Universiteit Leiden
Opleiding Wiskunde & Informatica

Predicting the Risk of Overload in

Overcommitted Server Clusters

Name: Sander Wubben

Date: 21/07/2016

1st supervisor: mr. Dr. S.G.R. Nijssen
2nd supervisor: ms. Dr. F.M. Spieksma

BACHELOR THESIS

Mathematical Institute & Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The Virtual Machine Placement (VMP) problem is the problem of finding an optimal assignment of virtual machines (VMs) to physical server clusters. Platform as a Service (PaaS) providers - companies that rent out server space - try to fit as many VMs as possible on the available physical servers, but need to consider a risk of overload on these servers. Therefore, when an assignment is made, the (highly dynamic) utilization of VMs should be considered. This work will characterize the utilization of VMs as accurately as possible while considering the needs for real-time adjustment and assessment of current configurations, limiting the complexity of the proposed solutions. We will characterize the utilization for 1) pre-existing VMs and 2) new VMs. These characterizations will be used to implement a model for quick risk assessment for possible assignments of VMs to server clusters. The proposed method will be evaluated on real life utilization traces provided by the Dutch IT company KPN.

Contents

Chapter 1

Introduction

We are observing server cluster (SC) configurations in KPN[1] server parks. KPN is a telecommunications, internet service and IT services provider, with a large division focused on corporate clients. These corporate clients rent server space in the KPN data centers where they can run their own IT operations - also known as "Platform as a Service" (PaaS). The server parks that contain these servers span around 160 physical SCs, managed through the VMware *vSphere* [VMw15] server management software. Every SC runs the VMware *vSphere Hypervisor* [VMw16] for deploying and controlling virtual machines (VMs). This software manages around 14.000 VMs, which are all separately rented out to customers as servers. KPN constantly monitors the performance of these VMs and the SCs they run on, to maximize customer satisfaction and minimize operational costs.

The vSphere managing software allows KPN to assign more resources to VMs on an SC than physically available. We will illustrate this concept using CPU capacity (it can, and usually will, also be applied to memory and storage) in SCs: it would mean that an SC with 30 CPU's can run 30 machines all with 4 virtual CPU's (a total of 120), under the assumption that these 30 machines will not all be utilizing their full capacity at the same time. We see a ratio of the virtual resources versus the physical resources of 4, called the *consolidation factor*. When the consolidation factor for any resource is greater than 1, we call an SC *overcommitted* with respect to that resource.

Overcommitment is a simple and efficient policy to maximize the utilization of hardware. When handled correctly, a lot less hardware needs to run idle, while the end-experience for the users should not change. Overcommitment does, however, incur a risk for the performance of the individual VMs. Whenever the VMs try to claim more resources from their SC than physically available, the vSphere software needs to apply aggressive resource reclamation techniques or temporarily pause the VMs. We will not delve too deep into challenges

[1] This thesis is the result of a cooperation between the University of Leiden and Royal KPN N.V.

imposed by running these configurations, as they are discussed extensively by Soundararajan et al. [SG10], but it should be clear that a shortage in resources causes performance issues.

If we could predict the use of any VM at any point in the future, we could make sure that enough resources are available at any time. However, these VMs are used by customers, and there is little to no information on the applications that run on these servers. Even if we did know the exact contents of any server, the programs running on it would still be unpredictable and subject to external influences.

Right now, an arbitrary consolidation factor is used for all SCs. An SC is usually filled until the consolidation factor reaches 4, after which it will be adjusted on an as-needed basis. An adjustment is commonly caused by inspection of utilization charts or complaints over performance issues. The first of these causes usually means hardware has been running idle, which is wasteful, and the second is bad for customer satisfaction, which could cause revenue-loss.

The configurations of SCs are maintained by employees (capacity managers) who have some very large server parks to attend to. It is nearly impossible to constantly monitor all SCs and if a problem is identified it is essential that they come to a solution instantly. It is therefore essential to find a framework that not only provides the capacity managers with quick insight into SCs that are at risk of overload or under-utilization, but is also able to predict the risk of overload in a changed configurations.

As discussed in Related Work, Chapter 2, most earlier works have addressed this problem - known as the Virtual Machine Placement (VMP) problem - by implementing automatic detection of overloaded hardware and determining a new (pseudo-)optimal placement of VMs without human interaction. This makes the VMP problem a two-part problem: the detection of overload and the optimal placement. However, the placement of most VMs in the observed server parks is dependent on business ruling, so we are not able to make changes on-the-fly. In this thesis we will not implement a solution to the whole VMP problem, but we will focus on the first part of this problem: implementing heuristics to determine the risk of overload in SCs, to inform capacity managers of the impact of their placement decisions.

We will make a characterization of utilization of every individual VM by, as a basic tool, an estimate of normally distributed variables[2]. Since the utilization of an SC is the sum of utilizations of all VMs on that SC, we can characterize the utilization of the SC as the sum of these normally distributed variables. Since this method can be applied to an arbitrary collection of VMs, it provides us with a way to determine the risk of overload in any SC that may be changed. The main contributions of this thesis will be methods to characterize VMs as accurately as possible, since that will directly allow us to more correctly predict the risk of overload in SCs.

[2]The normal distribution is defined on $\mathbb{R}$, while the utilization of a VM is in the interval $[0, Vcpu_{crj}]$ for VM V_{crj} and capacity $Vcpu_{crj}$ (see Section 3.3). However, for a small enough standard deviation the normal distribution will largely be restricted to this interval.

A change of an SC will usually happen in one of two ways:

1. A VM (or group of VMs) can be migrated from one SC to another;

2. A new VM, or multiple new VMs, can be added to an SC.

In the first scenario we know the previous utilization of all VMs in the changed SCs, so we only need to accurately characterize their behavior. For a new VM this is not applicable, so we will have to characterize them based on other information. For the moved VMs we will, as a basis, characterize the utilization of a VM as a single, time-independent, normally distributed variable to compare with earlier works, and as an extension model the utilization at every point in time as a time-dependent normally distributed variable. For new VMs we will group them by relevant contextual features - like the renting customer, operating system, etc. - and use these groups to deduce a characterization for the new VMs in the same group. We will also apply a predictive clustering algorithm to see whether this provides us with a more accurate characterization.

These methods will be compared to each other in terms of the accuracy of the results. If we disregard the previous utilization of a moved VM we have the same information as for a new VM, so the methods for new VMs can be applied to moved VMs. However, we expect to be able to provide a better prediction based on previous use than on contextual features, so we can use the results for moved VMs as a benchmark for new VMs. We will implement a general method for calculating the expected time an SC experiences overload, which will allow us to incorporate the best methods for both previous mentioned changes 1) and 2). This method should be applicable to any changed SC configuration, but will be evaluated using current configurations.

The remainder of this thesis is organized as follows: In Chapter 2 we will review earlier work and applicability to our objectives, and Chapter 3 will introduce the available data, observed configurations, notation and machine learning problems. In Chapter 4 we will determine the periodicity of the utilization of the observed system, which will be used to create utilization characterizations for (groups of) existing machines in Chapter 5 and new VMs in Chapter 6. Chapters 4 through 6 will be brought together in the risk assessment method in Chapter 7. In Chapter 8 we will evaluate the implemented methods and Chapter 9 will conclude the thesis.

Chapter 2

Related work

The issue of finding an optimal assignment of VMs to physical SCs is mainly referred to as the Virtual Machine Placement (VMP) problem. Extensive research has been performed on the VMP problem, as described in [LPB15]. As a result, multiple approaches and a variety of platforms have been identified and characterized. Most works propose an online approach (one with real-time monitoring and adjustment), because the workload generated by VMs is usually highly dynamic.

According to the definitions provided by [OF16] the platform that we observe, is a platform characterized as one with overcommitment, but without horizontal or vertical elasticity. About 50% of the works reviewed by [OF16] concern a similar platform. Through this characterization a notation has been proposed which will be the basis for the notation introduced in Chapter 3 of this thesis.

There have been two distinctive approaches to the VMP problem. One of these is reactive in nature. The works using this approach have used an online formulation of the VMP problem which constantly or periodically monitors the utilization of hardware configurations and makes changes as soon as an overload has been detected. Once a configuration needs to be changed, a bin-packing [CD14, FDH11], pseudo-boolean [RSM+13], reinforcement learning [Ven07] algorithm or a variation on these is applied to find the best new configuration. However, depending on the thresholds for reconfiguration, these methods will not prevent an overload of hardware when the threshold is too progressive or can still waste space when when the threshold is too conservative.

The methods in the previous paragraph rely on the freedom to move VMs when necessary. We do not have this much freedom, since the observed platform in this thesis is dependent on business ruling that is hard to implement in VMP solutions. However, there have been other works that have taken a more proactive approach in determining the risk of current or objective configurations, as most of these rely on forecasting. They characterize the utilization of a VM as a random variable [HP13, CZS+11, WMZ11] or combine methods of time-series forecasting [GP12, BKB07]. When the future utilization has been estimated, this is used to determine

the best configuration for the near future.

To the author's knowledge most earlier works seem to use a deterministic (fixed) value for the resources a VM needs, as in [MN08, RSM+13]. This usually means that the configuration is evaluated on the worst-case scenario where all VMs are at their maximum utilization. It is better to consider the utilization of a VM as a stochastic process to take into account the fact that most VMs will usually utilize far less than their capacity, as proposed in [CZS+11, KZL+10, WMZ11]. In [CZS+11] the utilization of a VM is regarded as a single time-independent random variable with a mean and a standard deviation with an unknown distribution. In [HP13, GP12] however, they are regarded as a normally distributed variable or a Poisson random variable.

We determined that the mean and variance of the utilization of VMs are not nearly equal after preliminary inspection. We will therefore not model the utilization as Poisson random variables. In this thesis, we will make use of the periodicity that VMs show (see Chapter 4) to create a characterization by normally distributed variables depending on the period. The utilization of VMs will first - for reference - be modeled as a single time-independent normally distributed variable, as proposed in [HP13]. To improve the accuracy of the characterization, the possibilities and strengths of using multiple time-dependent normally distributed variables will be investigated as well.

Chapter 3

System model

This chapter will describe the data, notation and problems used and discussed in this thesis. Throughout the following chapters we will use one pool of data out of which we will extract more specific data sets. We will first describe this data. We will then describe the SC configurations we are to observe. Subsequently, we define a notation we will use for the rest of this thesis and, to conclude, we will define the machine learning problems we have effectively worked on.

3.1 Available data

The available data mainly describes the individual CPU utilization of every VM. This means that, over the past year, we have a data point for every five minute interval containing average *CPU usage* in that interval. To observe the current configurations we also know the current *resource pool* (RP) any VM is deployed in and which *server cluster* (SC) this RP is on. To accompany this information, we know the configured maximum *capacity*, current *operating system*, current renting *customer* and managing *department* within KPN for every VM. Table 3.1 shows some extra information regarding the available features. We know, for example, that there are 34 different Server Clusters and for each VM which SC it is on. That means we have 100% coverage and 34 categories. We have this information on most of the VMs in SCs with the *standard configuration* (see Section 3.2), which amounts to 6.852 VMs.

3.2 Standard configuration

Within vSphere the user is very free in the configuration of an SC. This configuration can be seen as a tree with the root node being the SC, the internal nodes the RPs, and the leaf nodes being the VMs. It is even possible

Data	Type	Data points per VM	coverage	range/number of categories
Server Cluster	Categoric	1	100%	34
Resource Pool	Categoric	1	100%	157
Customer	Categoric	1	100%	38
Operating System	Categoric	1	$> 99\%$	48
CPU Capacity	Numeric	1	100%	0-40.000
CPU Usage	Temporal	± 105120	$> 90\%$	0-40.000

Table 3.1: Available attributes per VM.

to have RPs as leaf nodes (empty RPs) or a configuration without RPs (VMs directly on an SC).

Within KPN the most frequently used form of a configuration, is one where there are two layers of RPs: one layer with only a RP named "Flex" and one layer with a RP for any customer on an SC containing all VMs of this customer on this SC, as pictured in Figure 3.1. We will further refer to this form of configuration as a *standard configuration*. When SCs do not adhere to this standard configuration, it will usually mean that specific business ruling is applied to them, which could imply that our methods are not directly applicable to them.

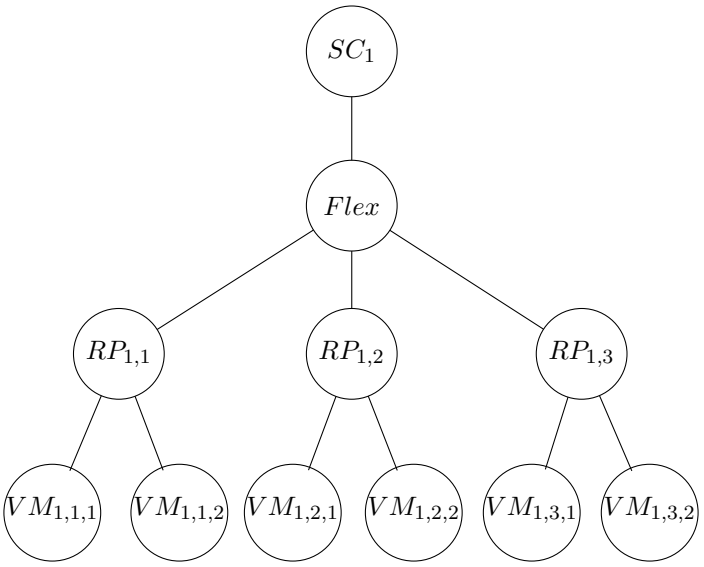


Figure 3.1: Standard configuration of an SC in KPN vCenters.

3.3 Notation

We will introduce a notation to describe the server park and all contained entities with their relations to each other and their CPU usage in time. We will extend the notation suggested in [OF16] to fit our situation. Say we have

- mSP: number of Server Clusters in all server parks;
- SC_c: Server Cluster c, where $c \in \{1, ..., mSP\}$;

- mSC_c: number of Resource Pools in SC_c;

- RP_{cr}: Resource Pool r in SC c, where $r \in \{1, ..., mSC_c\}$;

- mRP_{cr}: number of Virtual Machines in RP_{cr};

- V_{crj}: VM j on RP r in SC c, where $j \in \{1, ..., mRP_{cr}\}$.

We can then define the contextual features of VMs and SCs as

- $SCcpu_c$: CPU capacity of SC c;

- $Vcpu_{crj}$: CPU capacity of VM j in RP r on SC c;

- mC, mO, mD: the number of customers, operating systems and departments, respectively;

- $VC_{crj} \in \{1, ..., mC\}$: renting customer of VM j in RP r on SC c;

- $VO_{crj} \in \{1, ..., mO\}$: operating system of VM j in RP r on SC c;

- $VD_{crj} \in \{1, ..., mD\}$: managing department within KPN of VM j in RP r on SC c.

We consider the CPU utilization of a VM on the time points between t_{init} to t_{end}, where we model the time points as $t = 1, ..., T$ with T the number of 5 minute intervals between t_{init} and t_{end}, as

- $VU_{crj}(t)$: CPU utilization at time t of VM j in RP r on SC c.

and define a representation of RPs, SCs or the server park as

$$RP_{cr} = \{V_{crj}\}, j \in \{1, 2, ..., mRP_{cr}\}, SC_c = \bigcup_{r=1}^{mSC_c} RP_{cr} \text{ and } SP = \bigcup_{c=1}^{mSP} SC_c \tag{3.1}$$

we can define the CPU usage of a whole SC c at a timepoint $t \in \{1, ..., T\}$ as

$$SCU_c(t) := \sum_{V_{crj} \in SC_c} VU_{crj}(t). \tag{3.2}$$

We now know the utilization $SCU_c(t)$ of the SC for the time points $t \in \{1, ..., T\}$. The goal is to analyse the expected time an SC is in overload in the next $S \in \mathbb{N}$ time points, or

$$\mathbb{E}\left[\sum_{s=T+1}^{T+S} \mathbb{1}_{SCU_c(s) > SCcpu_c}\right]. \tag{3.3}$$

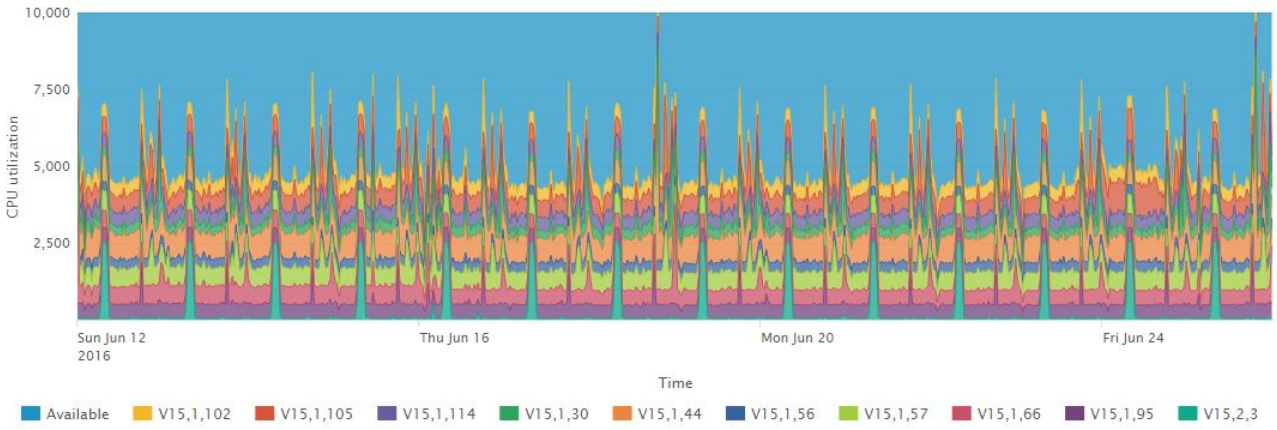


Figure 3.2: The utilization of an SC over a two week period, split by VM.

3.4 Machine learning

We will introduce three machine learning problems to be addressed in this thesis. The first two will be approaches to characterize the utilization of a VM, either using the previous utilization of an *existing* VM, or characterize a *new* VM using the previous utilization of multiple equivalent VMs where equivalency needs to be assessed using contextual features of the machine. The characterizations of utilization will be used to predict the risk of overload in an SC in the encompassing machine learning problem where we have a different objective: to identify an SC's risk for overload as accurately as possible.

3.4.1 Utilization of an existing VM

We want to characterize the utilization of an *existing* VM based on its previous utilization. We usually have tens of thousands of data points in a time series for any VM, so the difficulty lies in finding a representation small enough to remain usable in applications, yet large enough to hold most information necessary to characterize the VM.

The data we will use to solve this problem is the time series of five minute averages of VM utilization for up to a year in the past. We want to characterize the VM's utilization to apply to a period in the future. So, say we have all data for time points for last year $\{1, ..., T\}$ where t_{init} represents 'a year ago' and $T = 105120$ is the amount of five minute intervals in a year, we want to find an approximation for the utilization in the coming S time points where we define $VU^*_{crj}(s)$ as the approximation for utilization on time point s, with $T < s \leq T + S$.

To be able to validate the accuracy of our methods we will not use all available data for training. The training set will be the CPU utilization of the VM at the time points $1, ..., T - S$, with S a positive integer to be determined at a later stage. The test set will then be the data for the time points $T - S + 1, ..., T$.

Since large errors at single time points can immediately cause an error in the greater picture, we want to use a

performance measure that penalizes these errors, and not necessarily one that penalizes multiple smaller errors. We will therefore use the Mean Squared Error, a performance measure with, as described by [AA13]:

- Low tolerance for extreme errors;
- No distinction for the direction of the overall error. However, the directions of error will be expected to cancel out in the greater picture since we will usually be assessing a great number of VMs;
- Sensitivity for scale and data transformations. However, within this problem we will not be applying data transformations, and the scales (e.g. capacities of VMs) do not differ significantly.

We can then define as performance measure P_1:

$$P_1 := \frac{1}{mSP} \sum_{c=1}^{mSP} \frac{1}{mSC_c} \sum_{r=1}^{mSC_c} \frac{1}{mRP_{cr}} \sum_{j=1}^{mRP_{cr}} \frac{1}{S} \sum_{s=T-S+1}^{T} \left(VU_{crj}(s) - VU^*_{crj}(s)\right)^2. \tag{3.4}$$

3.4.2 Utilization of a new VM

We want to predict the utilization of a *new* VM. Since the proposed solution to the problem in Section 3.4.1 will provide us with a characterization of the utilization of existing VMs, we can use these characterizations together with contextual features of VMs to predict a characterization for the utilization of the new VM.

The data we will use to solve this problem will be the characterizations of utilization of all VMs produced from the data between t_{init} and t_{end}. The characterizations will be represented as a set of S (see Section 3.4.1) attributes later referred to as the *behavioral attributes*, present for every existing VM. For every VM we also have some *contextual attributes*, like the *department* controlling the VM, the *customer* renting the VM, et cetera.

For validation we will split the data set into a training set and a test set, where a random sample of 90% of the VMs will be in the training set and the remaining 10% of the VMs will be the test set. We will then use the contextual attributes of the VMs in the test set to create a prediction for the behavioral attributes, based on the contextual and behavioral attributes of the training set.

For a performance measure of the proposed methods we have equal conditions to the machine learning problem in Section 3.4.1. Consequentially we will use the Mean Squared Error between the predicted behavioral attributes and the actual behavioral attributes as a performance measure. So if we had a test set N of VMs, with $N \subset SP$, the performance measure P_2 can be defined as

$$P_2 := \frac{1}{T} \sum_{V_{crj} \in N} \left(VU^*_{crj}(t) - VU^*_{crj}(t)\right)^2 \tag{3.5}$$

3.4.3 Probability of overload in an SC

We not only want to be able to find the probability of overload in a current SC configuration, but also want to apply this process to new configurations. This encompasses the two preceding problems in this section. We will define a *new configuration* as a subset of all existing machines united with a set of new machines. The data used will thus be all data mentioned in Sections 3.4.1 and 3.4.2, for all VMs in scope.

We will use two methods for validation. For the situations where no new VMs are added to the new configuration (a subset of all existing machines in the server park) we will directly produce a probability of overload for all existing SC configurations through the characterization of the utilization of VMs. For the situations where new VMs are added to the new configuration, we will regard a few existing VMs in every SC as a new machine, provide them with a prediction for behavior and subsequently produce a probability of overload.

We will define the error e_c for SC c as the absolute difference between the predicted probability f_c^* of overload and the actual frequency f_c of overload in the test set, so $e_c = |f_c^* - f_c|$. Because a single large error has much more impact on business (this implies a serious over- or underload frequency that goes unnoticed), we will use the mean error over the error of all SCs $\{SC_c | c = 1, 2, ..., mSP\}$ as a performance measure P_3 for this problem, e.g.

$$P_3 := \frac{1}{mSP} \sum_{c=1}^{mSP} e_c = \frac{1}{mSP} \sum_{c=1}^{mSP} |f_c^* - f_c|. \tag{3.6}$$

We will use the Mean Absolute Error - not the Mean Squared Error - for this performance measure, since this provides us with more insight in the errors. For the previous problems (Section 3.4.1 and 3.4.2) we had a base model to compare the results to. Since we do not have a reference model for this problem more insight is needed to assess the effectiveness of the proposed solutions.

Chapter 4

Periodicity in behavior of VMs

If VMs show a repeating pattern of utilization (periodicity), we can summarize their behavior in this pattern. To determine if VMs show periodicity and if so, with which period, we have applied autocorrelation. Autocorrelation produces a second-order [Gra03] summary of time series data that shows the degree in which a time series correlates with itself after any lag l. This entails that, if we have a time series $X = \{X_t\}_{t=1,...,n}$ for a certain n with variance $\mathrm{Var}[X] > 0$ and we define the (empirical) mean of a stochastic process Z as $\bar{Z} := \frac{\sum_{i=1}^{n} Z_i}{n}$, we can calculate for lag $l \in \{0, ..., n\}$ [VR02]:

$$c_l = \frac{1}{n} \sum_{s=1}^{n-l} [X_{s+l} - \bar{X}][X_s - \bar{X}] \text{ and } r_l = \frac{c_l}{c_0}. \tag{4.1}$$

We will try to provide insight into these entities:

- c_l: For $l = 0$, we see that $c_0 = \frac{1}{n} \sum_{s=1}^{n} [X_s - \bar{X}]^2 = \mathrm{Var}[X]$ and for $l \in \mathbb{Z} \backslash \{0\}$ we see that this is almost the formula for the covariance of the stochastic processes $\{Y_j\}; j = 1, 2, ..., n - l)$ and $\{Z_k\}; k = l, ..., n - 1, n$ with $Y_j = X_j$ and $Z_k = X_k$ except that the divisor is n instead of $n - l$. The divisor is changed to ensure that the sequence of autocovariances for increasing l is positive-definite [VR02] for certain applications.

- r_l: For $l = 0$, we clearly see that $r_0 = 1$. For $l \in \mathbb{Z} \backslash \{0\}$, we divide the autocovariance of X with lag l by the variance of X. The intuition here is that if the autocovariance is as large as the variance there is perfect correlation between the function and itself at a later time.

In our case we define these entities for a VM j on SC c in RP r with $\overline{VU_{crj}} := \frac{\sum_{s=T-n}^{T} VU_{crj}}{n}$ as

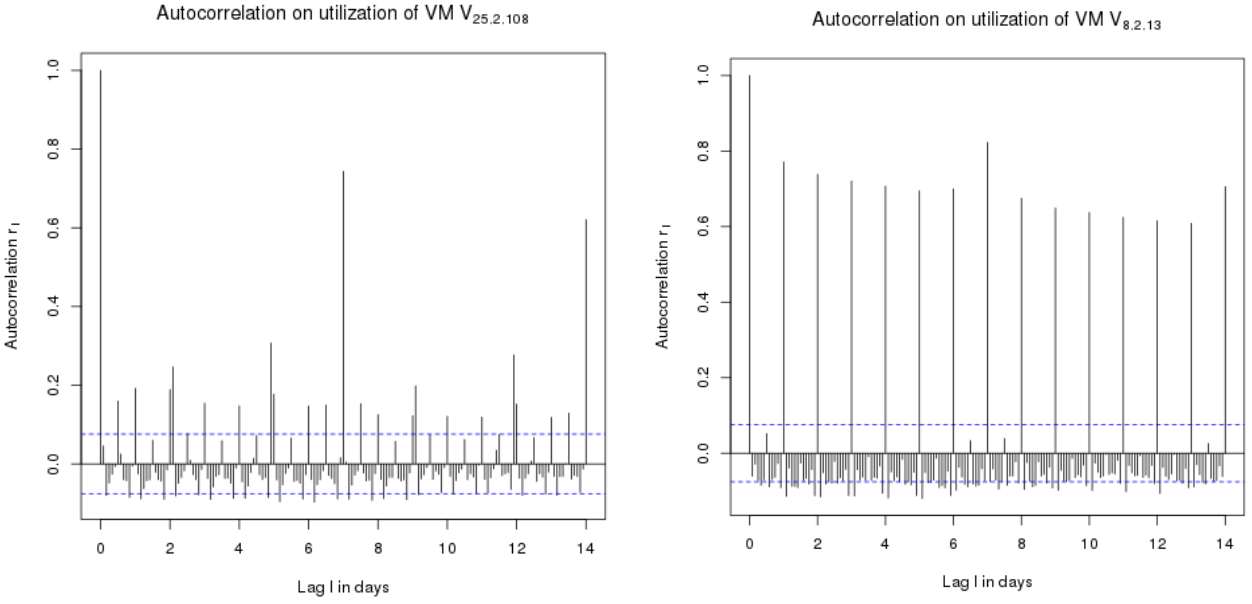
$$cov_{crj}(l) := \frac{1}{n} \sum_{s=T-n+1}^{T-288l} \left[VU_{crj}(s + 288l) - \overline{VU_{crj}}\right] \cdot \left[VU_{crj}(s) - \overline{VU_{crj}}\right]; \tag{4.2}$$

$$cor_{crj}(l) := \frac{cov_{crj}(l)}{cov_{crj}(0)}, \tag{4.3}$$

for $l \in \{0, 1, ..., \lfloor \frac{n}{288} \rfloor\}$.

We then have $cov_{crj}(l)$ and $cor_{crj}(l)$ the estimated autocovariance and autocorrelation, respectively, as described in [VR02]. We use the scalar 288 for l as we want the lag to be calculated in days, and there are 288 time points (12 per hour, 24 hours in a day) in a day. The method described is exactly the estimate of the autocorrelation used in R [R C14].

We applied autocorrelation to the time series data of the CPU utilization of a random sample of 15 VMs using R. For each VM in the random sample we have created a data set showing the average CPU usage every two hours over a span of one month. This data was imported into R, where autocorrelation was performed for a lag $0 \leq l \leq 14$ specified in days. This produced graphs like in Figure 4.1a, with every peak showing the strength of periodicity on the time span (in days) specified on the x-axis. The resulting 15 graphs (see Appendix A) were



(a) Weekly periodicity

(b) Daily periodicity

Figure 4.1: Autocorrelation on the CPU usage of VMs showing weekly (a) and daily (b) periodicity. The x-axis shows lag l specified in days and the y-axis shows the autocorrelation $cor_{crj}(l)$.

inspected. Most of them showed a high autocorrelation factor on a period of a week, or at least on a period of a day with a slight increase on the 7 day mark as in Figure 4.1b. We can thus conclude that characterizing the CPU in the period of a week should produce the most accurate results in determining their behavior.

This periodicity is the key for modeling the characterization of the utilization of a VM as multiple time-dependent normally distributed variables, as it allows us to model one full week from multiple weeks and apply the model to new weeks. In Chapter 8 we use the data of 18 weeks of utilization, which provides us with 18 observations for every time point in a week, to be considered as samples from the same random variable. If we then apply

the Central Limit Theorem [Dur10, p. 106-110] we know the characterizing random variables tend to normally distributed variables.

Chapter 5

Characterizing the utilization of an existing VM

We will propose two methods to characterize the utilization of a pre-existing VM using normally distributed variables, as an extension of the heuristic of [HP13]. In the earlier works (see Section 2) the characterization has not taken more freedom than a single time-independent random variable per VM, which will be our base implementation. To try to implement a more precise characterization of a VM we will create a normally distributed variable for every five minute interval of a week (that is $12 \cdot 24 \cdot 7 = 2016$ variables), since we determined periodicity of a week in Chapter 4.

5.1 Normal distribution

The data on utilization of VMs is collected on a five minute interval, where the value given on a time point is the average utilization within the previous five minute interval. *vSphere* [VMw15] takes a snapshot of CPU utilization every 20 seconds, and our observations are averages of 15 snapshots in every 5 minute interval. Even though the workload of a VM will usually not drastically change within a five minute interval, we still see that the utilization within an interval will highly fluctuate. We will therefore assume that the 20-second snapshots within a 5 minute interval can be modeled as random variables with similar means and standard deviations. We can then apply the Central Limit Theorem [Dur10, p. 106-110], which states that the average of a great number of random variables with equal means and standard deviations tends to a normally distributed variable. If we add the fact that we use observations from multiple weeks (17 in Chapter 8), every normally distributed variable has been formed on 17 observations of measurements nearing normally distributed variables, as these are based on 15 snapshots. This does require the assumption of independence between weeks, which we cannot

falsify after inspection of the utilization of VMs.

5.2 Time-independent normally distributed variable

To create a single normally distributed variable to characterize a VM V_{crj}, we need to calculate the mean

$$\mu^*_{crj} = \frac{1}{T}\sum_{t=1}^{T} VU_{crj}(t) \tag{5.1}$$

and the standard deviation

$$(\sigma^*_{crj})^2 = \frac{1}{T}\sum_{t=1}^{T}(VU_{crj}(t) - \mu^*_{crj})^2 \tag{5.2}$$

of the utilization of the VM, to create a normally distributed variable $VU^*_{crj} \sim \mathcal{N}(\mu^*_{crj}, (\sigma^*_{crj})^2)$.

5.3 Time-dependent normally distributed variables

In Chapter 4 we concluded that most VMs show strong periodicity on a period of a week or (in some cases) a day. Since daily periodicity can still be captured in a characterization spanning a week, we will provide a characterization of the utilization of a VM over a week.

Periodicity over a week implies that the utilization of a VM over a five minute span $VU_{crj}(t)$ highly correlates with the utilization of the same VM w weeks earlier $VU_{crj}(t - 2016w)$. We could then extrapolate that the values of utilization at the set of time points

$$N_t := \{t + 2016k \,|k \in \mathbb{Z}, 0 < t + 2016k \leq T\} \tag{5.3}$$

are all samples of the normally distributed variable that can be represented by

$$VU^*_{crj}(\hat{t}); \hat{t} \in \{1, ..., 2016\}. \tag{5.4}$$

We can then approximate the representative normally distributed variables $VU^*_{crj}(t); 0 \leq t < 2016$ as

$$VU^*_{crj}(t) \sim \mathcal{N}\left(\mu^*_{crj}(t), (\sigma^*_{crj}(t))^2\right) \tag{5.5}$$

with

$$\mu^*_{crj}(t) = \frac{1}{|N_t|}\sum_{s \in N_t} VU_{crj}(s) \tag{5.6}$$

and

$$(\sigma^*_{crj}(t))^2 = \frac{1}{N_t} \sum_{s \in N_t} \left(VU_{crj}(s) - \mu^*_{crj}(t)\right)^2. \tag{5.7}$$

5.4 Implementation

In this section we will show how we can implement the proposed methods in the *Splunk* [Spl16b] big data software. Since the data used is stored in Splunk, this implementation can quickly provide results without a need for data manipulation or transfers. We have implemented the above methods to create a single normally distributed variable or a set of 2016 normally distributed variables. To this end we have created queries in the Search Processing Language (SPL) [Spl16a] used by Splunk to directly model the behavior of VMs.

For a single random variable this query is shown in Listing 5.1. In the first two lines of this query we select the data we want to use, e.g., the five minute averages of CPU utilization of all VMs for the past 18 weeks. We then restrict the data to all VMs in scope in lines 3-4, where the scope is all VMs that are in SCs that conform to the *standard configuration* as defined in Section 3.2. Finally, in line 5, we calculate the mean $\mu^*_{crj}(t)$ and variance $(\sigma^*_{crj}(t))^2$ of all representative random variables for all VMs.

```
1 index=vcenter_script host=vcenter_statistics MetricId=cpu.usagemhz.average
2   Type=VM earliest=@w-18w latest=@w-w
3 | lookup thesis_configurations_in_scope.csv VMName OUTPUT in_scope
4 | search in_scope=1
5 | stats mean(Value) as mu, var(Value) as sigma by VMName
```

Listing 5.1: SPL query determining mean and variance of utilization of VMs.

For multiple random variables this query is shown in Listing 5.2. In the first two lines of this query we select the data we want to use, e.g. the five minute averages of CPU utilization of all VMs for the past 18 weeks. We then restrict the data to all VMs in scope in lines 3-4. Following this, in lines 5-6, we map each five minute average to a time point between 0 and 2015 and to conclude we calculate the mean $\mu^*_{crj}(t)$ and variance $(\sigma^*_{crj}(t))^2$ of all representative random variables for all VMs in line 7.

The queries in Listings 5.1 and 5.2 result in we get a result for every VM in scope (for every time point) with the mean and standard deviation of the single time-independent normally distributed variable or multiple time-dependent normally distributed variables.

```
1 index=vcenter_script host=vcenter_statistics MetricId=cpu.usagemhz.average
2   Type=VM earliest=@w-18w latest=@w-w
3 | lookup thesis_configurations_in_scope.csv VMName OUTPUT in_scope
4 | search in_scope=1
5 | eval date_wday=strftime(_time, "%w")
6 | eval timepoint=288*date_wday+12*date_hour+(date_minute/5)
7 | stats mean(Value) as mu, var(Value) as sigma by timepoint, VMName
```

Listing 5.2: SPL query determining mean and variance of utilization of VMs for all time points 0-2015.

Chapter 6

Predicting the behavior of a new VM

We have implemented two methods to create a prediction for the characterization of utilization of new VMs based on their contextual features. The first one is a general grouping by the available features, and the second is by applying a predictive clustering algorithm. We will discuss both methods and their implementation in this Chapter. We will first describe what data we will use as behavioral attributes for VMs before describing the methods of k-means clustering and Principal Components Analysis and their results.

6.1 Attributes

Since we have determined periodicity of a week in Chapter 4 and created a characterization of virtual machines in Chapter 5, we will use these characterizations of behavior in a week as behavioral attributes and try to find correlation between them and the available contextual attributes. We have prepared a dataset of VMs with their contextual attributes ($VD_{crj}, VC_{crj}, VO_{crj}$ and $Vcpu_{crj}$) and their behavioral attributes - the variables $\mu^*_{crj}(t); t = 1, ..., 2016$ as defined in 5.3.

6.2 Grouping

The base method for characterizing a new VM will be to group them on contextual features. For example, it could be expected that all VMs of the same *company* show relatively similar behavior. If any of the known contextual features have a strong correlation with behavior of a machine, then combining all of those could improve the association. To apply this method we first need to find which contextual features correlate highly with behavior of VMs. We have implemented two methods to gain insight in this correlation.

6.2.1 k-means clustering

In an attempt to find relevant contextual attributes, we have applied k-means clustering to the data set. The k-means clustering algorithm places the p components of the behavior of all n VMs on a p-dimensional coordinate system and tries to find positions of k cluster (not to be mistaken for Server Clusters/SCs) centroids in this coordinate system to associate these n VMs with.

It accomplishes this in the following way. Say we have n observations $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$, where $\boldsymbol{x}_i = (x_{i1}, ..., x_{ip})$. We place k centroids $\boldsymbol{c}_1, ..., \boldsymbol{c}_k$ where $\boldsymbol{c}_j = (c_{j1}, ..., c_{jp})$ with $c_{jl}; j = 1, ..., k; l = 1, ..., p$ chosen at random on the p-dimensional plane. We then:

1. Assign each observation i to the centroid j with the smallest squared euclidean distance between them

$$a_i = \text{argmin}_{j=1,...,k} \left\{ \sqrt{\sum_{l=1}^{p} (x_{il} - c_{jl})^2} \right\}; i = 1, ..., n. \tag{6.1}$$

2. Calculate the new location of each centroid as

$$\boldsymbol{c}'_j = \frac{\sum_{i=1,...,n;a_i=j} \boldsymbol{x}_i}{\sum_{i=1,...,n;a_i=j} 1}; j = 1, ..., k. \tag{6.2}$$

3. If $|\boldsymbol{c}'_j - \boldsymbol{c}_j| < r; j = 1, ..., k$ with r a certain threshold: We have found an assignment of the n observations to k clusters. Else: $\boldsymbol{c}_j := \boldsymbol{c}'_j; j = 1, ..., k$, go to step 1.

The k-means clustering algorithm can thus find an assignment of all n observations to k clusters without supervision, where all observations assigned to a cluster are assumed to have small distances between them. This algorithm could have a problem of *local optima*, as stated in [Mac67], which is usually resolved by running the algorithm multiple times with different initial placements of centroids and using the best of the results. If we apply this algorithm to the behavioral attributes of VMs we could, by inspection, assess whether VMs with equal contextual attributes are assigned the same clusters. This could give some clues for a correlation between contextual and behavioral attribues.

The k-means has been applied to the behavioral features for different values of k. The values of k were determined by the number of distinct values for each contextual feature. We know for instance that there are only two departments within KPN, so if the department managing a VM has high influence on the behavior of the machine we expect two distinct clusters after running the k-means clustering algorithm with $k = 2$. When inspecting these clusters we would expect a lower entropy with respect to the department.

We have run the k-means clustering algorithm for the values $k = 2, k = 3, k = 20$, and $k = 40$ for the attributes department, operating system, capacity, and company respectively. This was done using the Splunk [Spl16b]

big data software, which has a built-in [PVG+11] k-means clustering algorithm. This was achieved using the SPL [Spl16a] query in Listing 6.1. This query selects the values $\mu^*_{crj}(t); t \in [0, 2015]$ as obtained in Section 5.4, to transform the data points for all time points to the behavioral attributes. It then applies the k-means clustering algorithm to the behavioral attributes, and to conclude it adds the contextual attributes for further inspection.

```
1 index=summary search_name=thesis_normally_distributed
2 | eval timepoint="T".substr("T000".timepoint, -4)
3 | chart limit=0 values(mu) OVER VMName BY timepoint
4 | fit KMeans T* k=2
5 | fields VMName, cluster
6 | lookup thesis_configurations_in_scope VMName
7   OUTPUT Department, company, os, capacity
```

Listing 6.1: k-means clustering in SPL

Upon inspection we see that most VMs will be placed in a single cluster, which would imply that most VMs behave similarly. For example in Table 6.1 we see the amounts of VMs in each cluster, after applying the k-means clustering algorithm with $k = 2$ for each of the two departments. We then see that most VMs are placed in cluster 0, irregardless of their department. We get equivalent results for the different operating systems with $k = 3$ (as shown in Table 6.2), and for the other contextual attributes with their associated values for k.

Department	CT2	UH
Cluster 0	346	3973
Cluster 1	53	125

Table 6.1: Number of VMs of each department per cluster.

Operating system	Linux	Other/Unknown	Windows
Cluster 0	971	26	2549
Cluster 1	9	1	8
Cluster 2	95	0	155

Table 6.2: Number of VMs of each operating system per cluster

From these results we could concludes that - when regarding all behavioral attributes - most VMs in scope show too much similarity when placed on a 2016-dimensional coordinate system to distinguish them based on the contextual attributes that we can distill. It could however be the case that these VMs are very similar on most behavioral attributes, but can still be distinguished when regarding only some more varying features. We will therefore apply a Principal Components Analysis in the following section to try to find these features.

6.2.2 Principal Components Analysis

In the previous section we concluded that most VMs are too similar, when plotting their behavioral attributes on a 2016-dimensional coordinate system. This could be the result of a big subset of these behavioral attributes with high correlation. A Principal Component Analysis is a variable reduction procedure that can help us with precisely that problem. This procedure will create an orthogonal base for a new coordinate system with p dimensions on which the original data with $q \geq p$ dimensions shows the highest variance, retaining as much information on the data as possible and leaving out most of the attributes that show high correlation with others.

Say we have a vector of variables $\boldsymbol{x} = (x_1, x_2, ..., x_p)$ of p dimensions, where we have n observations for each of these variables. We want to find a vector $\boldsymbol{\alpha}$ of weights with $\boldsymbol{\alpha}^\top \boldsymbol{\alpha} = 1$ which maximizes the variance of

$$\boldsymbol{\alpha}^\top \boldsymbol{x} = \sum_{j=1}^{p} \alpha_j x_j. \tag{6.3}$$

As derived in [Jol02, p. 5-6], the normalized eigenvector $\hat{\boldsymbol{\alpha}}_1$ corresponding to the largest eigenvalue λ_1 of the covariance matrix Σ of $\boldsymbol{x}$ is the vector we are looking for. So if we have $\mu_j = \mathbb{E}[x_j]$ for $j = 1, 2, ..., p$ we can find covariance matrix

$$\Sigma = \begin{pmatrix} \mathbb{E}[(x_1 - \mu_1)(x_1 - \mu_1)] & \dots & \mathbb{E}[(x_1 - \mu_1)(x_p - \mu_p)] \\ \vdots & \ddots & \vdots \\ \mathbb{E}[(x_p - \mu_p)(x_1 - \mu_1)] & \dots & \mathbb{E}[(x_n - \mu_n)(x_n - \mu_n)] \end{pmatrix}. \tag{6.4}$$

If we then find all p eigenvalues of Σ and order them as $\lambda_1, \lambda_2, ..., \lambda_p$, where λ_1 is the largest eigenvalue, λ_2 is the second to largest eigenvalue, et cetera, we can find the corresponding eigenvectors $\hat{\boldsymbol{\alpha}}_1, \hat{\boldsymbol{\alpha}}_2, ..., \hat{\boldsymbol{\alpha}}_p$ such that

$$\Sigma \hat{\boldsymbol{\alpha}}_j = \lambda_j \hat{\boldsymbol{\alpha}}_j. \tag{6.5}$$

As a result, we will see that the component of $\boldsymbol{x}$ in the direction of $\boldsymbol{\alpha}_1$

$$\hat{\boldsymbol{\alpha}}_1^\top \boldsymbol{x} = \sum_{j=1}^{p} \hat{\boldsymbol{\alpha}}_{1j} x_j \tag{6.6}$$

shows the highest variance [Jol02] on the n samples that we have, and $\hat{\boldsymbol{\alpha}}_2$ will give us the component of $\boldsymbol{x}$ in the direction of $\hat{\boldsymbol{\alpha}}$ which shows the second highest variance, et cetera.

We will apply a Principal Components Analysis to gain insight in the influence of contextual attributes on the behavior of VMs. The objective is to gain insight in the power of the available attributes to predict the behavior of new VMs. Because of the large feature set, we have applied a principal component analysis to the data set

to reduce the behavioral data from 2016 dimensions to 2.

The Splunk [Spl16b] big data software has a built-in method [PVG+11] for performing a Principal Components Analysis. We can thus find the first two principal components of our data - the characterizations of all virtual machines in scope - using the SPL [Spl16a] query as shown in Listing 6.2.

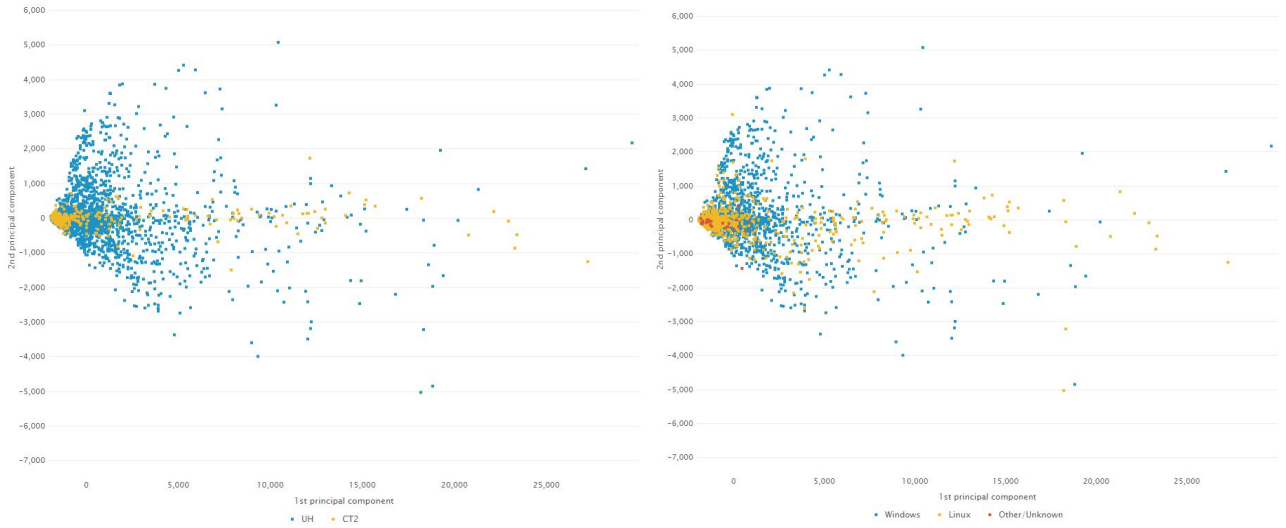
```
1 index=summary search_name=thesis_normally_distributed
2 | eval timepoint="T".substr("T000".timepoint, -4)
3 | chart values(mu) OVER VMName BY timepoint
4 | fit PCA T* k=2
5 | fields VMName, PC_1, PC_2
6 | lookup thesis_configurations_in_scope VMName
7   OUTPUT Department, company, os, capacity
```

Listing 6.2: Performing PCA on the dataset of all characterizations of existing VMs

The first line of this query selects the characterizations in normally distributed variables of virtual machines that we have created as described in 5.4. We then rename the time points to be usable as attribute names, and fill all behavioral attributes. We can then apply the Principal Components Analysis and add the contextual attributes.

We have plotted the two principal components on a box plot with colors that represent groups that differ on the available attributes. Upon inspection of the box plots it does not seem possible to distinguish the different groups. As seen in Figures 6.1 and 6.2, all groups overlap around the point where the first two principal components are around zero, so we cannot find significant differences between groups, even on the linear combinations of attributes with maximum variance.

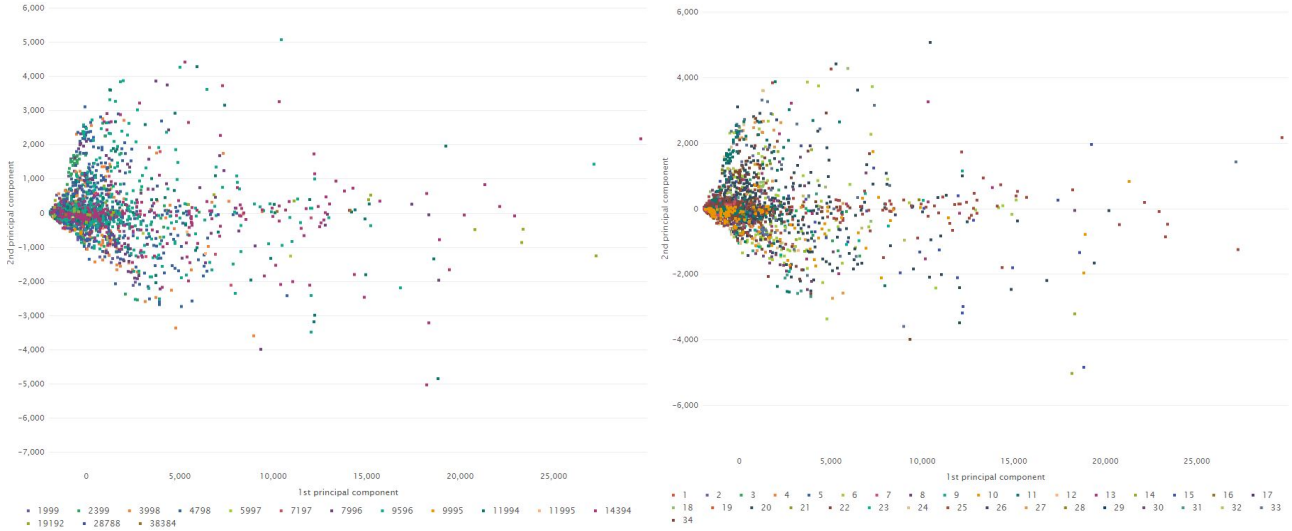
We will create a group of machines for every unique combination of department, operating system, company and capacity (254 groups) and use a combination of the behavior of all VMs in a group to determine the behavior of any single machine, since we cannot directly distinguish influence of individual attributes on the behavior of virtual machines. This would be a very quick way to associate new VMs with existing ones and creating a prediction. However, a problem could arise if we do not know all features of a new machine or if very small groups occur. If we do not know all contextual attributes for a new VM we are not able to create a prediction, and if small groups occur over-fitting could arise.



(a) First 2 principal components by department.

(b) First 2 principal components by operating system.

Figure 6.1: Scatterplot visualization of first two principals components of behavior of VMs, with the first PC on the x-axis, the second PC on the y-axis and color depicting the different departments (left) or operating systems (right).



(a) First 2 principal components by capacity.

(b) First 2 principal components by company.

Figure 6.2: Scatterplot visualization of first two principals components of behavior of VMs, with the first PC on the x-axis, the second PC on the y-axis and color depicting the different capacities (left) or companies (right).

6.3 Predictive clustering

Another way to associate new VMs with existing ones is by applying a predictive clustering algorithm. Such algorithms can create a search tree based on contextual features with a prediction for the behavioral attributes on the leaves of the tree. Say we have n observations of vectors $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ of p behavioral attributes, where $\boldsymbol{x}_j = (x_{j1}, ..., x_{jp}); j = 1, ..., n$. This algorithm makes use of a distance between instances, in our case the euclidean distance

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{j=1}^{p} (x_j - y_j)^2}, \tag{6.7}$$

and a prototype $\boldsymbol{p}(C)$ for a cluster $C \subset \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, in our case the mean of the vectors in the cluster

$$\boldsymbol{p}(C) = \frac{\sum_{j=1,...,n;x_j \in C}}{|C|}. \tag{6.8}$$

The goal of the algorithm is to split the observations into clusters with a low distance $d(\boldsymbol{x}, \boldsymbol{y})$ between observations $\boldsymbol{x}, \boldsymbol{y} \in \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ within clusters and a high distance $d(p(C_i), p(C_j))$ between the prototypes of clusters $C_i, C_j \subset \{\boldsymbol{x}_1 ..., \boldsymbol{x}_n\}$.

It will try to achieve this by creating boolean conditions on descriptive (contextual) attributes. For categoric attributes (department, company, os) this is a set comparison, e.g.

$$\text{company IN } D \subset \{1, ..., mC\} \tag{6.9}$$

and for numeric attributes (capacity) this is an inequality, e.g.

$$\text{capacity} <= 11.000. \tag{6.10}$$

The algorithm will:

1. Start with one cluster C containing all observations;
2. If $\sum_{\boldsymbol{x} \in C} \sum_{\boldsymbol{y} \in C} d(\boldsymbol{x}, \boldsymbol{y}) > r$ for a certain threshold r and $|C| > k$ for a certain minimum size of a cluster k: Go to step 3.

   Else: Go to step 5;
3. Find the splitting comparison to find clusters $C' + C'' = C$ that maximizes the distance $d(p(C'), p(C''))$ between C_1 and C_2 based on the available descriptive attributes. This essentially minimizes the distance within the clusters [BDR98].
4. Apply Step 2. to C' and C'';
5. The prediction $\boldsymbol{x}_j^*$ for the observations $\boldsymbol{x}_j \in C$ is the prototype of C: $\boldsymbol{x}_j^* := \boldsymbol{p}(C)$.

The predictive clustering algorithm was implemented with:

- Descriptive/contextual attributes: the four available features *department*, *company*, *operating system* and *capacity*;
- Target/behavioral attributes: the 2016 mean values $\mu^*_{crj}(t)$ for any VM j in RP r on SC c, one for every time point in a week $t = 1, ..., 2016$, as described in 5.4;
- Distance measure: the euclidean distance;

- Cluster prototype: the mean between the vectors of behavioral attributes of VMs in the cluster.

We used the Splunk [Spl16b] big data software to create a data set and exported it to the *.arff* format, the format that is used by predictive clustering program [Str11], using the SPL [Spl16a] query in Listing 6.3. This query first creates the header of the *.arff* file with a name for the relation and all attributes, including their (for categoric attributes) categories. It then places the data set in the file. When we export the results of this query to the *.csv* format, we only need to remove the strings '"&&' and '&&"' from the beginnings and ends of the lines - a process that can be automated - before Clus can interpret the *.arff* file.

```
| makeresults | eval text="&&@relation 'Predictive Clustering'&&"
| append [
  search index=summary search_name=thesis_predictive_clustering Try=3
  | table VMName, Department, company, os, capacity
  | eval t=1 | untable t, "Field", "Value"
  | stats values(eval(if(isnum(Value), null(), Value))) as Values by Field
  | eval text="&&@attribute ".Field.if(isnull(Values), " numeric", " {".
    mvjoin(Values, ",")."}")."&&"
]
| append [
  | makeresults count=2016 | streamstats current=f count
  | eval text="&&@attribute T".substr("T0000".count, -4)." numeric"
]
| append [ | makeresults | eval text="@data" ]
| append [
  search index=summary search_name=thesis_predictive_clustering
  | fields VMName, Department, company, os, capacity, info
  | eval text="&&".VMName.",".Department.",".company.",".os.",".
    capacity.",".info."&&"
]
| table text
```

Listing 6.3: SPL query of output of predictive clustering data set to the for Clus usable .arff format.

We then ran the predictive clustering algorithm using Clus [Str11], with 10 cross-validations and a minimal cluster size of 30. The predictions can then be imported into Splunk, where they are incorporated into the model described in Chapter 7. We will later refer to these predictions as $\nu^*_{crj}(t)$ for VMs j in RP r on SC c at time points $t = 1, ..., 2016$.

Chapter 7

Risk assessment of a configuration

The preceding three chapters lay the basis for the method described in this chapter. In Chapter 4 we determined periodicity of a week or a day for most VMs, which led us to believe the behavior of a VM can be characterized by creating a model for a full week which should repeat itself. In Chapter 5 we characterized the behavior of any VM by either

1. a single normally distributed variable;
2. multiple normally distributed variables,

representing any point in time, or all time points in a week, respectively. In Chapter 6 we implemented two methods for creating a characterization of new VMs based on their supposed similarity to other VMs with the same contextual attributes. We created

3. characterizations for groups of VMs with the same department, customer, operating system, and capacity in multiple normally distributed variables;
4. predictions for the characterization of behavior of new VMs based on department, customer, operating system, and capacity using predictive clustering.

Methods 1-3 provide us with normally distributed variables characterizing the behavior of VMs. Even the deterministic values $\nu^*_{crj}(t); t = 1, ..., 2016$ produced by method 4 can be viewed as normally distributed variables $VU^*_{crj}(t) \sim \mathcal{N}(\nu^*_{crj}(t), 0)$. We will therefore use an important characteristic of normally distributed variables to produce the expected time an SC is in overload. Say we have n normally distributed variables $x_1, ..., x_n$ where $x_j \sim \mathcal{N}(\mu_j, \sigma_j^2); j = 1, ..., n$, we then know that the sum of these variables $X = \sum_{j=1}^{n} x_j$ is normally distributed $X \sim \mathcal{N}(\mu, \sigma^2)$ with mean

$$\mu = \sum_{j=1}^{n} \mu_j \tag{7.1}$$

and variance

$$\sigma^2 = \sum_{j=1}^{n} \sigma_j^2. \tag{7.2}$$

If we now want to predict the expected fraction of time an SC configuration - a subset of all machines in the server park - $SC_c \subset SP$ is in overload, we can model the utilization of the SC at time points $t = 1, ..., 2016$ in a week as

$$SCU_c^*(t) \sim \mathcal{N}(\mu^*(t), (\sigma^*(t))^2), \tag{7.3}$$

where

$$\mu_c^*(t) = \sum_{V_{crj} \in SC_c} \mu_{crj}^*(t) \tag{7.4}$$

and

$$(\sigma_c^*(t))^2 = \sum_{V_{crj} \in SC_c} (\sigma_{crj}^*)^2. \tag{7.5}$$

We can then calculate the probability of SC overload at every time point $t = 1, ..., 2016$ as [Wei16]

$$\mathbb{P}(SCU_c^*(t) > SCcpu_c) = 1 - \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{SCcpu_c - \mu_c^*(t)}{\sigma_c^*(t)\sqrt{2}}\right)\right) \tag{7.6}$$

and the expected time an SC is in overload in a week

$$\mathbb{E}\left[\sum_{t=1}^{2016} \mathbb{1}_{SCU_c(s) > SCcpu_c}\right] = \sum_{t=1}^{2016}\left(1 - \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{SCcpu_c - \mu_c^*(t)}{\sigma_c^*(t)\sqrt{2}}\right)\right)\right). \tag{7.7}$$

The expected frequency of overload can then be found as

$$f_c^* = \frac{\mathbb{E}\left[\sum_{t=1}^{2016} \mathbb{1}_{SCU_c(s) > SCcpu_c}\right]}{2016}. \tag{7.8}$$

Chapter 8

Evaluation

This chapter will evaluate the proposed methods in Chapters 5 through 7. To this end we ran evaluation queries in the Splunk [Spl16b] big data software containing vCenter vSphere [VMw15] configuration and utilization data over the past year of all KPN vCenters.

We have created a *scope* of SCs, RPs and VMs, where we only consider those that conform to the *standard configuration* as described in Section 3.2. This scope contains 34 SCs, that facilitate 157 RPs that span 6.852 VMs.

A single VM provides us with 2016 data points every week, so even the 6016 VMs within our scope generate around $12 \cdot 10^6$ data points every week. With the performance of the Splunk platform in mind, we have only used the data between 13-02-2016 (t_{init}) and 18-06-2016 (t_{end}) for our validation, spanning 18 weeks (approximately one third of a year). This has the auxiliary advantage that, considering a data retention time of a year in the KPN Splunk platform, the experiments are repeatable in the foreseeable future.

We then have $T = 18 \cdot 7 \cdot 288 = 36.288$, where $t = 1$ is the first 5 minute interval after midnight of 13-02-2016 and $t = 36.288$ is the last 5 minute interval before midnight of 18-06-2016. In this time range we determined 4333 VMs in scope to be *active*, where we see an *active* machine as one that has given data points for at least the points $\{T - 2 \cdot 2016, ..., T\}$. This definition of *active* is necessary because we need at least one week of data to create a characterization and at least one more week of data to evaluate this characterization.

8.1 Existing VM

We have implemented two ways of characterizing the behavior of a VM in Chapter 5. The first uses a single normally distributed variable with mean μ^*_{crj} representing the CPU utilization of the VM V_{crj}, the second

takes advantage of the periodicity that VMs show over a week and characterizes a VM as a collection of 2016 independent normally distributed variables with means $\mu^*_{crj}(t); 0 < t \leq 2016$, each representing the mean use of a five minute interval in the week.

We have a training set of the utilization of all active VMs captured on timepoints $T_1 = \{1, ..., T - 2016\}$. Based on this training set we have deduced characterizations of all active VMs where a normally distributed variable U^*_{crj} with mean μ^*_{crj} and σ^*_{crj} characterizes the utilization in a single variable and the normally distributed variables $U^*_{crj}(t); t \in \{1, ..., 2016\}$ characterize the utilization in multiple independent variables.

For comparison we have a test set of the utilization of these VMs captured on timepoints in T_1, defined as $U_{crj}(t); t \in T_1$. We can then apply the performance measure P_1 as defined in Section 3.4.1. For a single time-independent normally distributed variable this gives us

$$Psingle_1 := \frac{1}{|T_1| \cdot mSP} \sum_{c=1}^{mSP} \frac{1}{mSC_c} \sum_{r=1}^{mSC_c} \frac{1}{mRP_{cr}} \sum_{j=1}^{mRP_{cr}} \sum_{s \in T_1} \left(VU_{crj}(s) - \mu^*_{crj}\right)^2 \tag{8.1}$$

and for multiple time-dependent variables

$$Pmultiple_1 := \frac{1}{|T_1| \cdot mSP} \sum_{c=1}^{mSP} \frac{1}{mSC_c} \sum_{r=1}^{mSC_c} \frac{1}{mRP_{cr}} \sum_{j=1}^{mRP_{cr}} \sum_{s \in T_1} \left(VU_{crj}(s) - \mu^*_{crj}(s)\right)^2. \tag{8.2}$$

We then find the performance measures

$$Psingle_1 = 490,690.04 \text{ and } Pmultiple_1 = 368,172.54, \tag{8.3}$$

so we can conclude a model with multiple random variables provides considerably better characterizations, a 20% decrease in error. This was expected, as we create a more refined characterization of utilization when we use more variables.

If we apply the mean square error to every single machine, e.g.

$$E_{crj,1,1} = \frac{1}{|T_1|} \sum_{s \in T_1} \left(VU_{crj}(s) - \mu^*_{crj}\right)^2, E_{crj,1,2016} = \frac{1}{|T_1|} \sum_{s \in T_1} \left(VU_{crj}(s) - \mu^*_{crj}(s)\right)^2 \tag{8.4}$$

and plot these as an error distribution (see Figure 8.1) we see some overfitting has taken place, as for a few VMs the error on a single random variable is smaller than for multiple random variables. However, on most VMs a significant improvement has been achieved using multiple variables.

Since we can apply this performance measure to any subset of VMs $N \subset SP$ as

$$P_{N,1,2016} = \frac{1}{|N||T_1|} \sum_{V_{crj} \in N} \sum_{s \in T_1} \left(VU_{crj}(s) - \mu^*_{crj}(s)\right)^2, \tag{8.5}$$

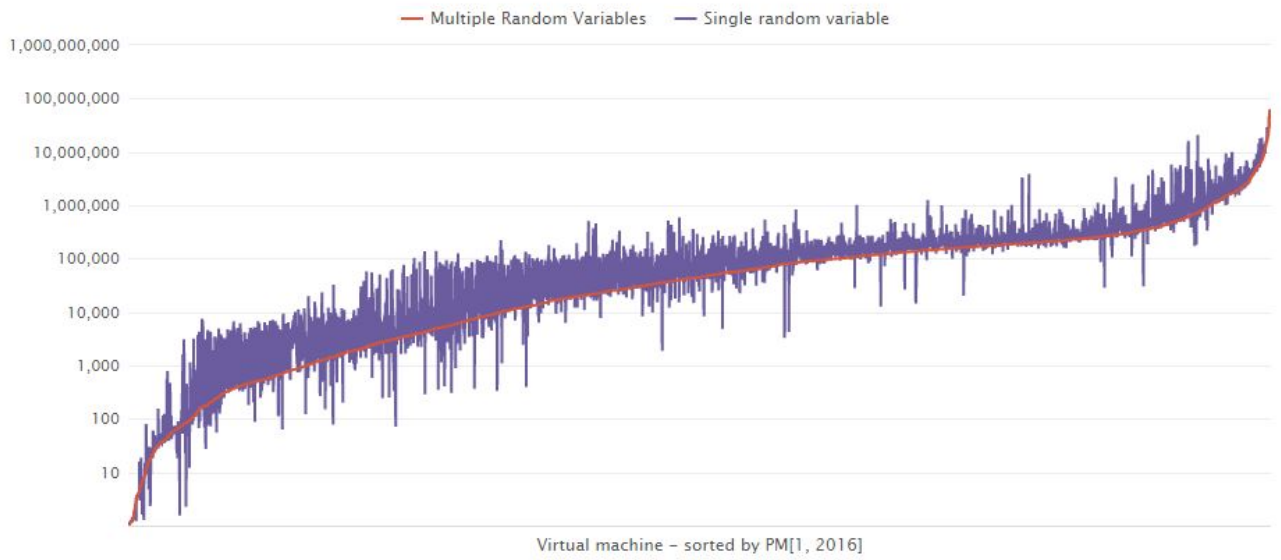
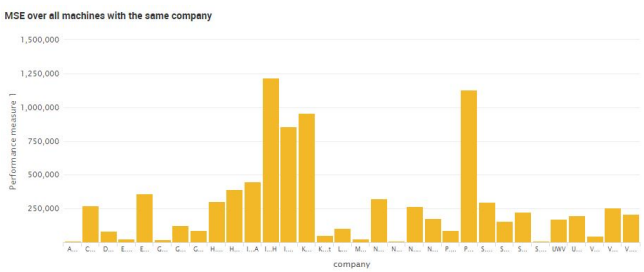
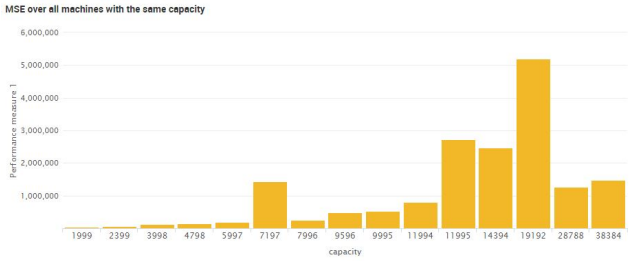


Figure 8.1: Performance measure on individual VMs for both a single random variables and multiple random variables, sorted by the size of $E_{crj,1,1}$. The calculated error is on the logarithmical scaled y-axis.

we have determined performance on the distinctions in the contextual attributes mentioned in Chapter 6. We observed that the height of the performance measure is highly dependent on the renting company (Figure 8.2a) and the capacity (Figure 8.2b) of the VM. A higher error on a VM with higher capacity was expected, as a higher capacity allows for larger variation, but at this point we cannot provide an explanation for a higher error on specific companies.



(a) Performance measure on all VMs of a renting company.



(b) Performance measure on all VMs with the same capacity.

Figure 8.2: Performance measure on groups of VMs, grouped by (left) company and (right) capacity

8.2 New VM

In Chapter 6 we proposed two methods to predict the behavior of a new VM. The first groups the VMs on all available contextual attributes, the second is an implementation of the predictive clustering algorithm. We will evaluate the accuracy of these methods as described in 3.4.2 in this section.

For evaluation we used the characterizations of VMs derived in Chapter 5. Out of these characterizations we randomly picked 10% (approximately 380 VMs) as a test set. The remaining 90% (approximately 3500 VMs)

was used as a training set. For the VMs in the training set we used both their contextual and their behavioral attributes, while we regarded the VMs in the test set as new machines where only the contextual attributes were known.

The method of grouping the VMs provided us with a way to predict the behavior of new VMs based on the similarity within groups. We created normally distributed variables $U^*_{crj}(t); t = 1, ..., 2016$ with means $\mu^*_{crj}(t)$ and standard deviations $\sigma^*_{crj}(t)$ for every group in the training set. The means $\mu^*_{crj}(t)$ of the associated would then predict the behavior of the new VMs in the test set. The predictive clustering algorithm applied to the training set provided us with deterministic predictions $\nu^*_{crj}(t); t \in [0, 2015]$ for the behavior of VMs in the test set.

We can then apply the performance measure P_2 as described in 3.4.2. For the grouping method this evaluates to

$$Pgrouping_2 = \frac{1}{n} \sum_{V''_{crj} \in N} \left(VU_{crj}(t) - \mu^*_{crj}(t)\right)^2 \tag{8.6}$$

and for predictive clustering

$$Ppredclus_2 = \frac{1}{n} \sum_{V''_{crj} \in N} \left(VU_{crj}(t) - \nu^*_{crj}(t)\right)^2 \tag{8.7}$$

We then find for our test set that

$$Pgrouping_2 = 1,074,347.11 \text{ and } Ppredclus_2 = 825,888.84, \tag{8.8}$$

where *predictive clustering* gives us the best results, but still performs 125% worse than the characterization of existing VMs. It should be noted that this error was calculated on a much smaller test set (10% of all active VMs) than the errors $Psingle_1$ and $Pmultiple_1$. We could, however, conclude that we do not have enough contextual information on new VMs to predict a characterization for their utilization.

8.3 Risk of SC configuration

We have proposed a method to assess the risk of any (changed) SC configuration in Chapter 7. This method was based on the results obtained in the previous Chapters 4-6, where we distilled characterizations for *existing* VMs and predicted the behavior of *new* VMs. As most proposed methods rely on normally distributed variables, the risk assessment was based on summing these variables at every time point and determining the risk of overload based on these summed variables. Even the predictions obtained by predictive clustering can be applied to this method, since we can view those as normally distributed variables with as means the obtained values and zero-valued standard deviations.

Since the characterization of behavior is expected to be more accurate than the prediction of behavior of a new VM, the accuracy of this model should be assessed for two different scenarios, as described in Section 3.4.3. At first we will evaluate the risk assessment method as if all machines in all new configurations are *existing*, so we will only apply the characterization techniques. Afterward, we will regard the machines in the test set of Section 8.2 as *new* VMs. We will apply the performance measure described in Section 3.4.3 to both scenarios.

Because most current configurations do not regularly experience overload, we will use fractions $x \in (0,1) \in \mathbb{R}$ of the actual CPU capacity of SCs. We can then define the frequency of overload as

$$f_c(x) = \frac{\sum_{t=T-2.016}^{T} \mathbb{1}_{SCU_c(s)>x \cdot SCcpu_c}}{2016} \tag{8.9}$$

We have chosen x to be in $\{0.25, 0.5, 0.75\}$ to be able to identify the effectiveness of our method on several levels of overload. For $x = 0.25$ most configurations will have a very high frequency of overload, nearly 1, while for $x = 0.75$ this should be near zero.

We have used all utilization data for all existing VMs on timepoints $T_1 = \{1, ..., T - 2.016\}$ as the training set. From this data we obtained:

- A characterization defined by 2016 normally distributed variables as described in Section 5.4;
- For the test set of Section 8.2: a prediction of the characterization as described in 6.3. This prediction will give deterministic values, but we will interpret them as normally distributed variables with as means the values given and as standard deviations 0. For the training set of Section 8.2: a characterization defined in 2016 normally distributed variables as described in Section 5.4.

We then used these characterizations to assess expected frequency $f_c^*(x); c = 1, ..., mSP$ of overload in every current SC configuration for $x \in \{0.25, 0.50, 0.75\}$.

The data points $T_2 = \{T - 2015, T\}$ were used as a test set. We determined the utilization $SC_c(t)$ for all SCs $c = 1, 2, ..., mSP$ for all time points in the test set $t \in T_2$ and determined the frequency of overload $f_c(x)$ for $x \in \{0.25, 0.50, 0.75\}$.

We can then apply the performance measure as described in 3.4.3 to determine the quality of our method, where in this case we have

$$P_3(x) = \frac{1}{mSP} \sum_{c=1}^{mSP} |f_c^*(x) - f_c(x)|.$$

We present the heights of the performance measure in Table 8.1. We then see that on theoretical high risk configurations the method for SC configurations with only *existing* VMs performs reasonably well, but with *new* VMs the method deteriorates quickly. On lower risk configurations the performance is (surprisingly) better when *new* virtual machines are in the configuration.

Existing		New	
x	$P_3(x)$	x	$P_3(x)$
0.25	0.0344	0.25	0.1145
0.50	0.0344	0.50	0.0418
0.75	0.0345	0.75	0.0345

Table 8.1: Performance measures for *risk assessment* with and without new VMs for values $x \in \{0.25, 0.5, 0.75\}$.

In Table 8.2 the errors for individual SCs are shown. For every SC c we show the values $e_c(x) = |f_c^*(x) - f_c(x)|$ for $x \in \{0.25, 0.50, 0.75\}$ for the scenarios without ($e_c(x)$) and with ($e_c(x)$ *new*) new machines. The value 'Fraction of VMs *new*' is the fraction of the VMs on that SC that were regarded as *new*. If we look at the errors in the table we see that *new* VMs usually vastly (negatively) influence the performance of the prediction method.

SC c	$e_c(0.25)$	$e_c(0.50)$	$e_c(0.75)$	Fraction of VMs *new*	$e_c(0.25)$	$e_c(0.50)$	$e_c(0.75)$ *new*
1	0.0126	0.0000	0.0000	0.066	0.5424	0.0000	0.0000
2	0.0000	0.0000	0.0000	0.060	0.0392	0.0000	0.0000
3	0.0000	0.6143	0.8814	0.066	0.0402	0.8057	0.8814
4	0.0000	0.0000	0.0000	0.091	0.0000	0.0000	0.0000
5	0.0000	0.0036	0.0000	0.087	0.0400	0.0040	0.0000
6	0,0000	0,0000	0,0000	0.067	0,0000	0,0000	0,0000
7	0,0004	0,0000	0,0000	0.082	0,0004	0,0000	0,0000
8	0,0000	0,0000	0,0000	0.148	0,0000	0,0000	0,0000
9	0,0000	0,0000	0,0000	0.000	0.0000	0.0000	0.0000
10	0.0000	0.0000	0.0000	0.229	0.0000	0.0000	0.0000
11	0.0000	0.0000	0.0000	0.097	0.0000	0.0000	0.0000
12	0.0004	0.0000	0.0000	0.099	0.0093	0.0000	0.0000
13	0.0000	0.0000	0.0000	0.082	0.0000	0.0000	0.0000
14	0.0100	0.0004	0.0000	0.118	0.0101	0.0004	0.0000
15	0.0030	0.0005	0.0000	0.097	0.0271	0.0005	0.0000
16	0.0073	0.0001	0.0000	0.082	0.0421	0.0001	0.0000
17	0.0000	0.0000	0.0000	0.148	0.0000	0.0000	0.0000
18	0.0146	0.0000	0.0000	0.122	0.0010	0.0000	0.0000
19	0.0689	0.0000	0.0000	0.056	0.3182	0.0000	0.0000
20	0.0080	0.0000	0.0000	0.129	0.0355	0.0000	0.0000
21	0.0002	0.0000	0.0000	0.098	0.0000	0.0000	0.0000
22	0.0000	0.0000	0.0000	0.062	0.0000	0.0000	0.0000
23	0.0000	0.0000	0.0000	0.119	0.0000	0.0000	0.0000
24	0.0000	0.0000	0.0000	0.133	0.0000	0.0000	0.0000
25	0.0000	0.0000	0.0000	0.122	0.0000	0.0000	0.0000

Table 8.2: Prediction error on SCs with or without *new* VMs.

If we plot the expected (predicted) overload frequency against the (actual) overload frequency in Figure 8.3, we see that for a frequency smaller than 0.10 the prediction usually comes out at near zero, while for larger than 0.10 the prediction does give a high probability. This is exactly where we want the turning point to be. A low risk ($0 \leq f_c^* \leq 0.10$ for SC c) is acceptable for a configuration, and a high risk should be signaled by the model.

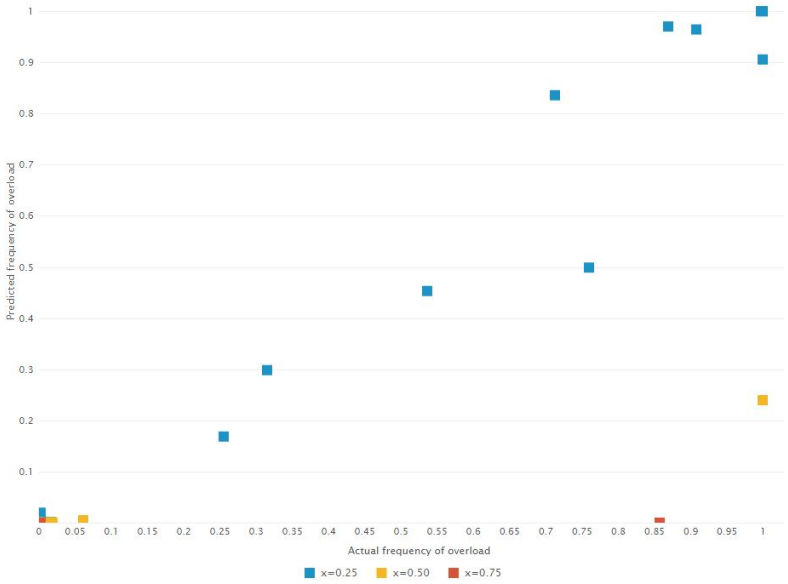


Figure 8.3: The expected overload frequency of an SC configuration for different actual overload frequencies.

Chapter 9

Conclusions

Physical server clusters can be overcommitted with respect to their resources to maximize utilization of the available infrastructure. When a server cluster is overcommitted it is possible that the virtual machines will request more resources than physically available. Therefore, when managing server cluster configuration, a line is constantly tread between too much overcommitment (causing overload and performance issues) and too little overcommitment (causing under-utilization of the hardware).

Earlier works have proposed solutions to this (Virtual Machine Placement) problem. The solutions usually consist of two parts: first a heuristic is chosen for the risk of overload in a server cluster configuration, thern this heuristic is used in, for example, bin-packing problems to find an optimal placement of virtual machines. Where most works focus on the placement, this work focused on the heuristics.

In this thesis, first we proposed models that characterized virtual machines in their utilization over a week, since they showed periodicity in weeks. Second, we proposed two methods to predict the (characterization of) utilization of new virtual machines where no previous utilization was know. Both these methods resulted in multiple normally distributed variables that model the behavior of the virtual machines at every time point (five minute interval) of a week. Finally, we used these characterizations to find the expected time that a server cluster configuration would experience overload, in essence the time in a week that the virtual machines in the configuration requested more virtual resources than physically available.

We evaluated the proposed methods using real life utilization traces provided by KPN[1]. A vast improvement was shown when using multiple normally distributed variables versus a single variable to characterize the utilization of a pre-existing virtual machine. The available contextual data was insufficient to predict the characterizations of new virtual machines. The proposed method for risk assessment of a server cluster configuration was sufficient

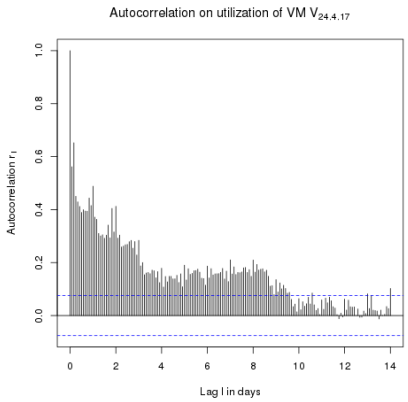
[1] A telecommunications, internet service and IT services provider, with a large division focused on providing Platform as a Service to corporate clients.

to correctly classify configurations with very high and low risk, but lacked some accuracy when the expected frequency of overload was around 0.05, in essence the critical area.

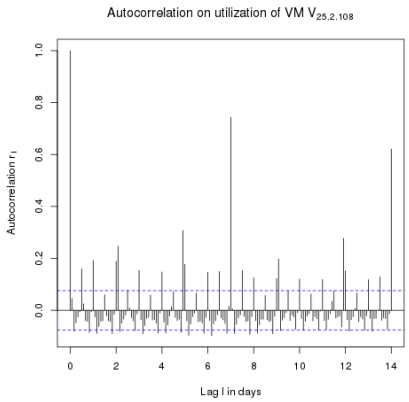
Future work could implement the proposed heuristic as a heuristic for methods solving the VMP problem, to determine if it improves performance. It is also possible that the proposed methods are more effective on other platforms or in a more controlled (simulated) environment.

Appendix A

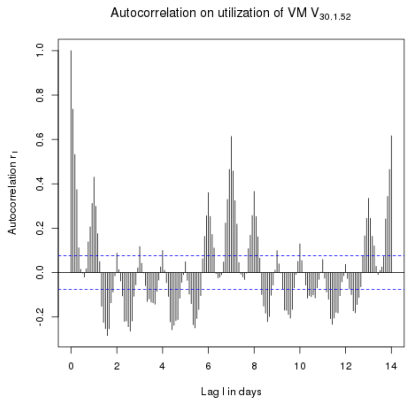
Autocorrelation results



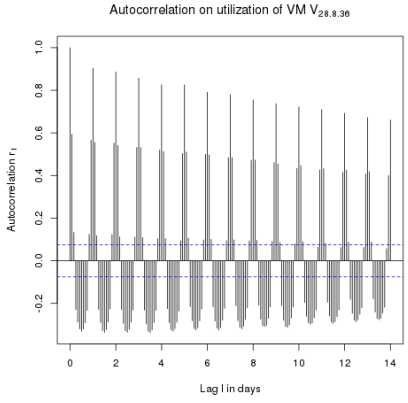
(a) Autocorrelation results for VM $V_{24,4,17}$.



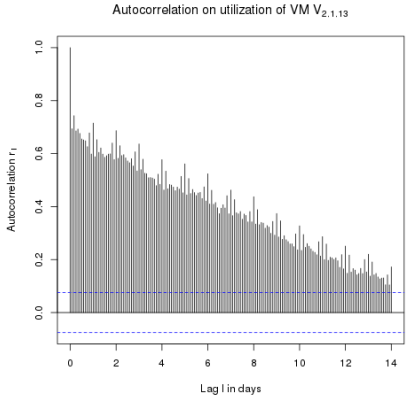
(b) Autocorrelation results for VM $V_{25,2,108}$.



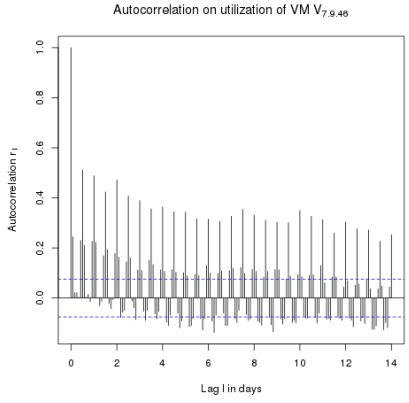
(c) Autocorrelation results for VM $V_{30,1,52}$.



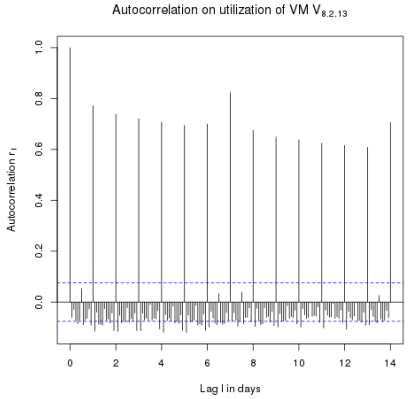
(d) Autocorrelation results for VM $V_{28,8,36}$.



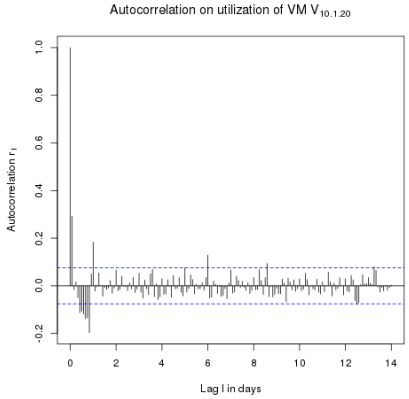
(e) Autocorrelation results for VM $V_{2,1,13}$.



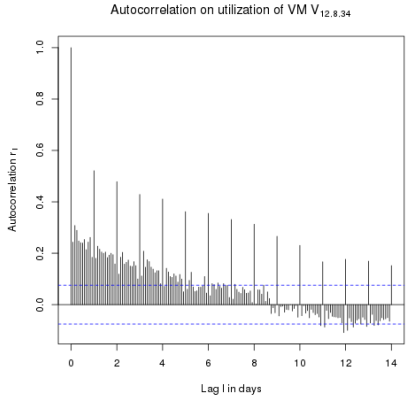
(f) Autocorrelation results for VM $V_{7,9,46}$.



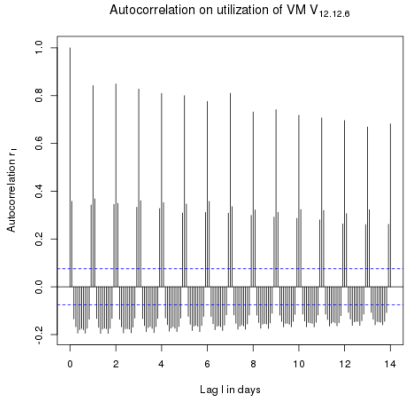
(g) Autocorrelation results for VM $V_{8,2,13}$.



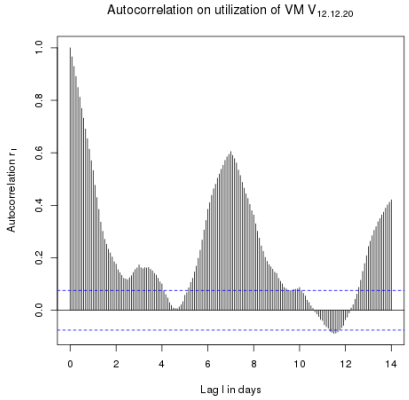
(h) Autocorrelation results for VM $V_{10,1,20}$.



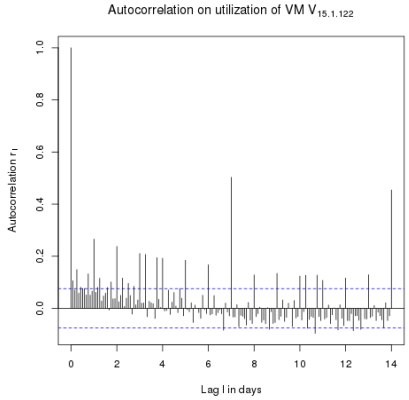
(i) Autocorrelation results for VM $V_{12,8,34}$.



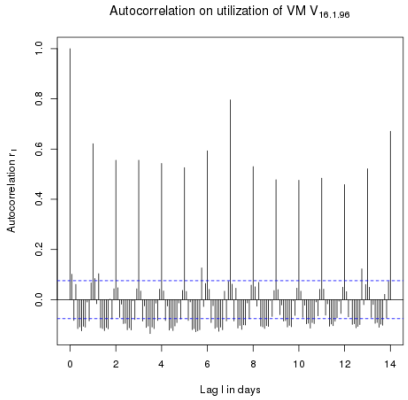
(j) Autocorrelation results for VM $V_{12,12,6}$.



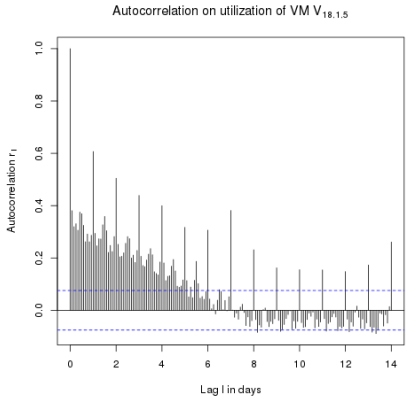
(k) Autocorrelation results for VM $V_{12,12,20}$.



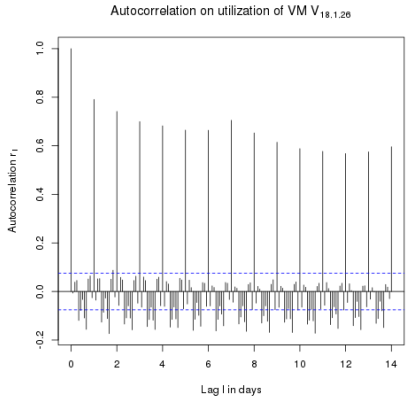
(l) Autocorrelation results for VM $V_{15,1,122}$.



(m) Autocorrelation results for VM $V_{16,1,96}$.



(n) Autocorrelation results for VM $V_{18,1,5}$.



(o) Autocorrelation results for VM $V_{18,1,26}$.

Bibliography

[AA13] R. Adhikari and R. Agrawal. An introductory study on time series modeling and forecasting. *CoRR*, 1302.6613, 2013.

[BDR98] H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 55–63, 1998.

[BKB07] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing SLA violations. *10th IFIP/IEEE International Symposium on Integrated Network Management*, 5:119–128, 2007.

[CD14] Z. Cao and S. Dong. An energy-aware heuristic framework for virtual machine consolidation in Cloud computing. *Journal of Supercomputing*, 69(1):429–451, 2014.

[CZS+11] M. Chen, H. Zhang, Y. Y. Su, X. Wang, G. Jiang, and K. Yoshihira. Effective VM sizing in virtualized data centers. *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, IM 2011*, pages 594–601, 2011.

[Dur10] R. Durrett. Probability: Theory and examples. *Biometrics*, 49(3):497, 2010.

[FDH11] T. Ferreto, C. A. F. De Rose, and H. U. Heiss. Maximum migration time guarantees in dynamic server consolidation for virtualized data centers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6852 LNCS(1):443–454, 2011.

[GP12] H. Goudarzi and M. Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, pages 750–757, 2012.

[Gra03] C. W. J. Granger. Time Series Concepts for Conditional Distributions. *Oxford Bulletin of Economics and Statistics*, 65(SUPPL.):689–701, 2003.

[HP13] I. Hwang and M. Pedram. Hierarchical virtual machine consolidation in a cloud computing system. *IEEE International Conference on Cloud Computing, CLOUD*, pages 196–203, 2013.

[Jol02] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 2nd edition, 2002.

[KZL+10] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. *1st ACM Symposium on Cloud Computing (SoCC '10)*, pages 39–50, 2010.

[LPB15] F. Lopez-Pires and B. Baran. Virtual Machine Placement Literature Review, 2015.

[Mac67] J. B. MacQueen. Some Methods for classification and Analysis of Multivariate Observations. *5th Berkeley Symposium on Mathematical Statistics and Probability 1967*, 1(233):281–297, 1967.

[MN08] S. Mehta and A. Neogi. ReCon: A tool to recommend dynamic server consolidation in multi-cluster data centers. In *NOMS 2008 - IEEE/IFIP Network Operations and Management Symposium: Pervasive Management for Ubiquitous Networks and Services*, pages 363–370, 2008.

[OF16] J. Ortigoza and L. Fabio. Dynamic environments for virtual machine placement considering elasticity and overbooking, 2016.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, P. Blondel, M. and Prettenhofer, V. and Weiss, R. and Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[R C14] R Core Team. R: A language and environment for statistical computing, 2014. `http;//www.R-project.org/`.

[RSM+13] B. C. Ribas, R. M. Suguimoto, R. A. N. R. Montaño, F. Silva, and M. Castilho. PBFVMC: A new Pseudo-Boolean formulation to virtual-machine consolidation. *Proceedings - 2013 Brazilian Conference on Intelligent Systems, BRACIS 2013*, pages 201–206, 2013.

[SG10] V. Soundararajan and K. Govil. Challenges in building scalable virtualized datacenter management. *ACM SIGOPS Operating Systems Review*, 44(4):95, 2010.

[Spl16a] Splunk. Search Processing Language, 2016. `http://docs.splunk.com/Documentation/Splunk/latest/Search/Aboutthesearchlanguage`.

[Spl16b] Splunk. Splunk Enterprise, 2016. `https://www.splunk.com/`.

[Str11] J. Struyf. Clus, 2011. `http://clus.sourceforge.net/doku.php`.

[Ven07] D. Vengerov. A reinforcement learning approach to dynamic resource allocation. *Engineering Applications of Artificial Intelligence*, pages 1–18, 2007.

[VMw15] VMware. VMware vSphere, 2015. `http://www.vmware.com/products/vsphere/overview.html`.

[VMw16] VMware. VMware vSphere Hypervisor, 2016. `https://www.vmware.com/products/vsphere-hypervisor/`.

[VR02] W. N. Venables and B. D. Ripley. Modern Applied Statistics with S. *Issues of Accuracy and Scale*, 1(March):868, 2002.

[Wei16] E. W. Weisstein. Normal Distribution, 2016. `http://mathworld.wolfram.com/NormalDistribution.html`.

[WMZ11] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *Proceedings - IEEE INFOCOM*, pages 71–75, 2011.