Data Center Optimization Using Virtual Machine Profiles

Guilherme Quintino
guilherme.quintino@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2019

Resumo

The Virtual Machine (VM) consolidation problem consists on finding a placement of a group of VMs in physical servers in order to optimize a certain objective, usually to minimize the number of active servers. The VM consolidation problem has been proven to be NP-Hard, which means it is important to have a good algorithm to approximate an optimal solution. The recent growth of Cloud Services, and the consolidation around a small number of very large cloud providers, means that a scalable way to approximate an optimal solution is even more important. In this document, we cover the different approaches used to solve the VM consolidation problem. We propose the use of VM profiles, a VM's trace of resource usage variation over time, as a way to improve upon the solutions found by traditional algorithms. This improves the solution due to the fact that VMs do not fully use the resources allocated to them all the time. Considering the resource usage of a VM to vary over time, allows for only the necessary resources to be allocated to that VM. This allows more VMs to be placed in a server without overloading it, minimizing the number of active servers. We provide a formal definition of the problem, describe our solution, detail experimental results and discuss their significance.
Keywords: Constraint Satisfaction, Multi-Objective Combinatorial Optimization, Virtual Machine Consolidation.

1. Introduction

The scalability and low entry and upkeep costs associated with cloud services caused their usage to steadily increase over the years. Due to economies of scale, hardware components can be acquired for a lower price by buying in large quantities. The costs associated with managing a data center scale in a similar way. The problem lies with the fact that the price of electricity is less flexible than other costs, which means that it is the main cost factor in modern large scale data centers. This has caused an increasing attention to be put on the optimization of a data center's efficiency, in order to minimize operating costs.

In order for a data center to run efficiently, it is important to keep the least amount of servers active. For this to happen it is important to decide which servers handle which tasks, commonly known as the Virtual Machine (VM) placement problem. The VM placement or consolidation problem, tries to solve what is the best way for VMs, or tasks, to be assigned to available servers in order to optimize a certain goal, usually power consumption. In the real world it is common to optimize for more than a single objective, making the problem far more complex. In this work, we propose the use of VM profiles, i.e. data that traces the variation of a VM's resource usage over time, as a way to improve upon the solutions of traditional optimization methods.

The structure of this thesis is as follows. Section 1 provides an introduction to the problem of VM consolidation as well as motivation for the importance of better ways of solving it. Section 2 details the fundamental concepts that are necessary to understand this thesis. Section 3 provides an overview of related work for the VM consolidation problem and briefly describes a few state of the art algorithms for its solution. Section 4 outlines an adapted VM consolidation model that considers the resource utilization of a VM to vary over time. Section 4 also describes the way data was collected and processed in order to create the VM profiles. Section 5 covers the experimental results and discusses their significance. Section 6 concludes the document with final remarks on the work presented in this thesis as well as some suggestions of future work.

2. Preliminaries

In this section, we describe the basic concepts that are underlying the work that will be presented th-

roughout this thesis.

2.1. Optimization

2.1.1 Pseudo-Boolean Optimization

Let $X = \{x_1, x_2, ..., x_n\}$ be a set of n Boolean variables such that $\neg x = 1 - x$. Pseudo-Boolean Optimization aims to find the minimum of a function of X (objective function) while subject to certain constraints, and is defined as:

$$\min \sum_{i=1}^{n} a_i \times x_i$$

Such that the solution respects a set of m pseudo-Boolean constraints of the form:

$$\sum_{i=1}^{n} b_{ij} \times x_i \bowtie c_j, \ \bowtie \in \{\geq, \leq, =\} \tag{1}$$

where $j \in \{1, ..., m\}$ and $b_{ij}, c_j \in \mathbb{Z}$.

2.1.2 Multi-Objective Combinatorial Optimization

In certain problems it is desirable to optimize multiple cost functions whose importance can not be consolidated in a single weighted objective function. Referred to as multi-objective combinatorial optimization, it is similar to pseudo-Boolean Optimization described previously, although with multiple objective functions.

Let n be the number of variables and m be the number of objective functions, multi-objective combinatorial optimization is defined as follows:

$$\min \sum_{i=1}^{n} a_{i_1} \times x_i$$

$$\dots$$

$$\min \sum_{i=1}^{n} a_{i_m} \times x_i$$

Such that the solution respects a set of constraints of the same form as the ones formulated in Equation 1.

Let $F = \{f_1, f_2, ..., f_m\}$ be a set of objective functions to minimize and $C = \{c_1, c_2, ..., c_p\}$ a set of pseudo-Boolean constraints to satisfy. Consider two different assignments that satisfy C, a and b, if $\forall_{f \in F} f(a) \leq f(b)$ and $\exists_{f \in F} f(a) < f(b)$ then a dominates b, formally $a \prec b$. If there is no assignment d that satisfies C such that $d \prec a$, a is said to be non-dominated or Pareto-optimal [12]. The group of all Pareto-optimal solutions is called the Pareto front, which is the solution to a multi-objective combinatorial optimization problem.

2.1.3 Minimal Correction Subsets and Multi-MCSs

A hard constraint is a constraint that must be satisfied. A soft constraint is a constraint that has a cost associated with not satisfying it. Consider an unsatisfiable set of pseudo-Boolean constraints $F = (F_H, F_S)$, such that F_H are the hard constraints and F_S the soft constraints. A Minimal Correction Subset (MCS) of F is a minimal subset $C \subseteq F_S$ so that $F \setminus C$ is satisfiable. For an unsatisfiable set of pseudo-Boolean constraints $F = (F_H, F_S)$ and a subset $C \subseteq F_S$, C is an MCS of F only if $F_H \cup (F_S \setminus C)$ is satisfiable and for each constraint $c \in C$, $F_H \cup (F_S \setminus C) \cup \{c\}$ is unsatisfiable.

In a multi-objective combinatorial optimization problem there is a set of hard constraints and a set of soft constraints for each objective. Multi-MCSs are an extension of MCSs to multi-objective formulas [15]. For a given multi-objective combinatorial optimization problem $W = (F_H, O_S)$, with O_S being the objective function set such that $O_S = \{F_{S1}, ..., F_{Sl}\}$ and C a tuple of constraint sets such that $C = (C_1, ..., C_l)$ and $C_i \subseteq F_{Si}$, with $1 \leq i \leq l$. For C to be a multi-MCS of W, $F_H \cup \bigcup_{i=1}^{l} (F_{Si} \setminus C_i)$ must be satisfiable and for each constraint $c \in \bigcup_{i=1}^{l} C_i$, $F_H \cup \bigcup_{i=1}^{l} (F_{Si} \setminus C_i) \cup \{c\}$ must be unsatisfiable.

The definition of domination for multi-objective combinatorial optimization solutions can be extended to multi-MCSs. For a given multi-objective combinatorial optimization problem $W = (F_H, O_S)$, consider two multi-MCSs of W, $C = (C_1, ..., C_l)$ and $C' = (C'_1, ..., C'_l)$. C dominates C', formally $C \prec C'$, only if $\forall_{1 \leq i \leq l} \sum_{(c,w) \in C_i} w \leq \sum_{(c',w') \in C'_i} w'$ and $\exists_{1 \leq j \leq l} \sum_{(c,w) \in C_j} w < \sum_{(c',w') \in C'_j} w'$.

2.2. Virtual Machine Consolidation

The VM consolidation problem consists on consolidating the VMs from multiple low usage servers into a smaller number of high usage servers. This problem aims to find in which server each VM should be placed in order to optimize one or more goals. There are many different approaches, but due to the complexity of the problem, most approaches make some compromises to model the problem in a way that can be solved efficiently. There are many variations between models of the problem, as they differ in complexity, the goals to be optimized, the resources considered in defining server capacity, and the constraints that a VM placement must satisfy [8].

The capacity of a server is most often considered to be just a single resource. The most commonly used resource to define server capacity is

CPU usage followed by memory usage. We will consider both CPU and memory usage.

The most common, and often only, goal is to minimize the number of active servers. Minimizing or limiting the number of VM migrations and minimizing resource wastage are also common [15]. There are many other objectives that can be optimized in the VM consolidation problem. Another example would be to ensure an even power distribution across the data center, in order to avoid excessive power dissipation in a small area, that could cause cooling problems.

A VM migration is the operation of moving a running VM from one server to another. A migration uses resources of the source and target servers which leads to an increased load in the data center. VMs that are not yet running on a server do not require a migration and can therefore be placed on any machine. A particular case of the VM consolidation problem is the reassignment problem [10], in which all VMs are already running on some server.

2.3. Data Analysis

In this subsection we cover the topics necessary to understand the data processing stage of this work.

2.3.1 Time Series

A time series is a sequence of data points ordered by the time they were registered. Time series are often generated by measurements performed at a fixed interval. They are used in many fields of science, usually when they involve the collection of time-sensitive data. A line chart is the most common visual representation for a time series.

2.3.2 Barycenter

In a time series, a Barycenter is the time series generated by the arithmetic mean of the data points between two or more time series for each time interval. Barycenters are used when there is the need to aggregate multiple time series into a single one. A common use case is when multiple measurements are performed and they need to be averaged to obtain a final result.

2.3.3 Dynamic Time Warping

Dynamic Time Warping (DTW) is an algorithm used to determine the optimal match between two time series. For time series a and b, it works as follows: Every data point in time series a must be mapped to one or more data points in b and every data point in time series b must be mapped to one or more data points in a. The first data point of a must be matched to the first data point of b. The last data point of a must be matched to the last data point of b. The matches between time series points cannot go backwards, so if $a[x]$ matches to $b[y]$, no new points can be matched to $a[z]$ for any $z < x$, or to $b[w]$ for any $w < y$. The chosen match is the one that has the lowest absolute difference between the value of the data points. DTW can be used in conjunction with a Barycenter [13] in order for the resulting time series to be less sensitive to the alignment of the data as well as providing a better representation of irregular time series.

3. Related Work

Several algorithms have been proposed for multi-objective combinatorial optimization. In this section we briefly review the most relevant ones.

3.1. NSGA-II

Genetic operators simulate the way genes are passed on from generation to generation. The crossover operator combines two parent solutions to create a new child solution composed of parts of each parent solution. The mutation operator makes a random change to a solution. NSGA-II [3] is a multi-objective evolutionary algorithm that starts with a randomly generated parent population. Genetic operators are applied to the parent population to create a child population. The parent and child populations are merged and sorted. A new parent population, that corresponds to a new generation, is created with the best of both populations. This process is repeated until a stopping condition is reached.

3.2. MOEA/D

MOEA/D [17] is a multi-objective evolutionary algorithm that divides the multi-objective problem into multiple single objective subproblems by optimizing a weighted sum of the objectives. The set of weights used in the weighted sum of the objectives is a weight vector, and it is different for each subproblem. Each subproblem starts with a randomly generated solution. A new candidate solution is created for each subproblem by applying genetic operators to the solutions of other subproblems. The candidate solution is compared to the original subproblem's solution and to the solutions of the other subproblems. The new solution replaces the current solution if it is superior to it.

3.3. VMPMBBO

Migration is an operation that is particular to biogeography based methods and consists in replacing a part of one solution with a part of another solution. VMPMBBO [18] is a biogeography based multi-objective evolutionary algorithm that divides a multi-objective problem into single objective subsystems. Each subsystem, also referred to as an island, keeps a population of N solutions and optimizes only one of the objectives of the problem.

Migration along with mutation are applied to each subsystem to create new a population of solutions. In each subsystem, the best solutions from the previous generation are used to replace the worst solutions in the population.

3.4. Pareto Minimal Correction Subsets

A Pareto Minimal Correction Subset [15] is a non-dominated multi-MCS. Consider a multi-objective combinatorial optimization problem W, and C, a multi-MCS of W. C is only a Pareto-MCS if a multi-MCS D, such that $D \prec C$, does not exist. Pareto-MCSs have been proven to be able to approximate the set of Pareto-optimal solutions for a multi-objective combinatorial optimization problem in the same way that MCSs approximate solutions to single objective optimization problems. By enumerating all the multi-MCSs of the problem and selecting the non-dominated ones, it is possible to find all the Pareto-optimal solutions. This turns a multi-objective combinatorial optimization problem into an MCS enumeration problem, which allows algorithms for MCS enumeration [2, 5, 7, 9, 11] to be used to solve multi-objective combinatorial optimization problems.

3.5. Integer Linear Programming and Hybrid Methods

Integer linear programming defines the problem formally using mathematical programming. Once defined, the problem is solved using a mathematical solver. Integer programming is an exact method that is able to guarantee that an optimal solution will be found.

Hybrid methods are a type of approach that uses multiple algorithms. GeNePi [14] is an example of this type of algorithm. GeNePi uses GRASP [6], a variation of a greedy search that includes a random factor, to optimize different weighted sums of the objectives to generate a population of solutions. This population is used as a starting point for the NSGA-II algorithm. Next, Pareto local search [1], a variation of local search, is used to find similar solutions that improve upon the ones found by NSGA-II. Using different algorithms allows GeNePi to continue to improve the solution quality, without getting stuck due to the shortcomings of a particular algorithm.

4. Adapted Model, Data Collection and Processing

In this section we describe the adapted VM consolidation model as well as the data collection and processing stages, from which result the virtual machine (VM) profiles to be used in the optimization algorithms.

4.1. Adapted Virtual Machine Consolidation Model

To define the problem, we follow the model from Terra-Neves *et al* in [15] and extend it to consider that the resource usage of a VM varies over time. Let $J = \{j_1, j_2, ..., j_m\}$ be a set of m jobs to be run in a data center, with each job being made up of several VMs. For a job $j \in J$, we use $V_j = \{v_1, v_2, ..., v_{k_j}\}$ to represent the group of k_j VMs in that job. To represent the set of all VMs we use $V = \bigcup_{j \in J} V_j$. For a VM $v \in V$, and a time period $t \in T$, $req_{CPU}(v,t)$ and $req_{RAM}(v,t)$ are, respectively, the CPU and memory requirements of VM v for time period t. The average resource utilization of resource r for VM v is represented as $avg\_req_r(v)$ and is given by:

$$avg\_req_r(v) = \frac{\sum_{t \in T} req_r(v,t)}{|T|}.$$

Let $S = \{s_1, s_2, ..., s_n\}$ be the set of n servers in the data center. For each server $s \in S$, we use $cap_{CPU}(s)$ and $cap_{RAM}(s)$ to represent, respectively, the CPU and memory capacity of server s. A server that has one or more VMs placed on it is active, and a server with no placed VMs is inactive.

We use $ld_r(s,t)$ to represent the sum of the requirements of a certain resource $r \in \{CPU, RAM\}$ of all the VMs placed in server s for time period t. The average server load over all time periods for resource r of server s is represented as $avg\_ld_r(s)$ and is given by:

$$avg\_ld_r(s) = \frac{\sum_{t \in T} ld_r(s,t)}{cap_r(s) \times |T|}.$$

The average residual capacity for resource r of server s, represented as $avg\_rsd_r(s)$, is computed as:

$$avg\_rsd_r(s) = 1 - avg\_ld_r(s).$$

We consider the energy consumption of a server to be a linear function of its CPU load according to [4] and [18]. We use $enr_{idle}(s)$ to represent the energy consumed by s when it is does not have any VMs placed on it and $enr_{full}(s)$ to represent the energy consumed by s when its capacity is fully utilized by VMs. If $avg\_ld_{CPU}(s) = 0$, it means server s does not have any VMs placed on it and is therefore inactive. Inactive servers are considered to not consume any energy. For an active server s, its energy consumption is given by:

$$energy(s) = (enr_{full}(s) - enr_{idle}(s)) \times avg\_ld_{CPU}(s) + enr_{idle}(s). \quad (2)$$

The way VMs are distributed across servers will leave different amounts of resources left in each server. If the available amounts of a server's resources are unbalanced it could prevent more VMs from being assigned to that server. An example would be when a server cannot run any more VMs

because it does not have any memory left, despite its low CPU utilization. This is known as resource wastage, which is computed as:

$$wastage(s) = |avg\_rsd_{CPU}(s) - avg\_rsd_{RAM}(s)|.$$

In the VM consolidation problem, the following constraints must be satisfied: each VM $v \in V$ must be placed in only one server and for each server $s \in S$, the sum of the amount of resource $r \in \{CPU, RAM\}$ required by each VM placed in s, for time period $t \in T$, cannot exceed the capacity of s of resource r. To make an application more resilient to failure, it may be necessary to ensure redundancy by having the application's VMs assigned to different servers. This is ensured by the anti-collocation constraint: given a job $j \in J$ and two VMs $v_1, v_2 \in V_j$, then v_1 and v_2 must not be assigned to the same server.

To indicate if a server $s_k \in S$ is active or inactive we use a Boolean variable y_k such that $y_k = 1$ if s_k is active and $y_k = 0$ otherwise. To indicate if a VM $v_i \in V$ is placed on a server $s_k \in S$, we use a Boolean variable $x_{i,k}$, such that $x_{i,k} = 1$ if VM v_i is placed on server s_k. Otherwise, $x_{i,k} = 0$.

For a resource $r \in \{CPU, RAM\}$, the load of a server $s_k \in S$, for time period $t \in T$, is computed as:

$$ld_r(s_k, t) = \sum_{v_i \in V} req_r(v_i, t) \times x_{i,k}.$$

Let n be the number of servers in a data center and m the number of VMs to be assigned to those servers. We define the multi-objective VM consolidation problem as a minimization, respectively, of the energy consumption, the resource wastage and the migration costs:

$$\min \sum_{s_k \in S} energy(s_k)$$

$$\min \sum_{s_k \in S} wastage(s_k)$$

$$\min \sum_{(v_i, s_k) \in M} avg\_req_{RAM}(v_i) \times \neg x_{i,k}$$

Such that it must satisfy the constraints that follow. Each VM has to be placed in only one server:

$$\sum_{s_k \in S} x_{i,k} = 1 \quad i \in \{1, ..., m\}.$$

The resource capacities of each server cannot be exceeded:

$$\forall t \in T,\ ld_r(s_k, t) \leq y_k \times cap_r(s_k)$$
$$k \in \{1, ..., n\},\ r \in \{CPU, RAM\}. \quad (3)$$

The migration budget cannot be surpassed:

$$\forall t \in T, \sum_{(v_i, s_k) \in M} req_{RAM}(v_i, t) \times \neg x_{i,k} \leq b.$$

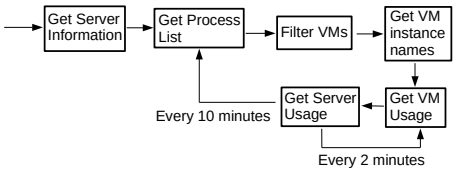

Figura 1: Steps of the data collection process.

Anti-collocation for all VMs in the same job must be ensured:

$$\sum_{v_i \in V_j} x_{i,k} \leq 1 \quad j \in J, k \in \{1, ..., n\}.$$

Finally, all problem variables are Boolean:

$$x_{i,k}, y_k \in \{0, 1\} \quad i \in \{1, ..., m\},\ k \in \{1, ..., n\}.$$

4.2. Data Collection

4.2.1 Simple Network Management Protocol

Simple Network Management Protocol (SNMP) is a protocol part of the Internet Standard that is used for managing devices over IP networks. SNMP is able to collect and modify information on managed devices. Changing device information with SNMP enables devices to be reconfigured remotely.

SNMP exposes data from each device as variables on the managed systems. The variables exposed by SNMP are organized in a hierarchical structure. These structures are defined as a management information base (MIB) that describes the structure of data available for each device. MIBs use a hierarchical namespace composed of object identifiers (OID). An OID represents a variable that can be read or set via SNMP.

In the SNMP context, a manager is a computer that performs monitoring or management tasks. Each managed device runs a software component called an agent that is responsible for reporting information to the manager via SNMP.

4.2.2 Data Collection

To create VM profiles, it is necessary to gather information on the variation of VM resource utilization throughout the day. Therefore, we created a Python script to request and register performance data from the servers, about the VMs executing, using PySNMP (a python SNMP library), as the information was accessible via SNMP. The script was placed in a VM inside the data center's OpenStack, and the VM's IP was whitelisted in order to be able to request SNMP data from the servers in the data center.

Figure 1 shows a diagram of the steps taken in the data collection process. The script starts by requesting from each server information about the hardware configuration. This includes CPU model

and its number of threads, as well as total available memory. To get performance data from the VMs on the server, we start by requesting the process list from each physical server. The process list is then filtered by processes that have the name "qemu-kvm", which correspond to VMs. The VMs' Process Ids (PIDs) are then used to get process parameters, from which we get each VM's instance name, which is used as a VM's identifier when logging performance data.

After the initialization, the script will request performance data for all VMs every 2 minutes and look through the process list for new VMs every 10 minutes. This timing was chosen as not to cause excessive load on the data center, while still maintaining a sampling interval short enough for the data to be able to represent short resource consumption peaks accurately. The sampling rates for VM performance data and to look for new VMs are different as a VM's performance varies a lot faster and more often than new VMs are launched. Another reason for this is that requesting the full process list is more demanding for the servers than just requesting data from each of the already known VM processes. The script registers information about CPU and memory usage for each VM as well as CPU load, memory utilization and temperature for each server. The CPU load for a VM is given as the total amount of CPU time used by the VM up to that point, in centi-seconds. This requires the script to keep track of the last value for each VM in order to determine the CPU time used by the VM since the last request. The memory usage for VMs and servers is given in KB. The temperatures are given in degrees Celsius with accuracy of one degree. Each data point also has a timestamp to allow the data to be associated to a certain time period. The server's utilization metrics were used to check that the servers were not overloaded, which was confirmed to be true. The server's temperature was used to check for any overheating issues or hotspots, which proved not to be a problem, as the temperatures remained within the expected range.

4.3. Data Processing

The data processing stage aims to use the collected data to create a profile that may accurately represent a VM's expected behaviour. The VM load was not as steady as expected, instead registering very high load spikes when compared to its baseline utilization. The load peaks were still detected at similar times of the day, and even though some deviations are present, there is still a temporal pattern in resource utilization.

The first intuition on a good way to represent a VM's load variation over time was to use a barycenter. The barycenter was determined using the resource utilization data aligned by time of day as a time series. This proved to be ineffective as the data was less steady than expected, presenting very large utilization peaks under a very short period of time followed by large periods of low to moderate utilization. These peaks occured at roughly the same time of day but the low utilization data points were a lot more common and that meant the barycenters ended up representing the peaks as much smaller than they were originally. This is a problem with the barycenter concept as it averages out values for a typical behaviour as opposed to considering a worst case scenario. Even for rare peaks we aim to represent the worst case scenario for a VM's load, as overloading a server can cause it to become unresponsive or crash, which would lead to critical services becoming unavailable. The fact that some peaks were only registered once a week means that the majority of the data points are at baseline utilization and peaks are lost in the barycenter representation. DTW barycenters, as described in subsection 2.3.3, were a better representation of the data but were still far from what was desired, as the barycenter was simply not compatible with low frequency peaks. To better represent the worst case scenario for a VM's resource utilization we determined the maximum value of resource utilization within a certain time window. As peaks will not always occur at exactly the same time, when computing the maximum, we also consider values outside the time window up to a certain point. The maximum values were calculated for every 1 hour period.

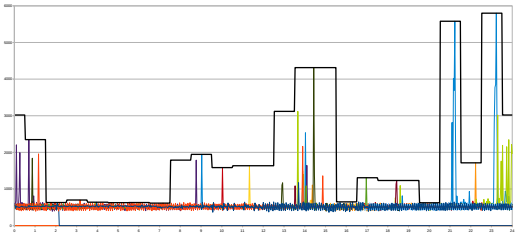

Figura 2: Cpu usage and Maximum values for each hour for VM_8509.

In figure 2, the CPU utilization data of VM_8509 is shown as a time series, with each color representing a different day. The x axis represents 2 minute time slices and the y axis represents the value of CPU used in that time slice, in centi-seconds. A Barycenter is a poor representation of the bursty nature of the data collected as it flattens most deviations from baseline load. Figure 2 also shows a discretization of the data as a set of 24 values, one for each hour. In hour 8 there are no usage peaks but the maximum is above the baseline utilization. This is because the maximum for each hour

also considers peaks 15 minutes before and after that hour, to account for temporal variability in load peaks.

This processing was only done to the CPU utilization data. The VMs' memory utilization showed minor variations which means that there is very little to be gained by considering the variation of memory usage over time. Thus, memory usage of a VM can be represented as a constant value without reducing the quality of the solutions found by optimization algorithms. Making the memory constant makes the placements easier as it reduces the number of restrictions the algorithm has to consider. The memory requirements of a VM were assumed to be the maximum value registered during the entire logging period.

The VM CPU usage data was measured in centiseconds of CPU time. To use this metric in VM placement we assumed the total CPU capacity of a server to be:

$$ServerCapacity = NrServerThreads \times SamplingPeriod \times 60 \times 100 \quad (4)$$

$ServerCapacity$ is the amount of centi-seconds of CPU time the server has available during every $SamplingPeriod$. $NrServerThreads$ is the number of threads the server has available. $SamplingPeriod$ is the time interval at which VM loads were registered, in minutes. The result is multiplied by 60 to turn minutes into seconds and then multipied by 100 to turn seconds into centiseconds. Both CPU and memory were considered to have a 10% overhead that cannot be used to place VMs, which means the final capacities were 90% of the values described above. The data was then used to generate a file with the optimization problem defined as a set of pseudo-Boolean constraints, with the objective being to minimize the number of active servers.

5. Experimental Results

In this section we present and discuss the experimental results of a series of tests that we ran, in order to evaluate our work. We tested the usage of VM profiles with pseudo-Boolean optimization (PBOVMP) along with traditional pseudo-Boolean optimization (PBO) and First Fit (FF). We compare the optimization with VM profiles to traditional optimization in order assess whether there is a real benefit in the use of VM profiles. The First Fit approach, in which VMs are placed in the first server that they fit into, is a good representation of the way a VM placement would be done manually by a system administrator or by a simple automated process.

This section is organized as follows. We start by detailing the experimental setup under which our experiments were conducted. Afterwards, we present and discuss the results, both for single objective and multi-objective optimization.

5.1. Experimental Setup

5.1.1 Configuration

All the tests were run in a server with a dual socket Intel Xeon Silver 4110 CPU and 64 GB of memory. The tests were set with a 1 hour time limit. The First Fit algorithm was implemented in Python while the pseudo-Boolean formulations were generated using a Python script and solved using a pseudo-Boolean solver.

PBOVMP had 24 time slots for CPU usage, one per hour, while memory usage was assumed to be constant, with the maximum value recorded used as the VM's memory requirement. For PBO, we used the maximum value of the 24 values in the VM profile as CPU usage and used the same memory requirement as PBOVMP. For First Fit we used the same resource usage values as PBO. For a multi-objective problem we optimized the uniform distribution of CPU load across active servers as well as the number of active servers.

The server capacity on the problem formulation matches the hardware from the servers used in the OpenStack Virtualization infrastructure of Instituto Superior Técnico, Universidade de Lisboa. This was chosen so as to make the experiments closer to a real world scenario, moreover since the VMs for which we created profiles were running in that same virtualization infrastructure. The servers have a dual socket Intel Xeon E5-2630 v4 with 40 threads and 128 GB of memory.

5.1.2 Datasets

For test data, we used the data we collected while creating VM profiles. The initial dataset had logged the resources consumed by 132 VMs over the period of 1 month. There were more VMs active during this time period, but the creation of VM profiles requires at least 1 day of data. This means that VMs that were active for less than 1 day were not considered. In order to be able to conduct testing with a larger dataset we have generated additional synthetic VM profiles. The goal when generating such profiles was for them to be similar to the real VM traces that we had collected, in order to prevent testing with unrealistic data. We generated VMs to complete the original dataset to 150 VMs, and then proceeded to generate more VMs for datasets with sizes of 300, 450, 600, 750 and 1500 VMs.

To generate a new VM profile, we randomly select any VM in the original dataset. We randomly generate a number between 0 and 23, and then shift the CPU values to the right by that amount.

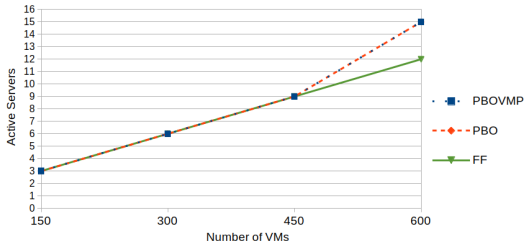

Figura 3: Active servers for different dataset sizes without a CPU multiplier.

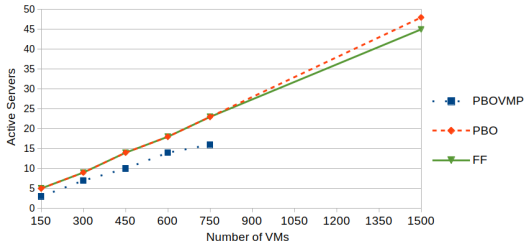

Figura 4: Active servers for different dataset sizes with a $\times 4$ CPU multiplier.

For example, for a shift right of 1, a CPU value for hour 8 would be in hour 9 of the new VM, while the value of hour 24 would move to hour 1. Next, we randomly generate two numbers between 0.75 and 1.25 and multiply one by the CPU values and the other by the memory value. The new VM profile is now finished and is added to the new dataset. This process is repeated until the desired number of VMs is reached. The values 0.75 and 1.25 were chosen in order to create variation in the data without creating unrealistic VMs that are too large or too small.

We observed that the dataset we had collected was quite light on processor usage and, as a result, all the generated datasets were also light on processor usage. Due to the VM profiles only considering variation of CPU load, a dataset that is light on CPU usage would make PBO and PBOVMP reach similar solutions, as the placement of VMs would be dependent only on memory. To simulate a more CPU intensive workload, we applied different multipliers to the CPU requirements of all time periods of every VM in a dataset. The multipliers were $\times 2$, $\times 3$, $\times 4$ and $\times 5$.

5.2. Test Results

5.2.1 Single Objective Optimization

The results reported in this subsection correspond to tests that aim to place the VMs in a way that minimizes the number of active servers. The optimization algorithms that were employed were PBOVMP, PBO and First Fit.

Figure 3 shows the results for the datasets without any CPU multiplier. Results for the datasets with 750 and 1500 VMs are not shown as neither of the pseudo-Boolean optimization methods were able to find a solution within the 1 hour time limit. This is due to the symmetry of the problem, as this problem has many similar solutions. In this case, many similar VMs can be swapped between servers to obtain a similar placement, making the search for a solution a lot more difficult, as many similar placements need to be tested. The servers all have the same amount of available resources, which also contributes to the symmetry of the problem. The symmetry of a problem gets worse as the problem size gets larger and leads to the solver not finding solutions for the larger datasets. To allow the algorithm to scale to larger problem sizes, symmetry breaking techniques [16] could be added to the problem formulations, as they have proven to provide a significant improvement when solving symmetric problems. The results show that this dataset is light on CPU usage and the placements are only memory limited, as PBOVMP and PBO perform equally well, and they differ only in whether or not they consider CPU load variation. The results show the need for CPU multipliers to simulate scenarios with more demanding CPU usage. Both pseudo-Boolean formulations perform worse as problem size increases as they cannot get close to an optimal solution within the time limit. Due to the large amount of VMs on each server, we can conclude that each VM is relatively small, which leads to the wastage caused by First Fit to be reduced. This enables the algorithm to find optimal solutions for many tests, which in most scenarios would not be expected of such a simple algorithm. Results for CPU multipliers $\times 2$ and $\times 3$ were similar to the ones without a CPU multiplier.

Figure 4 shows the results for the datasets with a $\times 4$ CPU multiplier. PBOVMP was not able to find a solution for the dataset with 1500 VMs within the time limit. The results show that the placements for this problem are now mostly CPU limited, as First Fit does well in one-dimensional problems, with both PBO and First Fit reaching the same optimal solutions. The exception is when the problem gets larger and PBO can no longer get to the optimal solution within the time limit. The CPU heavy dataset showcases the effectiveness of PBOVMP as it finds the best solutions of all algorithms. The lower complexity of PBO enables it to solve larger problems than PBOVMP.

Figure 5 shows the results for the datasets with a $\times 5$ CPU multiplier. The results show that the problem is now completely CPU dependent, as First Fit and PBO perform similarly. In this instance, PBOVMP dominated as it significantly outperformed the other algorithms and was able to find solutions for the larger problem sizes. The CPU heavy tests show how the efficient placements that result

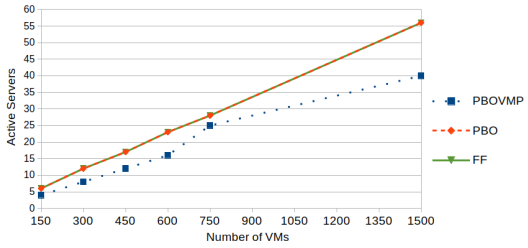

Figura 5: Active servers for different dataset sizes with a $\times 5$ CPU multiplier.

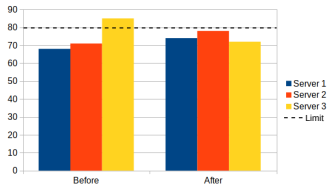

Figura 6: Maximum server load after balancing for 150 VMs with a $\times 3$ CPU multiplier.

from the usage of VM profiles can lower the number of active servers significantly.

5.2.2 Multi-objective Optimization

The results reported in this subsection correspond to tests that aim to place the VMs, while minimizing the maximum CPU load, in order to distribute the load uniformly across the active servers, as well as minimizing the number of active servers. The uniform load distribution was a feature requested by the management team of IST's OpenStack Virtualization infrastructure in order to prevent large amounts of heat from being produced by a single server. This creates hotspots in the data center that can lead to cooling problems in that specific region. The multi-objective optimization tests have two stages: first we solve the problem as in subsection 5.2.1, in order to minimize the number of active servers. Then, we define the problem with only that number of available servers, and progressively lower the maximum CPU capacity of all servers until the problem is unsatisfiable or no solution can be found within the time limit. When the algorithm reaches an unsatisfiable problem or a timeout occurs, it returns the placement of the last satisfiable formulation. Reducing the maximum capacity of all servers forces a more balanced VM distribution across them.

We used 10% steps as it was a good balance between providing a meaningful optimization improvement without requiring an excessive amount of time. Problems using datasets with sizes of 750 and 1500 did not always find a solution within the time limit, so they were not used in these tests. Figure 6, shows the maximum load as percentage of available capacity for each server, from VM placements before and after load distribution. The figure

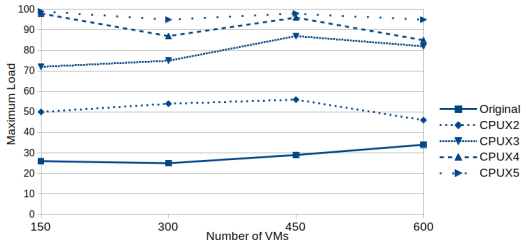

Figura 7: Maximum server load after balancing for different dataset sizes and CPU multipliers.

is from the load distribution of the dataset of 150 VMs, for a multiplier of $\times 3$. The results show how the load distribution stage is able to improve the uniformity of load distribution.

Figure 7 shows the load distribution results, of maximum server load in percentage of available capacity, achieved for different dataset sizes and CPU multipliers. The results show how that as the CPU usage of VMs increases, the available CPU capacity of each server decreases and is not able to be used to redistribute load, leading to maximum load increasing. The results for $\times 4$ and $\times 5$ multipliers show little to no improvement in load distribution, as the datasets are CPU heavy and leave very little CPU capacity left in each server. The results for the multi-objective tests, where we observe how the algorithm can deal with more than one objective, show that the algorithm is able to optimize the load distribution without sacrificing the number of active servers, as long as there is some capacity left to distribute the load.

6. Conclusions

The VM consolidation problem consists on determining the optimal placement of VMs in physical servers. The VM consolidation problem is NP-Hard, which makes it very important to find new ways to find a good approximation to the optimal solution. There are many algorithms that have been used to solve the problem, but most of them consider the resource requirements of VMs to be constant. Considering the resource utilization of VMs to be constant wastes server capacity, as a VM will not fully use the resources allocated to it all the time. In this work we use VM profiles, which are traces of VM resource utilization over time, to compute an optimized placement for VMs. This enables the algorithm to take advantage of variations in the resource utilization of VMs to place more VMs in each server, without the risk of exceeding the server's available resources.

Our work proposes a SAT based method to solve the VM consolidation problem with the use of VM profiles. We evaluated our work by running a series of tests. We compare the results of our algorithm to traditional pseudo-Boolean optimization as well

as the First Fit approach. The results show that an algorithm that considers the variation of a VM's CPU usage over time can lead to a placement with a much lower number of active servers. Due to only considering the variation of CPU load, our algorithm performs better the higher the CPU load is in a set of VMs. For such cases, our proposed method can lead to a significant improvement over traditional optimization methods. We have also demonstrated how the algorithm can handle multiple objectives by ensuring an uniform load distribution across active servers, after the number of active servers has been determined.

For future work, we would add symmetry breaking techniques to the formulation, in order to improve the time the solver requires to find the solution of the problem. This would allow our work to be used to solve larger problems.

Acknowledgments

This work was partially supported by national funds through FCT with references UID/CEC/50021/2019 and PTDC/CCI-COM/31198/2017. I would like to thank Professor Vasco Manquinho and Professor Luís Guerra e Silva, for their support and guidance, that accompanied me throughout this entire process. I would also like to thank my friends and family, for their patience and support.

Referências

[1] A. Alsheddy and E. P. K. Tsang. Guided pareto local search based frameworks for biobjective optimization. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[2] F. Bacchus, J. Davies, M. Tsimpoukelli, and G. Katsirelos. Relaxation search: A simple way of managing optional clauses. In *AAAI*, pages 835–841. AAAI Press, 2014.

[3] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation*, 6(2):182–197, 2002.

[4] X. Fan, W. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA*, pages 13–23. ACM, 2007.

[5] A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.

[6] M. Gabay and S. Zaourar. A grasp approach for the machine reassignment problem. 07 2012.

[7] É. Grégoire, J. Lagniez, and B. Mazure. An experimentally efficient method for (mss, comss) partitioning. In *AAAI*, pages 2666–2673. AAAI Press, 2014.

[8] Z. Á. Mann. Allocation of virtual machines in cloud data centers - A survey of problem models and optimization algorithms. *ACM Comput. Surv.*, 48(1):11:1–11:34, 2015.

[9] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On computing minimal correction subsets. In *IJCAI*, pages 615–622. IJCAI/AAAI, 2013.

[10] D. Mehta, B. O'Sullivan, and H. Simonis. Comparing solution methods for the machine reassignment problem. In *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 782–797. Springer, 2012.

[11] A. Nöhrer, A. Biere, and A. Egyed. Managing SAT inconsistencies with HUMUS. In *VaMoS*, pages 83–91. ACM, 2012.

[12] V. Pareto. *Manuale di economia politica*, volume 13. Societa Editrice, 1906.

[13] F. Petitjean, A. Ketterlin, and P. Gancarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44:678–, 03 2011.

[14] T. Saber, A. Ventresque, X. Gandibleux, and L. Murphy. Genepi: A multi-objective machine reassignment algorithm for data centres. In *Hybrid Metaheuristics*, volume 8457 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 2014.

[15] M. Terra-Neves, I. Lynce, and V. M. Manquinho. Introducing pareto minimal correction subsets. In *SAT*, volume 10491 of *Lecture Notes in Computer Science*, pages 195–211. Springer, 2017.

[16] M. Terra-Neves, I. Lynce, and V. M. Manquinho. Virtual machine consolidation using constraint-based multi-objective optimization. *J. Heuristics*, 25(3):339–375, 2019.

[17] Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evolutionary Computation*, 11(6):712–731, 2007.

[18] Q. Zheng, R. Li, X. Li, N. Shah, J. Zhang, F. Tian, K. Chao, and J. Li. Virtual machine consolidated placement based on multi-objective biogeography-based optimization. *Future Generation Comp. Syst.*, 54:95–122, 2016.