VM and Workload Fingerprinting for Software Defined Datacenters

ARCHIVES

by

Dragos Ciprian Ionescu

S.B., C.S. M.I.T., 2012

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2013

© Massachusetts Institute of Technology 2013. All rights reserved.

Author ....
Department of Electrical Engineering and Computer Science
February 1, 2013

Certified by ....
Saman Amarasinghe
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by ....
~~Christopher J. Terman~~
Chairman, Masters of Engineering Thesis Committee
Dennis M. Freeman

VM and Workload Fingerprinting for Software Defined Datacenters

by

Dragos Ciprian Ionescu

Submitted to the Department of Electrical Engineering and Computer Science
on February 1, 2013, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this work we develop strategies for mining telemetry streams in virtualized clusters to automatically discover relationships between sets of virtual machines. Examples of relationships include correlations between virtual machines, similarities in resource consumption patterns or dominant resources, and similarities in metric variations.

The main challenge in our approach is to transform the raw captured data consisting of resource usage and VM-related metrics into a meaningful fingerprint that identifies the virtual machine and describes its performance. In our analysis we try to determine which of these metrics are relevant and how they can be expressed as a light-weight and robust fingerprint that offers insight about the status of the machine.

Thesis Supervisor: Saman Amarasinghe
Title: Professor of Computer Science and Engineering

Acknowledgments

This thesis would not have been possible without the generous help and support of multiple individuals to whom I would like to express sincere gratitude.

To begin, I would like to thank VMware, the company at which I developed this project. As an MIT VI-A student, I was granted this unique industrial research opportunity and the chance to collaborate with a team of highly skilled engineers. I am grateful to VMware for assigning me a cutting-edge project and for providing me with the resources necessary for its completion.

I would also like to extend gratitude to the VMware colleagues who had an impact on this work. To Rean Griffith, you are an exceptional mentor and have given me immeasurably valuable guidance since day one. I'm especially appreciative of your unwavering support and dedication to the project. Naveen Nagaraj, I was honored to have you as my manager and I am thankful for your feedback. Additionally, special thanks is in order for my extended colleagues and collaborators: Xiaoyun Zhu, Anne Holler, Pradeep Padala, the members of the DRM team, Rishi Bidarkar, Christian Dickmann, Adam Oliner (U.C. Berkeley), Zhelong Pan, E. Lewis, Uday Kurkure, Sunil Satnur, Ravi Soundararajan, Steve Muir, Steve Herrod.

Outside VMware, I am indebted to those from within the MIT community who have helped me during both my undergraduate and graduate education. To my thesis advisor, Professor Saman Amarasinghe, thank you for your direction and mentorship. To the VI-A program, you coordinate and provide one of the best opportunities for MIT students. I wholeheartedly recommend your program to any and all MIT students seeking practical experience at one of the world's leading companies. Furthermore, I wish to specifically thank the course administrators and assistants for their patience and support.

Finally, I would not be where I am today without my loving, supportive and encouraging family and my great friends. I am lucky and blessed to have all of you in my life. Thank you.

Contents

1 Introduction **13**

1.1 Fingerprinting Motivation 14

1.1.1 VM Relationships Exploitation 14

1.1.2 Performance Explanation 16

1.2 Thesis Outline 16

2 Theoretical Background **19**

2.1 The Pearson Correlation Coefficient 19

2.2 Principal Component Analysis 20

2.3 K-Means Clustering 21

2.4 Selecting the Number of Clusters 22

2.4.1 The Elbow Method 22

2.4.2 The Bayesian Information Criterion 23

2.5 Logistic Regression 26

2.6 Entropy, Relative Entropy and Mutual Information 28

2.7 Silverman's Test for Multimodality 29

2.8 Bayesian Inference 30

3 Data Primer **33**

3.1 Virtual Data Center Architecture 33

3.2 Data Collection 35

3.3 Data Storage 37

3.4 In-memory Data Representation 40

3.5 Data Filtering 41
3.6 Data Sources 42

4 Pipeline Overview **43**
4.1 Correlation Pipeline 44
4.2 Clustering Pipeline 46
4.3 Dominant Resources Pipeline 47
4.4 Bayesian Inference Pipeline 48

5 Results **51**
5.1 ACluster 51
5.2 View Planner 54
5.2.1 View Planner Experiment Overview 55
5.2.2 Correlation Analysis 57
5.2.3 Clustering Analysis 60
5.2.4 Bayesian Inference Analysis 70
5.3 Nimbus 76
5.3.1 Clustering Analysis 76
5.3.2 Dominant Resources Analysis 77
5.3.3 Cluster Diagnosis 79

6 Related Work **83**

7 Summary of Contributions and Future Work **87**
7.1 Contributions 87
7.2 Future Work 87

References **89**

List of Figures

1-1 VM Placement and Resource Management . . . 15

1-2 Key Performance Indicator Distribution . . . 16

2-1 Elbow Method Example 1 . . . 23

2-2 Elbow Method Example 2 . . . 23

2-3 Bayesian Information Criterion Example . . . 25

2-4 Bayesian Information Criterion Example . . . 25

3-1 Virtual Data Center Architecture . . . 34

3-2 Performance Counter Categories . . . 36

4-1 Conceptual Pipeline Overview . . . 43

4-2 Conceptual Correlation Pipeline . . . 44

4-3 VM Correlation Heatmap . . . 45

4-4 Cluster Correlation Heatmap . . . 46

4-5 Clustering Pipeline . . . 46

4-6 Logistic Regression Fingerprint . . . 47

4-7 Dominant Resources Fingerprint . . . 48

4-8 Bimodal Distribution Example . . . 48

4-9 Conditional Distribution Example . . . 49

5-1 Pairwise Pearson Correlation Heatmaps - ACluster . . . 53

5-2 Cluster Correlation Heatmap - ACluster . . . 53

5-3 ACluster PCA K-Means . . . 54

5-4 VMware View Planner Architecture . . . 56

5-5 Pairwise Pearson Correlation Heatmaps 57
5-6 Pairwise Distance Correlation Heatmaps 59
5-7 Cluster Correlation Heatmap - View Planner 59
5-8 View Planner PCA K-Means Clustering 61
5-9 View Planner Metric Filtering 63
5-10 View Planner Metric Distribution Percentiles 64
5-11 The Elbow Method - View Planner 65
5-12 The Bayesian Information Criterion - View Planner 65
5-13 Three Cluster Partition - View Planner 66
5-14 Precision, Recall and F-measure - View Planner 69
5-15 Performance Indicator Distribution 71
5-16 CPU Usage Conditional Distribution 1 73
5-17 CPU Usage Conditional Distribution 2 73
5-18 Revised K-Means Clustering 1 74
5-19 Revised K-Means Clustering 2 75
5-20 Nimbus PCA K-Means 77
5-21 Owner and Deployment Type - Nimbus 77
5-22 Nimbus Dominant Resources Distribution 78
5-23 VM Dominant Resources Types Analysis 79
5-24 Nimbus Subcluster Diagnosis 80
5-25 Nimbus Revised Deployment Type Color Map 82

List of Tables

3.1 Default Stat Collection Periods 35
3.2 Performance Metrics Roll-Up Options 36

5.1 View Planner Desktop VM correlations 58
5.2 View Planner Metric Filtering Statistics 63
5.3 Raw VS PCA Cluster Stability - View Planner 67
5.4 Blue Cluster Fingerprint - View Planner 68
5.5 Red Cluster Fingerprint - View Planner 69
5.6 Silverman Multimodality Test Results 71
5.7 Mutual Information Table 72
5.8 ViewPlanner Spread Metrics 75
5.9 Nimbus Subcluster Fingerprint 1 80
5.10 Nimbus Subcluster Fingerprint 2 81
5.11 Nimbus Spread Metrics 82

Chapter 1

Introduction

In this work we are exploring novel ways of modeling virtual machine (VM) metrics in order to get meaningful performance related information. More concretely, we aim to develop tools to automatically identify patterns for groups of virtual machines and application workloads and to do collaborative performance bug diagnosis.

In order to achieve these goals we design a fingerprint that can be used to identify a VM and characterize its performance. Having such a construct is important since it can help us to identify performance problems within a VM, compare the performance of one VM to that of another to get insight about compatible co-location, and compare between similar clusters of VMs run by different clients to identify the factors that degrade performance.

Our design problem is complicated since it is not clear what the building blocks of a fingerprint should be. For instance, we first need to understand which performance metrics we need to take into consideration. Given a set of performance counters, we need to determine whether there are any redundancies or if we are missing any relevant metrics. In addition, it is crucial to figure out how to transform the set of metrics into a robust fingerprint that accurately models the virtual machine. We elaborate more on these challenges in the remainder of this section.

The data we collect comes in a raw format, such as a CSV file populated with resource usage measurements for the VM and information about its corresponding host and resource pool. As a first challenge, we need to understand which of these

metrics are relevant to identifying a VM and reasoning about its performance. For example, we could look only at patterns in resource usage, such as changes in CPU or memory. Given this subset of all the metrics, we need to decide whether it is complete with respect to our purpose, i.e. whether it can be successfully used to fingerprint a VM.

Redundant data is yet another challenge that we plan to address using algorithms such as principal component analysis (PCA) to shrink the size of the original candidate metrics.

The input we collect is passed in a second stage to a lightweight sophisticated analytics pipeline that reveals the relationships between virtual machines and produces a fingerprint. More details on our approach are given in later chapters. The next two sections describe the motivation and outline of this thesis.

1.1 Fingerprinting Motivation

To give the motivation for this thesis we introduce its possible use cases in this section. We have identified three main applications for our work: two of them are the result of exploiting VM relationships, while the remaining one is focused on explaining performance.

1.1.1 VM Relationships Exploitation

Detecting VM relationships without human intervention is important because such information can be used to automatically refine placement and resource management decisions. The high level idea is depicted in Figure 1-1.

There are two use cases suggested by Figure 1-1:

Use Case 1: Automatic assignment of affinity / anti-affinity rules based on VM workload

The hardware resources are usually organized into resource pools, such that each resource pool contains several VMs. The Distributed Resource Manager (DRS) is

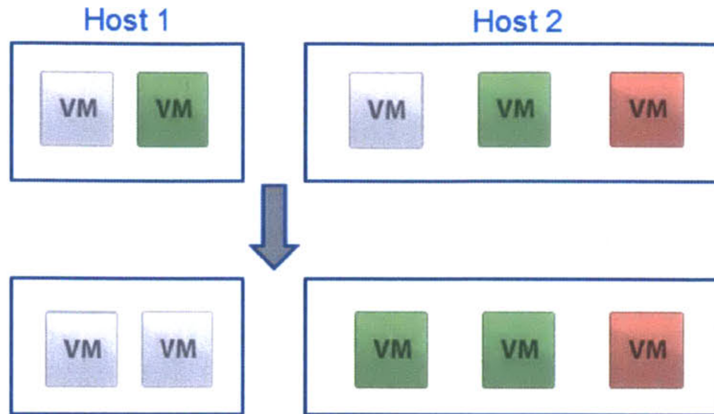

Figure 1-1: VM relationships. Placement and Resource Management.

the entity that dynamically does load balancing and resource allocation for the VMs based on pre-defined rules. Therefore, given the relationships that we discover we can reason about VM affinities and create new DRS rules. For example, in the scenario presented in Figure 1-1, our algorithm may hint that the grey VMs should always be placed on the same host due to their affinity. A rule to reflect this hint gets created, and after DRS executes it we get the new configuration shown in Figure 1-1 with both of the grey VMs on Host 1.

Use Case 2: Automatic grouping of VMs based on their behavior

We may discover that certain VMs are related with respect to their metric patterns. In this case, we will want to make use of the information in order to refine our resource allocation decisions. For example, we may create rules that stop DRS from stealing resources away from related VMs. In particular, we consider again Figure 1-1 and assume that the green VM placed initially on Host 1 needs more memory. However, Host1 is overloaded and DRS discovers that it could steal some memory from the green VM on Host2, not anticipating that the second VM will also need the resources in the near future. If we enforce our rule, DRS will take a different approach and decide instead to co-locate the two green VMs on Host2, which for the sake of our example has more available memory.

1.1.2 Performance Explanation

The detection of performance problems is in many cases not sufficient, and system administrators will often ask for an explanation. One possible approach is to look at performance abnormalities across deployments as depicted in Figure 1-2.

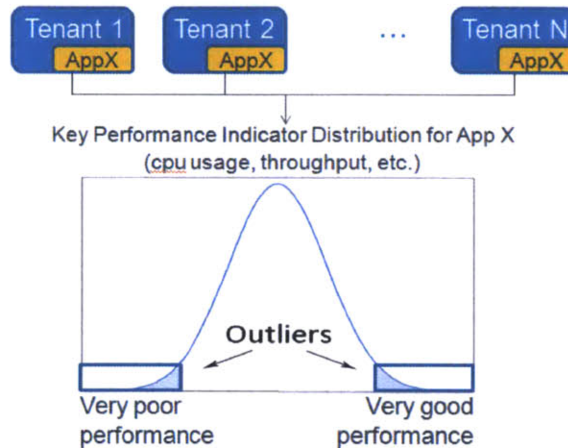

Figure 1-2: Key Performance Indicator Distribution. Explaining performance problems across deployments.

This brings us to our third use case:

Use case 3: Automatic detection and explanation of performance problems or differences across deployments using Bayesian Inference or Latent Variable Modeling

To illustrate our last application, we focus on the example shown in Figure 1-2. Assume you have multiple tenants, 1 through N, all running the same application X, but in different environments and with different configurations. The distribution plot of a chosen key performance indicator such as CPU usage will most likely have some outliers with very good performance and some with very poor performance. In this scenario, we will want to determine the factors that push the tenant to one extreme or the other. We can do so using Bayesian Inference tools.

1.2 Thesis Outline

The rest of this work is organized as follows: Chapter 2 introduces the reader to the algorithms and methods used in our pipeline. Chapter 3 describes our data collection

method and our data model. Chapter 4 outlines the conceptual details of each of our analysis pipelines, thus establishing the main scope of the thesis. Chapter 5 presents the results obtained by applying our analysis pipelines on 3 different systems. Chapter 6 discusses related work. Chapter 7 reviews the contributions of this thesis and gives possible directions for future work.

Chapter 2

Theoretical Background

The purpose of this chapter is to provide the reader with the theoretical background information necessary for understanding our analysis pipeline. At a high level, there are three main steps in our pipeline:

- robust and automatic identification of groups of similar virtual machines
- accurate fingerprint extraction in the form of a summarized/compressed representation of raw metrics for each group
- diagnosis (explaining differences)

We elaborate in the next sections on the different techniques used in our approach. At the beginning of each section we emphasize why the technique is relevant to our fingerprinting method.

2.1 The Pearson Correlation Coefficient

Relevance: We construct an estimate for the similarity between two virtual machines by looking at correlations between metric streams. This approach is a precursor to the clustering techniques we employ and serves as a way to identify relationships.

The Pearson Correlation Coefficient is a measure of the dependence between two quantities. It can be computed by dividing the covariance of two variables by the

product of their standard deviations. Formally, if we let X and Y be our random variables, with expected values μ_x and μ_y and standard deviations σ_x and σ_y, we have that the correlation coefficient ρ_{xy} is given by:

$$\rho_{xy} = \text{corr}(X,Y) = \frac{cov(X,Y)}{\mu_x \cdot \mu_y} = \frac{E[(X-\mu_x)(Y-\mu_y)]}{\mu_x \cdot \mu_y}$$

A correlations coefficient close to 1 corresponds to a positive linear relationship between the two variables, while a coefficient close to -1 corresponds to a negative relationship. Finally, a coefficient close to 0 suggests that there is no relationship between the variables we considered.

2.2 Principal Component Analysis

Relevance: In our approach each VM is characterized by a set of metrics. However, some of these metrics are redundant and by applying PCA we are able to get a more compact representation of the data without significantly affecting our results. By reducing to a 2-dimensional or a 3-dimensional data set we are also able to provide a graphical representation for the cluster of VMs, thus enabling visual inspection of the result.

Principal Component Analysis or PCA ([15]) is a data summarization technique used to explain the variance of a set of signals using the variance of a subset of them. It functions by identifying the key (principal) components and eliminating the redundant features. Therefore, one main use of PCA is finding patterns in high dimensional data for which a graphical representation is impossible. In addition to this, PCA also allows data compression with little information loss.

A series of steps need to be performed to apply PCA on a set of k-dimensional vectors X_i, $1 \leq i \leq n$:

1. Mean centering. Subtract the mean from each of the k data dimensions. We call the resulting vectors $\tilde{X}_i$. We also denote the matrix formed by selecting the $\tilde{X}_i$ vectors as rows by $\tilde{M}$ $(n \times k)$.

2. Calculate the covariance matrix C $(k \times k)$ corresponding to the n-dimensinal zero-mean data.

3. Calculate the unit eigenvectors $(evct_i, 1 \leq i \leq k)$ and the eigenvalues $(eval_i, 1 \leq i \leq k)$ of the covariance matrix. Choose the dimension d to which you want to reduce the data. This is done by first ordering the eigenvectors by eigenvalue (highest to lowest) and then forming a matrix of vectors F $(k \times d)$ out of the top d.

4. Construct the PCA data set by multiplying $\tilde{M}$ and F. In the resulting matrix, the ith row $(1 \leq i \leq n)$, which we denote by Y_i, is the reduced PCA data vector corresponding to X_i

2.3 K-Means Clustering

Relevance: We use K-Means Clustering to identify and group together the VMs that are 'similar' based on their metric values.

K-Means Clustering ([7]) is an unsupervised machine learning technique used to identify structure in a dataset. It works by partitioning the set of points / observations into a given number of clusters so that it minimizes the within-cluster sum of squares to each cluster centroid. Formally, given a set of n k-dimensional data points, X_i, $1 \leq i \leq n$, and a set of s clusters $C = C_1, C_2, ..., C_s$ among which the data needs to be partitioned, the k-means algorithm minimizes the quantity

$$\arg\max_{\mathbf{C}} \sum_{i=1}^{s} \sum_{X_j \in C_i} ||X_j - Z_i||$$

where Z_i is the centroid of C_i.

Given randomly generated initial values $Z_1^0, Z_2^0, ..., Z_s^0$ for the cluster centroids, the k-means algorithm proceeds by iterating through an assignment step and an update step until convergence. At step t we

- Assign each data point X_i to the cluster C_j^t with the closest centroid Z_j^t

- Update the centroid of each cluster based on the assignment: $Z_j^{t+1} = \frac{1}{|C_j^t|} \sum_{X_j \in C_i^t} X_j$

2.4 Selecting the Number of Clusters

Relevance: We employ the Elbow Method and the Bayesian Information Criterion to determine a good choice for the number of clusters in our data set.

The K-Means algorithm requires the number of clusters as an input. This implies that in addition to applying K-Means we also need to determine what is the optimal number of clusters for a given data configuration. A bad choice for the number of clusters can lead to poor results that greatly impact the rest of our analysis pipeline.

Two different methods are used to select the "right" number of clusters: The Elbow Method and The Bayesian Information Criterion. They are presented next.

2.4.1 The Elbow Method

The Elbow Method is one of the oldest and simplest methods for selecting the number of clusters. It was first mentioned by Thorndike in [19]. To understand how it works, we first have to define the distortion D_s for a given partition of the data into s clusters. Using the notation from 2.3, we have:

$$D_s = \frac{1}{n} \sum_{i=1}^{s} \sum_{X_j \in C_i} \|X_j - Z_i\|^2$$

The elbow criterion runs K-Means for $s = 1, 2, 3, ...$ and in each case computes the associated distortion D_s. It's worth mentioning that as the number of clusters increases, the distortion decreases until in eventually reaches the value 0 for $s = n$. At some point, adding more clusters does not produce a better model for the data. Therefore, to choose the best model we look for a sudden drop in the marginal loss of distortion. This drop is usually graphically visible in the form of an angle.

When the drop can be easily identified, the complete graph resembles an elbow, thus the name of the method. An example is given in Figure 2-1, for which the optimal number of clusters is 3.

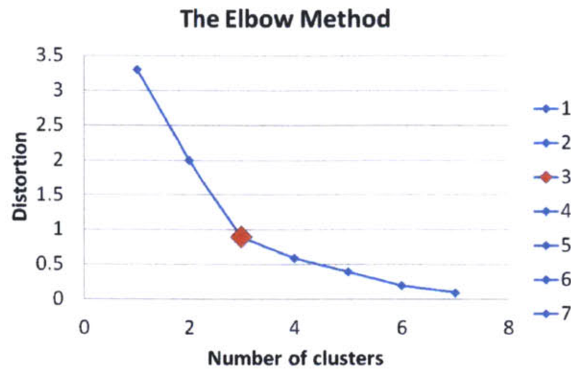

Figure 2-1: Cluster Distortion Plot. Using the elbow criterion to choose the number of clusters.

Unfortunately, it is not always easy to identify the "elbow" using this method. In some cases, like the one illustrated in Figure 2-2, the curve is smooth and has no visible "elbow".

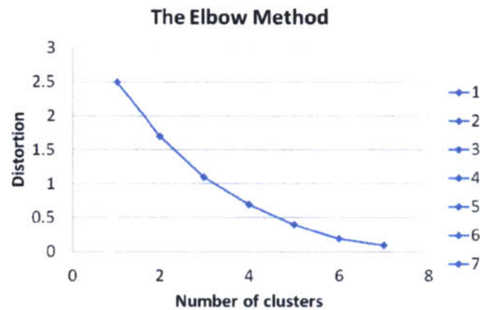

Figure 2-2: Cluster Distortion Plot. Using the elbow criterion fails.

The next criterion tries to improve on the elbow method by introducing a likelihood function.

2.4.2 The Bayesian Information Criterion

The Bayesian Information Criterion (BIC) [8] is a popular tool derived by *Schwarz* and widely used for statistical model selection. It is part of a family of information criteria that also includes the Akaike Information Criterion (AIC) and the Deviance Information Criterion (DIC). BIC is an improvement on the previously presented elbow method. In our approach, it provides a quantitative way for choosing the number of clusters in K-Means.

If we let $L(\theta)$ be the log-likelihood function of the proposed model and m be the number of clusters, then the BIC formula is given by

$$BIC = L(\theta) - \frac{f}{2} \cdot \ln n$$

where f is the number of free parameters and n is the number of observations.

According to [6], if we let s be the number of clusters and k be the number of dimensions, then the number of free parameters is the sum of $s-1$ class probabilities, $s \cdot k$ centroid coordinates, and $s \cdot k \cdot (k-1)/2$ free parameters in the co-variance matrix.

To compute $L(\theta)$ we follow the derivation presented in [16]. If we adopt the identical spherical Gaussian assumption, the maximum likelihood estimate for the variance of the ith cluster is given by

$$\Sigma_i = \frac{1}{n_i - m} \sum_{X_j \in C_i} \|X_j - Z_i\|^2$$

where n_i denotes the size of the ith cluster.

We can then compute $Pr(X_j)$, the probability of a given point in our data set that gets assigned to cluster C_i of size n_i:

$$Pr(X_j) = \frac{n_i}{n} \frac{1}{(2\pi)^{\frac{1}{2}} \Sigma_i^{\frac{k}{2}}} \exp(-\frac{\|X_j - Z_i\|^2}{2\Sigma_i})$$

Finally, the log-likelihood of the ith cluster and the BIC can be computed after some algebraic manipulations:

$$L(\theta_i) = \sum_{j=1}^{n_i} \log(Pr(X_j)) = \sum_{j=1}^{n_i} \log(\frac{n_i}{n} \frac{1}{(2\pi)^{\frac{1}{2}} \Sigma_i^{\frac{k}{2}}} \exp(-\frac{\|X_j - Z_i\|^2}{2\Sigma_i}))$$

$$= n_i \log(n_i) - n_i \log(n) - \frac{n_i}{2} \log(2\pi) - \frac{kn_i}{2} \log \Sigma_i - \frac{n_i - m}{2}$$

$$BIC_m = \sum_{i=1}^{m} (n_i \log(n_i) - n_i \log(n) - \frac{n_i}{2} \log(2\pi) - \frac{kn_i}{2} \log \Sigma_i - \frac{n_i - m}{2}) - \frac{f}{2} \cdot \ln n$$

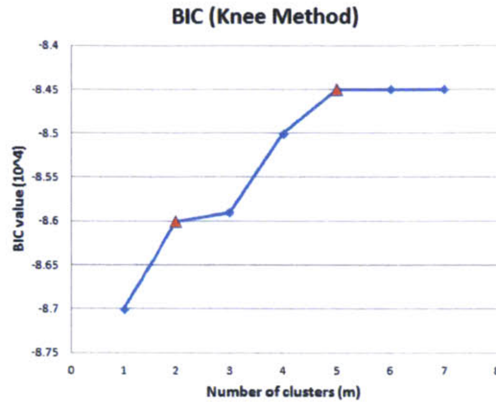

Figure 2-3: Cluster BIC Plot. Using the Bayesian information criterion to choose the number of clusters.

An example BIC plot is shown in Figure 2-3.

One easy way to determine based on the BIC values an appropriate number of clusters for the model is by selecting the *first* local maximum. In Figure 2-3 this occurs at $m = 2$. The knee point detection method is known to work better in general. In the example we provide, two knee shapes can be observed at $m = 2$ and at $m = 5$. The decision is often made based on a difference function F, where $F(m) = BIC_{m-1} + BIC_{m+1} - 2BIC_m$. The plot of F for our example is given in Figure 2-4. To discover a number of clusters that fits the model well, one has to look in this case for prominent troughs. The through that stands out occurs again at $m = 2$ and $m = 5$ an correspond tot the knee-shapes observed in Figure 2-3.

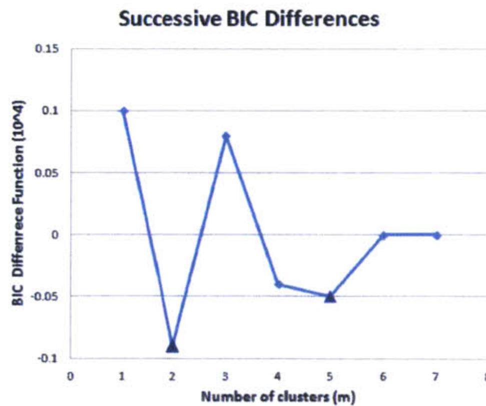

Figure 2-4: Cluster BIC Plot. Look for prominent troughs to choose the number of clusters.

In [24], the authors point out a disadvantage of the successive difference method: it fails to consider the whole curve so it finds local changes without a global perspective. For instance, in the example we provide there are troughs at both $m = 2$ and $m = 5$ but the successive difference method does not tell us which one to pick. To decide on the optimal number, [24] proposes an angle-based method. We summarize it below:

1. Find the local minimas among the successive differences and sort them in decreasing order of their absolute values. Keep pointers to the corresponding number of clusters for each local minimum.

2. Compute the angle associated with each local minimum. If i is the corresponding number of clusters, then we have:

   $Angle = \arctan(\frac{1}{F(i)-F(i-1)}) + \arctan(\frac{1}{F(i+1)-F(i)})$

3. Stop when the first local maxima is found among the angles.

In our example, we obtain 2.226 radians for $m = 2$ and 3.082 radians for $m = 5$, which suggests that the latter value is the optimal number of clusters.

2.5 Logistic Regression

Relevance: Logistic regression is used to describe the structure (statistical clusters) obtained from the K-Means Clustering algorithm by identifying the relevant features for each cluster. We manage to extract an accurate fingerprint for each group of virtual machines in the form of a summarized/compressed representation of their raw metrics.

Logistic Regression ([9]) is a classification technique for identifying the features that describe a labeled set of data points. The method is based on sigmoid function classifier $h_\Theta(X)$ that takes values between 0 and 1:

$$h_\Theta(X) = \frac{1}{1 + e^{-\Theta^T X}}$$

The goal of logistic regression is to assign a label ($y = 1$ or $y = 0$) to each new data point X based on a training set for which the labels are already known. The hypothesis output $h_\Theta(X)$ is interpreted as the estimated probability that $y = 1$ on input X. The rule that assigns labels given the Θ parameters is intuitive:

$$Label(X) = \begin{cases} 1 & h_\Theta(X) \geq 0.5 \\ 0 & h_\Theta(X) < 0 \end{cases}$$

The above classification rule can be simplified to

$$Label(X) = \begin{cases} 1 & \Theta^T X \geq 0 \\ 0 & \Theta^T X < 0 \end{cases}$$

Formally the equation $\Theta^T X = 0$ describes the decision boundary for our hypothesis. The points on one side of the boundary receive the label $y = 1$ while the others get $y = 0$.

The components of the Θ vector are determined by minimizing the cost function $J(\Theta)$ given below, where $h_\Theta(X_i)$ is our estimate corresponding to the $i^t h$ data point and y_i is the true label.

$$J(\Theta) = \frac{1}{k} \sum_{i=1}^{k} Cost(h_\Theta(X_i), y_i)$$

$$Cost(h_\Theta(X), y) = \begin{cases} -\log(h_\Theta(X)) & y = 1 \\ -\log(1 - h_\Theta(X)) & y = 0 \end{cases}$$

Finally, we can analyze the quality of our classification by looking at measures such as precision, recall and F-measure. As a reminder, we provide the formulas for these quantities.

$$precision = \frac{tp}{tp+fp} \qquad recall = \frac{tp}{tp+fn}$$

$$F - measure = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

where tp is the number of true positives, fp is the number of false positives, tn is the number of true negatives, and fn is the number of false negatives.

2.6 Entropy, Relative Entropy and Mutual Information

Relevance: Measures such as the *Kullback Leibler* distance and mutual information are used in our analysis pipeline to estimate the dependence between the studied VM metric distributions. For example, we can employ these techniques to identify the factors that have a nontrivial effect on the distribution of a metric of interest.

An in-depth coverage of entropy, relative entropy and mutual information is given in [3]. This section aims to summarize the definitions and properties that are relevant to our approach.

In information theory, entropy is also known as Shannon entropy and represents a measure of the uncertainty associated with a random variable. If we let X be a random variable over an universe Ω and $p_X(x)$ be its associated probability mass function, then the entropy of X is defined by

$$H(X) = -\sum_{x\in\Omega} p_X(x)\log p_X(x)$$

The concept of entropy is often employed to study the amount of information present in a given message. As a result, the most commonly used unit for entropy is the bit. For example, assume that the random variable X models a fair 6-sided die. By applying the formula above, we get that the die's entropy is equal to log(6) or 2.585. This makes sense since 2 bits can be used to represent at most 4 values, so more are needed to cover all the possible 6 outcomes of a die roll.

The notions of joint and conditional probability can be easily extended to entropy. If we let Y by a second random variable over a universe Γ such that X and Y have a joing distribution $p(X, Y)$, then the joint entropy $H(X, Y)$ and the conditional entropy $H(X|Y)$ are given by

$$H(X,Y) = -\sum_{x\in\Omega}\sum_{y\in\Gamma} p(x,y)\log p(x,y)$$

$$H(X|Y) = -\sum_{x\in\Omega}\sum_{y\in\Gamma} p(x,y)\log p(x|y)$$

The chain rule applies in this case and we have

$$H(X,Y) = H(Y) + H(X|Y)$$

Relative entropy measures the distance between two distributions. It is also known as the *Kullback Leibler* distance and gives the price one has to pay for assuming that the distribution is X' when the true distribution is X. If we let $p(x)$ and $q(x)$ be the probability mass functions of X and X', respectively, then the *Kullback Leibler* distance $D(X,Y)$ is given by

$$D(X,Y) = \sum_{x\in\Omega} p(x)\log\frac{p(x)}{q(x)}$$

Finally, the mutual information of two random variables is related to their mutual dependence. It is defined by

$$I(X,Y) = \sum_{x\in\Omega}\sum_{y\in\Gamma} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

2.7 Silverman's Test for Multimodality

Relevance: We consider a performance metric that has a multimodal distribution over a set of similar VMs to be unusual. We build probability distributions that are conditioned on such metrics in the diagnosis step. Silverman's method offers a way to test for multimodality.

Silverman ([18]) proposes a method based on kernel density estimates for investigating the number of modes in a distribution. For observations $x_1, ..., x_n$ and a window width of h Silverman defines the kernel estimate

$$\hat{f}(t,h) = \frac{1}{nh}\sum_{i=1}^{n} K(h^{-1}(t - x_i))$$

where K is the normal density function. Note that the parameter h corresponds to how much the data is smoothed. In particular, a large value of h would be necessary to obtain a unimodal estimate for a strongly multimodal density.

The author studies the null hypothesis that the distribution has k modes against the alternative that it has more than k modes. For this, he introduces the k-critical window width h^k_{crit} to be

$$h^k_{crit} = \inf(h; \hat{f}(., h) \text{ has at most k modes})$$

Simulation is employed to assess the significance h^k_{crit} for a given k. In each round, the new observations y_i, $1 \leq i \leq n$ are generated based on the formula

$$y_i = (1 + (h^k_{crit})^2/\sigma^2)^{-\frac{1}{2}}(x_{I(i)} + h^k_{crit}\epsilon_i)$$

where σ is the standard deviation of the data, ϵ_i is an independent sequence of standard normal random variables, and the $x_{I(i)}$ are chosen uniformly at random with replacement from the original data set. The p-value is equal to the percentage of time we get a distribution y_i with more than k modes during the simulations. The null hypothesis is rejected for p-values that are under a significance level α (0.1 in our tests).

2.8 Bayesian Inference

Relevance: We aim to discover the factors that have an impact on the distribution of key performance indicators using this technique.

Bayesian Inference ([1]) is a method of inference built on top of Bayes Rule and employed to compute the posterior probability of a hypothesis given its prior probability and a likelihood function for the observed data.

In our work, we take advantage of inference techniques in order to:

(a) Assess whether a given factor has an impact on a performance indicator

(b) Determine the factors that generate discrepancies in a cluster of VMs that are assumed to be "similar"

To outline the procedure for use case (a), consider a performance indicator I. Using mutual information we can identify potential candidate factors that have an impact on I. Given such a factor f, our original belief in the absence of any existing evidence is usually that f does not affect the distribution of I. Therefore, we start with the assumption that $P(I|f) \simeq P(I)$. Access to information about the distribution of I for several observed values, f_i, $1 \leq i \leq n$, can subsequently change our beliefs. For instance, we may notice that $P(I)$ differs considerably from $P(I|f > E[f])$, where $E[f]$, denoted the observed expected value for f.

For the second use case, we generally look at metrics that have an unexpected behavior over a set of VMs considered similar from a workload perspective. As an example, assume that you have a resource pool containing two types of VMs, X and Y. When you apply K-Means and PCA on the collected data, you expect the VMs to group into two clusters according to their type. However, you notice that the type X VMs do not closely group together, but rather split into 2 subclusters x_1 and x_2. In this case, you would want to identify the metrics you considered in the analysis that caused the spread. We call such metrics "spread metrics", since we expect that eliminating them from our analysis would bring the VMs closer together.

One option is to consider the multimodal metrics as possible candidates and select them with Silverman's test. The work done in Carat ([13]) inspired us to design a much faster and more accurate method. We conjectured that a metric causing spread would have significantly different expected values over the two sublusters. Formally, if we let m_{max} be the maximum median value for metric m over the VMs, then we label m as a "spread metric" if and only if

$$\frac{|E[m|x_1] - E[m|x_2]|}{m_{max}} > \theta$$

where θ is a threshold that characterizes how close together we want to bring the type X VMs. The smaller the θ, the more metrics we identify and can subsequently eliminate. This implies that as we decrease θ we bring the type X VMs closer together.

However, it should be noted that there exists the risk of eliminating the features that set apart the type X and type Y VMs if θ is too small.

Chapter 3

Data Primer

To identify the key features for a cluster of virtual machines we look at the associated data. The aim of this chapter is to provide an overview of the data format and to describe the different data interactions that occur. We start with a summary of the VMware virtual data center architecture. Next, we have a discussion of the data collection step and the two data types we consider: performance and metadata metrics. We then cover the data storage procedure and we argue that it provides fast access to the pieces of information that are needed in our analysis. The fourth section goes over the in-memory data representation adopted in our approach and explains how we deal with challenges such as big data and redundancies. Finally, we present the data sources that served as a base for our results.

3.1 Virtual Data Center Architecture

A detailed overview of VMware Infrastructure, "the industry's first infrastructure virtualization suite", is given in [20]. In this section we only focus on the parts that are necessary for understanding data collection and the analysis pipelines presented in the next chapter. VMware Infrastructure can take an entire IT infrastructure that includes heterogeneous resources such as servers, storage, and networks, and convert it into a pool of uniform virtual elements that can be easily managed - the virtual data center. The main building blocks of the virtual data center, depicted in Figure

3-1, are :

1. Hosts, Clusters, and Resource Pools (logical groupings of computing and memory resources)
2. Datastores (storage resources)
3. Networks (networking resources)
4. Virtual Machines

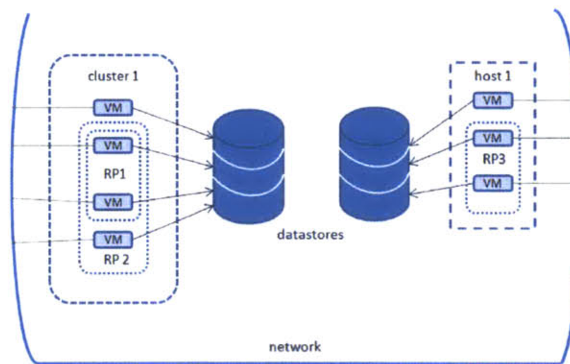

Figure 3-1: Virtual Data Center Architecture. The virtual building blocks.

The experiments we conducted were mainly concerned with virtual machines and their associated computing and memory resources.

A *host* incorporates the CPU and memory resources of a physical server. Therefore, its total memory and computing power can be calculated by adding up the resources present on the physical server. For example, 2 quad-core CPUs at 2GHz each would amount to 16 GHz for the host.

A *cluster* is defined as a set of hosts that are managed together. It follows that the resources corresponding to a given cluster are simply the sum of the host resources. As an example, if a cluster contains 10 hosts each with 8 GB of memory, then it has 80 GB of available memory.

The resources of a cluster or a host often need to be further divided and distributed according to the needs of the entities that requested them. The resulting resource

partitions are known as *resource pools.* Each resource pools can be further partitioned into smaller ones, thus allowing for resource allocation at a very fine granularity.

The *virtual machines* are the resource consumers. Each virtual machine is assigned to a host, cluster, and resource pool. The resources consumed are usually dependent on the workload executing on the virtual machine.

Finally, the VMware Virtual Center provides a wrapper around the around the existing resources and the resource consumers. It is a central point of control for managing, monitoring, provisioning, and migrating virtual machines.

3.2 Data Collection

Our data collection takes place at the virtual machine level. More exactly, we treat each VM as a black box and only collect its aggregate associated data without getting any information about the applications that are running.

The Virtual Center can collect data from the hosts at different interval lengths and with various interval periods. In this work we refer to the interval length as the epoch, and to the interval period as the sampling rate. Examples of collection periods/lengths are shown in Table 3.1.

Interval Length	**Sample Frequency**
1 hour	20 seconds
1 day	5 minutes
1 week	30 minutes
1 month	2 hours
1 year	1 day

Table 3.1: The interval lengths/periods at which the Virtual Center collects data metrics

The epoch length for the data we collect is 1 hour and the sampling rate is 20 seconds. Moreover, we have two categories for the received data: metadata and performance metrics. The *metadata* captures the general configuration of the VM and its associated host and resource pool. Examples of metadata quantities are the total number of CPUs for the VM, the VM friendly name, the VM host name, the VM

resource pool name, the host computing and memory resources, the host vendor, the number of VMs on the host, the resource pool computing and memory resources, etc. The *performance metrics* capture the effect of the workload on the virtual machine. Each of these performance counters is represented by a combination of its name, group, and roll-up type, in the form $< group > . < name > . < roll - up >$. For example, *cpu.usage.average* denotes the average CPU usage in the sample period.

Figure 3-2: Performance metrics. The different possible group types.

The possible group types are the ones shown in Figure 3-2. The roll-up types are summarized in Table 3.2.

Rollup Option	**Description**
none	instantaneous value
average	average over the sampling period
maximum	maximum value in the sampling period
minimum	minimum value in the sampling period
latest	last value in the sampling period
summation	sum of the values over the sampling period

Table 3.2: The performance metrics roll-up options with their description

Each performance counter also has an instance associated with it. For example, the cpu.usage.average counter is reported for each virtual cpu core on a VM in addition to the aggregate default measure. The combination of metric name and instance name completely identifies a counter. In our example, (cpu.usage.average, 1) would be the average CPU usage for virtual cpu 1. More information about the performance metrics is provided in [5].

We end this section with a sketch of the algorithm we use for data collection. It runs through an infinite loop that can be terminated once enough data has been

collected.

1. Connect to the Virtual Center (VC).

2. Set the epoch length (1 hour) and the sampling period (20 seconds).

3. Get the list of powered-on virtual machines.

4. Collect the metadata for each VM.

5. Collect the available performance counters for each VM at the specified sampling rate for the last hour.

6. Wait until the end of the epoch and go back to step 3.

3.3 Data Storage

The data we collect is stored directly on disk. We want to explore the possibility of using a database in the future, but at this time our current method is fast enough to satisfy our information retrieval needs. In particular, we have a few queries for which we require fast execution:

- Type 1 query: given an epoch, get all the virtual machines that are powered on.

- Type 2 query: given an epoch and a VM, get the associated metadata and performance counters.

- Type 3 query: given an epoch, a VM, and a performance counter, get the counter values at the target sampling rate.

We use 4 different files to store the data. Their format, described next, is the key to fast execution for the above mentioned queries.

metadata file

All the metadata gathered in step 4 of our collection algorithm is saved in this

file. Each entity we consider is described in the file by 2 lines. The first line mentions the entity type (virtual machine, host, or resource pool) and the name. The second line enumerates the corresponding metadata quantities. Here is an example of a resource pool description:

vim.ResourcePool:resgroup-9122

rp.cpuAllocMhz,17705; rp.memAllocMB,93307; rp.numVMs,12

vm_offsets file

This file usually has the smallest size among the 4 we consider and can be regarded as an index for fast access. For each epoch and each VM it keeps track of a series of byte offsets that are useful for fast access to certain pieces of information. In particular, each line in the file has the following format:

VM name, offset 1, offset 2, offset 3, offset 4, offset 5, epoch number

Each offset is the address of some specific VM data in of the the 3 other files, as follows:

offset1

VM byte offset in the stats file

offset2

VM byte offset in the metric_offsets file

offset3

VM byte offset in the metadata file

offset4

VM's host byte offset in the metadata file

offset5

VM's resource pool byte offset in the metadata file

metric_offsets file

Contains the byte offsets with respect to the stats file for the performance counters corresponding to each (VM, epoch) pair. The format is given below:

VM name

metric 1, metric 1 offset; metric 2, metric 2 offset; ...

stats file

This is the largest of the 4 files and contains the values for each performance counter. Since our epoch length is 1 hour and our sampling rate is 20 seconds, we have 180 samples for each 3-tuple (VM, metric, epoch). As a result, the data for each VM is represented in the following format:

VM name

metric 1 name

value 1, value 2, ..., value 180

....

metric k name

value 1, value 2, ..., value 180

The structure of the 4 files we employ for storage allows for fast execution of the queries we mentioned in the beginning of this section:

- Type 1 query: Use the *vm_offsets* file to get the virtual machines that have the specified epoch number

- Type 2 query: Find the VM in the *vm_offsets* file based on the VM name and the epoch number. Use offsets 3, 4, and 5 to get the metadata. Use offset 2 to get the performance counters.

- Type 3 query: Find the VM in the *vm_offsets* file based on the VM name and the epoch number. Use offset 2 to get the performance metrics and find the byte offset of the given one. Use the byte offset and the *stats* file to get the values.

3.4 In-memory Data Representation

The data we collect can be easily parsed using the storage files and the queries introduced in section 3.3. As we load the data into memory, we need to be careful about maintaining the fast access property. It then makes sense to have a data container for each of the entities we consider (virtual machine, host, resource pool). The architecture behind the virtual machine data container is rather straightforward. We present it below and skip the host and the resource pool containers since they are similar.

VM Data Container

name : the name of the VM
perf_data : dictionary mapping each performance metric to its values
meta_data : dictionary mapping each metadata quantity to its value

With the above in-memory data representation memory becomes a bottleneck for large clusters of virtual machines. For example, consider the case of a cluster with 1000 VMs and 300 collected performance counters over 10 epochs. If we estimate that each performance cluster name takes roughly 20 bytes and each counter value can be saved in 8 bytes, then the total memory required only for the *perf_data* dictionaries is just above 4 GB. This is not reasonable, especially since more memory gets allocated for other data structures in our analysis pipeline. There are several ways in which we can cut down the memory requirements:

- *Only load the data for one epoch at a time.* Our approach works on a per epoch basis, so loading all the data at once is not necessary.

- *Do not consider the performance metrics that have* 0 *variance.* In quite a few cases, the 180 values are all the same (usually 0). Disregarding the corresponding performance counters does not have an impact on our analysis.

- *Use a percentile value for each metric instead of the entire set of values.* Making the perf_data dictionary map each counter to its median value over the epoch, for instance, is a good approximation that does not alter significantly our results. We discuss this further in chapter 5.

- *Only focus on a subset of the collected performance metrics for the analysis.* The performance counters are not necessarily the same for each VM in a given epoch. However, in order to run through our main pipeline and group together "similar" virtual machines, we need to decide on a common set of metrics. One option is to take the intersection of the non-constant metrics over the set of VMs. This can reduce considerably the number of performance counters and still allow us to make accurate predictions in many cases. Nevertheless, in some scenarios, the intersection becomes too small and other methods need to be employed in order to decide on a common set of metrics. The option we employ in most of our experiments is the union of all non-constant metrics over the set of VMs.

Based on the observations made in this section, we decided to model the direct data input to our pipeline as a matrix of values, in which the rows represent the virtual machines and the columns correspond to the metrics we consider. As a result, at the intersection of the ith row and the $j^t h$ column we place the chosen percentile value of the jth metric over the current epoch for the ith virtual machine. We call this structure the *data matrix.*

3.5 Data Filtering

Reductions in the size of the *data matrix* introduced in section 3.4 are possible, as it turns out that in many cases some of the performance metrics are irrelevant or redundant. We define several levels of data filtering here, and we analyze their impact on our analysis in chapter 5.

Level 0 of filtering corresponds to the initial set of metrics that were chosen using the intersection or the union rule over the pool of virtual machines. The next levels are based on the following observations:

- **Level 1**: The system metrics (e.g. system uptime) can be discarded since they usually take the same values for a group of similar VMs in the cluster.

- **Level 2**: The *min*, *max*, *none*, *summation* and *average* rollups are strongly correlated so considering all 4 of them is redundant. We noticed that whenever the *min* and *max* rollups are available, the *none* and *average* are also present. As a result, we can filter out the performance counters with a *min* or *max* roll-up.

- **Level 3**: We take out the *summation* and *none* instances of a metric if and only if the *average* instance is present. This rule takes into account that some metrics only have the summation roll-up.

- **Level 4**: If the default metric instance exists, we take out all the other instances.

Note that each level of filtering assumes that the previous ones have been already applied. In chapter 5, dedicated to case studies and results, we show that even the highest levels of filtering (3 and 4) do not have a considerable impact on our approach. Therefore, metric filtering speeds up our algorithm and reduces the memory requirements at almost no cost with respect to the results.

3.6 Data Sources

The fingerprinting method is applied on 3 separate clusters, each having different properties:

- *ACluster*: A small controlled environment executing a VMware internal research workload

- *ViewPlanner*: A medium size controlled environment that executes a series of benchmarks.

- *Nimbus*: A large size heterogeneous environment dedicated to internal VMware testing.

We choose the 3 clusters to be very different in their properties (scale and diversity) on purpose, in order to prove that our approach works properly in a variety of scenarios.

Chapter 4

Pipeline Overview

The research done for this thesis focuses on discovering VM relationships and describing them in a concise way. In this section, we elaborate on the procedure that takes the raw captured data consisting of resource usage and metric information and transforms it into a meaningful and compact fingerprint that identifies a VM or a group of VMs.

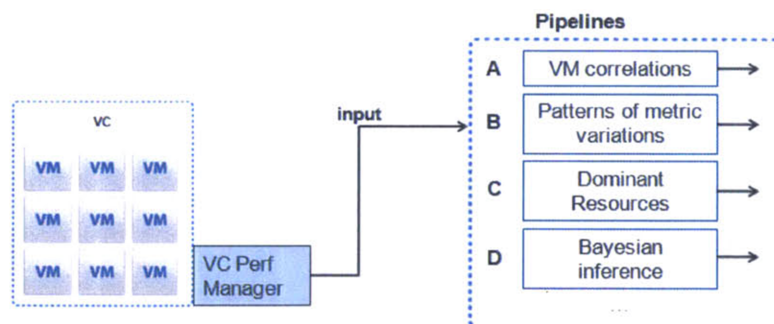

Figure 4-1: Approach overview. The conceptual pipeline.

At a conceptual level, our intended approach is summarized in Figure 4-1. To study a cluster of VMs, we first make a query to the VC and its associated Performance Manager. As a result of our request, we get the metric data and the metadata for each VM. The data we collect reflects the VM's state in the past hour and is sampled at a granularity of 20 seconds. In an intermediate step, the data is converted into an adequate format, as described in the previous chapter. It can then serve as

input to one of our 4 analysis pipelines, each of which we present next. We want to emphasize that the ways in which the collected data can be used are not limited to our approaches. The problem at hand is basically open-ended, and many more ways of interpreting the data may be possible. We hope that as this work progresses, we will both enhance the functionality of our pipelines and discover new ones.

4.1 Correlation Pipeline

Our first attempt involves using VM correlations for a rough approximation of VM relationships. Based on these results, we want to assess whether it is worth applying more advance statistical clustering techniques. This pipeline is fairly simple, and its main steps are outlined in Figure 4-2. Starting with the raw metric data, we compute pairwise VM correlations and then cluster the strongly correlated VMs together.

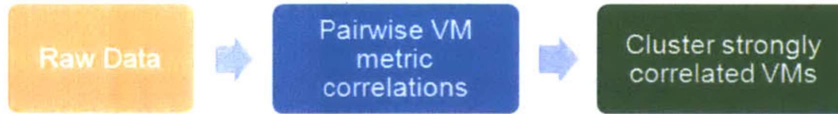

Figure 4-2: Correlation Pipeline. The focus is on VM metric correlations.

As a first step, we compute the metric correlations between any pair of virtual machines that we wish to analyze. For instance, given VM_1 and VM_2, we calculate the correlations coefficients between every metric m_i corresponding to the first VM and every metric m_j of the latter. The resulting coefficients can be arranged in matrix form and visualized using a heatmap, as shown in Figure 4-3. We use the color red for strong positive correlations, blue for strong negative correlations, and white for metrics that are uncorrelated.

We mentioned initially that our goal is to cluster strongly correlated VMs together. Considering each metric pair combination for each possible pair of virtual machines has a prohibitively expensive computational cost. As a result, we decided to only focus on the corresponding metrics, i.e. the ones on the main diagonal in Figure 4-3.

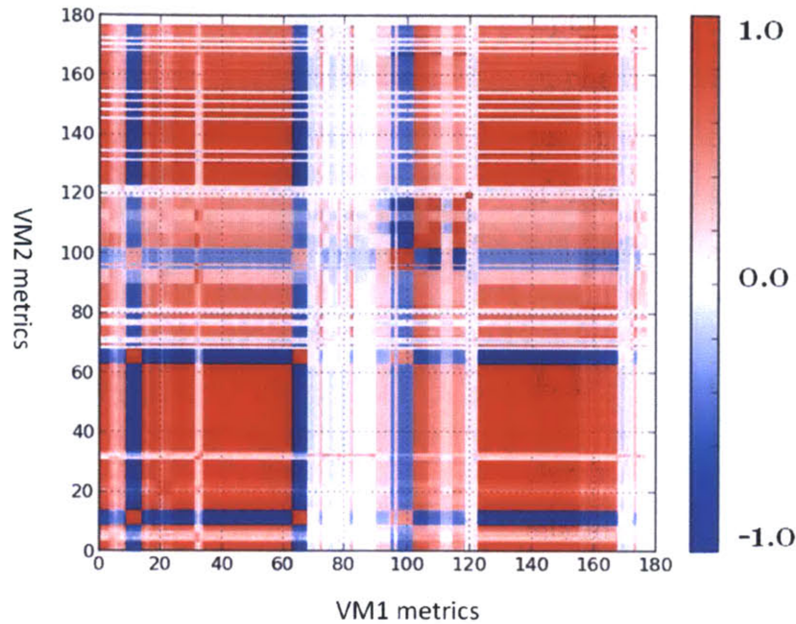

Figure 4-3: Pairwise Pearson Correlation Heatmap.

The clustering algorithm based on correlations works as follows:

- Given a pair of VMs, count for how many corresponding metrics the correlation coefficient is above a given threshold θ.
- Normalize using the highest count
- Cluster based on the normalized count
  - Count close to 1: high degree of correlation
  - Count close to 0: low degree of correlation

In our experiments we use $\theta = 0.4$. The result is a cluster correlation heatmap similar to the sample one shown in Figure 4-4. In this case, we notice that the virtual machines #4 - #8 appear to be clustered together.

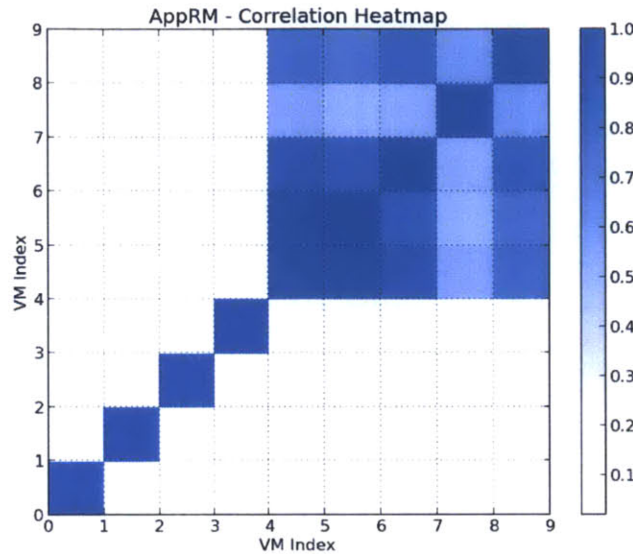

Figure 4-4: Sample cluster correlation heatmap. One cluster is visible for VMs 4-8.

4.2 Clustering Pipeline

This is our most elaborate pipeline which uses many of the techniques introduced in chapter 2 to produce a fingerprint. The process is summarized in Figure 4-5. Using the raw metric data we construct a feature vector for each VM. In particular, each entry in the vector equals a given metric percentile value for the associated VM. For example, the first entry may reflect the median CPU utilization.

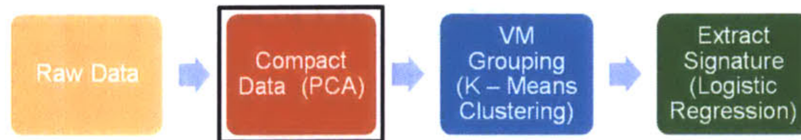

Figure 4-5: Clustering Pipeline. The focus is on VM metric variations.

Optionally, we can use PCA to eliminate the redundancies in our metric sets and get a compact data representation. This step would have the effect of reducing the dimension of the feature vectors. For instance, if we just select the two most important principal components we get a 2-dimensional projection of our original vector space. The clear advantage in this case is that we can visualize the resulting projection. However, we need to be careful and check whether PCA has a significant

impact on the accuracy of our results.

The feature vectors are passed as input to a K-Means clustering algorithm that groups 'similar' VMs together. We then do One-VS-All Logistic Regression on each group to identify the most relevant metrics that describe it. As suggested by Figure 4-6, the resulting fingerprint is a weighted expression of common metrics of importance.

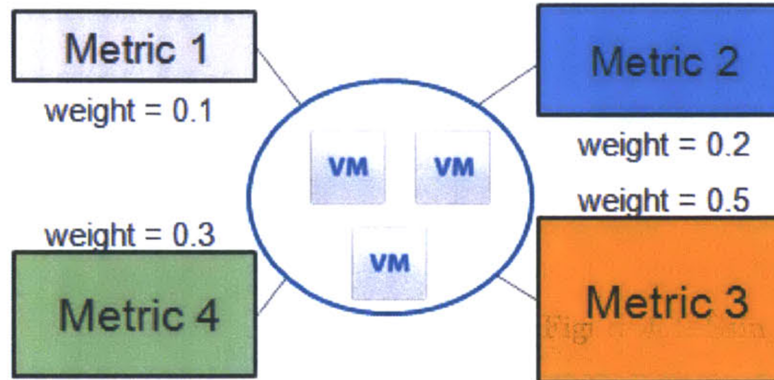

Figure 4-6: Logistic Regression weights. Fingerprint example for the Clustering Pipeline.

For example, getting a weight of 0.5 for Metric 3 and a weight of 0.1 for Metric 1 tells us that Metric 3 does a better job at describing the group of VMs then Metric 1.

4.3 Dominant Resources Pipeline

Our third pipeline is concerned with dominant resources. Here we only consider two metrics: CPU and Memory. Given a time interval, we compute for each VM the percentage of CPU/Memory usage with respect to its host's capacity. The resource for which the VM requests the highest fraction of host resources is its dominant resource. An alternative we explore is computing the percentage of CPU/Memory usage with respect to the entire cluster's capacity. One example of a resulting fingerprint is given in Figure 4-7.

The VM shown in Figure 4-7 has Memory as its dominant resource for the first 2 hours, and then switches to preferring CPU for the third hour, etc. Fingerprints

Figure 4-7: Dominant Resources. Fingerprint example for Pipeline C.

as the one depicted in the figure can be used for load balancing and placement hints. For example, we can place some VMs that prefer memory and some that prefer CPU on the same host, thus ensuring that neither resource is overloaded.

4.4 Bayesian Inference Pipeline

The last pipeline aims to provide explanations for the performance differences we observe in a cluster of 'similar' VMs or even across 'similar' deployments.

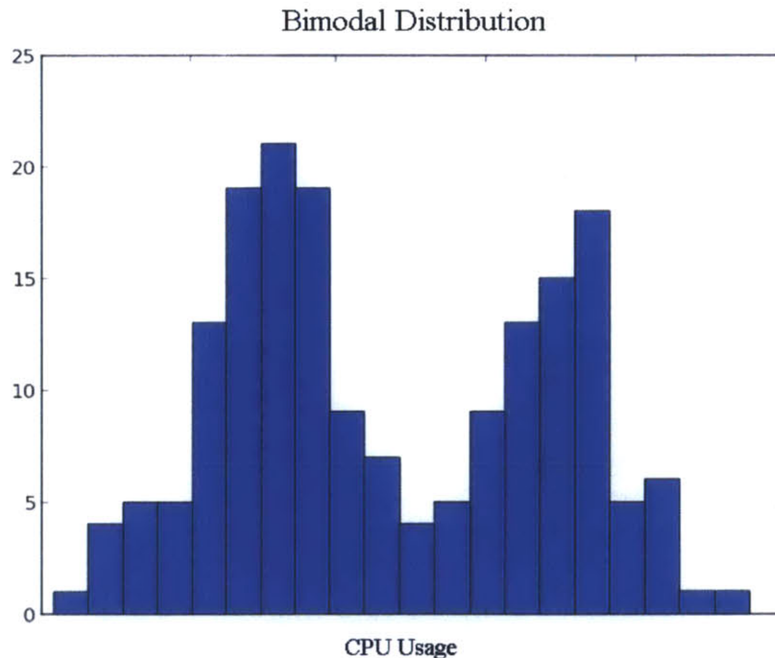

Figure 4-8: Bimodal Distribution. CPU usage distribution.

To better illustrate our intended approach, we will take an example. Assume that the cluster CPU usage has the bimodal distribution shown in Figure 4-8. In this case, we will want to better understand why a fraction of the VMs show a lower CPU

usage, while the rest have an increased utilization.

Remember that we are analyzing a group of 'similar' VMs so a bimodal distribution is not normally expected. Therefore, it becomes our aim to identify the factors that have an impact on the CPU utilization. For this example, assume that you have prior knowledge about the location of the VMs. In particular, you know that some of the VMs have 10 neighbors and are placed on heavily loaded hosts, while others have only 3 neighbors and are placed on lightly loaded hosts.

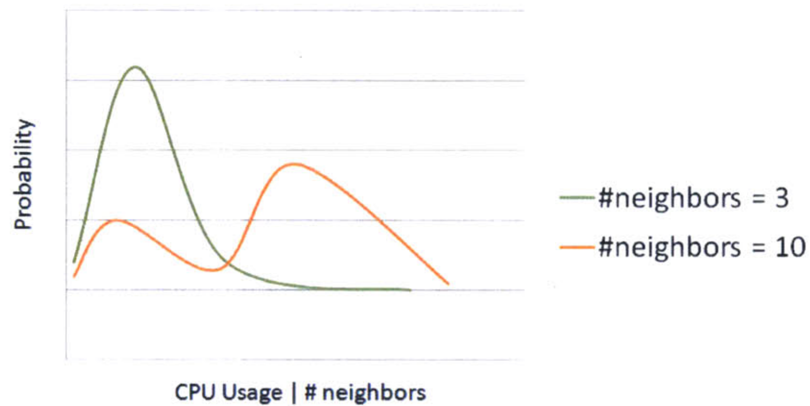

Figure 4-9: CPU Conditional Distribution. Usage given the number of neighbors of a VM.

By plotting the conditional distribution of CPU with respect to the number of neighbors (Figure 4-9) we see that the consolidation ratio may be indeed one of the latent factors behind the observed bimodal distribution.

Chapter 5

Results

This chapter presents the experiments we conducted on research and development clusters inside VMware. In our analysis, we are seeking to answer the following questions:

- Can we automatically group virtual machines based on their behavior in a robust way?
- Can we accurately extract markers or fingerprints for each group of virtual machines?
- How can we take advantage of our clustering analysis and the resulting fingerprints for diagnosis?

We describe our experience working with 3 clusters of virtual machines as we attempt to answer the above questions. Each setting we consider is chosen to have its own unique characteristics in order to prove that our methods give results in a variety of scenarios.

5.1 ACluster

ACluster is a small controlled environment of only 14 virtual machines and executes a VMware internal research workload. We began our analysis on ACluster due to its simplicity and small scale. The featured virtual machines are:

- 2 MongoDB VMs
- 2 rain VMs
- 4 shard VMs
- 1 testserver
- 5 workload VMs

In this section, we apply our correlation pipeline on ACluster. The key points we want to emphasize are the following:

- The workload each VM executes gets reflected in the performance metric values we collect. VMs executing similar workloads exhibit a higher degree of correlation than unrelated VMs.
- By analyzing the strength of correlation between each pair of VMs we can infer with decent accuracy the structure of the cluster.
- The correlation pipeline has several drawbacks (not scalable, not very accurate) but proves that there exists structure that can be exploited. As a result, it motivates more advanced approaches, such as our clustering pipeline.

As already mentioned, our first attempt was to to apply the correlation pipeline on ACluster. We suspected that virtual machines that are similar would show a higher degree of correlation and we were right. Figure 5-1 (a) shows the correlation heatmap between the two MongoDB virtual machines. In part (b) we have the heatmap between a MongoDB VM and the testserver. As explained in chapter 4, the color red corresponds to strong positive correlation, the color blue to strong negative correlation, and the color white to no correlation. It is then immediate to observe that the two MongoDB VMS are strongly positively correlated in almost all their metrics while the chosen MongoDB and the testserver are in general uncorrelated or very slightly correlated.

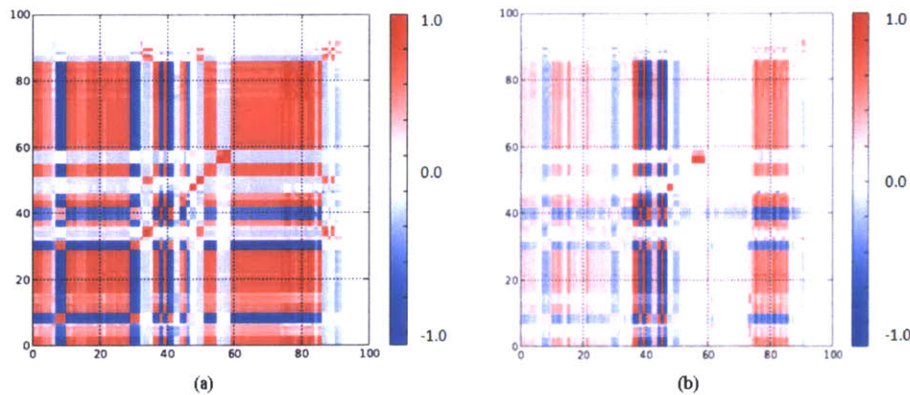

Figure 5-1: Pairwise Pearson Correlation Heatmaps. (a) two MongoDB VMs (b) a MongoDB VM and a testserver.

Using the algorithm described in subsection 4.1 we constructed a cluster correlation heatmap for ACluster, as shown in Figure 5-2. We can identify roughly 3 groups of virtual machines in the heatmap: indexes 0-7, index 8, and indexes 9-13. In addition to this, smaller subgroups are visible for indexes 0-7. It turns out that the MongoDB VMs correspond to indexes 0-1, the rain VMs to indexes 2-3, the shard VMs to indexes 4-7, the testserver to index 8, and the workloads to indexes 9-13.

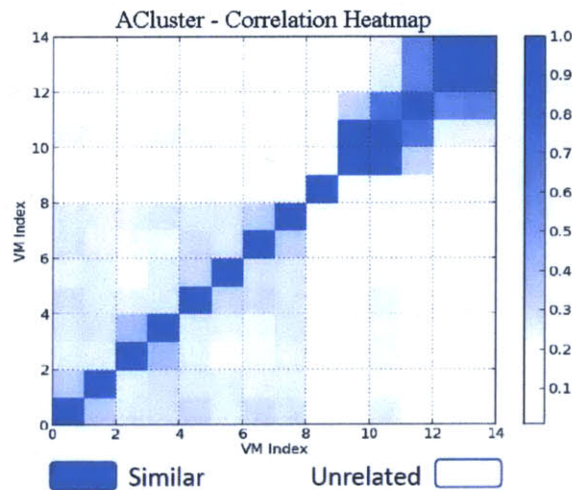

Figure 5-2: Cluster correlation heatmap for ACluster. Multiple groups are visible.

Our first cut at clustering based on correlations proves to be efficient when applied on ACluster. However, as we shall see in the next case study, this approach does not scale well with the size of the data. This is the main reason that motivated our clustering pipeline, a method that employs more sophisticated tools such as the K-

Means algorithm and Principal Component Analysis. The second pipeline is discussed in detail for View Planner, our second case study. We only give a preview here by showing the accurate grouping it produces for Acluster in Figure 5-3.

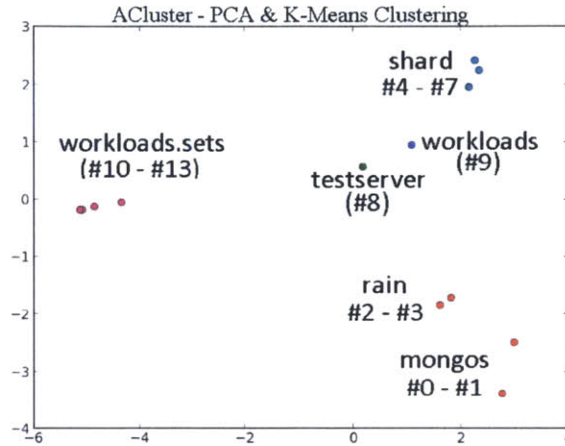

Figure 5-3: ACluster PCA K-Means. Cluster structure.

5.2 View Planner

Our second round of experiments are based on VMware View Planner, a tool used to simulate large-scale deployments of virtualized desktops. In its VMware User Guide ([22]), View Planner is presented as a powerful and flexible application that can account for a variety of testing and usage scenarios. We summarize some possible View Planner configurations below:

1. Three different run modes are possible:

    - Remote mode: each desktop virtual machine is paired with a client virtual machine
    - Passive mode: multiple desktop virtual machines are paired to each client virtual machine. This mode requires fewer hardware resources than the remote mode.
    - Local mode: no client virtual machines are used. This mode minimizes the hardware requirements.

2. Multiple display protocols are supported:
   - Microsoft RDP
   - View RDP
   - View PCoIP
3. Different applications can be selected to run in the workload:
   - Microsoft Word
   - Microsoft Excel
   - Microsoft PowerPoint
   - Microsoft Outlook
   - Microsoft Internet Explored
   - Mozilla Firefox
   - Adobe Acrobat Reader
   - Archiving Software
   - Video Playback Software

5.2.1 View Planner Experiment Overview

In our View Planner experiments we employ the remote mode and a PCoIP display protocol. The virtual machines in the clusters are divided into 3 categories based on their type:

- desktop virtual machines placed on one or more ESX hosts
- client virtual machines placed on one or more ESX hosts
- a single Virtual Desktop Infrastructure (VDI) controller appliance and several other infrastructure related virtual machines

A sketch of our cluster infrastructure presenting the relationships that exist between the different modules is given in Figure 5-4. In remote mode, the harness starts the same number of client and desktop virtual machines. In the configuration stage, each client VM is assigned to monitor one of the desktop VMs. Once the test begins, each client VM submits a series of predefined tasks to its corresponding desktop VM.

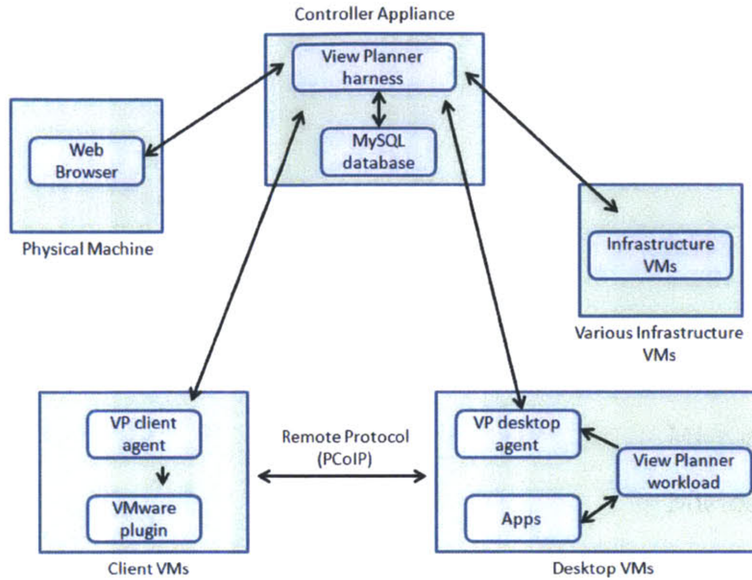

Figure 5-4: VMware View Planner Architecture. Operation overview in remote mode.

As the tasks execute, the client VMs measure the performance of their assigned desktop VMs. Once the results are ready, they are submitted back to the harness and can be visualized through a web browser.

For our experiment, we power on 84 desktop VMs, 84 client VMs, and 7 infrastructure VMs. The ViewPlanner run lasts for about 5 hours and generates a modest amount of data (~ 360 MB) in a concise format. In our results we show that:

- We can accurately identify the groups of VMs. Moreover, the groupings are stable over time and robust to various parameter choices.

- We can successfully compress the raw metric feature vectors using Principal Component Analysis and still maintain accurate and stable VM groupings.

- We can generate a fingerprint containing the subset of metrics that best describes each group.

- We can employ techniques from signal processing such as entropy-based measures to filter and select metrics useful for explaining/diagnosing differences *within* or *between* groups or VMs. Finally, we show how to use conditional probability distributions for diagnosis.

5.2.2 Correlation Analysis

As a starting point in our correlation analysis, we look at virtual machines in isolation and try to identify statistical relationships between the metrics of any two members of the cluster. Following the approach described in section 4.1, we begin by generating a few Pearson correlation heat maps for randomly chosen pairs of virtual machines.

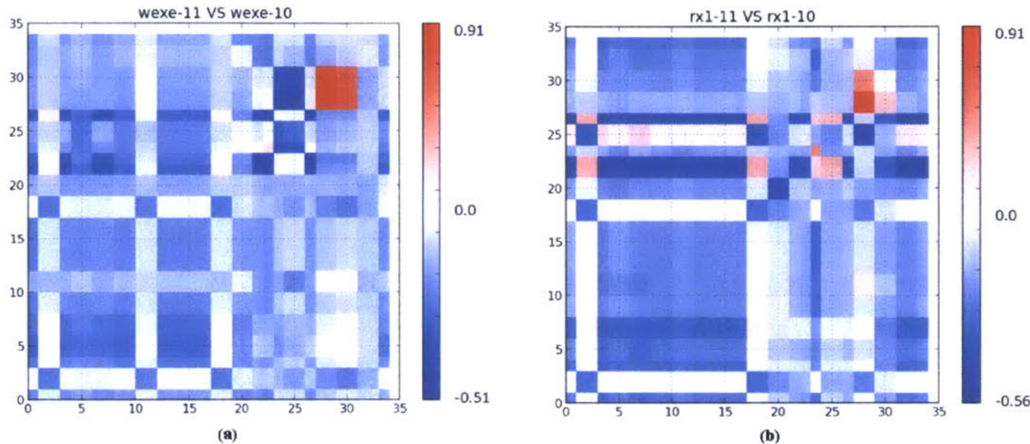

Figure 5-5: Pairwise Pearson Correlation Heatmaps: (a) two desktops (b) two clients

Examples of correlation heat maps are shown in Figure 5-5. On the left we have 2 desktop VMs, while on the right we have 2 clients. The most noticeable feature that happens to be present in both graphs is the red cluster in the top right corner. It corresponds to a group of metrics that are strongly correlated for both the clients and the desktops. Further inspection reveals that the metrics exhibiting this behavior are network related: net.broadcastRx.summation (number of broadcast packets received during the sampling interval) and net.packetsRx.summation (number of packets received during the interval). In particular, the exact correlation coefficients for the desktop VMs are given in Table 5.2. There is a clear drop of almost 0.5 between the net related correlation coefficients and the ones for the remaining metric pairs. We also point out that the are no strongly negatively correlated (close to -1) metric pairs for either the desktops or the clients.

The strong correlation we observe in the net related metrics is not unexpected. Given that in View Planner there is a lot of interaction over the network between

wexe-11 metric	wexe-10 metric	correlation
net.broadcastRx.summation	net.broadcastRx.summation	0.91
net.packetsRx.summation	net.broadcastRx.summation	0.86
net.broadcastRx.summation	net.packetsRx.summation	0.78
net.packetsRx.summation	net.packetsRx.summation	0.74
mem.usage.average	cpu.wait.summation	0.29

Table 5.1: The top strongly correlated metrics for two desktop VMs.

each client - desktop pair, we expect to see similar network patterns when we look at the clients or the desktops. However, since each client - desktop pair goes through a similar predefined process, we were anticipating prior to this experiment more metric pairs that are strongly correlated. In addition to this, we were hoping to see some correlation patterns that would set apart the desktops and the clients. Our investigation revealed that this is not the case.

We suspected that there are two possible reasons for which the correlation based approach presented above fails to distinguish between the clients and the desktops:

1. The information given by the Pearson correlation coefficient is not sufficient to capture the dependence structure which we believe exists in our data. Other correlation approaches may provide better results.

2. The approach itself is not adequate for the problem at hand. For example, there may be no robust or consistent notion of similarity at the individual sample level and we may have to choose a coarser granularity.

The two possibilities are not mutually exclusive. To assess how much each contributes to the observed results, we take turns and inspect each hypotheses.

The point made in (1) is valid, since it is known that the Pearson correlation coefficient can be employed to define dependence only in some particular cases (e.g. multivariate normal distribution). For instance, the Pearson correlation coefficient can take the value 0 even for dependent random variables. Stronger notions of correlation such as the Distance or the Brownian correlation were introduced to address these shortcomings. We decided to repeat the tests using the Distance correlation.

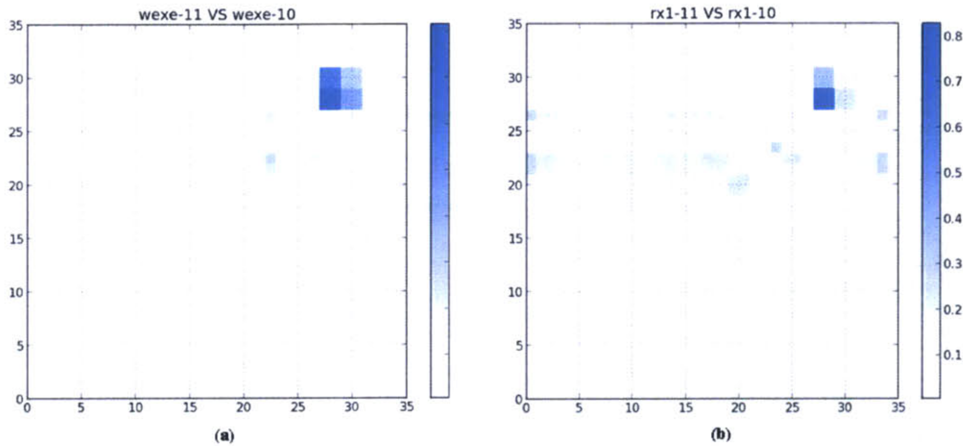

Figure 5-6: Pairwise Distance Correlation Heatmaps: (a) two desktops (b) two clients

The new results for the same set of virtual machines are shown in Figure 5-6. Our previous findings are confirmed and only the net related metrics show a noticeable correlation.

We could explore other measures, such as mutual information or total entropy, that are usually able to uncover more intricate dependencies. We choose not to do so at this point, and rather try to redefine the notion of similarity as suggested by (2). Instead of comparing each pair of virtual machines and then trying to infer the structure of the cluster, we will instead find a succinct representation for the data associated with each VM and cluster based on that.

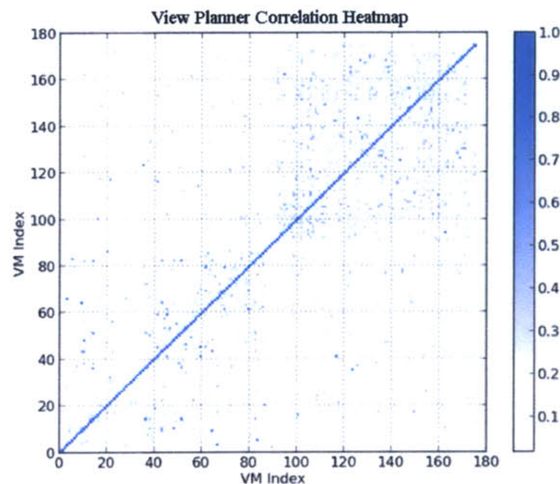

Figure 5-7: Cluster correlation heatmap for View Planner. Two slightly visible groupings visible in the bottom left and the top right corners.

Before transitioning to our next approach, it is worth including the overall cluster heatmap constructed using the algorithm outlined in section 4.1. At a first glance, two clusters are slightly visible indicating that we have indeed separated the clients and the desktops. However, the observed structure is unfortunately dependent on the order of the virtual machines. For instance, in this case they happened to be sorted by their friendly name, so all the client VMs were placed on the X and Y axes before all the desktop VMs.

There are algorithms able to reconstruct the separation shown in Figure 5-7 for randomly ordered VMs, but including them into the pipeline would add even more to the running time that is already a bottleneck: $\Theta((m \cdot n)^2)$ where n is the number of virtual machines and m is the number of metrics we consider for each. As a case in point, it took over 3 minutes to generate the cluster heatmap.

We conclude this section with a brief evaluation of the correlation pipeline. This method represented our first attempt at clustering and was chosen mainly due to its simplicity. Before trying more sophisticated tools on our data, we wanted to confirm first that there exists some underlying structure. The ACluster and View Planner experiments confirmed the existence of more pronounced dependencies between the virtual machines that we expected to be similar. For example, according to Figure 5-7, a desktop VM is highly correlated in more metrics with another dekstop VM than with a client. However, we also discovered that there are several drawbacks associated with the correlation pipeline in its current form: the Pearson correlation coefficient is not powerful enough to capture relationships at the VM level, two metric distributions may actually be similar (have almost identical mean, standard deviation, percentile values) without being strongly correlated, the approach running time exceeds what we would deem acceptable. These findings are what motivated our next pipeline that does a much better job on medium and large size data sets.

5.2.3 Clustering Analysis

In the context of our View Planner experiment, the clustering pipeline aims to produce a grouping that separates the clients and the desktops. Once this is done, we want to

generate fingerprints that capture the main characteristics of each of the two types of virtual machines.

We collected data over a period of 5 epochs (hours), out of which we omit the first one (epoch 0) since it corresponds to the the benchmark ramp-up process. The clustering plots for the remaining 4 epochs are shown in 5-8. Two main resulting clusters are observed. In each case, we identified post analysis that the green o-shaped cluster contains all the clients and the blue x-shaped one all the desktops, as desired. We keep this coloring convention in most of the subsequent plots. The infrastructure VMs are split between the two clusters and can be easily identified as the 7 visible outliers in each of the plots.

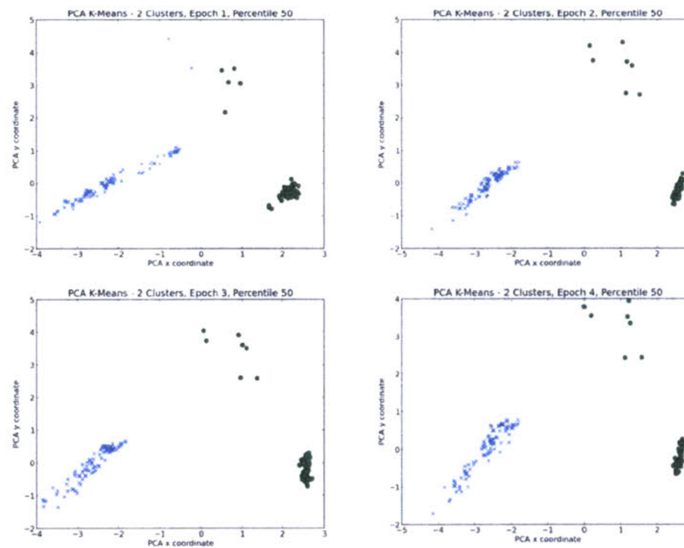

Figure 5-8: View Planner PCA K-Means. Cluster Structure over time.

To generate these graphs we first had to build the data matrix described in section 3.4. Our setting is described below:

- the common set of metrics is chosen by taking the union of non-constant metrics over the set of VMs
- the percentile value is the median.
- the number of clusters in K-Means is set to 2
- no data filtering is done

As part of the analysis, we would like to measure the impact of certain parameters on our clustering results. To start with, we want to:

- assess the effect of metric filtering on the clustering output
- study how different choices for the metric percentile value influence the cluster structure

The quality of a given cluster structure is judged in most cases based on two indexes: the distortion introduced in section 2.4.1 and the cluster score defined below.

$$\text{score} = \sum_{i=1}^{n} |c_i - d_i|$$

where n is the number of clusters, c_i is the number of clients in the ith cluster and d_i is the number of desktops. The goal is of course to minimize the distortion and to maximize the cluster score. To compare between 2 cluster structures A and B, we introduce a total order and say that $A \geq B$ if and only if $score_A \geq score_b$ or $score_A = score_B$ and $distortion_A \geq distortion_B$. Note that the score quantity cannot exceed 168 since there are 84 desktop VMs and 84 client VMs.

After exploring metric filtering and different percentile values, we focus on choosing the right number of clusters with the elbow method and the Bayesian Inference criterion (2.4). We then proceed to study the stability of the resulting K-Means clustering as we switch from the raw to the PCA space. It is important in our case to determine whether PCA distorts in any significant way the output of the K-Means algorithm. The last part of this section is dedicated to assigning labels to each cluster by building fingerprints with logistic regression.

Metric Filtering

Metric filtering was introduced in section 3.5 as a way to reduce the size of the data matrix by getting rid of the performance metrics that are irrelevant or redundant. Three levels of filtering were defined above Level 0. For space reasons we focus only on

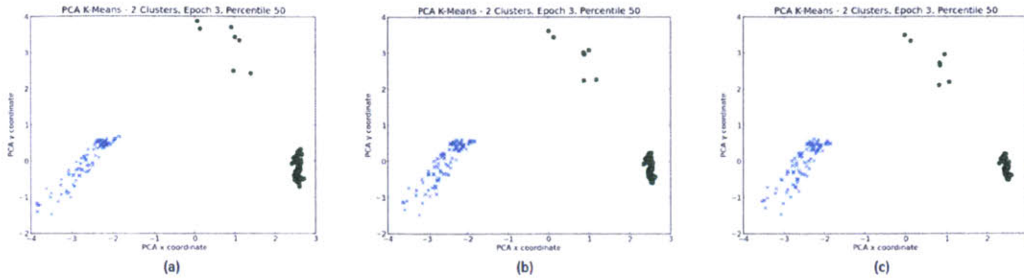

Figure 5-9: View Planner metric filtering. (a) Level 1 (b) Level 2 (c) Level 3

the 3rd epoch and analyze how each of the data filtering levels impacts the K-Means clustering. Figure 5-9 plots the clusters for each level.

The third level of filtering cuts out 51 metrics and the interesting part is that it does so without significantly affecting the clustering. Indeed, the corresponding plots for each level of filtering are almost identical. In addition to this, according to Table 5.2, the third level of filtering is actually the one with the highest quality index. The score remains the same (168) and indicates that we are always able to separate the desktops (on the left in the plots) from the clients (on the right). The distortion gradually decreases as we increase the filter level.

Filtering level	**Number of metrics**	**Score**	**Distortion**
0	242	168	0.925
1	239	168	0.903
2	207	168	0.761
3	191	168	0.691

Table 5.2: Metric filtering statistics for the 3rd epoch.

We observed the same trends for all the other epochs. Therefore, we assume it is safe to use either level 2 or 3 of filtering for the next tests.

Percentile Values

All the results presented so far used the 50th percentile when constructing the feature vector for each VM. We claim that fine tuning the percentile parameter is not necessary, and that adopting the median is a safe choice. To do so we examine what happens when we select the 10th or the 90th percentiles. The corresponding plots for

the 4th epoch are shown in Figure 5-10. Although the exact cluster shapes differ as we change the percentile from 10 to 50 and then to 90, the clients and the desktops remain separated.

In conclusion, even though varying the percentile changes the PCA and raw space coordinates corresponding to each VM, the overall structure stays the same: the client VMs are always close together, the desktops are more dispersed but still in their own cluster, and the infrastructure VMs are clearly visible outliers.

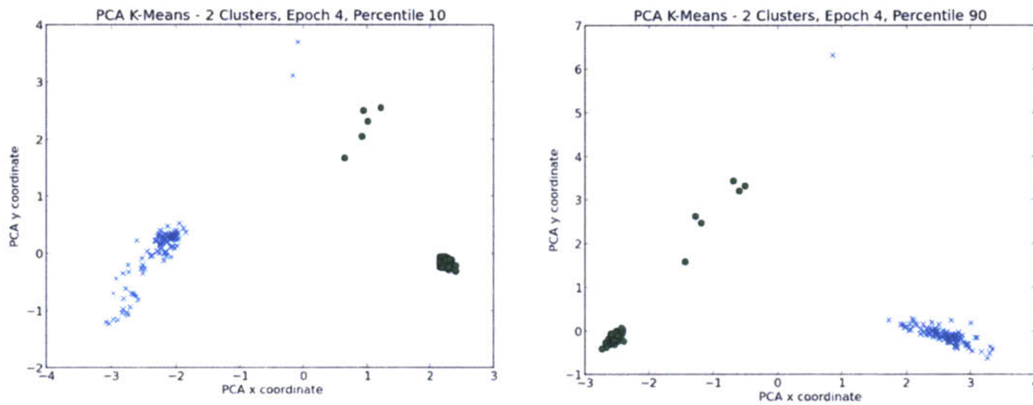

Figure 5-10: View Planner metric distribution percentiles. Comparing the 10th and the 90th percentiles.

Number of Clusters

K-Means Clustering is an unsupervised technique that requires the number of clusters (m) as input. Given our knowledge of the View Planner experiment, we would expect the optimal number of clusters to be either 2 or 3 (the latter if we take into account the effect of the infrastructure VMs). In accordance with these beliefs, we chose so far to work with only 2 clusters. However, we would like to be able to infer the optimal number of clusters without using any knowledge about the data.

To choose a number of clusters that fits our data the best we employ either the elbow method or the Bayesian information criterion (section 2.4). We consider again the 4th epoch as it presents an interesting cluster separation, as we shall see next. We assume that m ranges between 2 and 7. Figure 5-11 shows the associated distortion plot.

In the raw space, the elbow method graph has an ambiguous smooth shape and

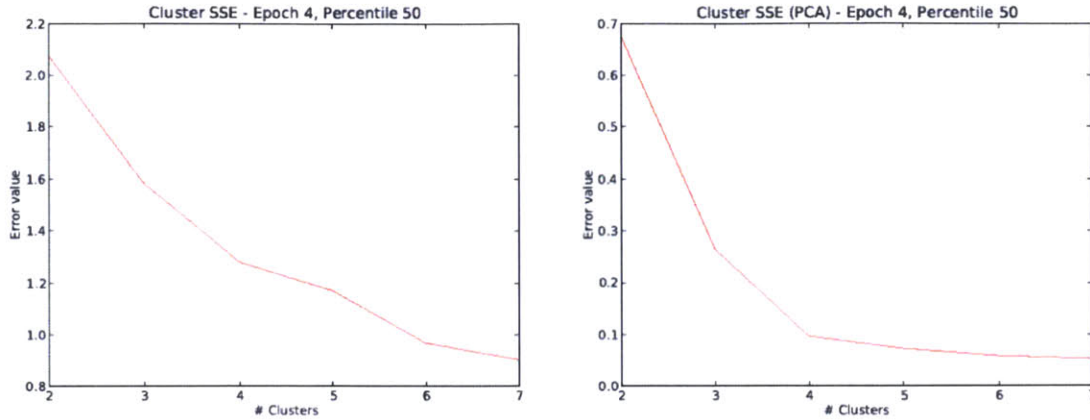

Figure 5-11: Distortion plots in the raw and PCA spaces. Choosing the number of clusters by identifying the sharp "elbows".

suggests that either 4 or 6 clusters should be chosen. Nevertheless, the distortion curve in the PCA space unambiguously reveals that 4 is good choice for the number of clusters. These results are confirmed by the BIC plots (Figure 5-12) which suggest 6 clusters in the raw space and either 4 or 6 clusters in the PCA space.

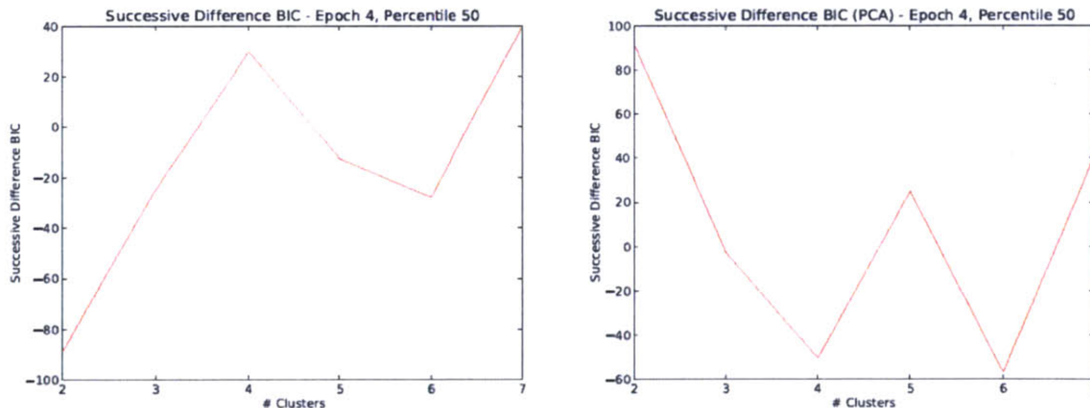

Figure 5-12: BIC successive difference plots in the raw and PCA spaces. Choose the number of cluster by looking for significant troughs.

The K-Means clustering obtained for $m = 4$ is shown in Figure 5-13. Given that experiments not shown here suggested 3 clusters for the previous epochs, it was surprinsing to get this data division for the virtual machines. The result is induced by the spread that causes the desktops to get split into 2 clusters (blue x-shaped and red +-shaped). We investigate further this outcome in the Bayesian Inference section.

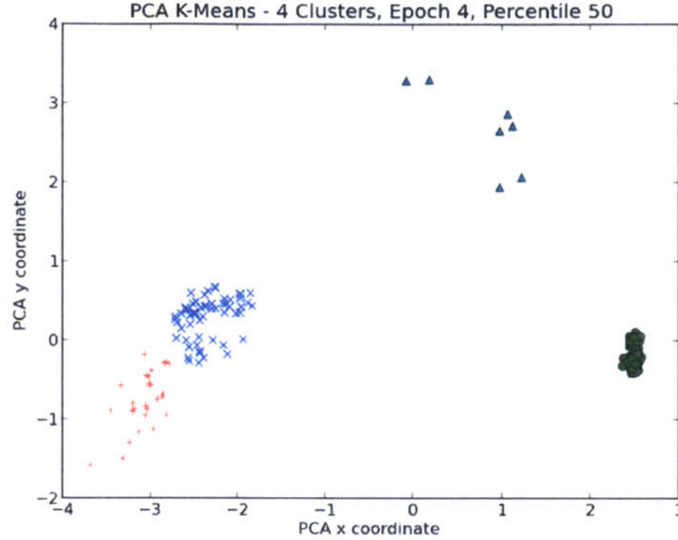

Figure 5-13: Three cluster partition. Desktop VMs are distributed among two clusters.

Cluster Stability

We implicitly assumed in our presentation of results that the K-Means clustering in the raw space is "similar" to the one in the PCA space. To prove the validity of our assumption, it is necessary to define what it means for two clusters to be similar. Formally, if V_i^{PCA} represents the set of VMs in PCA cluster i and V_j^{RAW} represents the set of VMs in raw cluster j, then we compute the matching ratio $m(V_i^{PCA}, V_j^{RAW})$ of cluster i with respect to cluster j using the formula:

$$m(V_{i^{PCA}}, V_j^{RAW}) = \frac{|V_i^{PCA} \cap V_j^{RAW}|}{size(V_i^{PCA})}$$

Note that $m(V_i^{PCA}, V_j^{RAW}) \neq m(V_j^{RAW}, V_i^{PCA})$ in general, hence the relationship we define is not symmetric. Next, let α^{PCA} and β^{RAW} be two partitions of the PCA and raw data into k clusters, V_i^{PCA} and W_i^{RAW} $(1 \leq i \leq k)$ respectively. Also let x be the total number of VMs in α^{PCA} and y be the one in β^{RAW}. The matching ratio of α^{PCA} with respect to β^{RAW} is computed using the following algorithm:

- Initialize $m(\alpha^{PCA}, \beta^{RAW})$ to 0.
- Sort the V_i^{PCA} sets in decreasing order of their size

- Pick each V_i^{PCA} and match it with the available W_{jRAW} for which $m(V_i^{PCA}, W_j^{RAW})$ is maximal.

- For each (V_i^{PCA}, W_j^{RAW}) matching remove W_j^{RAW} from β^{RAW} and add $\frac{m(V_i^{PCA}, W_j^{RAW}) \cdot (size(V_i^{PCA}) + size(W_j^{RAW}))}{x+y}$ to $m(\alpha^{PCA}, \beta^{RAW})$.

- Return $m(\alpha^{PCA}, \beta^{RAW})$.

There are a few tricks we use to speed up the above algorithm. The key idea is that $m(V_i^{PCA}, W_j^{RAW}) > \frac{1}{2}$ implies $m(V_i^{PCA}, W_j^{RAW})$ is maximal. Furthermore, to increase our chances of finding the best match for the current V_i^{PCA} as fast as possible, we also go through the remaining W_j^{RAW} candidates in decreasing order of their size. The matching ratio of partitions α^{PCA} and β^{RAW} is defined as

$$M(\alpha^{PCA}, \beta^{RAW}) = max(m(\alpha^{PCA}, \beta^{RAW}), m(\beta^{RAW}, \alpha^{PCA}))$$

It can be easily shown that $M(\alpha^{PCA}, \beta^{RAW}) = 1$ if and only if the α and β partitions are the same. In addition to this, the closer $M(\alpha^{PCA}, \beta^{RAW})$ is to 1, the more similar are the two partitions according to our definition.

The $M(\alpha^{PCA}, \beta^{RAW})$ values obtained by varying the epoch and the number of clusters while comparing the raw and PCA spaces are summarized in Table 5.3. All the values are close to 1, proving that the PCA space clustering is a good approximation of the raw space clustering.

		Epochs			
		1	2	3	4
Number of Clusters	2	1.0	1.0	1.0	1.0
	3	**1.0**	**1.0**	**1.0**	1.0
	4	1.0	0.967	0.977	**0.988**
	5	0.950	0.829	0.834	0.819
	6	0.885	0.823	0.826	0.743

Table 5.3: The raw-pca matching ratios for various epoch/number of clusters combinations.

We should point out that we are only interested in having a high matching ratio for the number of clusters that fit our data set the best according to the Bayesian

Information Criterion or the Elbow method. For each epoch, the best fit in terms of number of clusters is made bold. The high associated coefficients suggest that the clustering structure in the PCA and raw spaces are almost identical for the cases we care about.

Fingerprinting

For the rest of this subsection we set the number of clusters and the metric filtering level to 3, and we use the 50th percentile. We inspect the 4th epoch so the cluster structure is the one shown previously in 5-13.

Metric Name	**Metric Instance**	**Coefficient**
cpu.system.summation	0	2.48793139734
cpu.system.summation	cummulative	2.48076411647
mem.overheadTouched.average	cumulative	1.54899710642
mem.overhead.average	cumulative	1.54899710642
cpu.run.summation	cumulative	0.737469715863
cpu.used.summation	cumulative	0.705747421799
cpu.run.summation	0	0.702140766223
cpu.idle.summation	0	0.687808559368
cpu.wait.summation	0	0.687311909852
cpu.used.summation	0	0.671516557651
cpu.usagemhz.average	0	0.668466231252
cpu.wait.summation	cumulative	0.6557275407438
cpu.idle.summation	cumulative	0.655674855983
cpu.demand.average	cumulative	0.655674855983
cpu.idle.summation	cumulative	0.6352743498673
net.broadcastRx.summation	400	0.551896952279

Table 5.4: Blue Cluster Fingerprint (epoch 4, size 62)

The fingerprinting technique (section 4.2) is based on logistic regression and yields the performance metrics and significance weights enumerated in Table 5.4 and Table 5.5 for the two clusters containing desktops. As expected, the two fingerprints are very similar and they both contain a lot of CPU related metrics. However, the respective logistic regression weights are different. In the Bayesian inference analysis we return to these fingerprints and examine their relationship with the unusual spread characteristic to the desktops.

We end the subsection with a plot that captures the quality of our classification.

Metric Name	Metric Instance	Coefficient
cpu.idle.summation	0	1.90770616035
cpu.wait.summation	0	1.09385712865
cpu.overlap.summation	0	0.979734646251
cpu.overlap.summation	cumulative	0.973513079011
cpu.usagemhz.average	0	0.948671503677
cpu.used.summation	0	0.948660880027
cpu.used.summation	cumulative	0.946112180959
cpu.run.summation	0	0.934122274434
cpu.run.summation	cumulative	0.931474947522
net.broadcastRx.summation	4000	0.637473018679
net.broadcastRx.summation	cumulative	0.637473018679
cpu.idle.summation	cumulative	0.574910259872
cpu.wait.summation	cumulative	0.573011089866
cpu.demand.average	cumulative	0.571596830028
cpu.usage.average	cumulative	0.501795476174

Table 5.5: Red Cluster Fingerprint (epoch 4, size 23)

Figure 5-14 shows not only that we are doing a relatively good job, but also reinforces the fact that not a lot of information is lost in the transition from the raw to the PCA space.

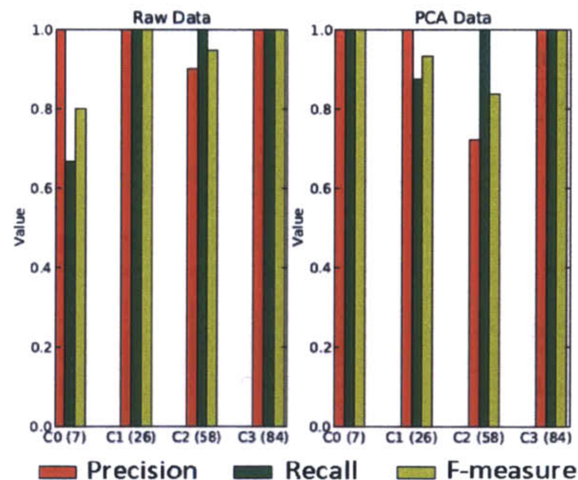

Figure 5-14: Precision, Recall, and F-Measure. Comparison between the raw and pca spaces.

5.2.4 Bayesian Inference Analysis

As explained in the motivation section of chapter 1, in most real-life systems the administrators are usually interested in explaining the values of a key performance indicator such as throughput, CPU usage, memory usage, or an alternative measure relevant to the system that is being studied. For example, the administrator may want to have tools that provide him or her with the intuition necessary for understanding why in specific scenarios the system falls short of meeting the requirements.

In the case we study, View Planner, there is no a priori knowledge about the performance indicators that we should analyze. As a result we try to infer in this section what are the metrics of interest that we should consider and what are the factors with which they share dependencies. In addition to this, we use our analysis to tackle one of the questions posed in the previous subsections. Specifically, we want to identify the causes for the observed spread in the group of desktops that leads to them being split between two clusters.

Since our analysis is strictly concerned with the desktops, we restrict our attention to them. To pinpoint some potential performance indicator candidates, we search for distributions of metric percentile values that are multimodal across the set of VMs we consider. This choice is motivated by the fact that one would expect in general a uniform or a normal distribution when computing the percentile values over a set of similar virtual machines. Obtaining a multimodal distributuion instead indicates a surprising behavior that can have multiple causes. On the one hand, it could be that our initial information is not accurate and there are further differences between the VMs we analyze. On the other hand, the unusual observation may be the result of a performance problem.

Some of the metrics found to be multimodal using Silverman's test and a 0.1 significance level ([18]) are listed in Table 5.6

To exemplify our method of finding dependencies we pick one of the multimodal metrics as our performance indicator. We choose *cpu.usage.average*, which measures the CPU usage as a percentage during the specified time interval. The distribution

Metric Name	Number of modes	P-value
cpu.idle.summation	2	0.33
cpu.latency.average	3	0.26
cpu.ready.summation	3	0.35
cpu.run.summation	3	0.20
cpu.used.summation	3	0.54
cpu.usage.average	3	0.95
cpu.usagemhz.average	3	0.96
cpu.wait.summation	2	0.30

Table 5.6: Multimodal metrics identified with Silverman's method.

plot is given ion Figure 5-15.

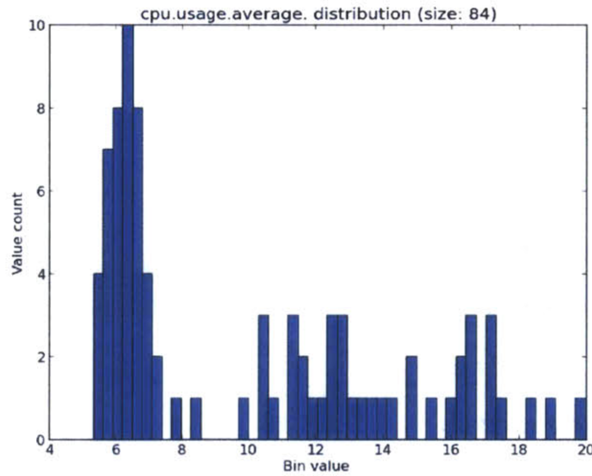

Figure 5-15: Performance indicator distribution. CPU usage (%) over the set of View Planner desktops.

Next we would like to determine which metrics share dependencies with our performance indicator. Ideally, if we let X and Y be random variables corresponding to the performance indicator and the candidate metric, we want to compare $P(X|Y)$ with $P(X)$. To avoid computing conditional probabilities, which can be time demanding, we take a shortcut and try to infer the potential metric candidates by looking at mutual information.

The key idea is that a value of 0 or close to 0 for the mutual information indicates no or very slight dependencies between the two metrics. In this case, $P(X|Y)$ and $P(X)$ are roughly equal, so the two distributions can be assumed to be independent.

Therefore, we need to search for candidate metrics that have a high mutual information with *cpu.usage.average*. Table 5.7 shows the top 10 such metrics. As one may expect, *cpu.usage.average* is strongly related to other CPU metrics.

Candidate Metric	**Mutual Information**
cpu.ready.summation	6.19
cpu.latency.average	6.19
cpu.idle.summation	6.16
cpu.wait.summation	6.16
cpu.run.summation	6.14
cpu.used.summation	6.12
cpu.usagemhz.average	6.04
cpu.demand.average	5.96
mem.entitlement.average	5.66
net.packetsRx.summation	5.51

Table 5.7: Top mutual information values between the performance indicator (*cpu.usage.average*) and the candidate metric.

The results suggested by the table can be confirmed by computing the actual conditional distributions. For instance, assume *cpu.ready.summation* (percentage of time that the virtual machine was ready, but could not get scheduled to run on the physical CPU) is picked as the candidate metric and let μ be its expected value over the set of desktops. In Figure 5-16 we plot the conditional distribution $P(cpu.usage.avarage | cpu.ready.summation > \mu)$. If there was no dependency between the two metrics, we would have expected to select uniformly at random from the original distribution and obtain a conditional similar in shape. However, it can be immediately observed that the overall *cpu.usage.average* distribution shown in Figure 5-15 is considerably different in its properties from the conditional distribution. In fact, the imposed condition seems to act as a filter that selects an almost contiguous part from the original distribution.

It is interesting to look back now at the associated K-Means clustering depicted in Figure 5-13 from the previous subsection. The desktops are split between the red and the blue clusters. If we only plot the average CPU usage distribution over the red cluster, we get the result shown in Figure 5-17, which is a large subset of the distribution in Figure 5-16. We believe that the two distributions are not identical due

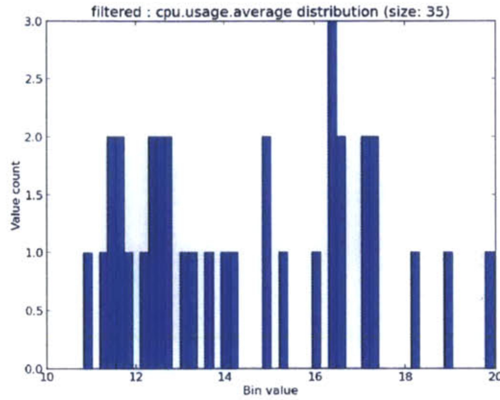

Figure 5-16: Conditional distribution. Average CPU usage conditioned on *cpu.ready.summation* being higher than its mean.

to our choice for the range of *cpu.ready.summation* values ($> \mu$) and to features of the K-Means algorithm that we cannot control. Nevertheless, we find that the analysis uncovers an interesting property of the candidate metric: the ranges of values it takes over the red and the blue clusters of desktops are almost disjoint. Similar results were obtained when considering other candidate metrics.

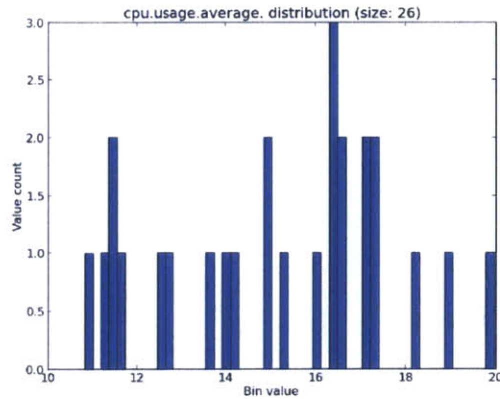

Figure 5-17: Conditional distribution. Average CPU usage conditioned on being in the red cluster of desktops.

Our analysis so far indicates that there are strong dependencies between many of the performance metrics we consider. In addition to this, it identifies the multi-modality characteristic to several metric distributions as the possible cause for the desktop spread. One last question we would like to answer is about the set of metrics

that should be discarded to bring the dekstop VMs as close together as possible while maintaining the separation with the client and the infrastructure VMs. A possible choice is to simply discard the metrics we found to be multimodal. The initial and the revised clustering in the PCA space are shown in Figure 5-18 (a) and (b), respectively . The K-Means algorithm manages now to automatically differentiate between the 3 classes of virtual machines. The distortion stays about the same: 0.096 for the 4-clustering and 0.105 for the 3-clustering.

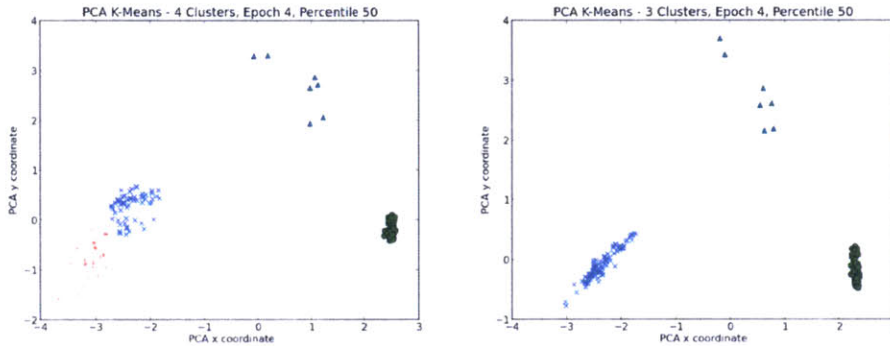

Figure 5-18: K-Means clustering. (a) the initial clusters (b) the clusters obtained after removing multimodal metrics.

The Carat ([13]) inspired method presented in section 2.8 proves to be a lot faster and more accurate for constructing the set of metrics to be discarded. In this context, if we let m_{max} be the maximum median value for metric m over the desktops, then we choose to remove m if the spread value

$$S(m) = \frac{|E[m|\text{red cluster}] - E[m|\text{blue cluster}]|}{m_{max}} > \theta$$

where θ is a threshold that characterizes how close together we want to bring the desktop VMs. The smaller the θ, the more metrics we eliminate and the closer we bring together the desktop VMs. However, if θ is too small, we risk to eliminate the features that set apart the desktops from the clients and the infrastructure VMs. The K-Means clustering obtained for $\theta = 0.2$ is shown in Figure 5-19. The PCA distortion dropped now to 0.099. The dark blue x-shaped cluster corresponds to the 84 desktops, the green o-shaped to the 84 clients, and the 7 triangle shaped points in the cyan cluster to the infrastructure VMs.

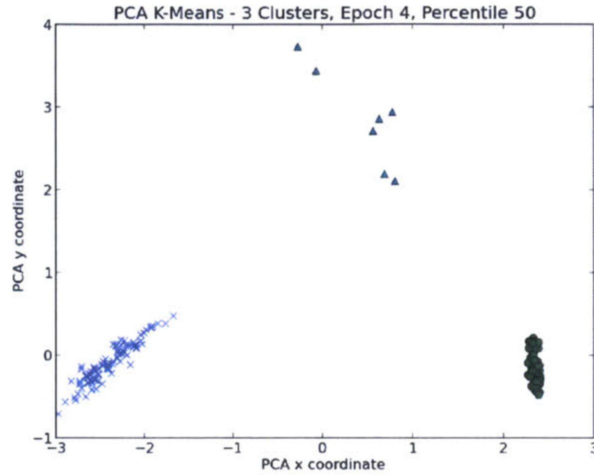

Figure 5-19: K-Means clustering. The clusters obtained after removing the spread metrics for $\theta = 0.2$.

The metrics we eliminated and their $S(m)$ values are listed in Table 5.8. Due to space constraints we only enumerate the metrics with a spread value above 0.3.

Spread Metric Name	Spread Metric Instance	Spread Value
cpu.used.summation	0	0.4068
cpu.used.summation	cumulative	0.4068
cpu.usagemhz.average	0	0.4067
cpu.run.summation	0	0.4052
cpu.run.summation	cumulative	0.4052
rescpu.actav5.latest	cumulative	0.3870
rescpu.runav1.latest	cumulative	0.3862
cpu.overlap.summation	0	0.3834
cpu.overlap.summation	cumulative	0.3834
cpu.usagemhz.average	cumulative	0.3783
cpu.usage.average	cumulative	0.3778
rescpu.actav1.lateste	cumulative	0.3649
cpu.demand.average	cumulative	0.3616
rescpu.actpk1.latest	cumulative	0.3649
cpu.ready.summation	0	0.3354
cpu.ready.summation	cumulative	0.3354
rescpu.actpk5.latest	cumulative	0.3001

Table 5.8: Top spread metrics and their associated spread values.

5.3 Nimbus

In this last case study we inspect a part of Nimbus, the VMware vSphere R&D developer/QA cloud. This large scale and heterogeneous cluster allows developers and QA to deploy all components of VSphere deployments (ESX, VC, NFS, etc.) in the cloud. Given that any developer can spawn one or more VMs for testing purposes, we start with the premise that Nimbus has a complicated and nonuniform structure that will challenge our analysis pipeline. Our goals for this case study are the following:

- Show that our clustering pipeline is able to cope with the size and heterogeneity of Nimbus. In particular, we use our methods to reveal that the cluster of VMs has a more homogeneous structure than we initially anticipated.
- Perform an analysis of dominant resources and determine the fraction of VMs that almost always prefer memory, almost always prefer CPU, or are a hybrid with respect to the these 2 main resources. Hint at how such information could be leveraged to perform improved placement and resource management decisions.

5.3.1 Clustering Analysis

We restricted our attention to a self-contained part of Nimbus encompassing roughly 800 virtual machines and 20 hosts.

The PCA grouping produced by our method is shown in Figure 5-20. The BIC recommended a number of clusters equal to 4. We would like, however, to have some type of validation for our results, and prove that they indeed reflect the state of Nimbus. Trying to identify the exact tasks executed by each VM in the time frame we study is of course unfeasible. Fortunately, it turns out that the VM friendly names to which we have access reveal in this case the owner and the deployment type. Since we expected to notice some sort of similarity for VMs that have the same owner/deployment type, we generated color maps for each of these 2 identifiers. There

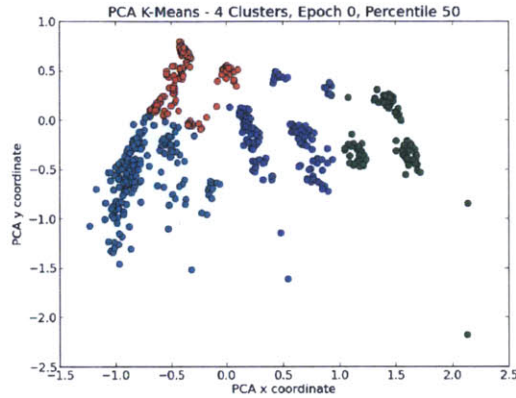

Figure 5-20: Nimbus PCA K-Means clustering. VM partition into 4 clusters.

are a large number of owners (130) and as a result the corresponding color map is hard to read and does not provide further insight (Figure 5-21 a). However, there are only 11 deployment types (esx-fullinstall, vc-vpxinstall, iscsi-default, vsm-vsminstall, db-mssqlvcd, generic-ovf, vc-vcvainstall, vcd-vcdinstall, vc-cloudvminstall, nfs-default, esx-pxeboot) that roughly match the structure we identified, as shown in Figure 5-21 b.

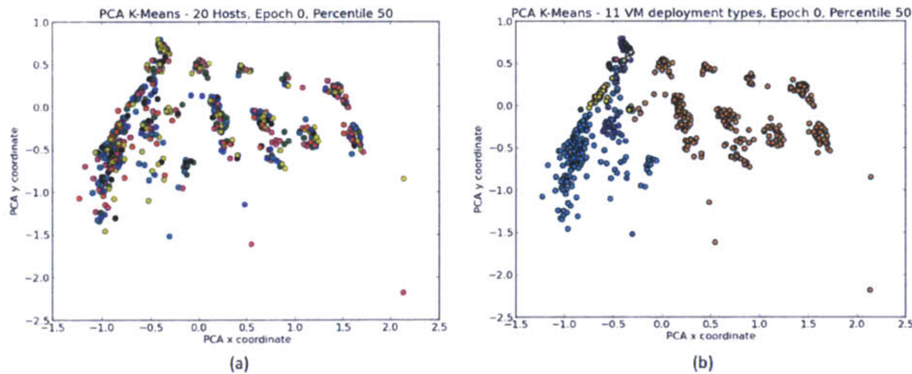

Figure 5-21: (a) VM owner color map (b) VM deployment type color map

5.3.2 Dominant Resources Analysis

After showing that our approach to clustering can cope with the large size of Nimbus, we thought it would be interesting to also apply our dominant resource pipeline to the data set. As shown in the corresponding section of chapter 4, we can divide the virtual

machines into 3 types for a given time interval based on their dominant resource with respect to the host: memory hungry (dominant resource is always memory), CPU hungry (dominant resource is always CPU), hybrid (dominant resource changes over time).

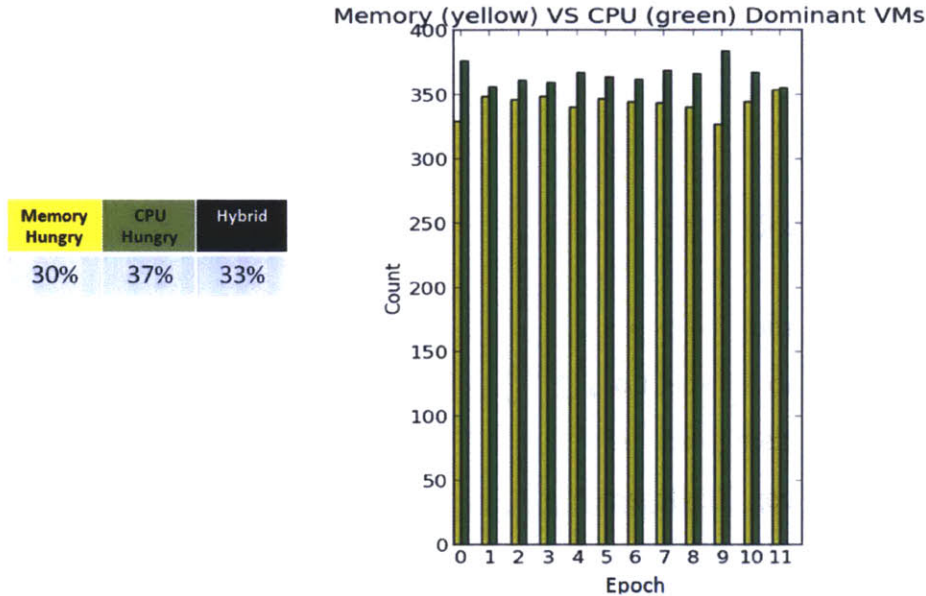

Figure 5-22: Nimbus dominant resources. Percentages of each VM type (left) and dominant resource per epoch (right)

The experiment we performed spanned 12 collection epochs (hours). Given the variety of tests performed on the Nimbus VMs, we expected initially to observe a lot a hybrid VMs, but that turned out not to be the case. In fact, as the left side of Figure 5-22 illustrates, 67% of the VMs are either memory or CPU hungry. We should also mention that most of the hybrid VMs have very few fluctuations in their dominant resource across epochs. Finally, it is interesting to note that in each epoch roughly half the VMs are memory dominant and half are CPU dominant(right side of Figure 5-22).

Studying dominant resources is important as it can help us make better decisions regarding placement and resource management. For example, consider the PCA space representation of the 3 types of VMs (memory hungry, CPU hungry, hybrid) shown in Figure 5-23. When a particular VM asks for more memory we can try to avoid getting it from VMs that are either close-by in the PCA space (thus similar and likely

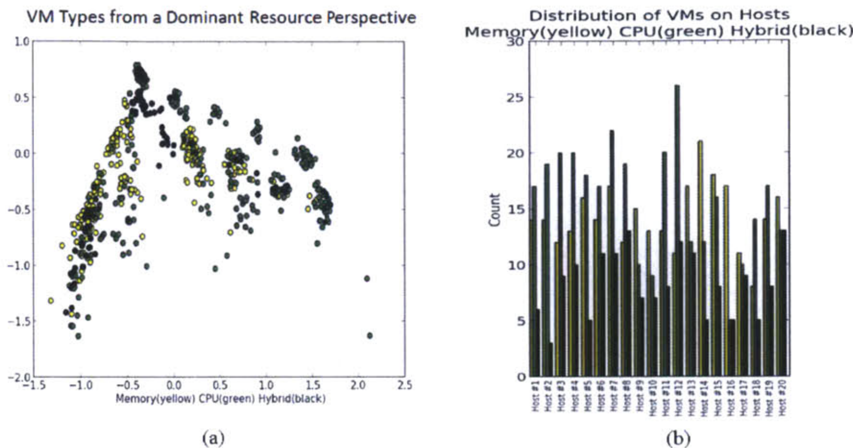

Figure 5-23: (a) VM dominant resource types in the PCA space (b) Distribution on VM dominant resource types on hosts

to also be needing more memory) and VMs that are memory hungry. Moreover, by balancing the number of memory and CPU hungry VMs on each host, we have better chances preventing resource overcommitment. Figure 5-23 (b) shows that Nimbus does a good job balancing the memory and CPU hungry VMs on each host.

5.3.3 Cluster Diagnosis

Virtual machines that fall within the same deployment type are usually close to each other in the PCA space, as illustrated in Figure 5-21 (b) . However, there is one visible exception to this expected behavior: the light brown ESX VMs ([21]) are split among multiple subclusters. Even if we judge by the K-Means clustering shown in Figure 5-20, the ESX VMs are roughly still separated into two clusters (blue and green), each made up of multiple subclusters.

In this section we identify the factors that cause the separation among the ESX VMs. We emphasize the fact that without any extra information such as the jobs running on the ESX machines or historical data associated with this subset of VMs, it is impossible for us to decide whether the separation we observe represents abnormal behavior. Therefore, our method can explain why we see the separation in terms of the metrics we consider, but does not have the ability to classify the resulting clustering as normal or abnormal. If we were to add extra information (e.g. historical

data that can be used to predict the state of the cluster), the latter would also be possible.

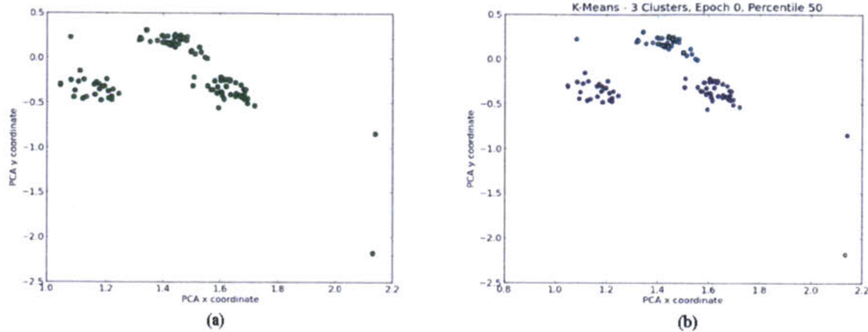

Figure 5-24: (a) Subcluster of ESX VMs (b) K-means partition of subcluster in the raw space projected into 2 dimensions

For simplicity, we only focus on the green cluster portrayed in Figure 5-20, and reproduced in Figure 5-24 (a). Without employing any algorithms, one can reasonably make the claim that the green cluster has 3 visible subclusters. Interestingly enough, when the same corresponding set of VMs is passed through K-means and separated into 3 clusters in the **raw** space then projected into 2 dimensions with PCA, we obtain the structure in Figure 5-24 (b). Therefore, the two lower subclusters are closer together in the raw space, and the point in the bottom right corner is treated as an outlier and placed in its own cluster.

Metric Name	Metric Instance	Coefficient
mem.usage.average	cumulative	3.25829049013
mem.active.average	cumulative	1.14010910839
mem.shared.average	cumulative	0.952924768235
mem.granted.average	cumulative	0.693351941959
cpu.idle.summation	cumulative	0.492365555619
cpu.wait.summation	cumulative	0.49209877748
cpu.capacitycontention.average	cumulative	0.4399936131343
cpu.latency.average	cumulative	0.435660108149
virtualDisk.totalWriteLatency.average	scsi0:2	0.423205392308
rescpu.runpk1.latest	cumulative	0.37673198228
rescpu.runpk5.latest	cumulative	0.366833553995
rescpu.runpk15.latest	cumulative	0.360381734335
rescpu.actpk1.latest	cumulative	0.347691573585
net.usage.average	cumulative	0.341082293824
net.throughputusage.average	cumulative	0.340241594087

Table 5.9: Fingerprint for the purple subcluster shown in Figure 5-24 (b)

As an extra confirmation for the raw space structure, we take a look at the fingerprints for the purple and light blue subclusters. Tables 5.9 and 5.10 show that these fingerprints are similar, which gives further motivation for our ongoing investigation that aims to figure out the metrics inducing the observed spread.

Metric Name	**Metric Instance**	**Coefficient**
mem.usage.average	cumulative	3.19277948962
mem.active.average	cumulative	1.15622057457
mem.shared.average	cumulative	0.935658415779
mem.granted.average	cumulative	0.732080387897
cpu.idle.summation	cumulative	0.463719636072
cpu.wait.summation	cumulative	0.463377063583
cpu.capacitycontention.average	cumulative	0.4607756987
cpu.latency.average	cumulative	0.44821374201
virtualDisk.totalWriteLatency.average	scsi0:2	0.413667819136
rescpu.runpk1.latest	cumulative	0.37289368725
rescpu.runpk5.latest	cumulative	0.36605187582
cpu.overlap.summation	cumulative	0.362877435766
rescpu.runpk15.latest	cumulative	0.359022347297
rescpu.actpk1.latest	cumulative	0.348501842961
net.multicastRx.summation	cumulative	0.347714907706

Table 5.10: Fingerprint for the cyan subcluster shown in Figure 5-24 (b)

Taking into account the hints provided by the raw space distribution and the fingerprints, we proceed by trying to identify the metrics that set apart the purple cluster from the cyan one. We adopt the carat-style method presented in section 2.8 and look to eliminate the "spread metrics" m that result in a large value for the expression:

$$S(m) = \frac{|E[m|\text{purple cluster}] - E[m|\text{cyan cluster}]|}{m_{max}}$$

where m_{max} is the maximum median value over all the VMs. The upper limit we place on this expression is directly related to how close together we want to bring the points. The lower the upper limit, the closer together we bring the points. For example, if we set an upper limit of 0.05 (i.e do not accept mean metric differences larger than 5% between the 2 clusters), we get the structure depicted in Figure 5-25 (a). As desired, the point are now strongly clustered together. The spread metrics we eliminated and their associated spread values are listed in Table 5.11. To save space, we only include the ones with a spread value above 0.1.

Spread Metric Name	Spread Metric Instance	Spread Value
mem.usage.average	cumulative	0.7268
mem.active.average	cumulative	0.5453
mem.shared.average	cumulative	0.5093
mem.granted.average	cumulative	0.3896
mem.capacity.usage.average	cumulative	0.3635
mem.swaptarget.average	cumulative	0.2518
mem.swapped.average	cumulative	0.2475
cpu.capacity.contention.average	0	0.2452
cpu.usage.average	cumulative	0.2397
virtualDisk.totalWriteLatency.average	scsi0:2	0.2394
cpu.latency.average	cumulative	0.2358
cpu.idle.summation	cumulative	0.2217
cpu.wait.summation	cumulative	0.2217
virtualDisk.throughput.usage.average	scsi0:2	0.1569
virtualDisk.read.average	scsi0:2	0.1561
net.throughput.contention.summation	0	0.1551
virtualDisk.write.average	scsi0:2	0.1550
disk.maxTotalLatency.latest	cumulative	0.1226
virtualDisk.totalWriteLatency.average	scsi0:1	0.1182

Table 5.11: Top spread metrics and their associated spread values.

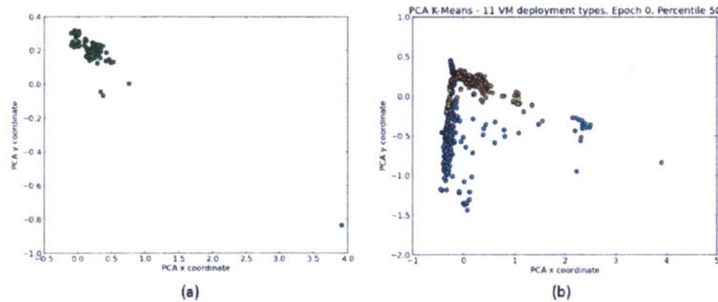

Figure 5-25: Structure after the removal of the "spread metrics" (a) Green cluster (b) The entire set of VMs

Our analysis so far only focused on a subset of the light brown ESX VMs shown originally in Figure 5-21. It is interesting to note that removing the "spread metrics" identified for this subset had actually a global effect and brought the bulk of the ESX VMs together, as illustrated in Figure 5-25 (b). Moreover, the metric elimination did not cause noticeable changes in the space structure of the other deployment types, showing that we were able to specifically target the ESX machines.

Chapter 6

Related Work

In this section we introduce a series of papers that have served as a source of inspiration. They discuss several techniques that we also decided to employ. We hope that the related work presented below will give the motivation for the different approaches that we take to design our pipeline.

Fingerprinting systems (applications, virtual machines, physical machines, datacenters) have been explored by others for a variety of purposes including problem diagnosis, VM placement/colocation and application scheduling. In [2], the authors use datacenter telemetry from machines running a single distributed application to construct fingerprints that can describe possible performance crises affecting the application. Known crises are labeled, enabling the use of SML classification techniques (logistic regression in particular) to identify the features (e.g. patterns of metric variations) associated with the event. Each fingerprint is linked to a resolution that provides a remediation for the problem. If no remediation is available, the fingerprints are linked to new, previously unseen, crises. In our work, fingerprints are used to represent the relationships (neighborhoods of similarity) between VMs based on their telemetry. The results are then exploited for explaining performance variations (explaining why VMs expected to be similar are behaving as if they were different), and for obtaining hints regarding placement and co-location. Similarly to [2], we design the fingerprints so that they are easy to compute and scale linearly with the number of performance metrics considered not the number of machines. In addition

to this, we also use K-Means for clustering and Logistic Regression for classification.

NetCohort [10] aims to influence the placement of VM ensembles VMs that co-operate and communicate jointly to provide a service or complete a task in order to minimize the usage of network bi-section bandwidth. The authors' work relies on identifying complementary communication patterns between VMs. These fingerprints of communication patterns are discovered using statistical correlations over a restricted set of metrics (those concerned with network communications such as packets in/out). In our experiments, we do not restrict ourselves to a specific set of resource metrics, but rather obtain all of the VM-level available statistics (e.g., cpu-related, memory-related, network-related, I/O-related counters) from the Performance Manager. Levels of filtering based on simple signal processing techniques (e.g., entropy) and statistical techniques (e.g., variance) are employed to exclude metrics from consideration. These exclusions do not prevent us from considering a diverse set of metrics and do not affect our results. In fact, even after the filtering stage we are still considering a large number of metrics from a large number of VMs. This is possible due to our efficient clustering (K-means) and summarization (PCA) techniques.

Statistical Demand Profiles (SDPs) [17] are an example of fingerprints used to capture the historical and/or anticipated resource requirements of applications. An SDP is constructed by collecting the history of an application's resource demands over an interval and using that history to estimate a sequence of time-indexed probability mass functions (pmfs) per resource type. The pmfs over resource types are utilized to influence how applications are run over shared pools of resources. In particular, the access to resources is controlled and applications with complementary resource demands (in time and in resource type) are co-located in the same resource pool for better statistical multiplexing of the underlying resources. In contrast, our pipeline does not require extensive historical observations to produce scheduling hints that can influence placement or co-location. For instance, relatively few observations (180 samples/hour) are collected to construct a VM's Dominant Resource fingerprint or its telemetry fingerprint. The Dominant Resource fingerprint can serve for co-locating VMs with complementary dominant resources for better statistical multiplexing. The

VM relationships expressed through telemetry fingerprints are important when making decisions about "stealing"/re-owing resources (e.g., to avoid victimizing related VMs).

Adam Oliner et al. design a method in [14] for identifying the sources of problems in complex productions systems based on influence. By their definition, two components share an influence if they exhibit surprising behavior at about the same time. We follow a similar approach and compute VM metric correlations to study how the behavior of one VM influences the performance of another VM.

In [12], Adam Oliner et al. aim to do online detection of sources of problems in complex systems. This is useful since big systems have a rapid turnaround so the answer to a question is relevant only if computed in real-time. The paper's online method for analyzing and answering questions about large systems is based on anomaly signals. They design such signals to be function representations of the system logs. It is then shown that understanding complex relationships between heterogeneous components reduces to studying the variance in a set of signals. We borrowed the idea of using Principal Component Analysis (PCA) to deal with redundancies. from this paper. Their approach also motivates the use of VM metric variations to construct our telemetry fingerprints.

Ali Ghodsi et al. study fair dominant resource allocation in [4]. Even though this paper focuses on fairness policies, we can reuse the notion of a dominant resource to group VMs and get a possible fingerprint (e.g. the dominant resource is the resource a VM cares about the most). This leads to the use of dominant resource patterns/fingerprints in our analysis pipeline.

In general, most state-of-the-art (performance) anomaly detection systems ([2, 23]) are dependent on the existence of extensive historical data. Without such information the system cannot distinguish between normal and abnormal behavior with low false positive and false negative rates. We take a different path in our work and use VM similarity as a complement for history to provide additional context for anomaly detection and diagnosis. As a result, we can perform diagnosis in many cases with little historical data.

Reasoning about the behavior and characteristics of an application deployed in various settings through methods based on group behavior has been explored in [11]. The authors put forth a procedure that enables collections of independent instances of the same application to cooperatively monitor their execution for flaws and attacks and notify the community when such events are detected. Similarly, we use clustering techniques to automatically detect instances of similar/identical VMs (a neighborhood). We are then able to not only extract and examine the key telemetry metrics of all the VMs in that neighborhood, but also detect and explain/diagnose differences in these metrics.

Carat ([13]) is another project similar to our work project that uses the idea of an Application Community to do collaborative detection of energy hogs/bugs for smart phones. More exactly, the authors define the Application Community to be a collection of multiple instances of the same application (or similar applications) running in different environments. The battery usage data is collected from the entire community and used to identify possible energy hogs (applications or settings that lead to a higher discharge rate) and energy bugs (some running instances of the app that drain the battery faster than other instances of the app). In Carat, the features used for diagnosis (e.g., whether WiFi is turned on, the number of concurrent applications, etc.) are selected by experts in the field. In our work, we automatically try to identify candidate features using techniques from signal processing (e.g., mutual information) and statistics (Silvermans test for multi-modality [18])

Chapter 7

Summary of Contributions and Future Work

7.1 Contributions

In this thesis we present a collaborative approach to performance bug diagnosis based on algorithms that automatically discover and leverage relationships between virtual machines. In addition to this, we believe that the information generated by our analysis pipelines can be used to improve placement and resource management decisions.

The main contributions we make are the following:

- We present a robust analysis pipeline that enables automatic discovery of relationships between VMs (e.g., similarity)
- We construct techniques for collaborative diagnosis
- We provide analysis/diagnosis examples for 3 different development and production clusters inside VMware

7.2 Future Work

There are two main direction in which our work can be extended in the future.

The first is taking our offline pipeline and transforming it into an interactive online system with improved performance. Several steps need to be taken in this case. For an online algorithm it is necessary to modify the collection procedures and the Performance Manager in order to ensure a real-time data feed. To provide easy interaction with the system we can build a GUI wrapper around the current functionality. Finally, we noticed that the performance bottleneck is not our analysis pipeline, but rather the step that collects and parses the data. Therefore, the bottleneck can be easily fixed by partitioning the VMs into disjoint sets and assigning each for processing to a different thread.

The second direction involves some of the ideas we presented in the motivation section. On the one hand we could try to use dominant resource patterns as placement/co-location hints for the Distributed Research Scheduler. On the other hand, we could test whether VM/workload "similarity" information can be exploited to minimize VM migration and re-balancing.

Finally, we would like to point out that this work can complement the pattern mining done in vCOPS ([23]).

References

[1] Peter J. Bickel and Kjell A. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics.* Pearson PrenticeHall, 2000.

[2] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B. Woodard, and Hans Andersen. Fingerprinting the datacenter: automated classification of performance crises. In *Proceedings of the 5th European conference on Computer systems*, 2010.

[3] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., 2005.

[4] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *USENIX NSDI*, 2011.

[5] Preetham Gopalaswamy and Ravi Soundararajan. *VI Performance Monitoring.* VMware, 2008. `http://vmware.com/files/webinars/communities/VI-Performance-Monitoring.pdf`.

[6] Ujjwal Das Gupta, Vinay Menon, and Uday Babbar. Parameter Selection for EM Clustering Using Information Criterion and PDDP. *International Journal of Engineering and Technology*, 2(4), 2010.

[7] John A. Hartigan. *Clustering algorithms.* John Wiley & Sons, Inc., 1975.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning.* Springer Series in Statistics, 2001.

[9] David W. Hosmer and Stanley Lemeshow. *Applied Logistic Regression.* John Wiley & Sons, Inc., 2000.

[10] Liting Hu, Karsten Schwan, Junjie Zhang Ajay Gulati, and Chengwei Wang. Netcohort: detecting and managing vm ensembles in virtualized data centers. In *Proceedings of the 9th international conference on Autonomic computing*, ICAC '12, pages 3–12, New York, NY, USA, 2012. ACM.

[11] Michael E. Locasto, Stelios Sidiroglou, and Angelos D. Keromytis. Application communities: using monoculture for dependability. In *Proceedings of the First*

conference on Hot topics in system dependability, HotDep'05, pages 9–9, Berkeley, CA, USA, 2005. USENIX Association.

[12] Adam J. Oliner and Alex Aiken. Online detection of multi-component interactions in production systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2011.

[13] Adam J. Oliner, Anand Iyer, Eemil Lagerspetz, Sasu Tarkoma, and Ion Stoica. Collaborative energy debugging for mobile devices. In *Proceedings of the Eighth USENIX conference on Hot Topics in System Dependability*, 2012.

[14] Adam J. Oliner, Ashutosh V. Kulkarni, and Alex Aiken. Using correlated surprise to infer shared inuence. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2010.

[15] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559 – 572, 1901.

[16] Dan Pelleg and Andrew Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

[17] Jerry Rolia, Xiaoyun Zhu, Martin Arlitt, and Artur Andrzejak. Statistical service assurances for applications in utility grid environments. *Perform. Eval.*, 58(2+3):319–339, November 2004.

[18] Bernard W. Silverman. Using kernel density estimates to investigate multimodality. *Journal of the Royal Statistical Society*, pages 97 – 99, 1981.

[19] Robert L. Thorndike. Who belong in the family? *Psychometrika*, 18(4):267 – 276, 1953.

[20] VMware. VMware Infrastructure Architecture Overview. `http://vmware.com/pdf/vi_architecture_wp.pdf`, 2006.

[21] VMware. VMware ESX and VMware ESXi. `http://www.vmware.com/files/pdf/VMware-ESX-and-VMware-ESXi-DS-EN.pdf`, 2009.

[22] VMware. VMware View Planner Installation and User Guide. `http://communities.vmware.com/docs/DOC-15578`, 2011.

[23] VMware Inc. Vmware vcenter operations enterprise. automated operations management for your enterprise. `http://www.vmware.com/files/pdf/vcenter/VMware-vCenter-Operations-Enterprise-Standalone-Automated-Operations\-Mgmt-WP-EN.pdf`.

[24] Qinpei Zhao, Ville Hautamaki, and Pasi Frnti. Knee point detection in BIC for detecting the number of clusters. *Advanced Concepts for Intelligent Vision Systems*, pages 664 – 673, 2008.