initial (does nothing)	dummy_reorder
First-Fit-Decreasing	ffd_reorder
First-Fit-Decreasing + Migration Optimizer	with_migopt(ffd_reorder)
VPSolver	solver_reorder
VPSolver + Migration Optimizer	with_migopt(solver_reorder)
SerCon	sercon_reorder

All algorithms are named as ... *_reorder* and have a uniform interface:
in: (list of hosts, list of virtual machines, mapping of virtual machines to hosts)
out: updated mapping of virtual machines to hosts
If the algorithm failed to consolidate, it returns the previous mapping to retain the correctness of mapping.

5.3 Tests design

Virtual machine sizes distributions in the following test cases are based upon rounded statistics from large cloud provider. All test cases take random seed, number of hosts and distribution as parameters. The algorithms to create test cases with selected parameters are provided below.

5.3.1 Shrink test generator

```
hosts = AllocateHosts(hosts_size)
vms = []
while True:
    vm = ChooseVirtualMachine(distribution)
    if not FirstFit(vm, hosts):
        break
    vms.append(vm)
for vm_type in vm_types:
    shrink_factor_cpu = random(0.5, 1)
    shrink_factor_ram = random(0.5, 1)
    for vm in vms:
        if vm == vm_type:
            vm *= [shrink_factor_cpu, shrink_factor_ram]
```

The inspiration for this test comes from real-world applications. Most customers do not fully use their provisioned quotas. Therefore it is useful to gather statistics of resource usage and allow resources overbooking, knowing their average

consumption. This step, gathering of statistics and overbooking is emulated via demands shrinkage.

5.3.2 Full-pack test generator

```
init_hosts = AllocateHosts(hosts_size / 2)
vms = []
for init_host in init_hosts:
    while not Full(init_host):
        vm = ChooseVirtualMachine(distribution)
        if AddOnHost(vm, init_host):
            vms.append(vm)
hosts = AllocateHosts(hosts_size)
for vm in vms:
    RandomFit(vm, hosts)
```

This test case has one major advantage: there is obvious optimal solution of 0.5 *active* score, since originally all virtual machines resulted on half of the hosts number. Because of fully loaded hosts in optimal solution, it is somehow hard case for simple heuristic such as SerCon.

5.3.3 Test parameters

Parameter	**Values**
Test type	Shrink, Full-pack
VM sizes distribution	3, 4
Hosts count	10, 20, 40, 80
Algorithm	Initial, FFD, FFD+migopt VPSolver, VPSolver+migopt SerCon

5.4 Results

Figure 5.1: Scores on full-pack tests

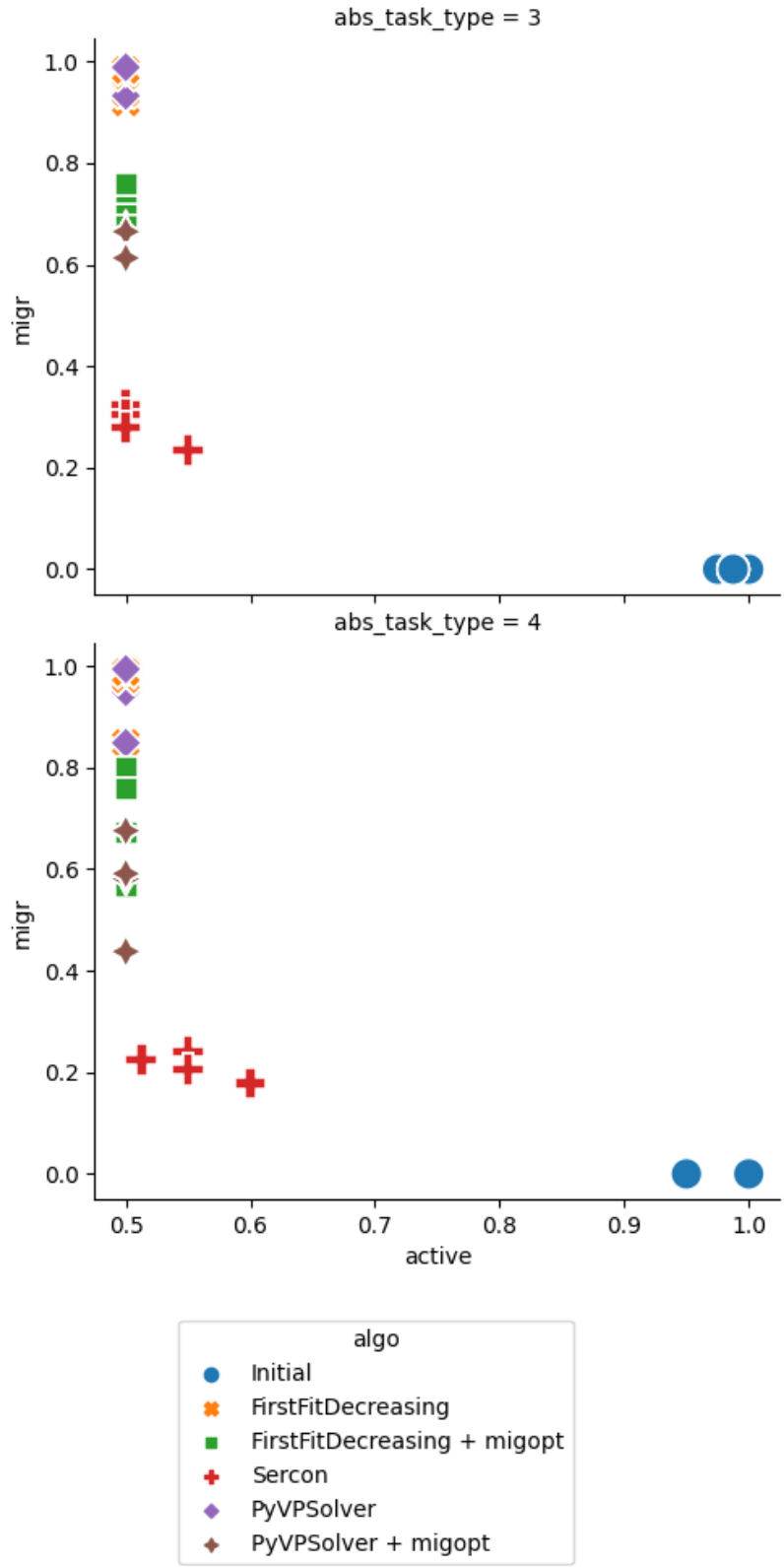


Figure 5.2: Scores on shrink tests

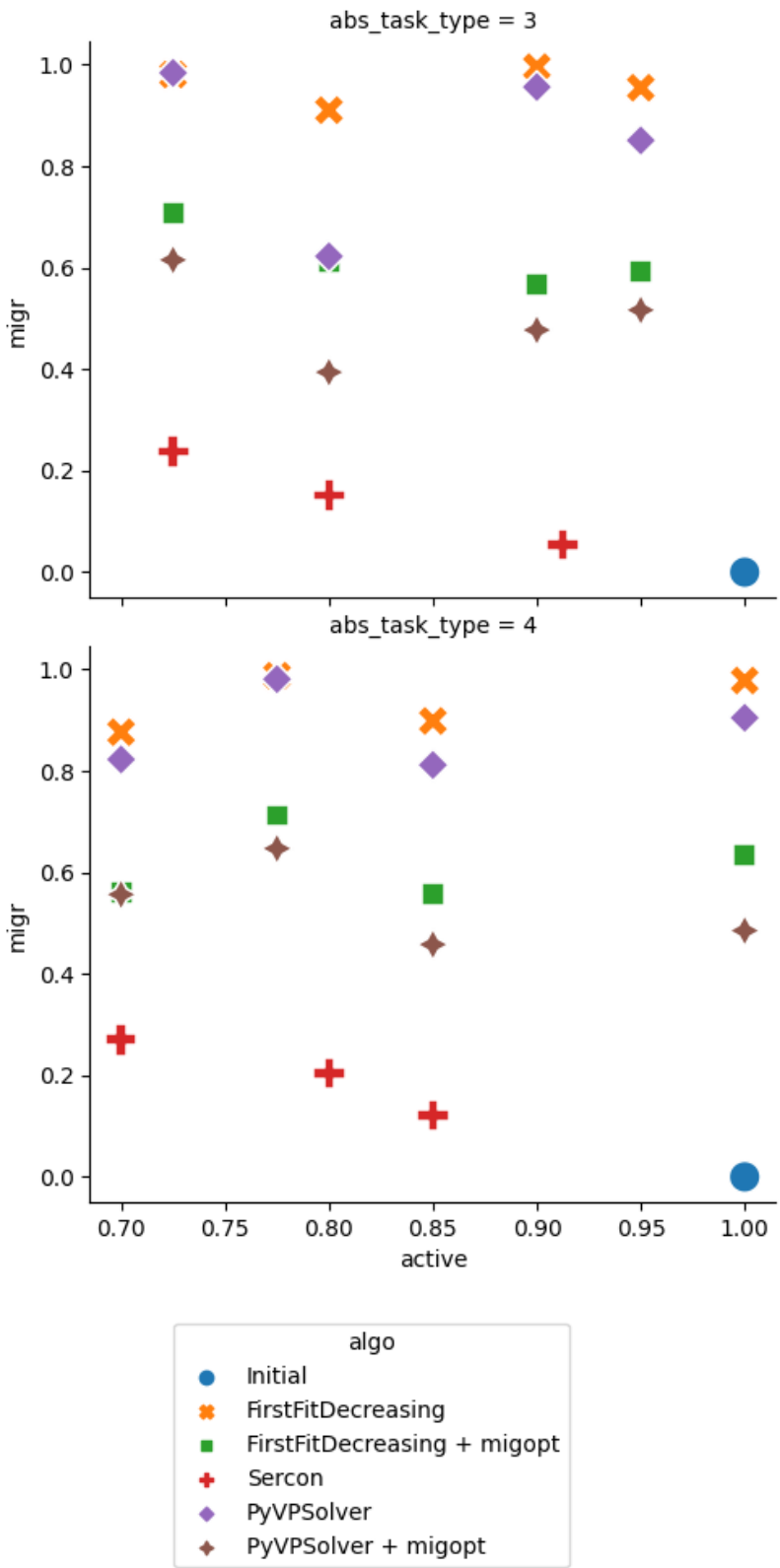


Figure 5.3: Working times on full-pack tests

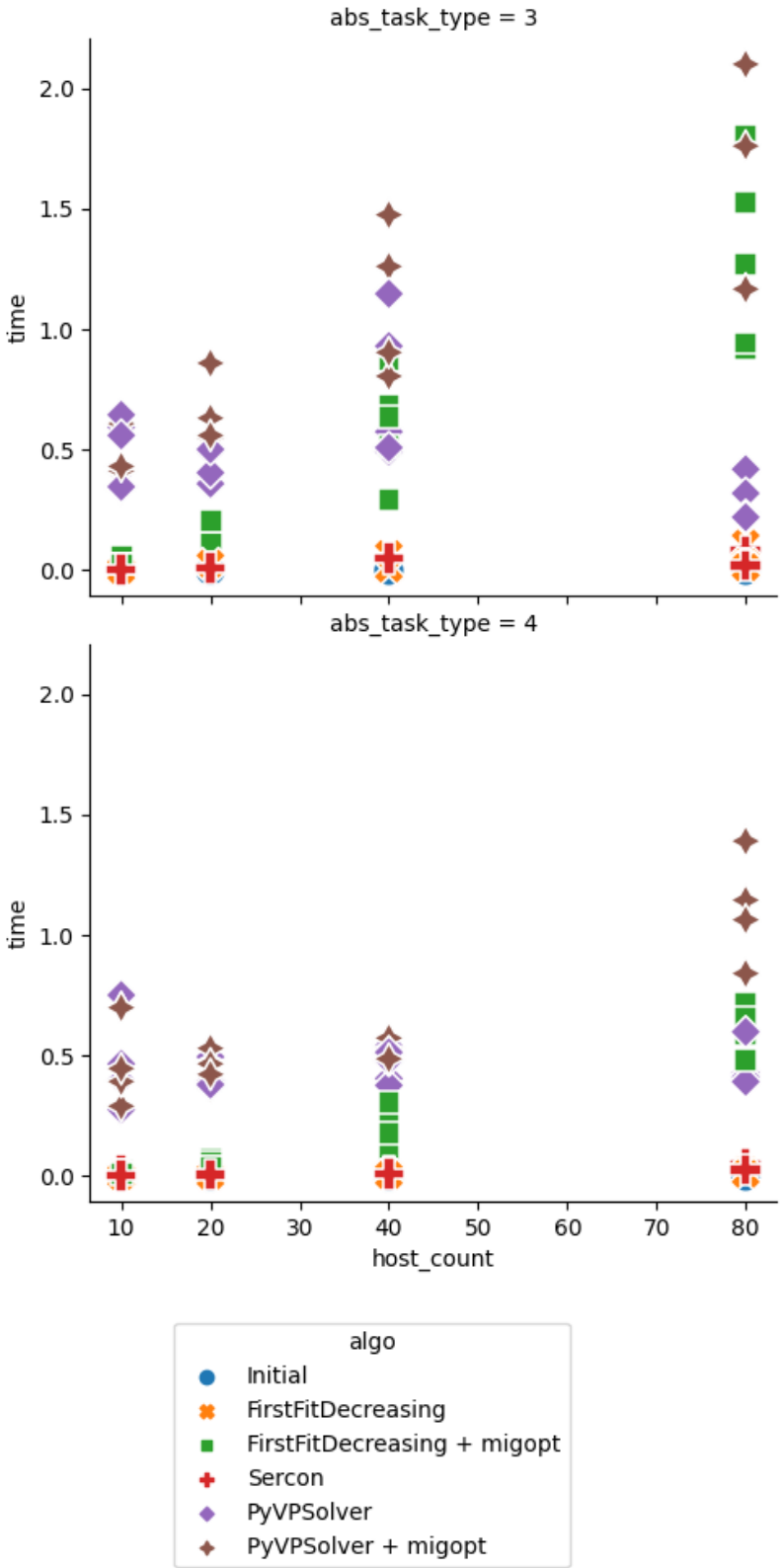
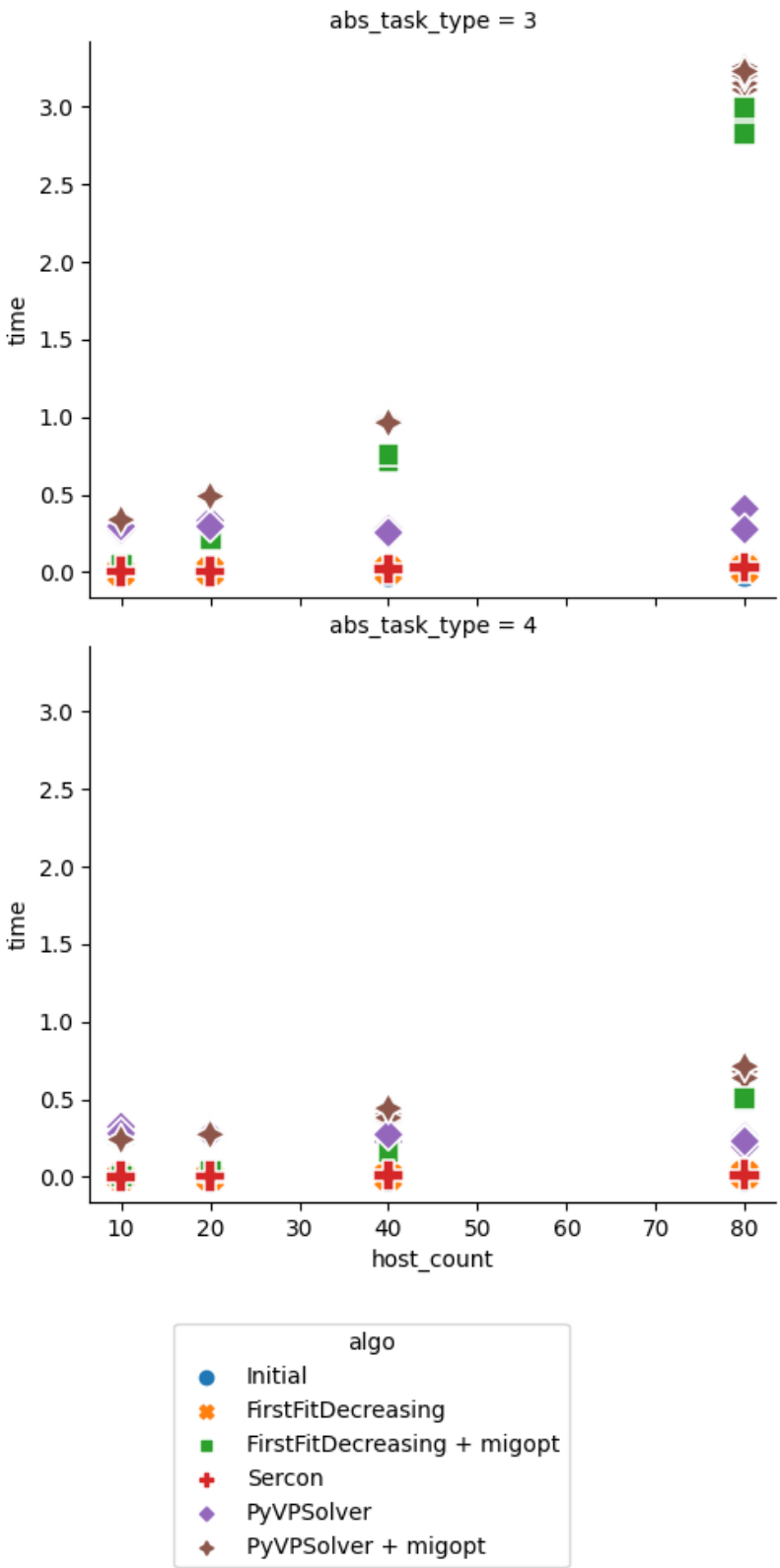


Figure 5.4: Working times on shrink tests



Figures 5.1, 5.2, 5.3, 5.4 are made from experiments results in *results.csv*. The absolute value of the task type corresponds to the virtual machines distribution type. Shrink means the choice between shrink and full-pack tests.

5.5 Discussion

Algorithm performance matches expectations: an increase in *migr* score is necessary for lowering the *active* score. Migration optimizer consistently lowers *migr* score without changing *active* score. The one unexpected result is that VPSolver provides generally smaller *migr* score while having no information about the previous mapping. The complete explanation of this phenomenon is unknown, a possible source of this behavior can be in the way how MIP solver branches, filling first hosts first.

Working time results also meet their expectation. Migration Optimizer is the longest one since it uses Kuhn algorithm with cubic complexity. The second slowest is VPSolver, being exponential, but thoroughly optimized.

Sercon is the winner of shrink tests due to its simplicity and field experience. Full-pack tests are harder and First-Fit-Decreasing + migopt provides to be the victor.

6 Conclusion

This paper presents a polynomial-time approximated algorithm for the virtual machine consolidation problem. Possible future improvements include (but do not limit to):

- Using a more sophisticated heuristic for the first step.
- Migration score optimization using more than one flavor.
- Taking into account balancing score: how far the load distribution across the hosts from uniform.
- Taking into account fixed VM amount score: how many VMs on certain size can be put into this configuration.

References

1. Dávid Bartók and Zoltan Mann. "A branch-and-bound approach to virtual machine placement". In: Jan. 2015.
2. Filipe Brandão. *VPSolver 3: Multiple-choice Vector Packing Solver*. 2016. arXiv: 1602.04876 [math.OC].
3. Stephanie Challita, Fawaz Paraiso, and Philippe Merle. "A Study of Virtual Machine Placement Optimization in Data Centers". In: Apr. 2017. DOI: 10.5220/0006236503430350.
4. Christopher Clark et al. "Live Migration of Virtual Machines". In: *Proceedings of the 2nd Conference on Symposium on Networked Systems Design&Implementation - Volume 2*. NSDI'05. USA: USENIX Association, 2005, pp. 273–286.
5. R Gilmore and Ralph Gomory. "A Linear Programming Approach to the Cutting Stock Problem I". In: *Oper Res* 9 (Jan. 1961). DOI: 10.1287/opre.9.6.849.
6. Fabien Hermenier et al. "Entropy: a Consolidation Manager for Clusters". In: *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'09* (Mar. 2009). DOI: 10.1145/1508293.1508300.
7. Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. 5th. Springer Publishing Company, Incorporated, 2012. ISBN: 3642244874.
8. H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics (NRL)* 52.1 (Feb. 2005), pp. 7–21. DOI: 10.1002/nav.20053. URL: https://ideas.repec.org/a/wly/navres/v52y2005i1p7-21.html.
9. Vimal Mathew, Ramesh Sitaraman, and Prashant Shenoy. "Energy-Aware Load Balancing in Content Delivery Networks". In: *Proceedings - IEEE INFOCOM* (Sept. 2011). DOI: 10.1109/INFCOM.2012.6195846.

10. Aziz Murtazaev and Sangyoon Oh. "Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing". In: *IETE Technical Review* 28 (May 2011), p. 212. DOI: 10.4103/0256-4602.81230.

11. Rina Panigrahy et al. "Heuristics for Vector Bin Packing". Jan. 2011. URL: https://www.microsoft.com/en-us/research/publication/heuristics-for-vector-bin-packing/.

12. Christos Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Vol. 32. Jan. 1982. ISBN: 0-13-152462-3. DOI: 10.1109/TASSP.1984.1164450.