M.S. THESIS

Machine Learning Models for Live Migration Metrics Prediction

Live Migration Metrics 예측을 위한 기계 학습 모델

FEBRUARY 2019

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Simeon Varbanov

M.S. THESIS

Machine Learning Models for Live Migration Metrics Prediction

Live Migration Metrics 예측을 위한 기계 학습 모델

FEBRUARY 2019

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Simeon Varbanov

Machine Learning Models for Live Migration Metrics Prediction

Live Migration Metrics 예측을 위한 기계 학습 모델

지도교수 Bernhard Egger

이 논문을 공학석사 학위논문으로 제출함

2018년 10 월

서울대학교 대학원

컴퓨터 공학부

사이먼

사이먼의 공학석사 학위논문을 인준함

2018 년 12 월

Abstract

Live migration of Virtual Machines (VMs) is an important technique in today's data centers. In existing data center management frameworks, complex algorithms are used to determine when, where, and to which host a migration of a VM is to be performed. However, very little attention is paid to the selection of the right migration technique depending on which the migration performance can vary greatly. This performance fluctuation is caused by the different live migration algorithms, the different workloads that each VM is executing, and the state of the destination and the source host. Choosing the right migration technique is a crucial task that has to be made quickly and precisely. Therefore, a performance model is the best and the right candidate for such a task.

In this thesis, we propose various machine learning models for predicting live migration metrics of virtual machines. We predict seven different metrics for twelve distinct migration algorithms. Our models achieve a much higher accuracy compared to existing work. For each target metric and algorithm, an input feature evaluation is conducted and a strictly specific model is generated, leading to 84 different trained machine learning models. These models can easily be integrated into a live migration framework. Using the target metric predictions for each migration algorithm, a framework can easily choose the right migration algorithm, which can lead to downtime and total migration time reduction and less service-level agreement violations.

Keywords: machine learning, live migration, virtualization, VM

Student Number: 2017-28499

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

In present days, virtualization is an important technology used by cloud service providers allowing them to better utilize their hardware resources. Virtualization allows better data utilization and provides an isolation between tenants that use the same physical resource. Live migration is a key technology in virtualization, allowing efficient client management of virtualized resources in data centers. As a result, data centers can migrate VMs between physical servers transparently without causing a service interruption for the user. Companies like Google use live migration technology in their data centers for migrating millions of VMs each month [30, 7]. Other cloud resource management systems adopt live migration for different purposes [8, 27, 31, 36]. Due to the absence of good and practical instructions, most of the live migrations are performed using a fixed live migration algorithm called pre-copy [4]. This migration algorithm works well for moderately loaded VMs, but often suffers from a long migration time and a high resource consumption. Therefore, alternative algorithms and optimization techniques have been proposed [9, 10, 11, 13, 14, 18, 20, 22, 33, 36] that perform better for specific metrics depending on the workload running inside the VM and the state of the data center. With these different live migration algorithms being available, guidelines to select the right migration algorithm that gives the best performance for a specific metric have been proposed [16, 25, 34]. In [12] a machine learning approach using support vector regression (SVR) has been proposed as a solution of selecting the right technique. SVR performs well, but we believe it can be outperformed. Again using a machine learning approach, but this time using neural networks, the prediction accuracy of our models is better than SVR in almost all target metrics. In this work, we present a number of models, specifically made for each algorithm and target metric, that outperform the current state-of-the-art SVR approach [12] and greatly improve the prediction accuracy. These models can be plugged into a VM migration framework, that can use the target metrics predictions to easily decide which migration algorithm will cause the lowest number of service level agreements violations. Another advantage of our approach is that our work uses the graphics processing unit (GPU) instead of the central processing unit (CPU) as in [12] where the training process is much slower.

The remainder of this thesis is organized as follows. Chapter 2 provides the necessary background on virtualization, live migration, and artificial neural networks. Chapter 3 introduces related work to this thesis. Chapter 4 gives information about design choices and an overview of the work. Chapter 5 gives information about the implementation of the different types of networks tried. Chapter 6 presents the different evaluation methods and compares the results with previous work. Chapter 7 gives a presentation and comparison of our models and the SVR with bagging model. Chapter 8 concludes this thesis and includes information about related and future work.

Chapter 2

Background

2.1 Virtualization

Nowadays virtualization is an important technology used in cloud computing environments. Live migration helps system administrators to better manage workloads by transforming traditional computing to make it more scalable. There are seven primary types of virtualization: storage, hardware, network, administrative, application, server and operating system virtualization. In this work, our main focus is on operating system virtualization.

Hardware Virtualization is the most common form of virtualization and is a long-established technology in cloud computing environments that gives the system administrators abilities to manage their resources, to perform server consolidation or load balancing, and to increase the system availability. It is a server virtualization technology that virtualizes hardware components so that the server can run different operating systems handled by multiple users on a single computer at the same time as shown in Figure 2.1.

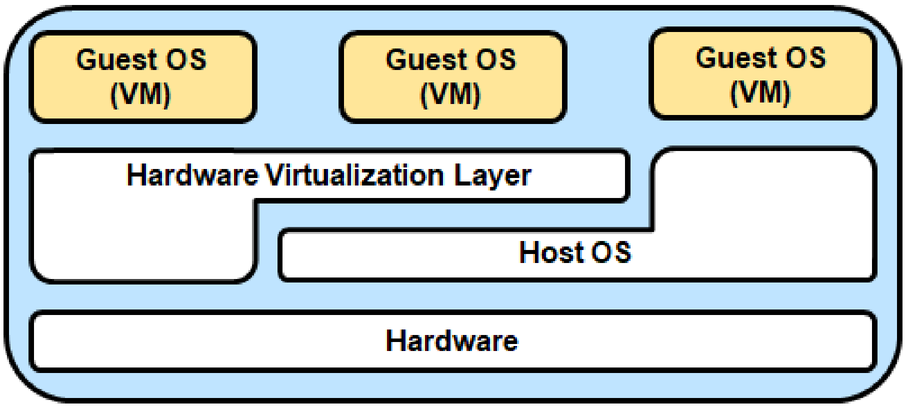


Figure 2.1: Hardware virtualization

From a VM point of view, the VM is the only one having control over the (virtualized) hardware, while from a physical host prospective each VM is just an application process. The processes in the virtualized OS environment are isolated and their interactions with the underlying hardware and OS instance are monitored. These qualities of hardware virtualization has attracted considerable interest in recent years, especially from data centers and cluster computing communities. In our work, we explore a further benefit allowed by virtualization, that of live OS migration.

2.2 Live Migration

Live migration is the process of transferring a live virtual machine from one physical host to another without disrupting its normal operation. Live migration enables the porting of virtual machines and is carried out in a systematic manner to ensure minimal operational downtime. Live migration is a key selling point for the state-of-the-art virtualization technologies, making virtualization of machines an even more appealing technology for the industry. It allows administrators to consolidate system load, perform maintenance tasks, and relocate cluster-wide resources quickly and with minimal downtime. First of all, migration of the whole OS together with its applications as one unit is saving many of the difficulties that we may face in process-level migration. Secondly, migrating the entire virtual machine means that the memory state of the virtual machine can be transferred as a chunk in a consistent and efficient way. This way we can migrate a running guest without making its clients reconnect, an impossible task for application-level migrations. Lastly, live migration of virtual machines allows the user or operator not to worry about application interruption or loss of data during migration. Because of the advantages that live migration provides, both user and operator benefit. The user is having an interrupt-free usage of the system and the operator can expect a smooth transition from the host to the destination virtual machine.

Depending on the time when the state of the virtual machine is transferred, there are two main approaches to live migration. The first one is called pre-copy and can be observed when the state is transferred before execution is switched from the source to the destination host. The second live migration approach is the opposite: the state is transferred after execution, also called post-copy. In this work, we will discuss in depth these two types of live migration and we will see more different types of migration, which we will call hybrid migration approaches. The hybrid approaches vary depending on many factors explained later, in Chapter 2.4.3.

Choosing the right type of live migration is an important task. If the wrong type of migration is selected the total migration time can be prolonged, the user experience can be slowed down or even interrupted. In this work, we propose an automatic selection of the best migration technique based on pre-profiling of the source and destination machine. This approach will predict the best migration technique.

2.3 SLA and SLO

As cloud computing is becoming more popular, an agreement between the service provider and client is an important aspect. The requirements from the client side can have many variations and some of them may not be able to be fulfilled. Therefore, a balanced agreement has to be made. Such an agreement is the Service Level Agreement (SLA). In SLAs between a service provider and a customer, a Service Level Objective (SLO) is a key element. SLOs are agreed as a means of measuring the performance of the Service Provider and are outlined as a way of avoiding disputes between the two parties based on a misunderstanding. While SLA is the entire agreement that specifies what service is to be provided, how it is supported, times, locations, costs, performance, and responsibilities of the parties involved, SLOs are specific measurable characteristics of a SLA such as availability, throughput, frequency, response time, or quality.

2.4 Live Migration Techniques

Migration of a running VM transfers the execution of a VM from one host to another. The execution context includes the entire volatile state of a VM: the state of the virtual CPUs (registers), the state of the attached virtualized hardware devices and the data stored in the VM's RAM. During intra-datacenter migration the permanent storage does not need to be moved since is typically provided by network-attached storage (NAS), see Figure 2.2. The largest volatile component of a VM is the memory, which can easily reach several gigabytes. Each live migration consists of the following steps: 1) dirty page transfer, 2) suspending of the host machine, 3) transfer of remaining pages 4) resume work at destination machine. As explained before depending when step two is performed, live migration can be classified into two approaches: pre-copy and

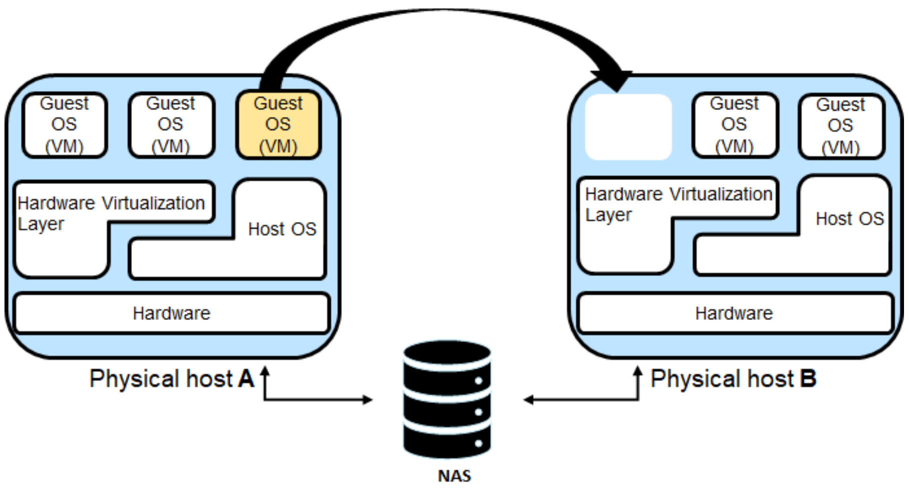


Figure 2.2: Live Migration

post-copy migration. In the pre-copy approach, we first copy the memory pages while the VM keeps running on the source machine, while in post-copy, we first stop the migrating VM on the source machine, restart it on the destination, and then transfer the memory pages.

2.4.1 Pre-copy (PRE)

Pre-copy as shown in Figure 2.3 is an iterative type of migration [28], this is due to the fact that the first stage of pre-copy migration is using an iterative approach. In the first stage, all pages are copied while the VM keeps running on the source. If a memory page that has already been copied to the destination host is modified, it is re-transmitted again in one of the following iterations. The time required for this stage is determined by the page dirty rate of the VM and the stop-and-copy threshold. The stop-and-copy threshold defines when the number of dirty pages is low enough to terminate that stage and proceed with stopping the source host VM and copying the remaining pages. This termination can happen if $dirty\ pages < threshold$, but also if $dirty\ rate > bandwidth_{max}$, where $dirty\ rate = \frac{dirty\ pages}{duration}$. In the first case, the stop-and-copy threshold is set so that the expected downtime is sufficiently short, where $expected\ downtime = \frac{remaining\ dirty\ pages \times page\ size}{bandwith}$ [33]. In the second case, we are stopping the dirty page transfer because the amount of memory dirtied is higher than the network transfer rate; this condition would prolong pre-copy migration infinitely. When the previous stage page transfer terminates, the stop-and-copy phase starts. The time when this transition is complete is not trivial since there is a trade-off between total migration and downtime. If the second stage starts too soon, more data must be sent over the network while the VM is down, which leads to a longer downtime. If stopped too late, the time for repeatedly copying dirtied pages is wasted which diminishes the use of pre-copy. After transferring the CPU state and the remaining dirty pages, the VM is resumed on the destination host.

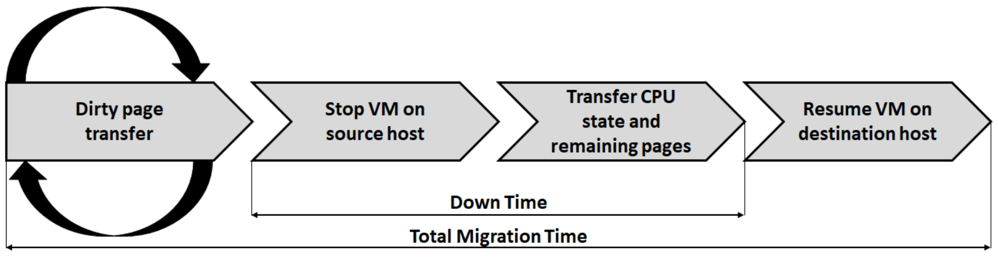


Figure 2.3: Pre-copy migration flow.

2.4.2 Post-copy (POST)

In post-copy, the step in which the memory pages are transferred is moved behind the stop-and-copy (see Figure 2.4). As a consequence, the host VM is stopped at the beginning of the migration. A minimal processor state is copied to the destination host and the work is immediately resumed on the destination node. The remaining pages are fetched from the source while the VM is running on the destination host. The main advantage of this technique is that each memory page is transferred at most once, this way the duplication overhead observed in pre-copy is avoided.

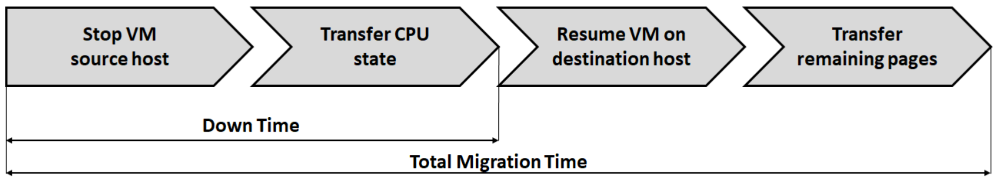


Figure 2.4: Post-copy migration flow.

2.4.3 Hybrid Migration Techniques

The main bottleneck of live migration is the transfer of the memory pages. To alleviate this problem there are two options. First, reduce the dirty page rate or, second, reduce the amount of data to be sent over the network. In order for this to be achieved, there are two main practices: slowing down the CPU of the host machine and compressing the VM memory that is transferred.

Delta Compression (DLTC)

Even with the constant improvement of network connections, the transfer of memory pages over the network is several times slower than the random-access memory (RAM) or disc access [33]. If the page dirty rate is higher than the network throughput, the migration (downtime) can be long. Therefore, in order to shorten the migration (downtime), the page throughput needs to be increased. This can be achieved by compressing the memory pages before the transfer. Delta compression uses XBZRLE (Xor Binary Zero Run-Length-Encoding) to compress the VM's memory pages and thus reduce the total live-migration time. On the sender side, XBZRLE is implemented as a compact delta encoding of page updates, retrieving the old page content from a Least Recently Used (LRU) cache. The receiving side uses the existing page content and XBZRLE to decode the new page content. This technique requires additional memory on the source host to store the memory pages for future delta computation [12].

Data Compression (DTC)

Delta compression is another technique that employs a standard data compression algorithm to compress the memory pages before transmitting them over the network [12]. This method can significantly increase the CPU utilization and therefore may not be a good option for hosts with high CPU utilization.

CPU Throttling (THR)

In this approach, to enforce convergence of the pre-copy process, the speed of the virtual CPU of the VM is deliberately reduced in order to reduce the page dirty rate. This technique typically incurs a significant performance degradation in the VM which may violate SLOs.

2.5 Live Migration Performance Metrics

In this work we are predicting the metrics proposed in [12] where six target metrics were suggested and additionally add network throughput as a metric which we consider useful for evaluating migration performance.
Total migration time (TT) Denotes the elapsed time between the start of a migration and its completion.
Downtime (DT) Represents the time duration of the stop-and-copy phase, the phase during which the VM is not available.

Total traffic (TD) The total amount of transferred data from the source to the destination machine.

Throughput (THRU) The rate of data sent from the host source to the destination.

Performance degradation (PERF) The relative performance degradation during live migration in terms of executed instructions per second (IPS).

Host CPU utilization (CPU) CPU load during migration on the source host.

Host Memory utilization (MEM) Denotes the amount of memory used by VM on the source host.

These metrics were chosen as important, because total time, total traffic, throughput, host CPU utilization, and host memory utilization are metrics of interest to data center operators in order to estimate the required resources for live migration. On the other hand, downtime and performance degradation may affect SLAs and the quality of service (QoS) experienced by the user.

2.6 Artificial Neural Networks

The core part of this work is the generated prediction models. These models are used to predict the aforementioned migration metrics using features collected during profiling of the VM. This prediction is a core part of the framework in [12], since based on these predicted values the framework selects the suitable migration technique based on SLA requirements. In this work, we tried three different neural networks types and compared the result of the best performing type with the prediction results from [12].

Artificial neural networks are a set of algorithms designed with similar intentions as the human brain. Their purpose is to recognize patterns in given data. Based on the provided data the neural networks can be classified as supervised and unsupervised. In supervised neural networks, the data and the expected answer are provided, while in unsupervised models only the input data is given and the answer has to be guessed. Each of these two types can be separated into two other subtypes depending on what is the purpose of the model. If the model is classifying the input into different categories then the model is called a classification model. If the model is predicting or approximating an answer then the model is called a regression model. Every neural network has layers, depending on the number of layers a network can be classified as deep or not.

2.6.1 Feedforward Neural Network (FNN)

A feedforward neural network is the simplest type of a neural network as shown in Figure 2.5. As any other neural network, this one also has an input layer with a size the number of input features x and an output layer with a size the number of desired output predictions y. As the name suggests, the flow of the data in this type of network goes only forward. That means the information flows through the function being evaluated from x, through the intermediate computations, and finally to the output y. There are no feedback connections in which outputs of the model are fed back. When feed-forward neural networks are extended to include feedback relationships, they are called recurrent neural networks. The layers between the input and the output are called hidden layers. In a feedforward neural network, the number of hidden layers is one. If the number of hidden layers is more that one then the network is called a Deep Neural Network. In theory, a feedforward network with a single layer is sufficient to represent any function, but in practice, the layer may be infeasibly large and may fail to learn and generalize correctly. In that case, we need deeper neural networks to learn and generalize the problem correctly, so the prediction can become more accurate.

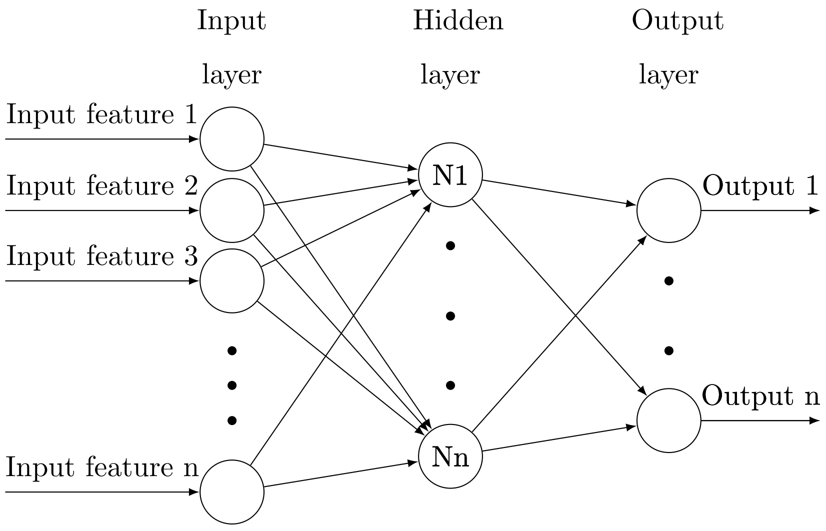


Figure 2.5: Feedforward neural network structure.

2.6.2 Deep Neural Network (DNN)

A definition for a deep neural network is a neural network which has more than one hidden layer (Figure 2.6). Deep neural networks are purposed for more complicated and nonlinear problems. DNNs can be regarded as a subtype of FNNs with more layers since the flow of the information also goes in one direction. DNNs can be more difficult to train, because they have an infinite number of variations. In this thesis, the networks we trained have different characteristics: number of input features, depth and width (number of layers and number of neurons), number of training epochs and more. Each DNN model also implements backpropagation. Backpropagation is an algorithm for supervised learning using gradient descent. Given a deep neural network and an error function, the backpropagation method can calculate the gradient of the error function with respect to the neural network's weights. Backpropagation works in a similar way as the delta rule for a deep neural network. To calculate the

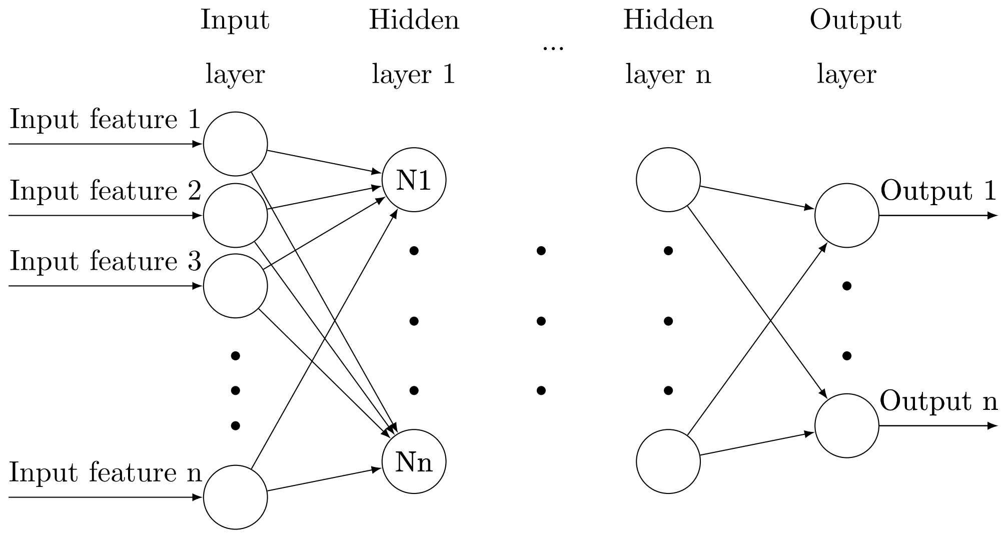


Figure 2.6: Deep neural network structure.

backpropagation the following three things are required:

1. Labeled dataset (with both the input and the output) denoted as (x_i, y_i) where x_i is the input and y_i is desired output. The set with size N samples is denoted as $X = \{(x_1, y_1), ..., (x_N, y_N)\}$

2. A fully connected DNN with weights w and biases b for each layer l. Where w_{ij}^k is the weight between the node j in layer l_k and node i in layer l_{k-1} and bias b_i^k, the bias for node i in layer l_k, which are collectively denoted as θ.

3. An error function, $E(X, \theta)$ which is defining the error between the expected output y_i and the predicted output $\hat{y_i}$ of the DNN on input x_i, for the set of pairs $(x_i, y_i)\ \epsilon X$ and the particular value of the parameters θ.

Training a neural network with gradient descent requires the calculation of the gradient of the error function $E(X, \theta)$ with respect to the weights w_{ij}^k and biases

b_i^k. Then, according to the learning rate α, each iteration of gradient descent updates the weights and biases according to:

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X,\theta)}{\partial \theta},$$

where θ^t denotes the parameters of the neural network at iteration t in gradient descent.

2.6.3 Convolution Neural Network (CNN)

Convolutional Neural Networks are known for their ability to recognize patterns in images. Their architecture makes the implicit assumption that the input is an image (or just any 2D argument of data). However, in this work, we feed our CNN with a numerical input and expect a continuous number prediction. Computers are not able to recognize shapes and patterns; they read the images as pixels that are arranged as a matrix (height $\times$ width $\times$ depth). Normally the layers of a CNN consist of Convolutional, Pooling, Activation and Fully-Connected layers. A common order of the layers is:

$$\text{Convolutional} \rightarrow \text{Activation} \rightarrow \text{Pooling} \rightarrow \text{Convolutional} \rightarrow \text{Activation} \rightarrow \text{Pooling} \rightarrow \text{Fully-Connected}.$$

The convolutional layer has filters that are used to detect a feature or a pattern in the image. Filters usually have smaller dimensions and are expressed as matrices. The filter is sliding (convolving) across the picture and at each place a dot product is computed. Different filters are convolving for different features on the input data. As a result, a set of activation maps are given as an output. The output dimension of the convolutional layer is calculated in the following way:

$$O = \frac{(W-K+2P)}{S} + 1,$$

where O is the output height/length, W is the input height/length, K is the filter dimension, P is the padding, and S is the stride of the filter. After the convolutional layer, an activation layer follows. The purpose of this layer is to nonlinearly transform the input from the previous layer and pass it to the next layer. The most widely used activation function is Rectified Linear Unit (ReLU). This function converts all the negative inputs to zero and this way the neuron does not activate with negative inputs. The activation layer is typically followed by a pooling layer. The purpose of this layer is to reduce the number of parameters and computation in the network. Because of the pooling layer, the network spatial size is reduced and the chances of overfitting are reduced. There are two types of pooling layers: average and maximum pooling (Figure 2.7). Max-pooling picks the maximum value from every neighborhood and average pooling computes the average of every neighborhood. The size of the neighbourhood can vary, but the most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples same as in Figure 2.7.

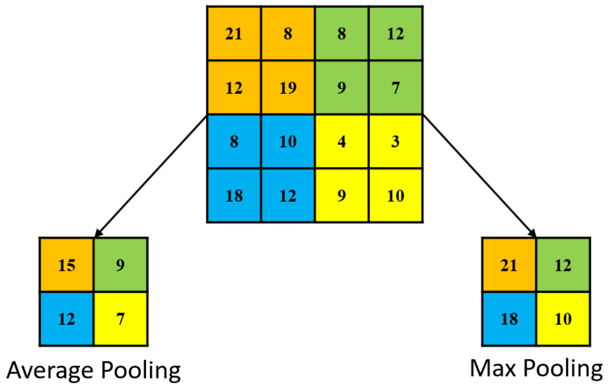


Figure 2.7: Pooling layer types

At the end a fully connected layer is needed to flatten the data and put it into a vector, then based on each value of this vector a prediction is made.

Chapter 3

Related Work

Machine learning is a good and powerful tool that can predict and solve complex problems by using past examples. Many researchers have used machine learning for solving problems in data centers or making quick decisions in a time-critical task. Such a time-sensitive and important decision can be the decision of the right migration technique. In this chapter, we explore some of the previously proposed models, observe what their advantages and disadvantages are, and compare them to our proposed approach.

Modeling live migration performance accurately has been a research topic by many researchers in the past. In [1] a simulation-based live migration modeling approach is proposed. In that work, the model has only two target metrics total time and downtime, also due to missing important input parameters the prediction accuracy is low. In a similar manner in [21] an online performance prediction is done. Additionally, to total time, downtime and total traffic in [21] a power consumption of a migration is predicted. In this work as well important input parameters are missing, leading to a not good prediction accuracy.

The authors of [24] propose more comprehensive model and reveals problems of twelve existing models [1, 2, 5, 18, 19, 21, 23, 26, 37, 38, 39, 40]. However, just as in the other works in [24] the predicted metrics are limited to total time, downtime and total traffic and the prediction accuracy is still not high.

In the current state of the art work for machine learning live migration metric prediction [12], three regression models were proposed - liner regression, support vector regression, and support vector regression with bootstrap aggregation also known as bagging. All three regression models use the sci-kit learn toolkit and were trained and tested using 10-fold cross-validation.

In [12] linear regression does not achieve good results due to the complex correlation that exists in migration data. A linear approach cannot capture the complexities that such data has and therefore fails. On the other hand, SVR and SVR with bagging gave more accurate predictions; this is because they can capture more of the correlation that this complex data has. SVR with bagging is showing best results and is outperforming the normal SVR. This is not surprising as it is commonly known that bagging outperforms single models [3].

Comparing to suggested work in [12] and their SVR with bagging model, we not only achieve a better prediction accuracy, but also add more prediction target metrics and, most importantly, seven more migration algorithm techniques. Along with the five migration algorithms (Pre-copy, Post-copy, CPU Throttling, Delta and Data compression) predicted in [12], we additionally predict seven more hybrid migration algorithms listed in Table 3.1. This way our work is more than doubling the possible migration options. Another advantage of our work is also the additional throughput target metric that we included. Throughput was added as possible useful target metric in case of future bandwidth prediction.

Migration Algorithm	Description
DLTC POST	Delta compression with post-copy.
DLTC DTC	Delta and data compression.
THR POST	CPU throttling with post-copy.
THR DTC	CPU throttling with data compression.
THR DLTC	CPU throttling with delta compression.
THR POST DLTC	CPU throttling with post-copy and delta compression.
THR DLTC DTC	CPU throttling with delta and data compression.

Table 3.1: Hybrid migration algorithms

The SVR models were trained and tested using a dataset with 40,000 samples. This dataset was produced by the migration of four identical machines. On the other hand, our models were trained with more than 130,000 samples on much more heterogeneous data. For the making of this dataset, four different types of machines and twelve machines in total were used. Because of this, we are sure our models are having much bigger chances to perform better than [12].

Besides the advantage that our models have because of the much more diverse and bigger dataset, we also have more features than the state of the art work. SVR with bagging model uses 20 input features, on the other hand in our deep neural network models the feature number vary from 49 and up to 72 features. The number of input variables is different because we found that not all features are important and give benefit for the targeted metric that will be predicted. More about this can be seen in the feature importance analysis section in Overview and Design.

With the stated above, we believe our models are much better trained, more adjustable to different data variation and offer much more migration techniques possibility compared to the current state of art.

Chapter 4

Overview and Design

This chapter gives an overview and discusses the design choices that influenced our models.

TensorFlow For this thesis, we decided to use the TensorFlow library version 1.5 together with python 3.4 [35]. TensorFlow is an open-source library for machine learning. In [12] the models were made using scikit-learn and this choice of machine learning library had some disadvantages. First of all, TensorFlow is a low-level library that allowed us to build machine learning models using a set of simple operations like add and matmul, while the scikit-learn is a higher-level library that includes already an implementation of several machine learning algorithms, so a model can be defined as in [12] just in a few lines. As a result, TensorFlow is more difficult to use but allows for more customization and agility than scikit-learn. The second advantage of using TensorFlow over scikit-learn is the ability to do automatic differentiation. TensorFlow's idea is that you build a computation graph for doing any computation and you always end up working on that graph. The nodes on the graph are the different operations and the edges are the tensors. This way of visualizing a problem allows TensorFlow to provide automatic differentiation to perform backpropagation easily. TensorFlow also allows us to use GPU or CPU, which is not possible in scikit-learn, where CPU is the only available option. As a result model training in TensorFlow is much faster.

In conclusion, scikit-learn is good for building standard machine learning models quickly and train classifiers like Logistic Regression or SVR but is not the right choice for our much more advanced and more customized neural network models.

Feature Analysis and Selection The collected dataset of VM migration has 90 input features (Appendix A), but not all of them are useful for our prediction. Some of these features decrease the accuracy of the models and confuse them. Therefore we built a feature importance classifier program using a scikit-learn library and Extra Trees Classifier [6]. We ran the importance classifier for each target metric and for each migration algorithm. This gave us 84 tables with the weight that each feature has for the specific target metric and specific migration algorithm. We combined the data of these tables in 7 graphs - one for each target metric. Each graph has 12 categories one for each migration algorithm type. These graphs can be seen in Appendix B. We can see the tendency that depending on the target metric and migration algorithm some features are more preferable to have more importance than others (Appendix B). Such an examples can be `max bandwidth` for Total time, `downtime limit` when trying to predict Downtime metric and `postcopy start time` for all the postcopy or postcopy like migration algorithms.

Feature Scaling Because each feature is measured in different scales their numerical value can vary. If we use raw data and do not scale, then some of the features will always dominate over the rest and our accuracy will not be good. Hence, we have to scale the input features in a way so they become easier to work with but also preserve their information and ratio. For that purpose, we used the scikit-learn library and the StandartScaler [32]. StandartScaler was also the scaler chosen for [12]. StandartScaler assumes that the data is normally distributed within each feature and will scale them such that the distribution is now centered around 0, with a standard deviation of 1. The mean and standard deviation are calculated for the feature. After that, the same feature is scaled using the following formula:

$$\text{StandartScaler} = \frac{x - \mu}{\sigma}$$

$$\text{with mean: } \mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

$$\text{and standard deviation: } \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

This way we transform the data so the variance is unitary and that the mean of the series is 0.

Cross Validation K-fold cross-validations is a useful technique for assessing model performance. Cross-validation is used as a way to determine if a change in the learning/test data is giving a positive or negative impact. In our work, we use 10-fold cross validation across the whole dataset. That means we divide our dataset into 10 chunks, train on nine and test on the remaining one, print the result and clean the TensorFlow graph. If we do not clean the graph after each testing, the model will keep the values and labels from the previous iteration, which will lead to retraining the model, a behavior which we do not want. We do this iterative train, test and clean the graph ten times, until all the 10 splits became at least ones a test set (Figure 4.1). A similar approach is used in [12]. Because of cross-validation, we were able to measure how good the model fits, both for accuracy and variance.

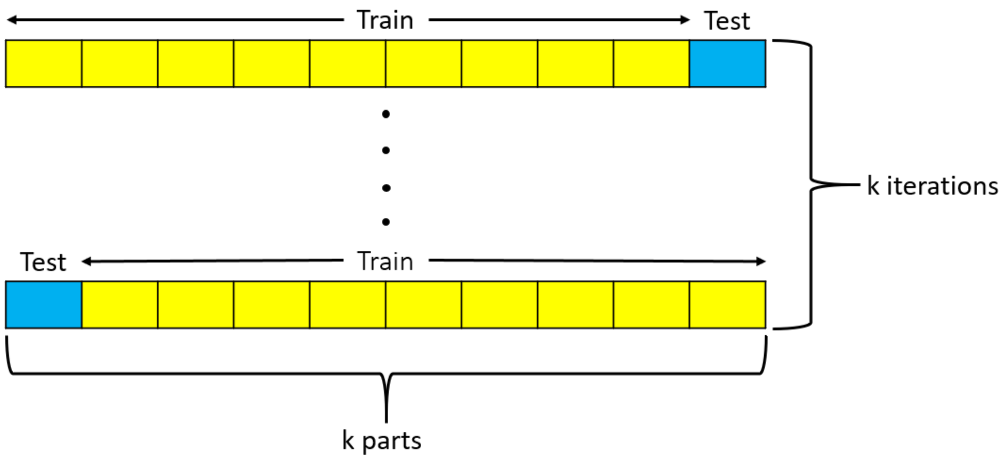


Figure 4.1: 10-fold cross validation

Model Selection In order to decide what kind of neural network we need, a short analysis of the data input and the expected result had to be done. Our dataset is labeled, that means we have the input features and also the expected result of the migration. For that reason, we were looking for a supervised neural network. Second, our input is continuous numbers and our output is a prediction of a continuous number, therefore we needed a regression type of model. As a 'rule of thumb' when starting a neural network design from scratch, it has to start simple and build up complexity and see what improves the network model. Following this approach, our first model choice was a simple feed-forward network with one layer. This type of model did not give us good results, just as the linear regression in [12] the network was not capable of finding the correlation between the inputs features and the output. Therefore this type of network was eliminated as a possible solution. The next step was to make a more complicated network structure. Such an alternative was a DNN. DNNs have many variations, so we spent a lot of time testing different hyperparameters in order to find the optimal structure. However, a DNN with simple feed-forwarding was not minimizing the error and additional improvement was necessary, hence we included a backpropagation functionality to our DNN model. Due to the backpropagation, our model improved more than 7 times for pre-copy total time prediction. The Mean Absolute Error for the network without backpropagation was approximately 35 seconds and when using back propagation, the Mean Absolute Error became 5 seconds. In order to expand the work of this thesis we decide to try another feedforward type of network and for this purpose, we choose CNN. CNNs are usually used for image input data and their primary purpose of convolution in the case of a CNN is to extract features from the input image. The convolution in CNNs preserve the spatial relationship between pixels. In our case, we used a CNN with an input of 90 features by putting them in a matrix of 9×10 in a similar manner as if we are using 9×10 pixels image. The results of the CNN were not better than a DNN with backpropagation. For pre-copy migration with total time as target metric, the CNN gave the best Geometric Mean Absolute Error (GMAE) prediction of 10 seconds while the DNN less than 1.4 seconds. We interpret this bad results because CNN was not able to take advantage of the spatial relationship as it does with picture input. Hence, we find in this particular case a DNN with backpropagation to be the best type of model.

Outliers removal In order to compare both the SVR and the DNN models, we had to use the same dataset for training and testing. Therefore we had to adjust our dataset of 131,957 samples to be same as the one the SVR with bagging has used. Luckily all the features used in [12] were present in our dataset so such an operation is just a matter of parsing. Because our data is consisting of 12 different migration techniques, the data that is matching with the same migration techniques as in [12] was 50 534 samples. When the new data was fed to the SVR model from [12] the prediction was unreasonably bad. We received a geometric mean absolute error of approximately 40 seconds for the pre-copy total time, while our DNN model was having a geometric mean absolute error of under 1.4 seconds (which is in the usual prediction range). After further investigation of the problem, we discovered that the SVR was not able to deal with the little number outliers in our dataset, these outliers were confusing the SVR model. This is the reason why we had to remove the outliers of each migration technique, by removing 0.01% of the miss-fitting data. This sample cleaning cost us in total 792 samples or 0.6% of the total data (Table 4.1). This problem with SVR was another indication that DNN models are more flexible and can adjust better.

Migration Algorithm Type *	Samples Before Refining	Samples After Refining
PRE	9694	9650
POST	10241	10215
DTC	7867	7810
DLTC	10545	10490
POST DLTC	10865	10814
DLTC DTC	10473	10372
THR	12187	12118
THR POST	12098	12057
THR DTC	10583	10485
THR DLTC	12455	12391
THR DLTC POST	12612	12547
THR DLTC DTC	12330	12216
Total	131957	131165

Table 4.1: Number of samples before and after refining

Chapter 5

Implementation

In this chapter, we describe the different parameters that we have selected to build our models and the reason behind these decisions.
This chapter is divided into two subsections. In the first part, we explore details regarding the hyperparameters and the structure of our Deep Neural Network model. In the second subsection, a discussion about our Convolutional Neural Network design will be given.

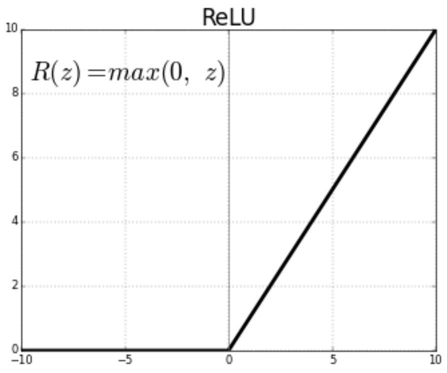
5.1 Deep Neural Network design

Since we have 84 different models we will not talk about each model individually, but we will discuss the one we find relevant. For all models we used similar hyperparameters except the number of layers, number of input features, number of epochs and batch size. Each one of these three parameters was selected individually for each model after extensive experiments. We found that number of layers, number of features, number of epochs and batch size can greatly

Target Metric	Number of layers	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7
Total Time	7	900	900	900	400	400	400	400
Downtime	5	900	900	850	850	850		
Total Traffic	5	900	900	450	450	200		
Throughput	5	90	90	90	45	20		
Performance	7	90	90	90	90	90	90	90
CPU	5	450	450	450	450	450		
CPU Post-copy	5	250	250	250	450	450		
CPU - DLTC	5	250	250	250	450	450		
CPU - DTC	5	100	150	150	450	450		
MEM	7	180	180	180	90	90	90	90

Table 5.1: DNN models structure

improve the prediction accuracy if they are selected correctly.
Because we were building our models from scratch, we started with a small number of layers and neurons and slowly increased them until no further prediction improvement was possible and our models started overfitting. For models like total time and downtime we found that it was difficult to get a good prediction with just a few network layers because of the complexity that these two metrics have. As a result, deeper and wider networks were used. Table 5.1 shows for each target metric the number of layers and neurons per layer. For every hidden layer in all the models, we have used Rectified Linear Units (ReLu) as the activation function. This activation function was selected because first of all ReLu is not a liner, therefore combinations of ReLu are also nonlinear (Figure 5.2). In ReLu if x is negative or 0 the output is also 0, in the rest of the cases ReLu gives an output x (see Figure 5.2). Another plus for ReLu compared



$$f(x) = \left\{ \begin{array}{ccc} 0 & for & x < 0 \\ x & for & x \geq x \end{array} \right\}$$

Table 5.2: Relu

to the other activation functions is that ReLu is more efficient than tanh and sigmoid because it involves simpler mathematical operations [17]. Because of all mentioned above reasons we choose ReLu for our models instead of any other activation function.
In Chapter 4 we analyzed the features and their importance for each target metric and migration type. Not every feature is useful and gives information to our model, therefore we started excluding the features with the lowest importance. For example for Pre-copy migration technique total time, the best prediction accuracy was achieved when only the top 49 features were in use, while for the other migration techniques we needed 65 input features. This was due to the fact that these 49 features were concentrating the most essential information that our model required in order to produce an accurate prediction. For the rest of the models and how many numbers of input features refer to Table 5.3.
Another parameter that was strictly individual for each model was the number of epochs. One complete epoch is when the entire dataset is passed forward and backward through the network once. As the number of epochs increases, the number of times our weights being updated also increase. If we have not enough

Target Metric	PRE	POST	DTC	DLTC	DLTC POST	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
Total Time	49	65	65	65	65	65	65	65	65	65	65	65
Downtime	65	65	63	65	57	65	65	65	65	65	65	65
Total Traffic	57	63	56	57	57	57	57	65	57	70	57	60
Throughput	60	57	67	57	57	57	57	57	57	57	57	66
Performance	61	57	72	65	56	65	65	57	58	67	59	65
CPU	58	57	65	57	57	57	57	57	57	57	57	57
Memory	66	72	65	65	65	65	65	66	65	65	65	65

Table 5.3: DNN models number of input features

epochs our model may not train well and underfit if we use more than the right number epochs we may overfit which will lead to good results while training but our model will not be capable to perform well on new data. Therefore the right number of epochs has to be selected for each model. Unfortunately, there is no rule what is the right number of epochs. Epochs are different for the different data sets and the diversity of the data is a determining factor. Therefore we ran our models in many different numbers of epochs and analyzed their results. In the end, the number of epochs that produced the best prediction accuracy was selected. Table 5.4 shows the different number of epochs for each model.
Batch size is the number of training examples in one iteration. The batch size can be one of the following three options: batch mode, mini-batch mode and stochastic. Batch mode is when the batch size is equal to the total dataset, the number of iteration and epochs is the same. A stochastic model is when the batch size is equal to one. As a result, the gradient and the neural network parameters are updated after each sample. Mini-batch mode size when the size of the batch is greater than one and less than the total data size. For our models, we tried all three options and mini-batch mode performed best. We tried

Target Metric	PRE	POST	DTC	DLTC	DLTC POST	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
Total Time	135	165	90	135	105	75	90	150	90	75	100	105
Downtime	90	90	100	75	75	90	75	100	150	165	90	75
Total Traffic	305	350	320	350	350	240	300	350	350	350	350	300
Throughput	135	420	150	420	90	150	150	75	150	420	240	320
Performance	60	75	150	150	60	105	75	60	150	75	75	60
CPU	45	60	120	45	105	90	90	135	120	90	90	120
Memory	120	120	105	150	105	120	200	105	120	150	105	150

Table 5.4: DNN models number of epochs

splitting the dataset into 5, 10, 15 and 20 batches but 10 batch split gave the best performance for all models. Since our dataset is having a different number of samples for each migration technique splitting them into 10 splits gave us different batch size for each migration algorithm. The size of the batch for each migration algorithm can be seen in the Table 5.5. In machine learning, a cost

Target Metric	PRE	POST	DTC	DLTC	DLTC POST	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
Batch size	868	919	702	944	973	933	1090	1085	943	1115	1129	1099

Table 5.5: DNN models batch size

function is used to estimate how good a model performs. Since our main goal was to improve prediction accuracy than the state-of-the-art approach[12] we decided that the best option for a cost function is the one that can give a good overview of our model accuracy. Therefore we decided to try two cost functions mean absolute error and mean absolute relative error. The mean absolute relative error gave us good results and in comparison with MAE was performing better. The mean absolute error was giving us with approximately 10% worse prediction than the MARE. Hence, we decided the formula for our cost function to be MARE and it looks like this:

$$\overline{cost} = \frac{|PV - TV|}{TV}$$

where TV is the true value and PV is the predicted value.
To make our predictions as correct as possible we need to minimize the cost function on each iteration. This minimization is done by changes in the parameters of our models, which is done by an optimizer. For our models, we decided to use the Adam optimizer, one of the most popular optimization methods nowadays[15][29]. Adam is a popular algorithm and commonly used in the field of deep learning because it is fast and gives good results. In our case, Adam optimization gave the best performance among the other options that TensorFlow has - AdadeltaOptimizer, GradientDescentOptimizer, and AdagradOptimizer. Table 5.6 shows an example of the different prediction accuracy that each one of these optimizers produced for the Total Time target metric, which is measured in milliseconds.

	Adam	Adadeltar	GradientDescent	Adagrad
MAE (ms)	5225	33745	35890	34613

Table 5.6: Optimizer comparison

The learning rate is an important hyperparameter, which controls the weight adjustment of our network with respect to the loss gradient. If this rate is low the time we need to converge will be longer, while if the learning rate is too large we may miss the global optimum and even diverge. A typical learning rate ranges between 0.1 and 0.001 and typically most models are using a learning rate around 0.01. This is also our case, we have tried many different learning rates from 0.1 until 0.001 but the one giving us the best result was 0.01, therefore we decided to stay with it.

5.2 Convolutional Neural Network design

Convolutional neural networks are a much more complicated type of networks. They are primarily used for image classification taking the image as a 3D matrix where the first 2 dimensions are the size of the image and the third one is the number of channels. In colored images, we have 3 channels red, green and blue. Since our input is not an image, but just a matrix of numerical values, we will have only one channel. Hence, we placed our input in a 2D matrix with size 9×10. Since we start our CNN from scratch we decided to build up complexity with time. We started with two convolutional layers with stride 1 and padding with extra 0. Stride controls how the filter convolves around our input, we choose 1 because this way our filter will convolve over similar data more frequently and hoped that this will increase the accuracy. Also, the stride should be set in a way so that the output volume is a whole number. We decided to use padding with extra zero because this way we can maintain the spatial dimensions and better preserve the information around the edges. After each convolutional layer, we applied a ReLu activation function and a max pooling layer. Pooling layers are often considered as part of the convolutional layers and are used to reduce the spatial dimension and select the strongest activation from the grid. After using convolutional layers to extract the spatial features of our data, we apply a fully connected layer to for the final output prediction. This way we flattened our output. After the first run of our two-layer convolutional network, we received results GMAE 10 seconds - for Pre-copy total time. This was seven times higher GMAE than our DNN model. Hence, we decided that this is because we have too little convolutional layers. We decided to try other variations with more layers (up to seven convolutional layers), without max-pooling and even with fully connected layers with bigger depth and width. All these experiments did not give positive results.

Initially, we were not having high hopes that CNN will work because we knew that CNN, as explained in Chapter 2, is made for pattern recognition and CNN works well with data that has spatial information. Our data does not contain such information, because the order of the input features does not give any information to the model. We decided to focus more on our DNN models and try to get more improvement from there instead of CNN.

Chapter 6

Evaluation metrics

In this chapter, we discuss the evaluation methods we used to compare our work and [12]. Also, we suggest the use of other evaluation metrics in the future since they will give a better idea of how accurate the prediction is.

6.1 Geometric Mean Absolute Error (GMAE)

One of the evaluation metrics that [12] uses is the Geometric Mean Absolute Error. The formula calculating this metric is as follows:

$$AE_1 = \|TV_1 - PV_1\|$$

$$...$$

$$AE_n = \|TV_n - PV_n\|$$

$$GMAE = \left(\prod_{i=1}^{n} AE_i\right)^{1/n} \equiv \sqrt[n]{AE_1 \cdot AE_2 \cdot ... \cdot AE_n}$$

Figure 6.1: Geometric mean absolute error formula

where PV is the predicted value for the specific sample, TV is the true value for the specific sample, AE is the absolute error for the specific sample, and n is the total number of samples. The geometric mean indicates the central tendency or typical value of a set of numbers by using the product of their values. GMAE does not give a good understanding of the model accuracy, therefore we believe is not a good metric for accuracy evaluation.

6.2 Geometric Mean Relative Error (GMRE)

The other evaluation metric used in [12] is the Geometric Mean Relative Error. The formula calculating this metric is as follows:

$$RE_1 = \left\| 1 - \frac{PV_1}{TV_1} \right\| \equiv \left\| \frac{TV_1 - PV_1}{TV_1} \right\|$$

$$\ldots$$

$$RE_n = \left\| 1 - \frac{PV_n}{TV_n} \right\| \equiv \left\| \frac{TV_n - PV_n}{TV_n} \right\|$$

$$GMRE = \left(\prod_{i=1}^{n} RE_i \right)^{1/n} \equiv \sqrt[n]{RE_1 \cdot RE_2 \cdot \ldots \cdot RE_n}$$

Figure 6.2: Geometric mean relative error formula

where PV is the predicted value for the specific sample, TV is the true value for the specific sample, RE is the relative error for the specific sample, and n is the total number of samples. As mentioned before in the GMAE section, the geometric mean can give an overall idea but can mislead about the accuracy that a model has, since the geometric mean is always smaller than the arithmetic mean and it gives more a model prediction tendency than a concrete value. Therefore we believe that the geometric mean is not suitable and the arithmetic mean should be used for a closeup comparison of the SVR and DNN results.

6.3 Mean Absolute Error (MAE)

One of the metrics which we believe is giving a better idea of the model's accuracy is the Mean Absolute Error. This is the simplest way to represent the model accuracy of our predictions since is just the average error. The lower the MAE, the better the prediction accuracy. The formula calculating this metric is as follows.

$$AE_1 = \|TV_1 - PV_1\|$$
$$\dots$$
$$AE_n = \|TV_n - PV_n\|$$

$$MAE = \frac{\sum\limits_{i=1}^{n} AE_i}{n}$$

Figure 6.3: Geometric mean absolute error formula

where PV is the predicted value for the specific sample, TV is the true value for the specific sample, AE is the absolute error for the specific sample, and n is the total number of samples.

6.4 Weighted Absolute Percentage Error (WAPE)

Another metric with which we can measure model's performance is the Weighted Absolute Percentage Error. The WAPE can help us judge the goodness of fit. The lower the WAPE, the better the prediction accuracy. The formula calculating this metric is as follows.

$$AE_1 = \|TV_1 - PV_1\|$$

$$\ldots$$

$$AE_n = \|TV_n - PV_n\|$$

$$WAPE = \frac{\sum_{i=1}^{n} AE_i}{\sum_{i=1}^{n} TV_i} \times 100$$

Figure 6.4: Weighted absolute percentage error formula

where PV is the predicted value for the specific sample, TV is the true value for the specific sample, AE is the absolute error for the specific sample, and n is the total number of samples. In the next chapter, we will present the results for SVR and DNN in both MAE and WAPE. A comparison between these two models using the MAE and WAPE will give a good idea of why our model is much better than the SVR with bagging.

Chapter 7

Results

We have conducted a wide range of experiments to evaluate our DNN models. In this chapter, we show the potential of our models and compare them with the state-of-the-art SVR with bagging [12] model.
This chapter first starts with presenting the results from our DNN models, followed by the results of SVR with bagging. Finally, an overall accuracy and training time comparison for SVR and DNN is given. The results are generated using 10-fold cross-validation: the dataset is first split into 10 equal-sized subsets. Each set serves as the test set once, while the union of the remaining nine forms the training dataset. The reported values represent the average of the 10 evaluations.

7.1 Deep Neural Network

We present the prediction accuracy of 84 different DNN models for twelve live-migration algorithms and the seven target metrics. The results are shown in

Table 7.1. As explained in Chapter 6 the MAE represents the average divergence of the predicted value to the actual value in absolute units of the metric (ms, MB or %). On the other hand, WAPE displays the prediction inaccuracy of our models in percentage. For both measuring methods, the smaller the value, the better the prediction accuracy.

Migration Algorithm	Target Metric	MAE	WAPE
PRE	Total Time (ms)	5,055	13%
	Downtime (ms)	309	17%
	Total Traffic (MB)	212.0	9%
	Throughput (Mbps)	31.2	6%
	Performance (%)	20.9	18%
	CPU (%)	1.5	20%
	Memory (MB)	1.8	111%
POST	Total Time (ms)	5,134	15%
	Downtime (ms)	236	17%
	Total Traffic (MB)	195.0	10%
	Throughput (Mbps)	29.1	5%
	Performance (%)	54.9	59%
	CPU (%)	2.1	34%
	Memory (MB)	1.8	124%
DTC	Total Time (ms)	14,875	32%
	Downtime (ms)	244	20%
	Total Traffic (MB)	283.3	22%
	Throughput (Mbps)	35.1	15%
	Performance (%)	24.1	19%
	CPU (%)	39.8	13%
	Memory (MB)	2.4	84%
DLTC	Total Time (ms)	5,073	14%
	Downtime (ms)	321	19%
	Total Traffic (MB)	194.4	9%
	Throughput (Mbps)	29.4	5%
	Performance (%)	19.8	17%
	CPU (%)	1.7	21%
	Memory (MB)	21.6	16%

Migration Algorithm	Target Metric	MAE	WAPE
POST DLTC	Total Time (ms)	5,608	16%
	Downtime (ms)	250	20%
	Total Traffic (MB)	179.6	9%
	Throughput (Mbps)	29.8	6%
	Performance (%)	53.9	61%
	CPU (%)	2.1	32%
	Memory (MB)	34.8	24%
DLTC DTC	Total Time (ms)	10,561	21%
	Downtime (ms)	350	27%
	Total Traffic (MB)	220.3	15%
	Throughput (Mbps)	35.9	14%
	Performance (%)	18.9	16%
	CPU (%)	53.3	21%
	Memory (MB)	27.9	19%
THR	Total Time (ms)	7,252	13%
	Downtime (ms)	346	19%
	Total Traffic (MB)	262.7	9%
	Throughput (Mbps)	26.7	5%
	Performance (%)	17.3	16%
	CPU (%)	1.6	22%
	Memory (MB)	2.1	131%
THR POST	Total Time (ms)	6,643	14%
	Downtime (ms)	278	19%
	Total Traffic (MB)	234.9	9%
	Throughput (Mbps)	27.4	5%
	Performance (%)	50.9	59%
	CPU (%)	2.0	33%
	Memory (MB)	2.1	133%
THR DTC	Total Time (ms)	20,404	29%
	Downtime (ms)	282	20%
	Total Traffic (MB)	334.1	17%
	Throughput (Mbps)	35.6	15%
	Performance (%)	18.0	16%
	CPU (%)	48.8	17%
	Memory (MB)	2.8	94%

Migration Algorithm	Target Metric	MAE	WAPE
THR DLTC	Total Time (ms)	6,455	13%
	Downtime (ms)	373	23%
	Total Traffic (MB)	226.7	9%
	Throughput (Mbps)	26.8	5%
	Performance (%)	22.0	20%
	CPU (%)	1.6	22%
	Memory (MB)	20.0	14%
THR DLTC POST	Total Time (ms)	6,547	14%
	Downtime (ms)	290	22%
	Total Traffic (MB)	216.7	9%
	Throughput (Mbps)	27.6	5%
	Performance (%)	49.4	60%
	CPU (%)	2.1	31%
	Memory (MB)	32.2	23%
THR DLTC DTC	Total Time (ms)	11,361	18%
	Downtime (ms)	379	28%
	Total Traffic (MB)	282.3	14%
	Throughput (Mbps)	33.2	12%
	Performance (%)	16.6	15%
	CPU (%)	41.7	19%
	Memory (MB)	26.5	19%

Table 7.1: Accuracy of the different DNN algorithms for the twelve live migration algorithms and the seven target metrics.

7.2 SVR with bagging

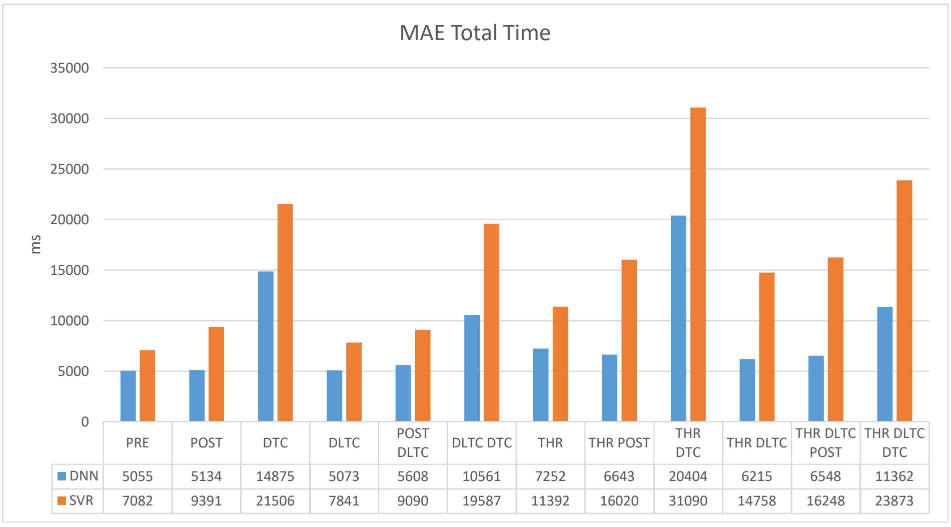
The current state-of-the-art in the live migration performance modeling is the work presented by Jo *et al.* in 2017 [12]. Using the same technique on our dataset as in [12], SVR with bagging give us the results show in Table 7.2. Jo *et al*'s work only make predictions for pre-copy, CPU-throttling, delta compression, data compression and post-copy migration algorithms and the six target metrics total time, downtime, total traffic, performance, CPU and memory.

Migration Algorithm	Target Metric	MAE	WAPE
PRE	Total Time (ms)	7,082	19%
	Downtime (ms)	967	52%
	Total Traffic (MB)	295.0	13%
	Performance (%)	21.1	18%
	CPU (%)	2.3	31%
	Memory (MB)	2.4	142%
POST	Total Time (ms)	9,391	27%
	Downtime (ms)	1,063	76%
	Total Traffic (MB)	456.8	22%
	Performance (%)	61.7	67%
	CPU (%)	5.3	89%
	Memory (MB)	2.3	155%
DTC	Total Time (ms)	21,506	46%
	Downtime (ms)	657	54%
	Total Traffic (MB)	332.4	26%
	Performance (%)	25.4	20%
	CPU (%)	96.7	32%
	Memory (MB)	2.7	94%
DLTC	Total Time (ms)	7,841	21%
	Downtime (ms)	955	58%
	Total Traffic (MB)	352.7	16%
	Performance (%)	19.1	17%
	CPU (%)	2.5	31%
	Memory (MB)	103.2	76%
THR	Total Time (ms)	11,392	20%
	Downtime (ms)	938	51%
	Total Traffic (MB)	490.0	16%
	Performance (%)	16.9	15%
	CPU (%)	2.4	33%
	Memory (MB)	2.6	161%

Table 7.2: Accuracy of the SVR with bagging algorithm for the five live migration algorithms and the six target metrics.

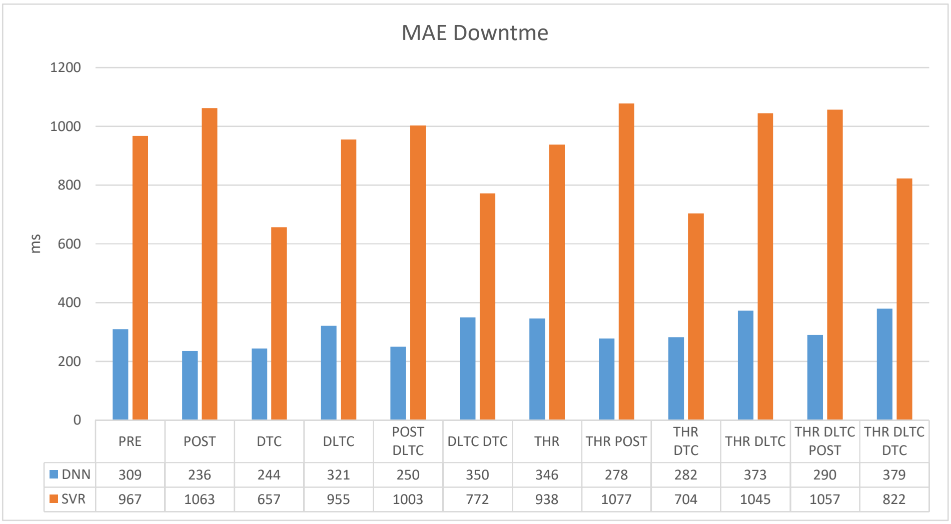
7.3 DNN vs. SVR comparison

The presented in [12] is limited to five migration algorithms and six metrics. However, for comparison purposes we additionally used SVR to predict the seven migration algorithm techniques proposed by our DNN models. From Figure 7.1 to Figure 7.7 we compare the DNN and the SVR models for the specific target metric and specific migration technique using MAE metric. From Figure 7.8 to Figure 7.14 we compare the DNN and the SVR models for the specific target metric and specific migration technique using WAPE metric. The lower the MAE and the WAPE, the better the prediction accuracy.



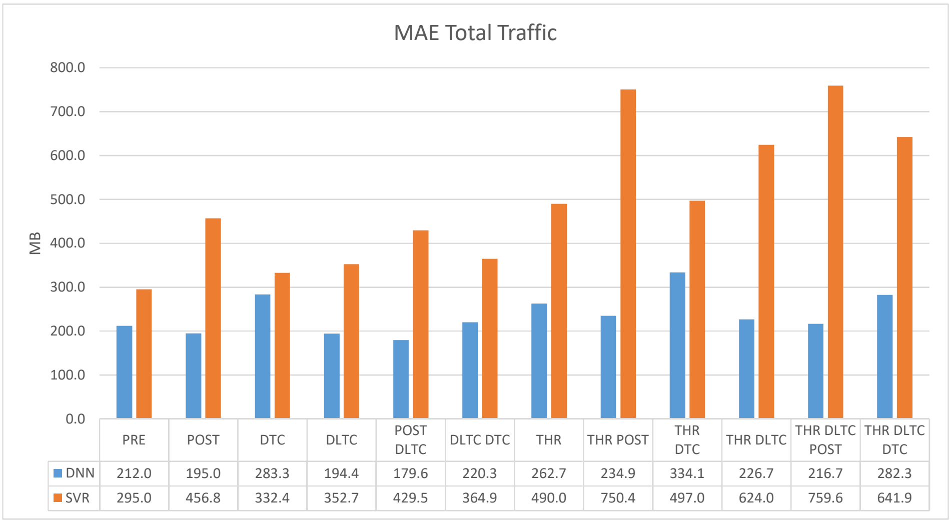
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	5055	5134	14875	5073	5608	10561	7252	6643	20404	6215	6548	11362
🟧 SVR	7082	9391	21506	7841	9090	19587	11392	16020	31090	14758	16248	23873

Figure 7.1: Total time - Mean absolute error



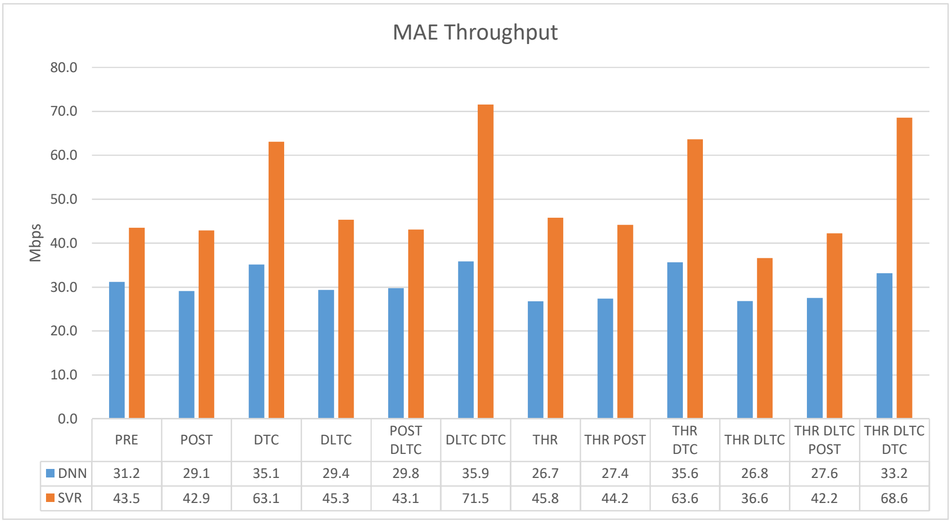
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
■ DNN	309	236	244	321	250	350	346	278	282	373	290	379
■ SVR	967	1063	657	955	1003	772	938	1077	704	1045	1057	822

Figure 7.2: Downtime - Mean absolute error



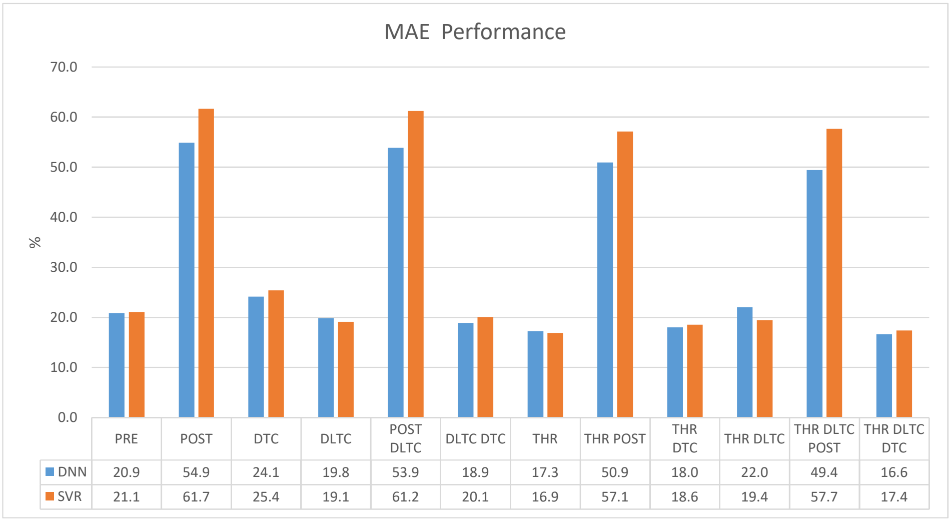
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
■ DNN	212.0	195.0	283.3	194.4	179.6	220.3	262.7	234.9	334.1	226.7	216.7	282.3
■ SVR	295.0	456.8	332.4	352.7	429.5	364.9	490.0	750.4	497.0	624.0	759.6	641.9

Figure 7.3: Total traffic - Mean absolute error



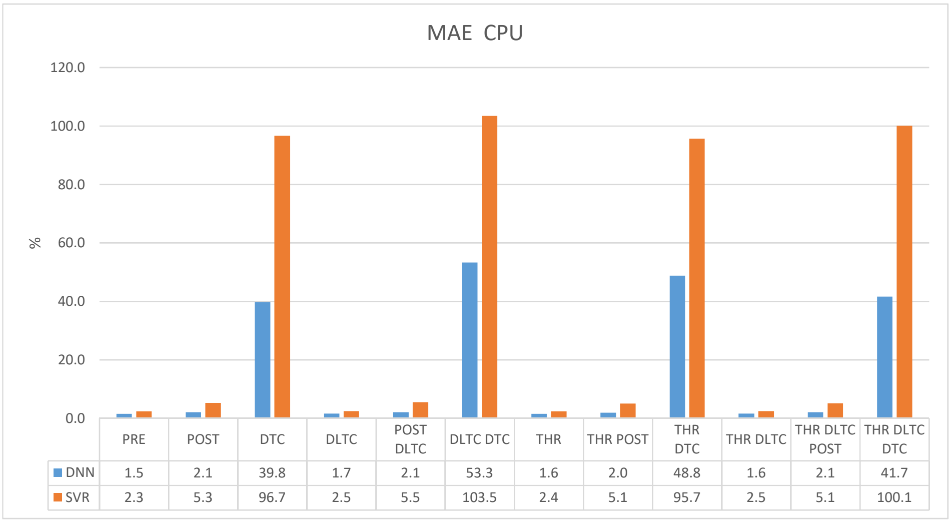
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	31.2	29.1	35.1	29.4	29.8	35.9	26.7	27.4	35.6	26.8	27.6	33.2
🟧 SVR	43.5	42.9	63.1	45.3	43.1	71.5	45.8	44.2	63.6	36.6	42.2	68.6

Figure 7.4: Throughput - Mean absolute error



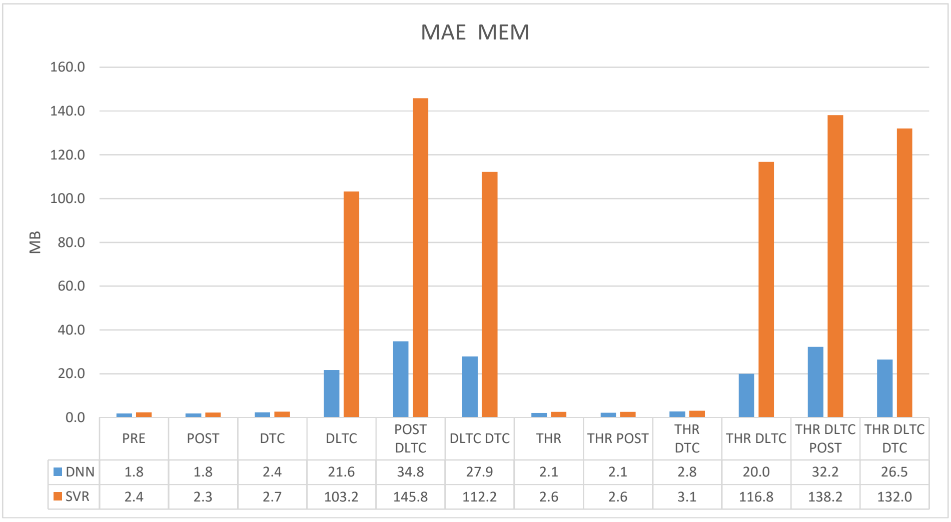
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	20.9	54.9	24.1	19.8	53.9	18.9	17.3	50.9	18.0	22.0	49.4	16.6
🟧 SVR	21.1	61.7	25.4	19.1	61.2	20.1	16.9	57.1	18.6	19.4	57.7	17.4

Figure 7.5: Performance degradation - Mean absolute error



	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	1.5	2.1	39.8	1.7	2.1	53.3	1.6	2.0	48.8	1.6	2.1	41.7
🟧 SVR	2.3	5.3	96.7	2.5	5.5	103.5	2.4	5.1	95.7	2.5	5.1	100.1

Figure 7.6: CPU utilization - Mean absolute error



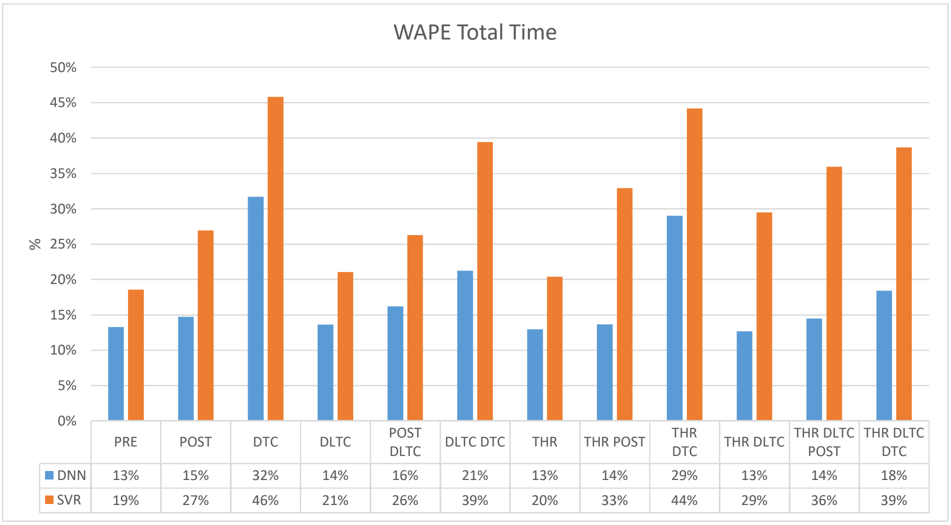
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	1.8	1.8	2.4	21.6	34.8	27.9	2.1	2.1	2.8	20.0	32.2	26.5
🟧 SVR	2.4	2.3	2.7	103.2	145.8	112.2	2.6	2.6	3.1	116.8	138.2	132.0

Figure 7.7: Memory utilization - Mean absolute error

As it can be seen from the figures above there are a few cases where the MAE for a specific migration algorithm is much bigger compared to the other migration algorithms. Such an example can be seen in Figure 7.1 where both DNN and SVR are having an increased MAE. We believe that this big MAE, for both SVR and DNN, is due to the dataset that the models are trained with. We believe our dataset is missing an important feature and therefore the models do not predict well the total time for migration algorithms that use DTC. In Figure 7.6 we can see that the accuracy for models that have DTC are having the highest MAE for both SVR and DNN. This behaviour is because of the migration technique itself. As explained in Chapter 2.4.2 this type of migration can increase significantly the CPU utilization which is difficult to predict. Another value that is distinctive compared the rest is the prediction accuracy of memory utilization for migrations that use DLTC. This is because DLTC needs additional memory, for storing the old pages in order to perform a XBZRLE compression. Due to this both models are having increased MAE. Lastly, we can see that the performance prediction accuracy for migrations that use post-copy is significantly increased, compared to the other migration techniques. This behaviour is due to the post-copy way of migration. As explained in Chapter 2.4.2 in case of post-copy migration initially a minimal processor state is copied to the destination host and then the work is immediately resumed on the destination host. As a result, not all memory pages are transferred on the destination host, therefore in case a page that is not yet transferred from the source to the destination host is required, the destination host should stop its work retrieve that page from the source host and resume work. This process can cause an increase on the performance degradation which can be difficult to predict.

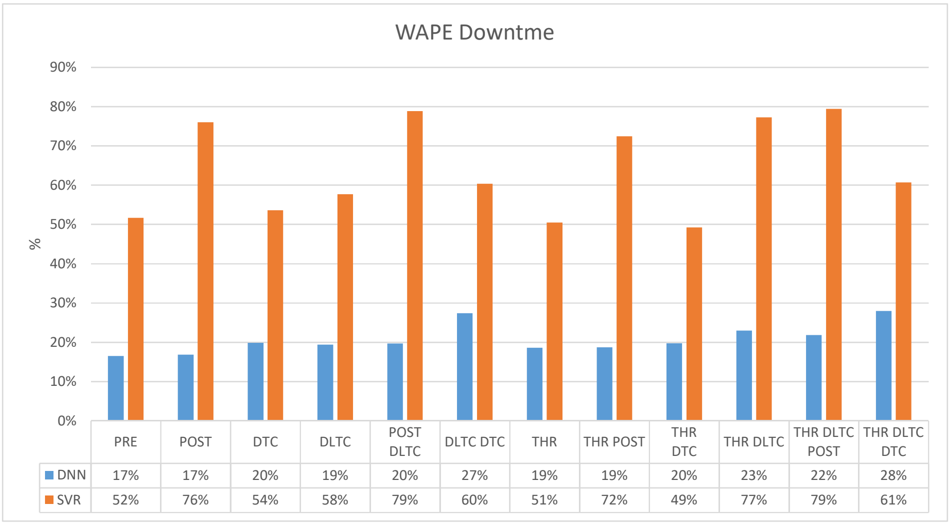
Another result that worth mentioning is the is the fact that our DNN models were able to perform much better in all target metrics except one. As it can be

seen in Figure 7.5 when predicting the performance utilization our DNN models predicted equally good or same as SVR except in the following three cases DLTC, THR and THR DLTC where our DNN prediction is with slightly worse prediction accuracy than the SVR. We assume this is due to the fact how the data for performance utilization is created. The performance utilization label is created as a median value of the whole migration performance utilization, because of this median transformation it is possible that some of the feature correlation be lost.
As explained in Chapter 6.4 WAPE can give us an overall idea of the goodness of the model. The lower the WAPE the better the prediction accuracy. As it can be seen from the WAPE results our model is better than the SVR in all migration metrics except in performance utilization where a few predictions of SVR are slightly better than the predictions of our DNN models.



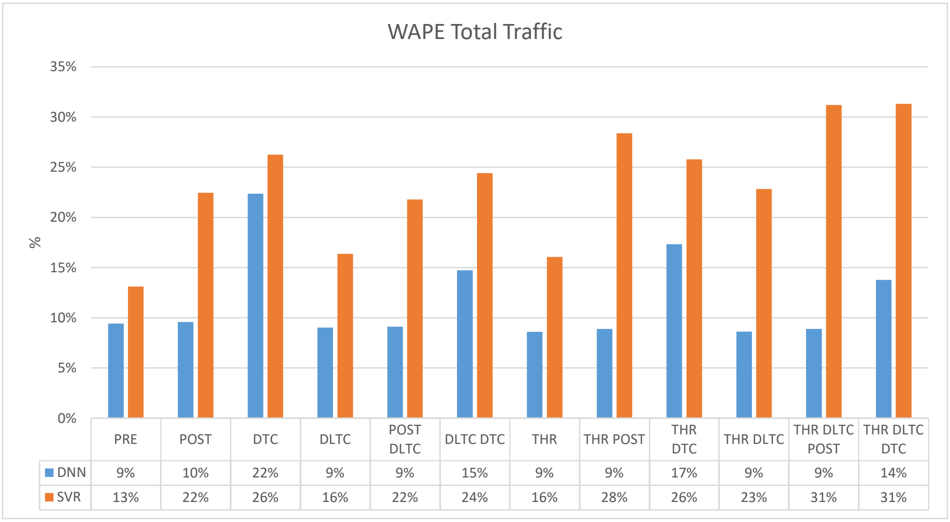
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
■ DNN	13%	15%	32%	14%	16%	21%	13%	14%	29%	13%	14%	18%
■ SVR	19%	27%	46%	21%	26%	39%	20%	33%	44%	29%	36%	39%

Figure 7.8: Total time - WAPE



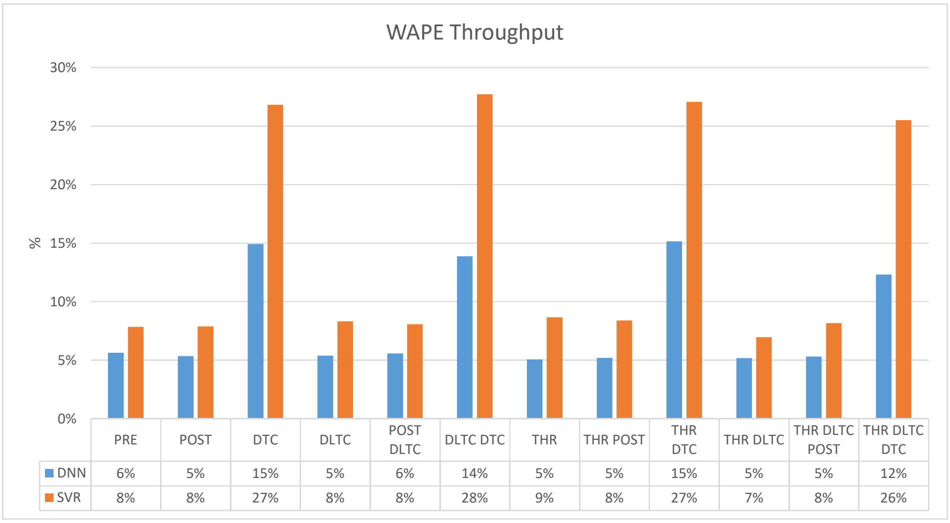
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	17%	17%	20%	19%	20%	27%	19%	19%	20%	23%	22%	28%
🟧 SVR	52%	76%	54%	58%	79%	60%	51%	72%	49%	77%	79%	61%

Figure 7.9: Downtime - WAPE



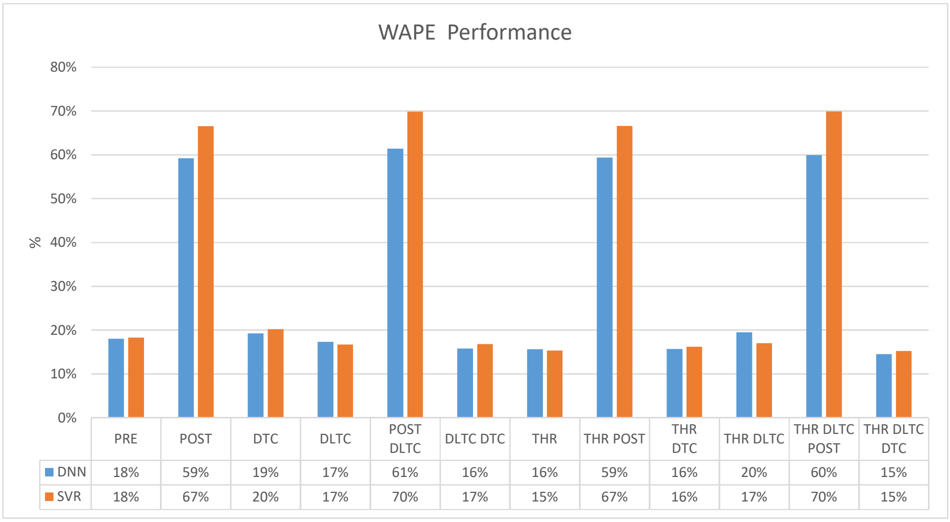
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	9%	10%	22%	9%	9%	15%	9%	9%	17%	9%	9%	14%
🟧 SVR	13%	22%	26%	16%	22%	24%	16%	28%	26%	23%	31%	31%

Figure 7.10: Total traffic - WAPE



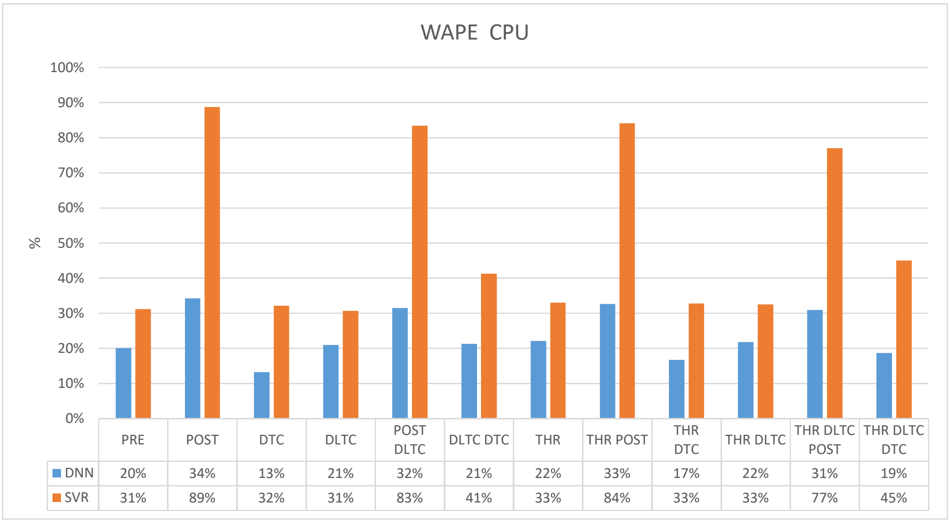
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	6%	5%	15%	5%	6%	14%	5%	5%	15%	5%	5%	12%
🟧 SVR	8%	8%	27%	8%	8%	28%	9%	8%	27%	7%	8%	26%

Figure 7.11: Throughput - WAPE



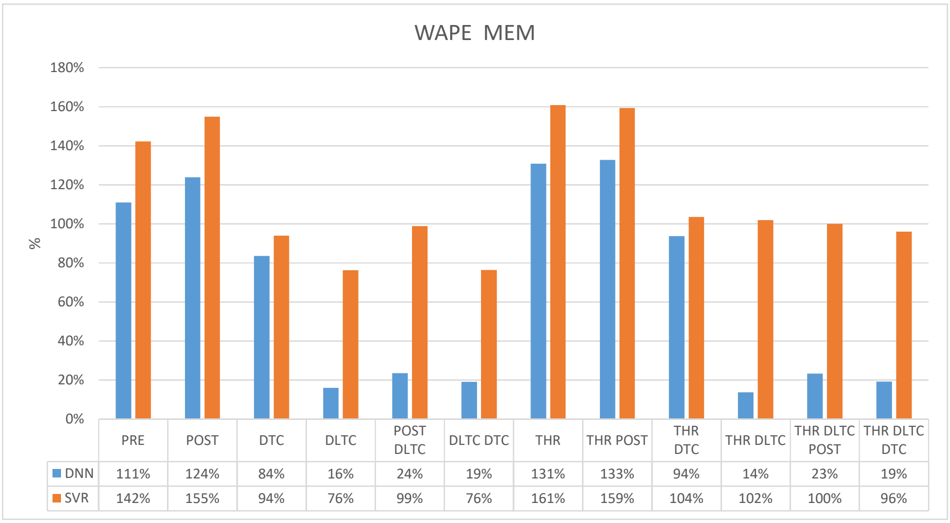
	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	18%	59%	19%	17%	61%	16%	16%	59%	16%	20%	60%	15%
🟧 SVR	18%	67%	20%	17%	70%	17%	15%	67%	16%	17%	70%	15%

Figure 7.12: Performance degradation - WAPE



	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	20%	34%	13%	21%	32%	21%	22%	33%	17%	22%	31%	19%
🟧 SVR	31%	89%	32%	31%	83%	41%	33%	84%	33%	33%	77%	45%

Figure 7.13: CPU utilization - WAPE

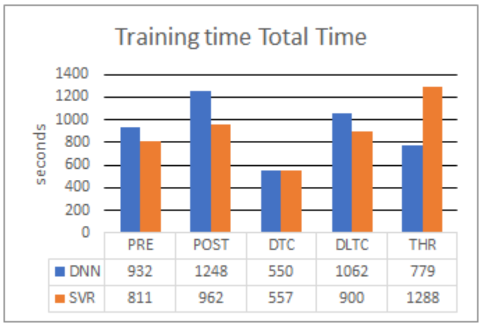


	PRE	POST	DTC	DLTC	POST DLTC	DLTC DTC	THR	THR POST	THR DTC	THR DLTC	THR DLTC POST	THR DLTC DTC
🟦 DNN	111%	124%	84%	16%	24%	19%	131%	133%	94%	14%	23%	19%
🟧 SVR	142%	155%	94%	76%	99%	76%	161%	159%	104%	102%	100%	96%

Figure 7.14: Memory utilization - WAPE

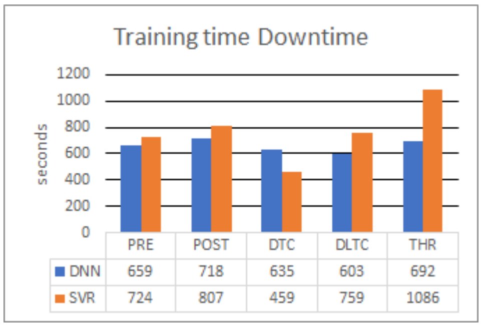
7.4 Overhead

This experiment was done to give an overview idea of how long the training process for each model takes. For this experiment, we used Intel i5-7500 CPU with processor base frequency of 3.40GHz and GPU Nvidia GeForce GTX 750. For the training of the SVR models we were using the CPU, since sckit learn can work only on CPU. On the other hand for the DNN models we used GPU. In Figure 7.15 a training time for each migration algorithm and target metric is presented. In 18 out of the 30 different models DNN is training faster. The reason why DNN is not as fast as the SVR is because of the different number of epochs and different models structure. An example can be total traffic Figure 7.15f where our models require the most number of epochs and train slower than the SVR. The rest of the training times can be seen in 7.15.



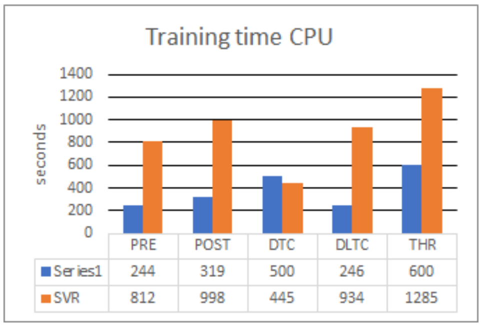
	PRE	POST	DTC	DLTC	THR
■ DNN	932	1248	550	1062	779
■ SVR	811	962	557	900	1288

(a) Total time



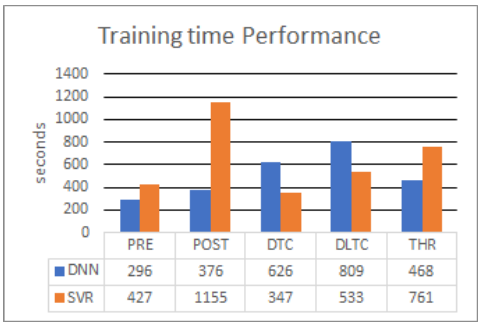
	PRE	POST	DTC	DLTC	THR
■ DNN	659	718	635	603	692
■ SVR	724	807	459	759	1086

(b) Downtime



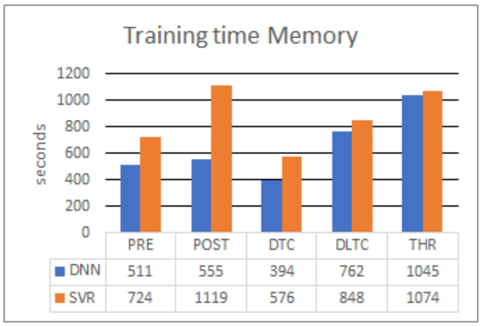
	PRE	POST	DTC	DLTC	THR
■ Series1	244	319	500	246	600
■ SVR	812	998	445	934	1285

(c) CPU



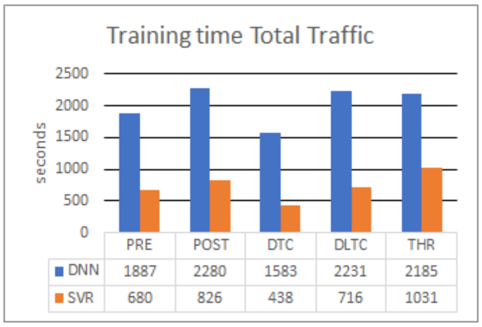
	PRE	POST	DTC	DLTC	THR
■ DNN	296	376	626	809	468
■ SVR	427	1155	347	533	761

(d) Performance



	PRE	POST	DTC	DLTC	THR
DNN	511	555	394	762	1045
SVR	724	1119	576	848	1074

(e) Memory



	PRE	POST	DTC	DLTC	THR
DNN	1887	2280	1583	2231	2185
SVR	680	826	438	716	1031

(f) Total traffic

Figure 7.15: Training time for each migration algorithm and target metric

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this work, we presented 84 deep neural network models that can predict important VM live migration metrics. We discussed the input that these models get and their relevance for each model. The reason and technique for selecting the right hyperparameters were discussed and explained. The proposed models achieve high prediction accuracy for all target metrics and migration algorithms. Compared to the state of the art, our models attain a significant prediction accuracy improvement over almost all the targeted metrics. We believe that if the presented work becomes part of a live migration framework, the proposed models will reduce the number of SLA and SLO violations.

8.2 Future Work

For a future work a model training on more heterogeneous and bigger dataset, that involves more complex migration scenarios can be done. If a creation of new dataset is done, adding timestamps can be useful for training other types of networks like Recurrent Neural Networks (LSTMs).

Appendices

Appendix A

List of Features

#	Name	Description
0	SRC_id	Id of the source node
1	DST_id	Id of the destination node
2	auto-converge	Cpu-throttling migration technique 0:disabled 1:enabled.
3	xbzrle	Delta compression migration technique 0:disabled 1:enabled.
4	compress	Data compressio nmigration technique 0:disabled 1:enabled.
5	postcopy-ram	Post-copy migration technique 0:disabled 1:enabled.
6	cpu-throttle-increment	Throttling step of auto-converge technique.
7	cpu-throttle-initial	Initial throttling level of auto-converge technique.
8	xbzrle_cache_size	Cache size of delta compression technique. Old data is stored in this memory and used later to compute delta.
9	compress-level	Compression level, from 0 to 10. Higher number gives better compression ratio.
10	compress-threads	Number of threads to compress data.
11	decompress-threads	Number of thread to decompress data.

#	Name	Description
12	postcopy_start_time	When to start postcopy ex) 0: start postcopy immediately 1: do precopy for one iteration and enable postcopy in the second iteration.
13	downtime-limit	Requested downtime from user. Iterative precopy only proceed to stop-and-copy phase only the expected downtime is less than the downtime-limit.
14	max-bandwidth	Reverved bandwidth for vm live migration.
15	total_pages	Total pages of the VM.
16	working_set_pages	Total working set pages of the VM.
17	non_working_set_pages	Total non-working set pages of the VM (total_pages = working_set_pages + non_working_set_pages)
18	zero_pages	The number of pages which contents are zero.
19	pdr	Page dirty rate.
20	mwpp	Average number of modified words (1-byte) per a page.
21	wse	Entropy of working set (0.0 ∼1.0). lower entropy gives better compressibility.
22	nwse	Entropy of non working set (0.0 ∼1.0). lower entropy gives better compressibility.
23	DST_cpu_idle	Percentage of time that the destination node CPU or CPUs were idle and the system did not have an outstanding disk I/O request.
24	DST_cpu_io	Percentage of time that the destination node CPU or CPUs were idle during which the system had an outstanding disk I/O request.
25	DST_cpu_system	Percentage of destination node CPU utilization that occurred while executing at the system level (kernel).
26	DST_cpu_user	Percentage of the destination CPU utilization that occurred while executing at the user level (application).
27	DST_io_bread	Total amount of data read from the destination node devices in blocks per second, blocks has size - 512 bytes.
28	DST_io_bwrtn	Total amount of data written to destination node devices in blocks per second.

#	Name	Description
29	DST_io_rtps	Total number of read requests per second issued to physical devices.
30	DST_io_tps	Total number of transfers per second that were issued to physical devices.
31	DST_io_wtps	Total number of write requests per second issued to physical devices.
32	DST_kbmemfree	Amount of free memory available on destination node in kilobytes.
33	DST_kbmemused	Amount of used memory on destination node in kilobytes.
34	DST_memused	Percentage of used memory on destination node.
35	DST_net_manage_ifutil	Utilization percentage of the network interface on destination node.
36	DST_net_manage_rxkb	Total number of kilobytes received per second on the destination node.
37	DST_net_manage_txkb	Total number of kilobytes transmitted per second on destination node.
38	DST_paging_fault	Number of page faults (major + minor) made by the system per second for on destination node.
39	DST_paging_majflt	Number of major faults the system has made per second, those which have required loading a memory page from disk for destination node.
40	DST_paging_pgpgin	Total number of kilobytes the system paged in from disk per second for destination node.
41	DST_paging_pgpgout	Total number of kilobytes the system paged out to disk per second for destination node.
42	DST_processor_cores	Number of processor cores of destination node.
43	DST_processor_speed	Processor clock speed of destination node.
44	DST_processor_threads	Number of processor threads of destination node.
45	DST_ram_size	Ram size of destination node.
46	DST_ram_speed	Ram clock speed of destination node.
47	DST_swap_kbswapfree	Size of free swap memory for the destination node.

#	Name	Description
48	DST_swap_kbswapused	Size of used swap memory for the destination node.
49	DST_swap_swpused	Amount of used swap space in kilobytes for the destination node.
50	SRC_cpu_idle	Percentage of time that the source node CPU or CPUs were idle during which the system had an outstanding disk I/O request.
51	SRC_cpu_io	Percentage of time that the source node CPU or CPUs were idle during which the system had an outstanding disk I/O request.
52	SRC_cpu_system	Percentage of source node CPU utilization that occurred while executing at the system level (kernel).
53	SRC_cpu_user	Percentage of source node CPU utilization that occurred while executing at the user level (application).
54	SRC_io_bread	Total amount of data read from the source node devices in blocks per second, blocks has size - 512 bytes.
55	SRC_io_bwrtn	Total amount of data written to source node devices in blocks per second.
56	SRC_io_rtps	Total number of read requests per second issued to physical devices.
57	SRC_io_tps	Total number of transfers per second that were issued to physical devices.
58	SRC_io_wtps	Total number of write requests per second issued to physical devices.
59	SRC_kbmemfree	Amount of free memory available on source node in kilobytes.
60	SRC_kbmemused	Amount of used memory on source node in kilobytes.
61	SRC_memused	Percentage of used memory on source node.
62	SRC_net_manage_ifutil	Utilization percentage of the network interface on source node.
63	SRC_net_manage_rxkb	Total number of kilobytes received per second on the source node.
64	SRC_net_manage_txkb	Total number of kilobytes transmitted per second on source node.
65	SRC_paging_fault	Number of page faults (major + minor) made by the system per second for on source node.
66	SRC_paging_majflt	Number of major faults the system has made per second, those which have required loading a memory page from disk for source node.

#	Name	Description
67	SRC_paging_pgpgin	Total number of kilobytes the system paged in from disk per second for source node.
68	SRC_paging_pgpgout	Total number of kilobytes the system paged out to disk per second for source node.
69	SRC_processor_cores	Number of processor cores of source node.
70	SRC_processor_speed	Processor clock speed of source node.
71	SRC_processor_threads	Number of processor threads of source node.
72	SRC_ram_size	Ram size of source node.
73	SRC_ram_speed	Ram clock speed of source node
74	SRC_swap_kbswapfree	Size of free swap memory for the source node.
75	SRC_swap_kbswapused	Size of used swap memory for the source node.
76	SRC_swap_swpused	Amount of used swap space in kilobytes for the source node.
77	SRC_vm_cpu_baseline	Source node CPU baseline.
78	SRC_vm_mem_baseline	Source node memory baseline.
79	vm_perf_LLC-load-misses	Average number of last level cache load misses of VM per a second.
80	vm_perf_LLC-loads	Average number of last level cache loads of VM per a second.
81	vm_perf_cache-misses	Average number of cache misses of VM per a second.
82	vm_perf_cache-references	Average number of cache references of VM per a second.
83	vm_perf_cycles	Average number of cpu cycles of VM per a second.
84	vm_perf_instructions	Average number of executed instructions of VM per a second.
85	RPTR	Relative page transfer rate.
86	DLTC_benefit	Expected benefit of delta compression.
87	THR_benefit	Expected benefit of delta compression.
88	ewss	Expected size of wss after compression.
89	enwss	Expected size of nwss after compression.

Table A.1: Features name and description.

Appendix B

Features Importance

Feature Name	AVG TT	AVG DT	AVG TD	AVG THRU	AVG PERF	AVG CPU	AVG MEM
SRC_id	0.8	0.6	0.7	0.8	0.8	0.6	0.8
DST_id	0.8	0.9	0.7	1.0	0.8	1.0	0.9
auto-converge	0.0	0.0	0.0	0.0	0.0	0.0	0.0
xbzrle	0.0	0.0	0.0	0.0	0.0	0.0	0.0
compress	0.0	0.0	0.0	0.0	0.0	0.0	0.0
postcopy-ram	0.0	0.0	0.0	0.0	0.0	0.0	0.0
cpu-throttle-increment	1.7	1.8	1.7	1.8	1.4	1.7	1.7
cpu-throttle-initial	1.8	1.8	1.7	1.8	1.4	1.7	1.6
xbzrle_cache_size	1.7	1.7	1.7	1.7	1.4	1.6	3.8
compress-level	1.6	1.7	1.6	1.8	1.3	1.7	1.6
compress-threads	1.7	1.7	1.6	1.8	1.4	2.2	2.7
decompress-threads	1.6	1.7	1.6	1.7	1.3	1.7	1.6
postcopy_start_time	1.5	2.6	1.5	1.6	1.7	1.8	1.7
downtime-limit	1.8	2.0	1.8	1.9	1.8	1.7	1.7

Feature Name	AVG TT	AVG DT	AVG TD	AVG THRU	AVG PERF	AVG CPU	AVG MEM
max-bandwidth	1.8	1.8	1.7	2.6	1.5	3.8	1.9
total_pages	0.0	0.0	0.0	0.0	0.0	0.0	0.0
working_set_pages	1.8	1.7	1.8	1.7	3.3	1.7	1.9
non_working_set_pages	1.8	1.7	1.7	1.7	3.0	1.7	2.0
zero_pages	1.7	1.7	1.8	1.7	3.5	1.7	2.1
pdr	1.7	1.7	1.8	1.7	2.2	1.7	1.7
mwpp	1.5	1.5	1.6	1.4	1.2	1.4	1.4
wse	1.7	1.8	1.7	1.8	1.8	1.7	1.7
nwse	1.7	1.7	1.8	1.7	1.5	1.6	1.6
DST_cpu_idle	1.7	1.7	1.7	1.7	1.2	1.6	1.5
DST_cpu_io	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DST_cpu_system	1.7	1.7	1.7	1.7	1.2	1.6	1.6
DST_cpu_user	1.7	1.7	1.7	1.7	1.2	1.6	1.5
DST_io_bread	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DST_io_bwrtn	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DST_io_rtps	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DST_io_tps	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DST_io_wtps	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DST_kbmemfree	1.7	1.7	1.8	1.7	1.2	1.6	1.4
DST_kbmemused	1.7	1.7	1.8	1.7	1.2	1.7	1.6
DST_memused	1.7	1.7	1.8	1.7	1.2	1.6	1.5
DST_net_manage_ifutil	1.6	1.6	1.6	1.6	1.2	1.6	1.5
DST_net_manage_rxkb	1.6	1.6	1.6	1.6	1.3	1.6	1.6
DST_net_manage_txkb	1.6	1.6	1.6	1.6	1.2	1.5	1.4
DST_paging_fault	1.6	1.5	1.7	1.5	1.2	1.5	1.6
DST_paging_majflt	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DST_paging_pgpgin	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DST_paging_pgpgout	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Feature Name	AVG TT	AVG DT	AVG TD	AVG THRU	AVG PERF	AVG CPU	AVG MEM
DST_processor_cores	0.5	0.6	0.5	0.7	0.6	0.7	0.7
DST_processor_speed	1.7	1.7	1.6	1.7	1.4	1.7	1.6
DST_processor_threads	0.6	0.7	0.5	0.8	0.6	0.8	0.7
DST_ram_size	0.3	0.4	0.3	0.4	0.4	0.4	0.4
DST_ram_speed	0.8	0.8	0.7	0.9	0.7	0.9	0.9
DST_swap_kbswapfree	1.4	1.4	1.4	1.4	1.0	1.3	1.2
DST_swap_kbswapused	1.4	1.3	1.4	1.3	1.0	1.3	1.3
DST_swap_swpused	1.4	1.3	1.3	1.3	1.0	1.3	1.3
SRC_cpu_idle	1.7	1.7	1.7	1.7	1.3	1.7	1.5
SRC_cpu_io	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_cpu_system	1.7	1.7	1.7	1.7	1.3	1.6	1.6
SRC_cpu_user	1.7	1.7	1.7	1.7	1.4	1.7	1.6
SRC_io_bread	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_io_bwrtn	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_io_rtps	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_io_tps	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_io_wtps	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_kbmemfree	1.7	1.7	1.8	1.7	1.2	1.7	1.6
SRC_kbmemused	1.7	1.7	1.8	1.7	1.3	1.7	1.7
SRC_memused	1.7	1.7	1.7	1.7	1.2	1.7	1.7
SRC_net_manage_ifutil	1.5	1.6	1.6	1.5	1.2	1.5	1.4
SRC_net_manage_rxkb	1.5	1.5	1.6	1.5	1.2	1.5	1.4
SRC_net_manage_txkb	1.6	1.5	1.6	1.6	1.3	1.6	1.5
SRC_paging_fault	1.6	1.6	1.7	1.6	1.2	1.5	1.8
SRC_paging_majflt	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_paging_pgpgin	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_paging_pgpgout	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SRC_processor_cores	0.5	0.4	0.5	0.6	0.4	0.2	0.6

Feature Name	AVG TT	AVG DT	AVG TD	AVG THRU	AVG PERF	AVG CPU	AVG MEM
SRC_processor_speed	1.7	1.9	1.6	1.7	2.0	3.1	1.6
SRC_processor_threads	0.6	0.5	0.5	0.7	0.6	0.2	0.6
SRC_ram_size	0.3	0.3	0.3	0.3	0.3	0.1	0.3
SRC_ram_speed	0.7	0.6	0.6	0.8	0.6	0.5	0.7
SRC_swap_kbswapfree	1.4	1.3	1.4	1.3	1.1	1.3	1.1
SRC_swap_kbswapused	1.3	1.3	1.4	1.3	1.0	1.2	1.2
SRC_swap_swpused	1.4	1.3	1.3	1.3	1.0	1.2	1.2
SRC_vm_cpu_baseline	1.4	1.4	1.5	1.4	3.7	1.5	1.4
SRC_vm_mem_baseline	1.8	1.7	1.7	1.8	3.2	1.6	2.0
vm_perf_LLC-load-misses	1.7	1.7	1.7	1.6	1.5	1.7	1.6
vm_perf_LLC-loads	1.7	1.7	1.8	1.6	1.6	1.6	1.5
vm_perf_cache-misses	1.7	1.7	1.8	1.6	1.8	1.6	1.6
vm_perf_cache-references	1.7	1.7	1.8	1.6	2.3	1.6	1.5
vm_perf_cycles	1.7	1.7	1.8	1.6	3.2	1.6	1.5
vm_perf_instructions	1.7	1.7	1.7	1.6	3.4	1.6	1.5
RPTR	1.7	1.7	1.7	1.7	2.1	1.7	1.9
DLTC_benefit	1.5	1.5	1.6	1.4	1.3	1.5	1.4
THR_benefit	1.5	1.4	1.6	1.4	2.1	1.4	1.4
ewss	1.7	1.7	1.7	1.7	1.9	1.6	1.8
enwss	1.7	1.7	1.8	1.7	1.8	1.6	1.8
TOTAL %	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table B.1: Average percentage importance for each feature, for each target metric.

The following figures present the importance of each feature for each target metric for specific migration technique.

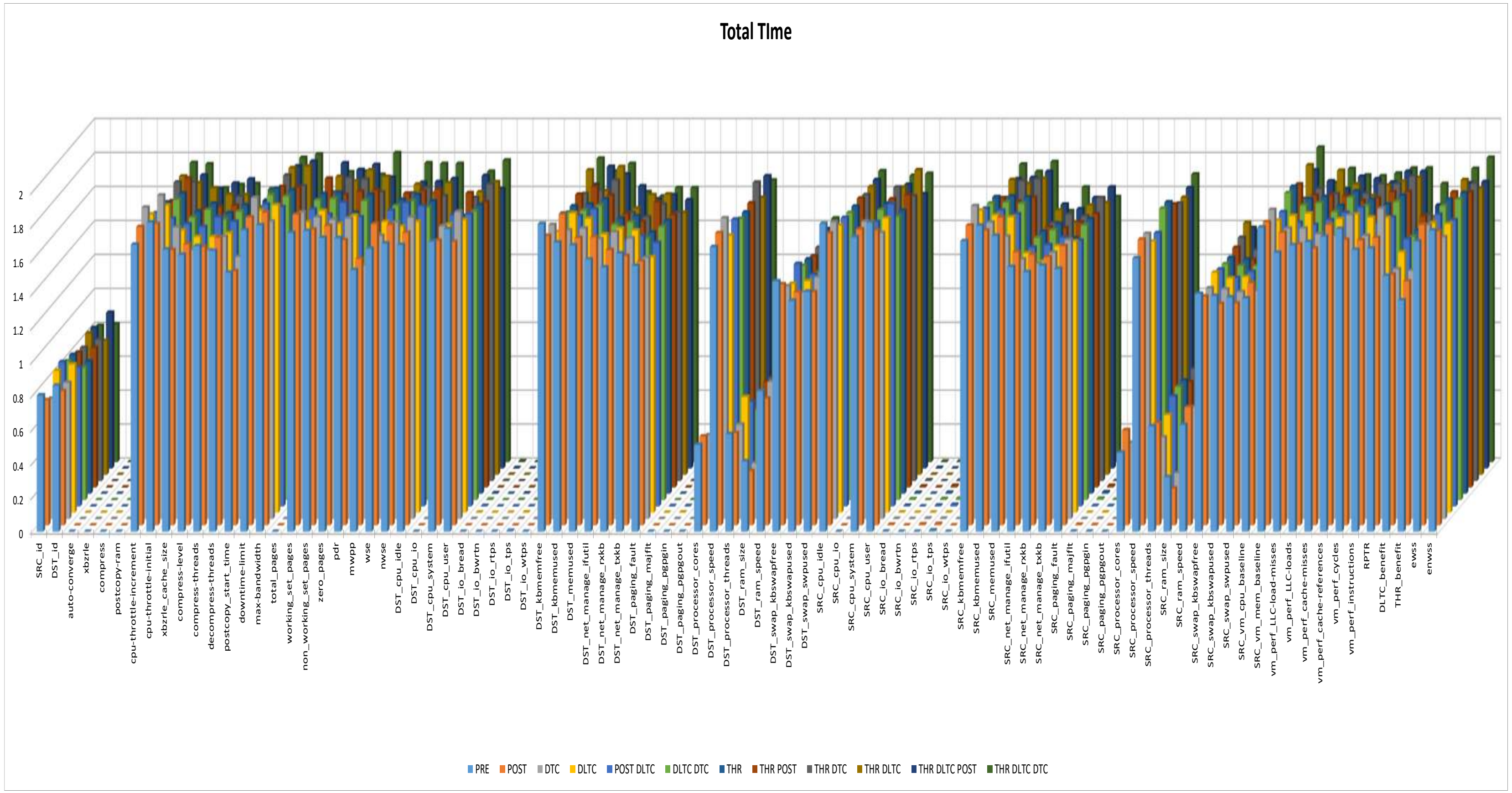


Figure B.1: Feature importance total time.

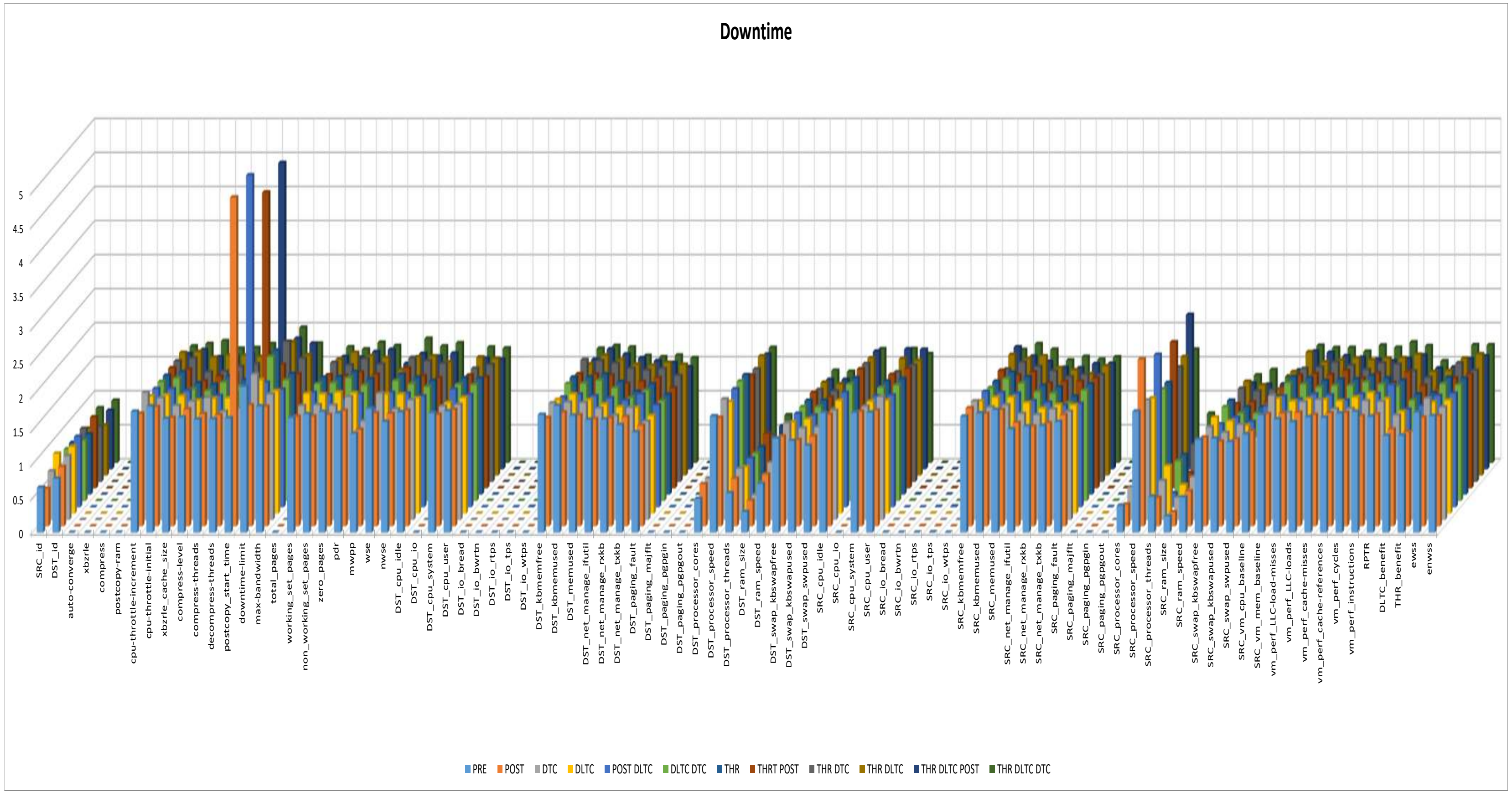


Figure B.2: Feature importance downtime.

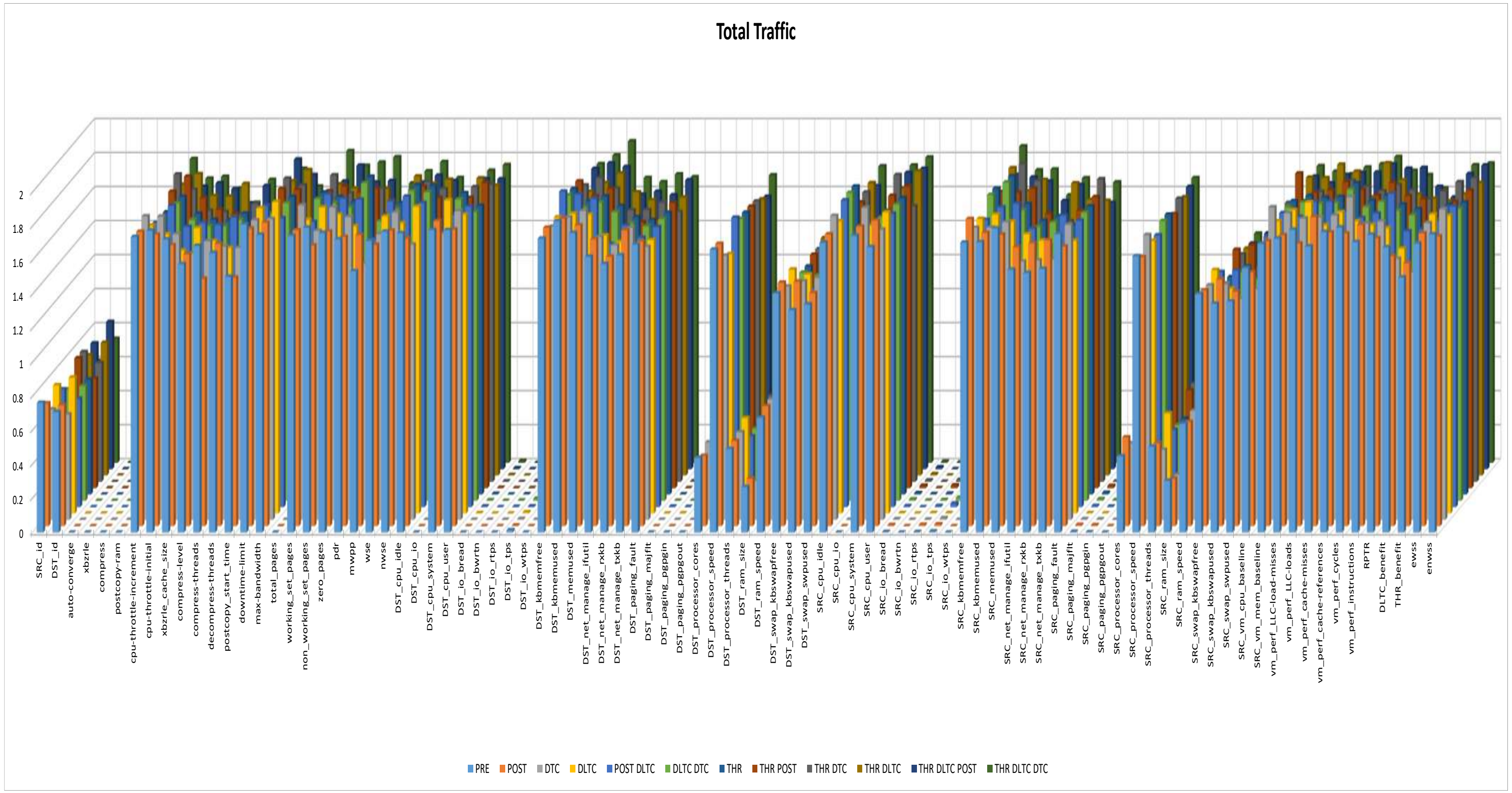


Figure B.3: Feature importance total traffic.

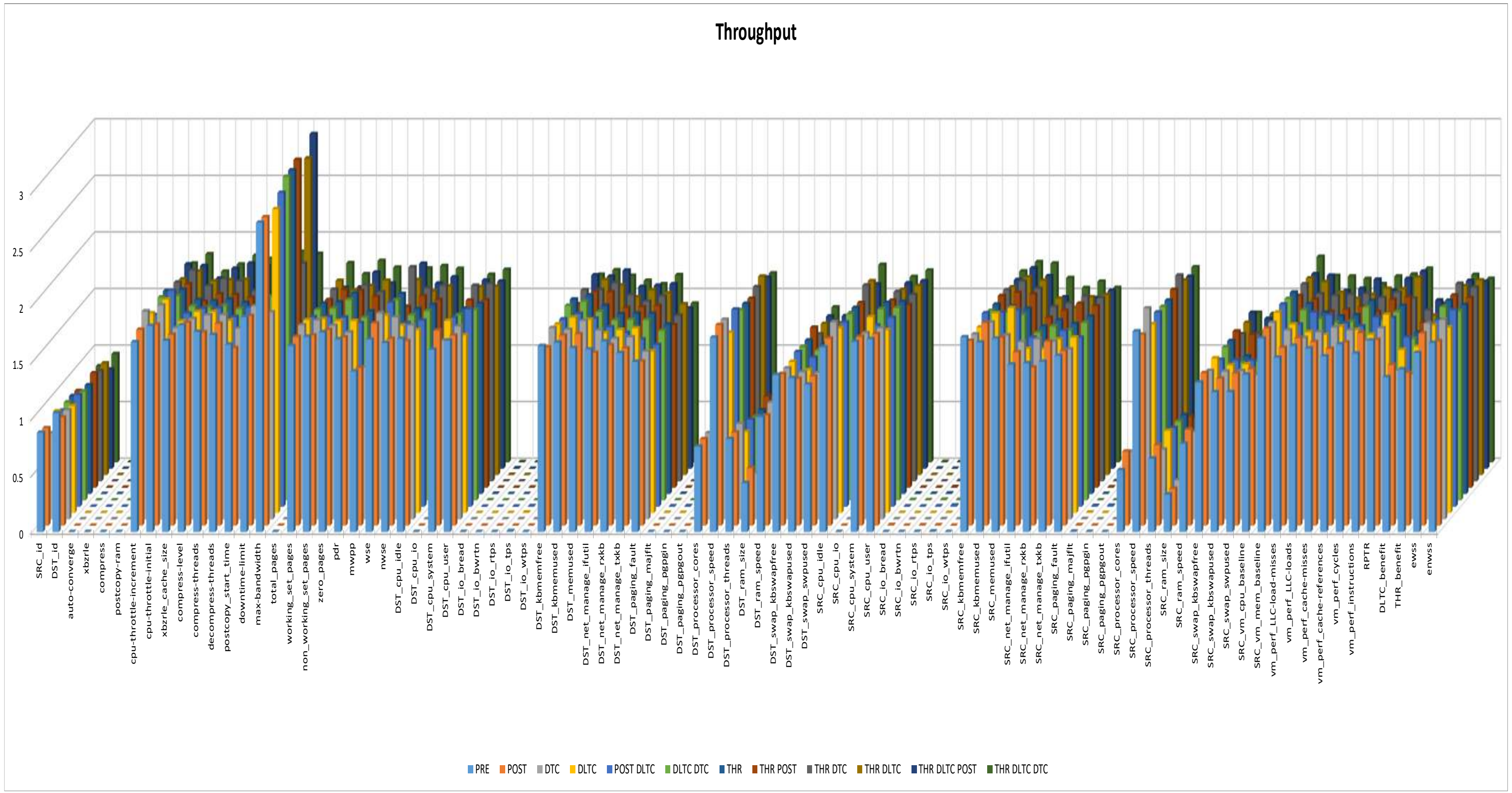


Figure B.4: Feature importance throughput.

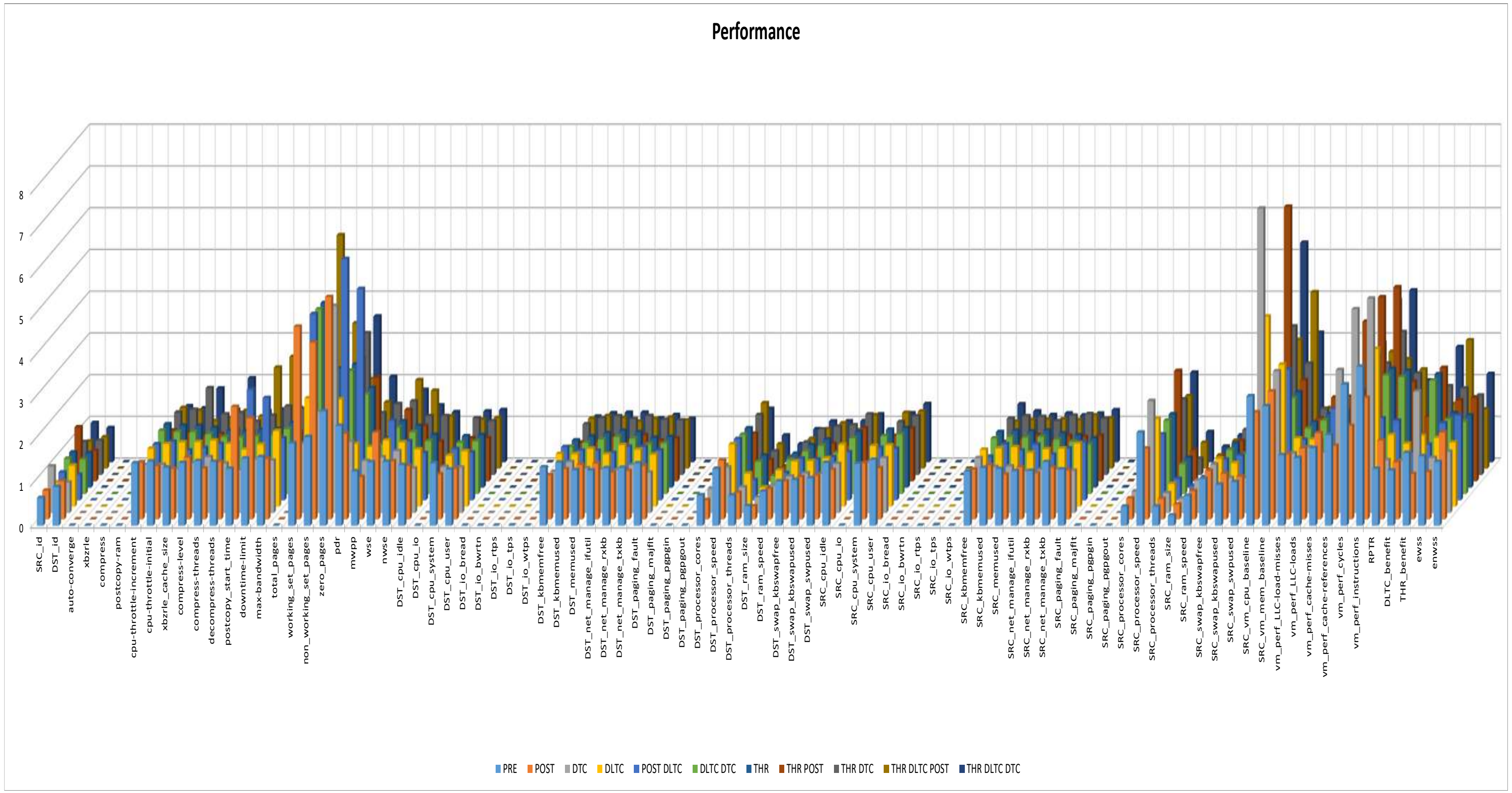


Figure B.5: Feature importance performance.

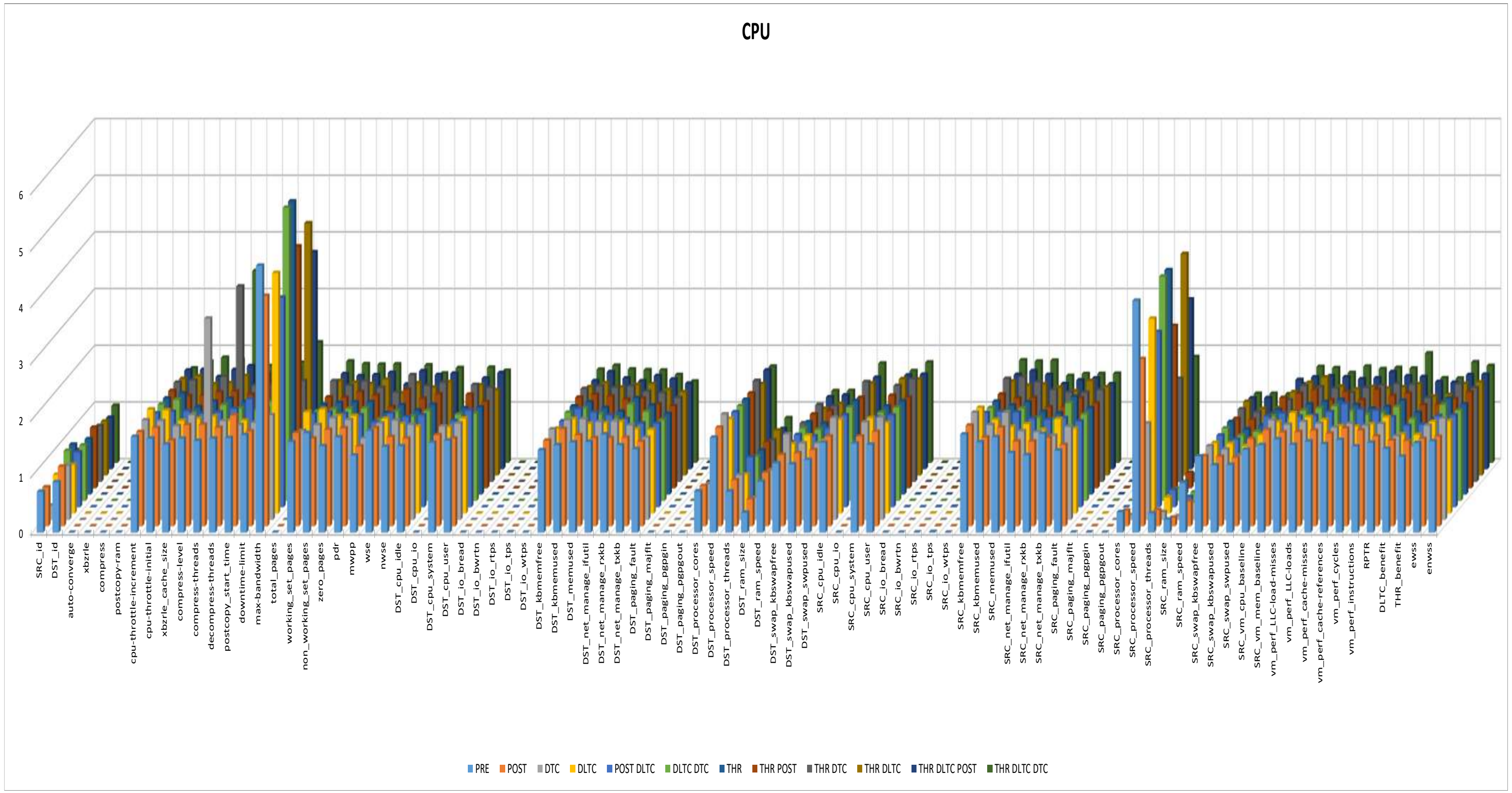


Figure B.6: Feature importance CPU.

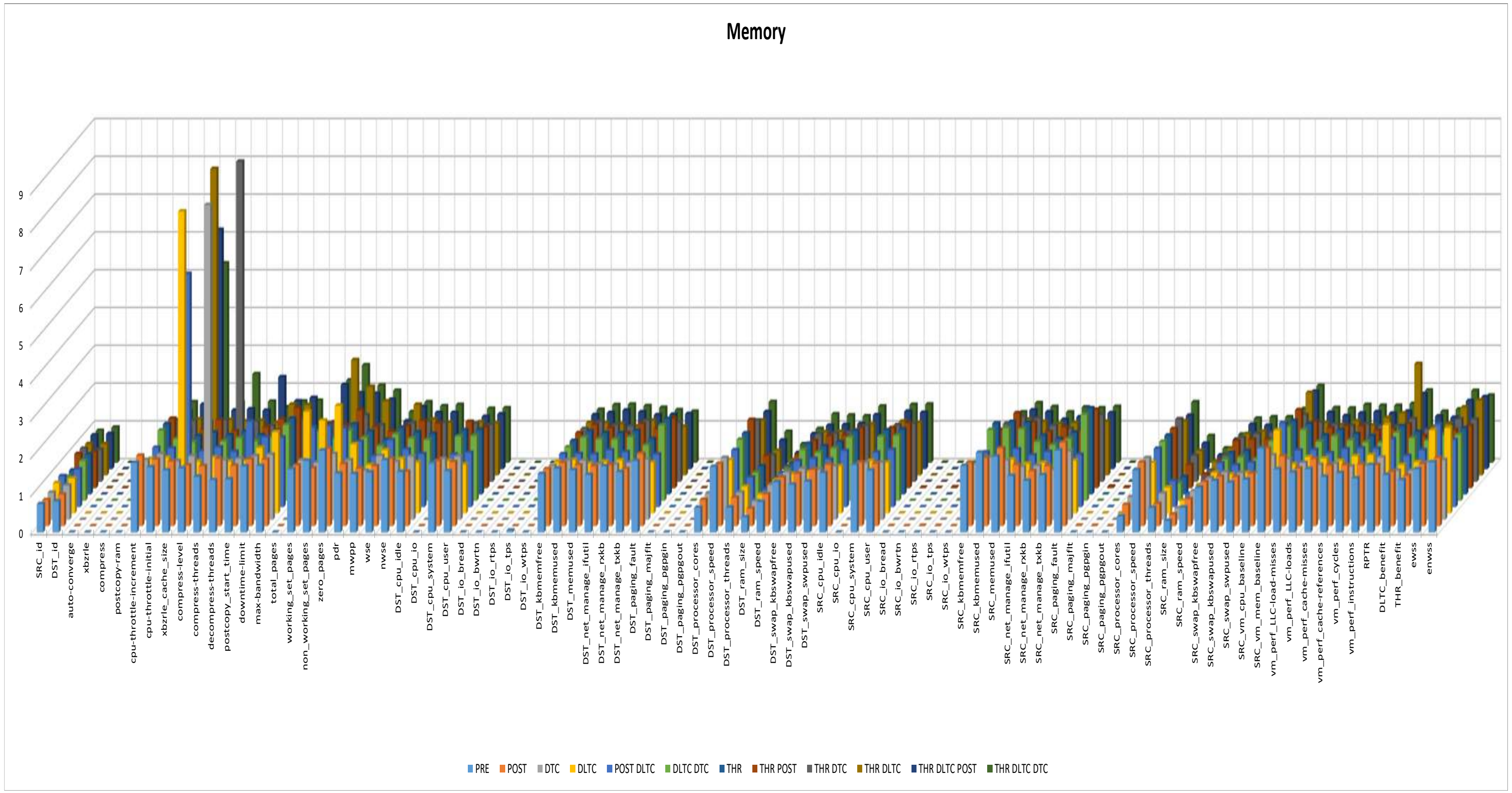


Figure B.7: Feature importance memory.

Bibliography

[1] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper. Predicting the performance of virtual machine migration. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, pages 37–46, Washington, DC, USA, 2010. IEEE Computer Society.

[2] A. Aldhalaan and D. A. Menascé. Analytic performance modeling and optimization of live vm migration. In M. S. Balsamo, W. J. Knottenbelt, and A. Marin, editors, *Computer Performance Engineering*, pages 28–42, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug 1996.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[5] L. Deng, H. Jin, H. Chen, and S. Wu. Migration cost aware mitigating hot nodes in the cloud. In *2013 International Conference on Cloud Computing and Big Data*, pages 197–204, Dec 2013.

[6] Feature importances with forests of trees. `http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html/`. Online; accessed on November.2018.

[7] Google compute engine uses live migration technology to service infrastructure without application downtime. `https://cloudplatform.googleblog.com`, Mar 2015. Online; accessed on November.2018.

[8] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. L. Lawall. Entropy: a Consolidation Manager for Clusters. In *VEE 2009 - 5th International Conference on Virtual Execution Environments*, pages 41–50, Washington, DC, United States, Mar. 2009. ACM.

[9] M. R. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 51–60, New York, NY, USA, 2009. ACM.

[10] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi. Reactive consolidation of virtual machines enabled by postcopy live migration. In *Proceedings of the 5th International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '11, pages 11–18, New York, NY, USA, 2011. ACM.

[11] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan. Live virtual machine migration with adaptive, memory compression. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, Aug 2009.

[12] C. Jo, Y. Cho, and B. Egger. A machine learning approach to live migration modeling. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 351–364, New York, NY, USA, 2017. ACM.

[13] C. Jo and B. Egger. Optimizing live migration for virtual desktop clouds. In *IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 1 of *CloudCom '13*, pages 104–111, Dec 2013.

[14] C. Jo, E. Gustafsson, J. Son, and B. Egger. Efficient live migration of virtual machines using shared storage. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '13, pages 41–50, New York, NY, USA, 2013. ACM.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[16] A. Koto, K. Kono, and H. Yamada. A guideline for selecting live migration policies and implementations in clouds. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 226–233, Dec 2014.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[18] J. Li, J. Zhao, Y. Li, L. Cui, B. Li, L. Liu, and J. Panneerselvam. imig: Toward an adaptive live migration method for kvm virtual machines. *The Computer Journal*, 58(6):1227–1242, 2015.

[19] H. Liu and B. He. Vmbuddies: Coordinating live migration of multi-tier applications in cloud environments. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):1192–1205, April 2015.

[20] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu. Live migration of virtual machine based on full system trace and replay. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, HPDC '09, pages 101–110, New York, NY, USA, 2009. ACM.

[21] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 171–182, New York, NY, USA, 2011. ACM.

[22] Z. Liu, W. Qu, W. Liu, and K. Li. Xen live migration with slowdown scheduling algorithm. In *2010 International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 215–221, Dec 2010.

[23] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer. Remedy: Network-aware steady state vm management for data centers. In R. Bestak, L. Kencl, L. E. Li, J. Widmer, and H. Yin, editors, *NETWORKING 2012*, pages 190–204, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[24] S. Nathan, U. Bellur, and P. Kulkarni. Towards a comprehensive performance model of virtual machine live migration. In *Proceedings of the Sixth*

ACM Symposium on Cloud Computing, SoCC '15, pages 288–301, New York, NY, USA, 2015. ACM.

[25] S. Nathan, U. Bellur, and P. Kulkarni. On selecting the right optimizations for virtual machine migration. In *Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '16, pages 37–49, New York, NY, USA, 2016. ACM.

[26] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. AGILE: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 69–82, San Jose, CA, 2013. USENIX.

[27] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 219–230, San Jose, CA, 2013. USENIX.

[28] D. Perez-Botero. A brief tutorial on live virtual machine migration from a security perspective. *University of Princeton, USA*, 2011.

[29] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[30] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, and T. Sanderson. Vm live migration at scale. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '18, pages 45–56, New York, NY, USA, 2018. ACM.

[31] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 5:1–5:14, New York, NY, USA, 2011. ACM.

[32] Sklearn standardscaler. `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html/`. Online; accessed on November.2018.

[33] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, pages 111–120, New York, NY, USA, 2011. ACM.

[34] P. Svärd, B. Hudzia, S. Walsh, J. Tordsson, and E. Elmroth. Principles and performance characteristics of algorithms for live vm migration. *SIGOPS Oper. Syst. Rev.*, 49(1):142–155, Jan. 2015.

[35] Python tensorflow tutorial - build a neural network. `http://adventuresinmachinelearning.com/python-tensorflow-tutorial/`, Sep 2018. Online; accessed on November.2018.

[36] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX Conference on Networked Systems Design &#38; Implementation*, NSDI'07, pages 17–17, Berkeley, CA, USA, 2007. USENIX Association.

[37] Y. Wu and M. Zhao. Performance modeling of virtual machine live migration. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 492–499, July 2011.

[38] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li. iaware: Making live migration of virtual machines interference-aware in the cloud. *IEEE Transactions on Computers*, 63(12):3012–3025, Dec 2014.

[39] J. Zhang, F. Ren, and C. Lin. Delay guaranteed live migration of virtual machines. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 574–582, April 2014.

[40] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu. Pacer: A progress management system for live virtual machine migration in cloud computing. *IEEE Transactions on Network and Service Management*, 10(4):369–382, December 2013.

요약

오늘날 데이터 센터에서 가상머신의 라이브 마이그레이션 기술은 매우 중요하게 사용된다. 현존하는 데이터 센터 관리 프레임워크에서는 복잡한 알고리즘을 이용하여 언제, 어디서, 어디로 가상머신의 마이그레션을 실행할지를 결정한다. 하지만 어떤 마이그레이션 방법을 사용하는지에 따라서 성능이 크게 차이가 날 수 있음에도 불구하고 이에 대한 논의는 주요하게 다뤄지지 않았다. 이러한 성능의 차이는 라이브 마이그레이션 알고리즘의 차이나 가상머신에 할당된 워크로드의 양의 차이 그리고 마이그레이션을 하는 곳과 목적 host의 상태 차이에 의하여 일어난다. 빠르고 정확하게 올바른 마이그레이션 방법을 정하는 것은 필수적인 과제이다. 이러한 과제를 performance model을 이용하여 해결할 것이다.

본 논문에서는, 가상머신의 라이브 마이그레이션 성능을 예측하는 여러 머신 러닝 모델을 제시한다. 여기서 12개의 서로 다른 마이그레이션 알고리즘에 대해 7가지의 다른 metric들을 예측한다. 이 모델은 기존 연구에 비해 훨씬 정확한 예측을 성공하였다. 각각의 target metric과 여러 알고리즘들에 대하여 input feature evaluation을 수행하였고 각각의 특성에 맞는 모델을 만들어 84개의 서로다른 머신 러닝 모델들을 훈련시켰다. 이러한 모델들은 실제 라이브 마이그레이션 프레임워크에 쉽게 적용 가능하다. 각각의 마이그레이션 알고리즘에 대하여 target metric 예측을 사용함으로써 올바른 마이그레이션 알고리즘을 쉽게 결정할 수 있고 이는 결과적으로 다운타임과 마이그레이션에 소요되는 총 시간의 감소 효과를 볼 수 있다.

주요어: 머신 러닝, 라이브 마이그레이션, 가상화, 가상머신

학번: 2017-28499

Acknowledgements

First of all, I would like to thank the National Institute for International Education for giving me this incredible opportunity to study and graduate in Korea.
I would also like to thank my advisor Professor Bernhard Egger of the Department of Computer Science and Engineering at Seoul National University to give me the opportunity to be part of his lab. During my stay in the lab and while writing this thesis Professor Egger was always giving me the right direction and advises whenever he thought I needed it.
I would also like to thank Changyeon and Youngsu for sharing with me all their knowledge and experience related to virtualization, live migration, and SoCC'17. I would like to thank also to Inpyo and Barend for answering all my questions and helping me understand better machine learning. To Minsu, Richard, Jiwoong, and Hyemi I am grateful for the great time that we had together as teacher assistants of system programming. Even though we have not talked much, I would like to thank Younghyun for being such a hardworking and ambitious person who inspired me during my stay in the lab.
I would also like to thank Camilo for all the help he gave me as a friend and a csap member, I am also grateful for his valuable comments on this thesis.
Finally, I must express my very profound gratitude to my family and to all my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.